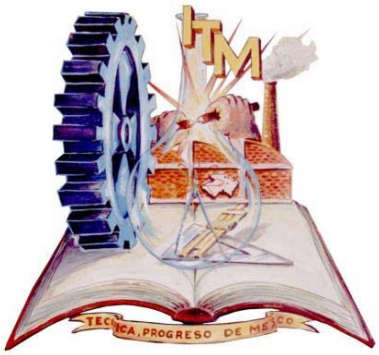


INSTITUTO TECNOLÓGICO DE MORELIA

José Ma. Morelos y Pavón

Departamento de Sistemas y Computación

PROTOTIPO DIDÁCTICO PARA LA MATERIA DE PROGRAMACIÓN WEB

Que para obtener el Título de

Ingeniero en Sistemas Computacionales

presenta

JULIO CÉSAR HERNÁNDEZ ZARAGOZA

Asesor: Ing. José Ma. Zepeda Florián

Morelia, Michoacán Agosto de 2007

A mi mamá, antes que todo por la vida, por el amor y el cariño, por el esfuerzo, por el sacrificio, por la felicidad y por el orgullo inmenso que siento de ser tu hijo.

A Lila, por el apoyo incondicional y el enorme ejemplo de vida que me ha enseñado a nunca dejarme vencer y salir adelante contra todos los obstáculos que me ponga la vida.

A Mami Gu, por estar siempre con sus hijos y ser para sus hijos, por el esfuerzo y la lucha que nunca terminan y nunca se vencen, por ser el cariño que une a nuestra bella familia.

A Gaby por llegar a mi vida, por darme apoyo, por ayudarme, por ser la felicidad que ilumina mi existir.

A ti… I.Z.R.

INDICE

INTRODUCCIÓN

El interés y la demanda por Internet crece y crece, y el uso de servicios como World Wide Web (www), Internet Mail, Telnet, File Transfer Protocol (FTP) y el almacén de información en bases de datos es cada vez más popular. Cada día más y más aplicaciones son desarrolladas sobre plataforma Web debido a la disponibilidad que ofrece tal tecnología.

Son varios los lenguajes y herramientas disponibles para el desarrollo de aplicaciones Web y cada uno de ellos ofrece funcionalidades muy diferentes uno del otro, por ejemplo, si queremos desarrollar un sistema de ventas por internet no hay nada mejor que utilizar PHP con MySQL, esto debido a que el lenguaje PHP funciona de manera muy transparente con el SMBD MySQL, además, con PHP se tiene un fácil control sobre los componentes HTML de una página y algo muy importane, el costo de los servidores para PHP con MySQL es muy bajo, por otro lado, si queremos hacer un sistema Web que esté realizando cálculos o mediciones en tiempo real no hay nada mejor que utilizar JSP o Servlets, ya que ambos lenguajes nos ofrecen el enorme mundo de posibilidades del lenguaje Java (clases, objetos, encapsulación, polimorfismo, herencia, etc.), en especial los Servlets.

Sin embargo, cada uno de los lenguajes para el desarrollo de aplicaciones Web manipulan la información de manera diferente, cada uno de ellos tiene sus propios métodos y funciones para realizar una misma tarea y en ocasiones los métodos que son muy fáciles en un lenguaje en particular, son mucho más complicados en otro.

La manipulación y validación de datos, el almacén de información y la seguridad son las principales características de los sistemas, por lo cual se desarrolló el presente trabajo que ilustra de manera clara y concisa los aspectos señalados anterioremente con los diferentes lenguajes para el desarrollo de aplicaciones Web. Este trabajo fue desarrollado haciendo uso de la experiencia obtenida con el sistema SIBAINES.

El desarrollo del Sistema Básico de Información Estadística “SIBAINES” fué propuesto por el Colegio de Estudios Científicos y Tecnológicos del Estado de Michoacán (CECyTEM) debido a la necesidad de contar con un sistema que permitiera el manejo de la información estadística que se requiere y genera por dicha institución.

Se planteó que dicho sistema debía estar disponible en todo momento, ser de fácil acceso y uso, y debe poder ser utilizado por todo el personal directivo y de planeación del CECyTEM así como por todos los planteles que forman parte de este subsistema educativo que se encuentran dispersos por todo el estado de Michoacán. La tecnología que nos da la capacidad de cubrir tales características y que es en la cual el sistema fue desarrollado es la tecnología Web, pero ésta presenta algunos problemas de seguridad, por tal razón se investigaron y desarrollaron algunos medios para asegurar los datos, validar información, cifrar contenidos, cifrar scripts del sistema, replicar BD, etc. los cuales se describen a lo largo del presente así como el funcionamiento e interacción de los diferentes lenguajes Web con los SMBD más utilizados y de distribución gratuita.

JUSTIFICACIÓN

Cada día los sistemas basados en tecnología web son más utilizados debido a la facilidad con la que cualquier persona desde cualquier lugar del mundo entero puede acceder a las funcionalidades para las cuales fueron programados. Por lo cual es importante proporcionar a los desarrolladores de este tipo de aplicaciones una guía base para comprender de manera sencilla aspectos que en varias ocasiones son difíciles de comprender.

Cada lenguaje utilizado para el desarrollo de aplicaciones Web manipula de manera distinta a la información, en ocasiones la documentación existente para hacer una tarea como la conexión con un SMBD no deja en claro muchos aspectos que se deben considerar o también muchos de ellos no se ejemplifican correctamente.

El uso de la tecnología no debe estar limitado al conocimiento que se tenga sobre ella sino a la flexibilidad que nos brinda el hacer uso de alguna herramienta para realizar una tarea.

Si la solución a un problema se realiza de manera mucho más sencilla en cierto lenguaje es mejor aprender a utilizar dicho lenguaje que intentar forzar a otro que ya conocemos para que también lo haga, por tal razón se desarrolla el presente trabajo que ilustra de manera clara y a base de ejemplos sencillos de comprender las diferentes formas para realizar validaciones de datos, conexiones a sistemas manejadores de bases de datos, ejecución de consultas, manejo de errores con el SMBD, empleo de sesiones y empleo de cookies para cada uno de los diferentes lenguajes de programación Web, así como protección de scripts del sistema en el servidor y cifrado de datos utilizando MD5, DES y 3DES con PHP, replicación de BD con MySQL y configuración de conexiones seguras utilizando SSL con la finalidad de comprender mejor los aspectos vistos en las materias que involucran programación en Web y materias relacionadas como base de datos para desarrollar sistemas cada vez más seguros y toleantes a fallos pero sobre todo con el conocimiento de la tecnología adecuada.

OBJETIVOS

- Dotar a la comunidad académica con la documentación, ejemplificada y probada, para la mejor comprensión de los temas cubiertos en la materia de Programación Web, así como en cualquier materia de Bases de Datos y Programación Distribuida ya que los ejemplos mostrados aplican para diversas actividades en estas áreas.

- Contar con un material didáctico en el cual se describan y ejemplifiquen de manera clara e interactiva los elementos necesarios para poder desarrollar sistemas seguros, íntegros y tolerantes a fallos sobre plataforma WEB.

- Adquirir los conocimientos necesarios para asegurar la calidad y confidencialidad de la información desde que se establece la comunicación con un "posible usuario" hasta que se almacenan los datos en la BD.

- Con el desarrollo e inclusión de las funciones de cifrado y descifrado de datos así como de cifrado de archivos del servidor se beneficia a los usuarios y empresas dedicadas al desarrollo de aplicaciones WEB con la misma tecnología, ya que estas funciones pueden ser instanciadas o pueden servir como base para implementaciones o mejoras propias.

- Entrega de un disco con ejemplos y documentos utilizados para ilustrar el uso de dichos mecanismos de seguridad, librerías de cifrado y descifrado de datos, configuraciones de replicado multilineal de bases de datos, conexiones con los diferentes SMBD y obtención de certificados digitales.

- Todo lo anterior basándose en un sistema probado y funcional (Sistema Básico de Información Estadística "SIBAINES") desarrollado para el CECyTEM, el cuál fue desarrollado en PHP, Javascript y MySQL pero los ejemplos serán traducidos a las diferentes tecnologías y lenguajes.

ALCANCES Y LIMITACIONES

Se describirá de manera general el esquema y funcionamiento de la tecnología WEB, la diferencia entre el código del servidor y el código del cliente, la manera en que se envían los datos de una entidad a otra y la forma en la cual el código del servidor puede ejecutar código del cliente y viceversa, así como la manera en la cual se puede hacer el paso de elementos variables y objetos entre el cliente y el servidor, todo esto, tomando como ejemplo los códigos de SIBAINES. También, se describirá la forma de instalar y configurar los servidores con AppServ, IIS y Tomcat así como el servidor FTP para la actualización de los archivos del servidor.

Se especificarán, describirán y ejemplificarán los mecanismos de seguridad y validación con los que cuenta el lenguaje Javascript para garantizar una comunicación segura con el servidor.

Se especificarán, describirán y ejemplificarán los mecanismos de seguridad y validación con los que cuentan PHP, JSP, ASP y los Servlets para garantizar una comunicación segura con el cliente, mecanismos tales como el manejo de sesiones, cookies, el cifrado lineal propio de PHP, cifrado y descifrado en DES y 3DES programado para SIBAINES, autenticación y validación de usuarios, manejo de la bitácora de la BD (para utilizar BEGIN, COMMIT y ROLLBACK) en caso de una detección de fallos en el SMBD, etc.

Se describirán y ejemplificarán las diferentes formas de conexión que existen, dependiendo del leguaje o tecnología utilizados, con algún SMDB (MySQL principalmente pero también PostgreSQL con PHP y ASP así como ASP). Así mismo, se describirán y ejemplificarán los métodos para realizar las consultas en el SMBD así como los objetos o tipos de datos que regresan para manipular la información que regresa una consulta tanto incorrecta como correctamente ejecutada.

Se describirá la forma de obtener certificados digitales para realizar el envío de datos por un canal inseguro y evitar que a pesar del sniffing o robo de paquetes la información sustraída sea útil. Además, se describirá la forma de proteger los scripts (que son el sistema en sí) para evitar que cuando los archivos, códigos, clases, rutinas o funciones sean copiadas sin autorización sean de alguna utilidad al raptor. Por último, Se especificará la forma en la cual obtener replicación de Bases de datos en MySQL de manera lineal (un solo sentido), en espejo (ambos sentidos) o multilineal (varias BD en diferentes sentidos: RAID), para que a pesar de un fallo en alguno de los servidores el sistema siga funcionando, así mismo, se especificará la forma en la cual establecer tiempos para reintentar la comunicación con el servidor caído.

CAPITULO I.- INTRODUCCIÓN A LA TECNOLOGÍA WEB

La web se encuadra dentro de Internet, no es más que un servicio de los muchos que presta la Red, entre los que podemos encontrar

- Correo electrónico
- IRC o chat
- FTP
- El propio web

El sistema con el que está construido el web se llama hipertexto y es un conjunto de páginas conectadas con enlaces.

Los sistemas de hipertexto se utilizan en otros contextos aparte del web, como la ayuda del Windows. Son muy fáciles de utilizar y también es muy fácil encontrar lo que buscamos rápidamente, gracias a que con enlaces vamos accediendo a la información que más nos interesa.

Páginas estáticas y páginas dinámicas

En la web podemos encontrar, o construir, dos tipos de páginas:

- Las que se presentan sin movimiento y sin funcionalidades más allá de los enlaces.
- Las páginas que tienen efectos especiales y en las que podemos interactuar.

Las primeras páginas son las que denominamos páginas estáticas, se construyen con el lenguaje HTML, que no permite crear efectos ni funcionalidades más allá de los enlaces.

Estas páginas son muy sencillas de crear, aunque ofrecen pocas ventajas tanto a los desarrolladores como a los visitantes, ya que sólo se pueden presentar textos planos acompañados de imágenes y a lo sumo contenidos multimedia como pueden ser videos o sonidos

El segundo tipo de páginas se denomina página dinámica. Una página es dinámica cuando se incluye cualquier efecto especial o funcionalidad y para ello es necesario utilizar otros lenguajes de programación, aparte del simple HTML.

Páginas dinámicas

Una página es dinámica cuando realiza efectos especiales o implementa alguna funcionalidad o interactividad.

Además, para programar una página dinámica necesitaremos otros lenguajes aparte del HTML. Sin embargo, nunca hay que olvidarse del HTML, ya que éste es la base del desarrollo web: generalmente al escribir una página dinámica el código de los otros lenguajes de programación se incluye embebido dentro del mismo código HTML.

Una razón por la que se construyen páginas dinámicas es la simple vistosidad que pueden alcanzar los

trabajos, ya que se pueden hacer presentaciones más entretenidas de las que se consiguen utilizando únicamente HTML. Pero vamos a ver con calma algunas razones menos obvias pero más importantes.

Supongamos que hemos decidido realizar un portal de televisión donde una de las informaciones principales a proveer podría ser la programación semanal. Efectivamente, esta información suele ser dada por las televisiones con meses de anticipación y podría ser fácilmente almacenada en una base de datos. Si trabajáramos con páginas HTML, tendríamos que construir una página independiente para cada semana en la cual introduciríamos "a mano" cada uno de los programas de cada una de las cadenas. Asimismo, cada semana nos tendríamos que acordar de descolgar la página de la semana pasada y colgar la de la actual. Todo esto podría ser fácilmente resuelto mediante páginas dinámicas. En este caso, lo que haríamos sería crear un programa (solo uno) que se encargaría de recoger de la base de datos de la programación aquellos programas que son retransmitidos en las fechas que nos interesan y de confeccionar una página donde aparecerían ordenados por cadena y por hora de retransmisión. De este modo, podemos automatizar un proceso y desentendernos de un aspecto de la página por unos meses.

Páginas dinámicas del cliente

Son las páginas dinámicas que se procesan en el cliente. En estas páginas toda la carga de procesamiento de los efectos y funcionalidades la soporta el navegador.

Usos típicos de las páginas de cliente son efectos especiales para webs como el control de ventanas, presentaciones en las que se pueden mover objetos por la página, control de formularios, cálculos, etc.

El código necesario para crear los efectos y funcionalidades se incluye dentro del mismo archivo HTML y es llamado SCRIPT. Cuando una página HTML contiene scripts de cliente, el navegador se encarga de interpretarlos y ejecutarlos para realizar los efectos y funcionalidades.

Las páginas dinámicas del cliente se escriben en dos lenguajes de programación principalmente: Javascript y Visual Basic Script (VBScript), este último no es utilizado comúnmente debido a que su uso se limita a la tecnología Microsoft.

Las páginas del cliente son muy dependientes del sistema donde se están ejecutando y esa es su principal desventaja, ya que cada navegador tiene sus propias características, incluso cada versión, y lo que puede funcionar en un navegador puede no funcionar en otro.

Como ventaja se puede decir que estas páginas descargan al servidor algunos trabajos, ofrecen respuestas inmediatas a las acciones del usuario y permiten la utilización de algunos recursos de la máquina local.

Páginas dinámicas del servidor

Podemos hablar también de páginas dinámicas del servidor, que son reconocidas, interpretadas y ejecutadas por el propio servidor justo antes de que se envíe la página a través de Internet al cliente. Las páginas que se ejecutan en el servidor pueden realizar accesos a bases de datos, conexiones en

red, y otras tareas para crear la página final que verá el cliente. El cliente solamente recibe una página con el código HTML resultante de la ejecución.

Estas páginas son útiles en muchas ocasiones. Con ellas se puede hacer todo tipo de aplicaciones web. Desde agendas, foros, sistemas de documentación, estadísticas, juegos, chats, etc. Son especialmente útiles en trabajos que se tiene que acceder a información centralizada, situada en una base de datos en el servidor, y cuando por razones de seguridad los cálculos no se pueden realizar en la computadora del usuario.

Las páginas dinámicas del servidor se suelen escribir en el mismo archivo HTML, mezclado con el código HTML, al igual que ocurría en las páginas del cliente. Cuando una página es solicitada por parte de un cliente, el servidor ejecuta los scripts y se genera una página resultado, que solamente contiene código HTML. Este resultado final es el que se envía al cliente y puede ser interpretado sin lugar a errores ni incompatibilidades, puesto que sólo contiene HTML

Para escribir páginas dinámicas de servidor existen varios lenguajes, como son Common Gateway Interface (CGI) comúnmente escritos en Perl, Active Server Pages (ASP), Hipertext Preprocesor (PHP), y Java Server Pages (JSP).

Las ventajas de este tipo de programación son que el cliente no tiene acceso a los scripts, ya que se ejecutan y transforman en HTML antes de enviarlos. Además son independientes del navegador del usuario, ya que el código que reciben es HTML fácilmente interpretable.

La siguiente figura corresponde al esquema del funcionamiento general de la tecnología Web, en ella se puede apreciar el ciclo que forman la solicitud, ejecución y envío de los datos a través de la red cada vez que accedemos a una página, un enlace, descargamos contenido, etc.

Figura 1.1 Esquema General de la Tecnología Web

Lenguaje HTML

Una página Web la vemos en nuestro navegador, o Cliente web, y parece una sola entidad, pero no es así, está compuesta por multitud de diferentes archivos, como son las imágenes, los vídeos y lo más importante: el código fuente.

El código de las páginas está escrito en un lenguaje llamado HTML, que indica básicamente donde colocar cada texto, cada imagen o cada video y la forma que tendrán estos al ser colocados en la página.

El HTML se creó en un principio con objetivos divulgativos. No se pensó que la web llegara a ser un área de ocio con carácter multimedia, de modo que, el HTML se creó sin dar respuesta a todos los posibles usos que se le iba a dar y a todos los colectivos de gente que lo utilizarían en un futuro.

El lenguaje consta de etiquetas que tienen esta forma <B> o <P>. Cada etiqueta significa una cosa, por ejemplo <B> significa que se escriba en negrita (bold) o <P> significa un párrafo, <A> es un enlace, etc. Casi todas las etiquetas tienen su correspondiente etiqueta de cierre, que indica que a partir de ese punto no debe de afectar la etiqueta. Por ejemplo </B> se utiliza para indicar que se deje de escribir en negrita. Así que el HTML no es más que una serie de etiquetas que se utilizan para definir la forma o estilo que queremos aplicar a nuestro documento.

Un documento HTML ha de estar delimitado por la etiqueta <html> y </html>. Dentro de este documento, podemos asimismo distinguir dos partes principales:

- El encabezado, delimitado por <head> y </head> donde colocaremos etiquetas de índole informativo como por ejemplo el titulo de nuestra página.
- El cuerpo, flanqueado por las etiquetas <body> y </body>, que será donde colocaremos nuestro texto e imágenes delimitados a su vez por otras etiquetas como las que hemos visto.

Construcción de Páginas con HTML

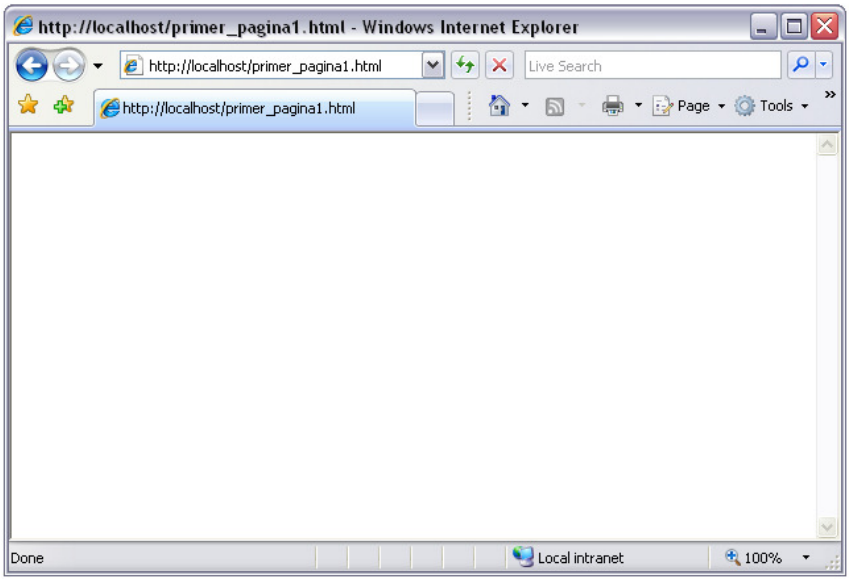
Como se mencionó anteriormente, el código HTML se utiliza sólo para indicar al navegador las posiciones y los tipos de elementos que nuestra página web va a contener. Sin embargo, el contenido HTML puede ser manipulado por lenguajes como PHP, JSP o los Servlets que serán vistos más adelante.

La página más simple en HTML podría ser la siguiente:

```
1       <html>
2
3       <head>
4       </head>
5
6       <body>
7       </body>
8
9       </html>
```

Archivo primer_pagina1.html, La más simple página HTML

Una página HTML se divide en dos secciones principales, el **head** y el **body**, dentro de las etiquetas <head> y </head> se colocan códigos que se ejecutan en el navegador del cliente (generalmente escritos en Javascript) y del encabezado de la página (por ejemplo el título de la misma), dentro de las etiquetas <body> y </body> se colocan los elementos de nuestra página web como son los campos de texto, los de contraseña, las imágenes, las animaciones, las listas desplegables, etc. y todos estos códigos siempre deben estar dentro de las etiquetas <html> y </html>. El código anterior nos genera la siguiente página de internet:



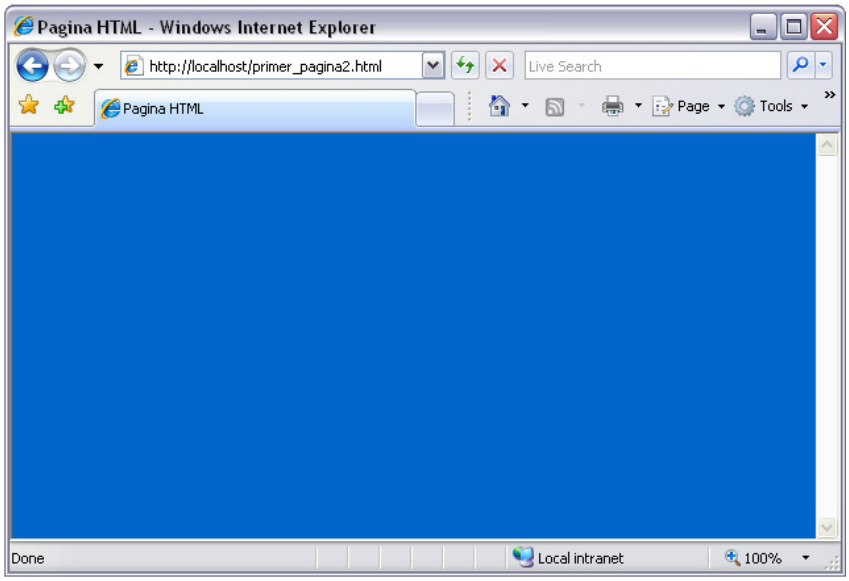
Ejecución del archivo primer_pagina1.html

Para añadir un color de fondo y un título a nuestra página web debemos incluir las etiquetas siguientes al código anterior:

```
1       <html>
2
3       <head>
4       <title>Pagina HTML</title>
5       </head>
6
7       <body bgcolor="#0066CC">
8       </body>
9
10      </html>
```

Archivo primer_pagina2.html, Página HTML con título y color de fondo

Y el resultado en pantalla sería el siguiente:



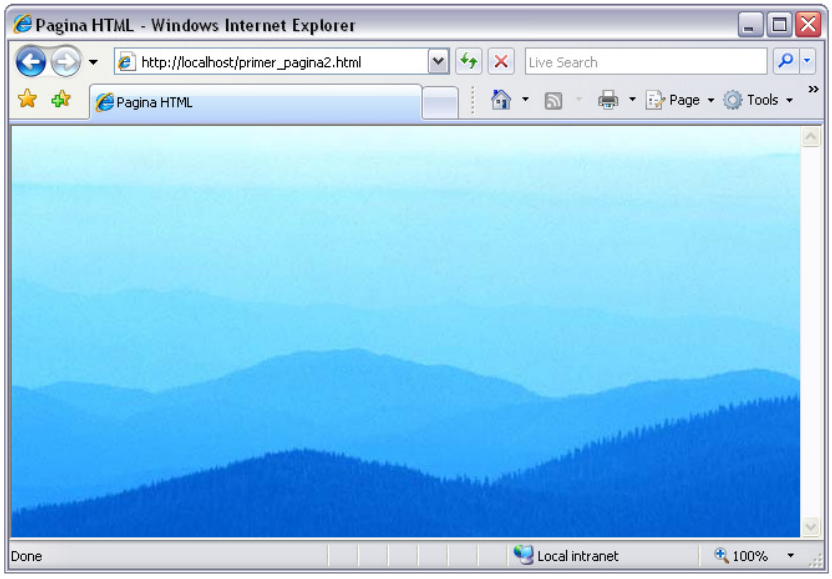
Ejecución del archivo primer_pagina1.html

Para agregar una imagen de fondo debemos agregar lo siguiente:

```
1       <html>
2       <head>
3       <title>Pagina HTML</title>
4       </head>
5
6       <body background="Blue hills.jpg"> //la ruta de la imagen debe ser la misma que la de la pagina
7       </body>
8
9       </html>
```

Archivo primer_pagina3.html, Incluyendo una imagen de fondo

Y el resultado en pantalla sería el siguiente, se debe notar que si el tamaño de la imagen es inferior al de la página, la imagen se copiará a si misma seguida una de la otra hasta “llenar” el tamaño total de la pantalla, similar a cuando se coloca un papel tapiz de mosaico en nuestro escritorio de la PC.



Ejecución archivo primer_pagina3.html

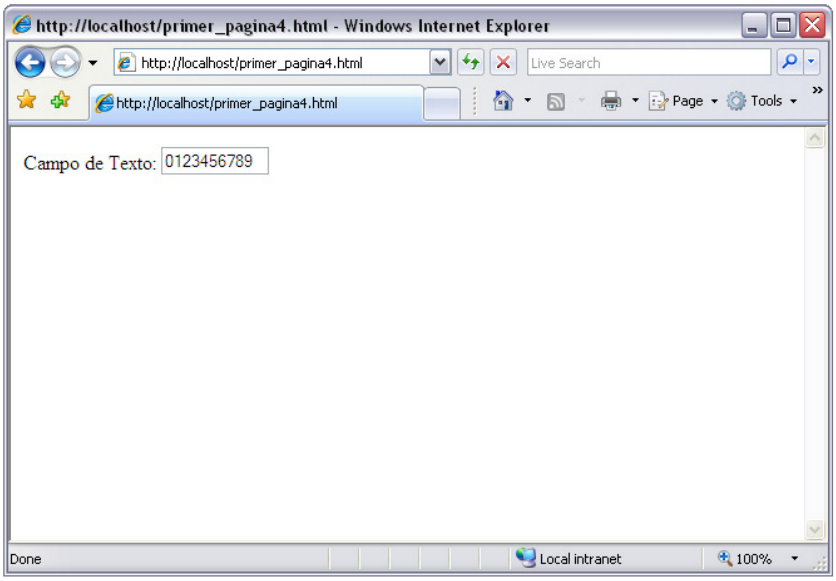
Para agregar un campo de texto debemos colocar las etiquetas siguientes:

```
1       <html>
2
3       <head>
4       </head>
5
6       <body>
7       <form name="form" method="post" action="mi_otra_pagina.html">
8         Campo de Texto: <input type="text" id="mi_campo" name="mi_campo" size="10" maxlength="50">
9       </form>
10      </body>
11
12      </html>
```

Archivo primer_pagina4.html, Página HTML con un campo de texto y formulario

Las etiquetas <form> y </form> indican que **todo** el código correspondiente a los elementos de nuestra página HTML que se encuentren dentro de estas etiquetas serán enviados a otra página (mi_otra_pagina.html) con la finalidad de analizar o validar sus valores, pero dichas validaciones se hacen sólo con códigos PHP, JSP, Servlets, ASP, etc.

En el ejemplo anterior, incluimos el código correspondiente a un campo de texto el cual se llama "mi_campo" y admite como máximo una cadena de 50 caracteres pero tiene un ancho de solo 10 caracteres. En pantalla tendremos lo siguiente:



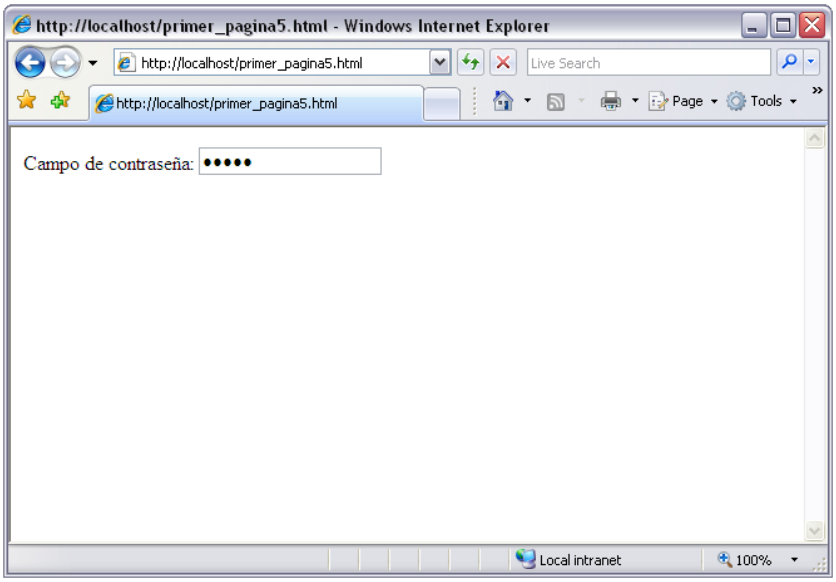
Ejecución archivo primer_pagina4.html

Para agregar un campo de tipo contraseña debemos colocar las etiquetas siguientes:

```
1          <html>
2
3          <head>
4          </head>
5
6          <body>
7          <form name="form" method="post" action="mi_otra_pagina.html">
8           Campo de contrase&ntilde;a: <input type="password" id="mi_contrasena" name="mi_contrasena" size="20"
maxlength="20">
9          </form>
10         </body>
11
12         </html>
```

Archivo primer_pagina5.html, Página HTML con un campo de contraseña

En el ejemplo anterior, incluimos el código correspondiente a un campo de tipo contraseña el cual se llama “mi_contraseña”, admite como máximo una cadena de 20 caracteres y tiene un ancho de 20 caracteres. La forma de validar y disponer de los datos introducidos en dichos campos (ya sea con SCRIPTS del lado del cliente o del servidor) es exactamente la misma que para los campos de texto normales (type=”text”). En pantalla tendremos lo siguiente:



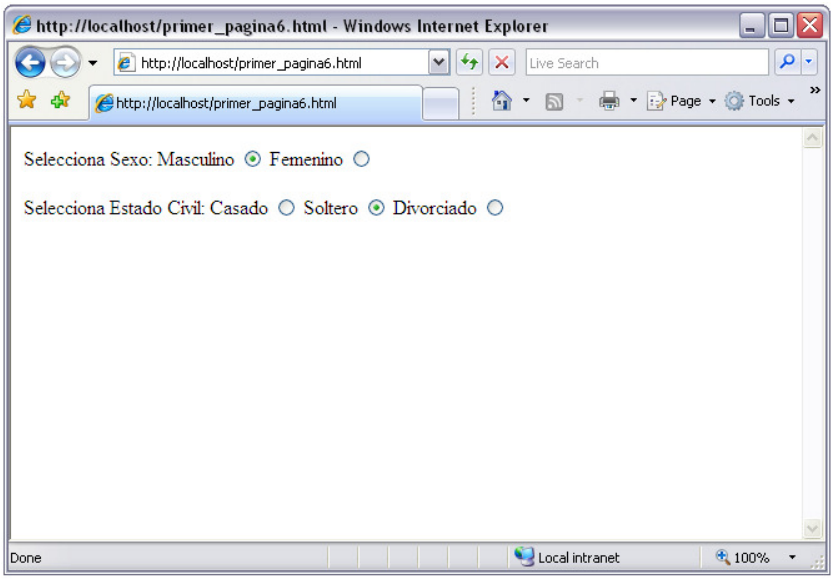
Ejecución archivo primer_pagina5.html

Para agregar un componente de tipo Radio Button debemos colocar las etiquetas siguientes:

```
1       <html>
2       <head>
3       </head>
4
5       <body>
6       <form name="form" method="post" action="mi_otra_pagina.html">
7               <p>
8                       Selecciona Sexo:
9                       Masculino
10                      <input name="sexo" id="sexo" type="radio" value="M">
11                      Femenino
12                      <input name="sexo" id="sexo" type="radio" value="F">
13              </p>
14              <p>
15                      Selecciona Estado Civil:
16                      Casado
17                      <input name="edo_civil" id="edo_civil" type="radio" value="C">
18                      Soltero
19                      <input name="edo_civil" id="edo_civil" type="radio" value="S">
20                      Divorciado
21                      <input name="edo_civil" id="edo_civil" type="radio" value="D">
22              </p>
23      </form>
24      </body>
25      </html>
```

Archivo primer_pagina6.html, Página HTML con Radio Buttons

En el ejemplo anterior se incluyó el código correspondiente a cinco componentes de tipo Radio Button, los primeros dos de ellos tienen el mismo nombre (sexo) y por lo tanto entre ellos solo se permite una selección, los siguientes tres tienen el mismo nombre (edo_civil) y como consecuencia entre ellos tres también sólo se permite una selección pero la selección entre los componentes llamados **sexo** y **edo_civil** son independientes pero cada uno de ellos tiene su propio valor (para sexo seria “M” o “F” y para edo_civil seria “C”, ”S” o ”D”). En pantalla tendremos lo siguiente:



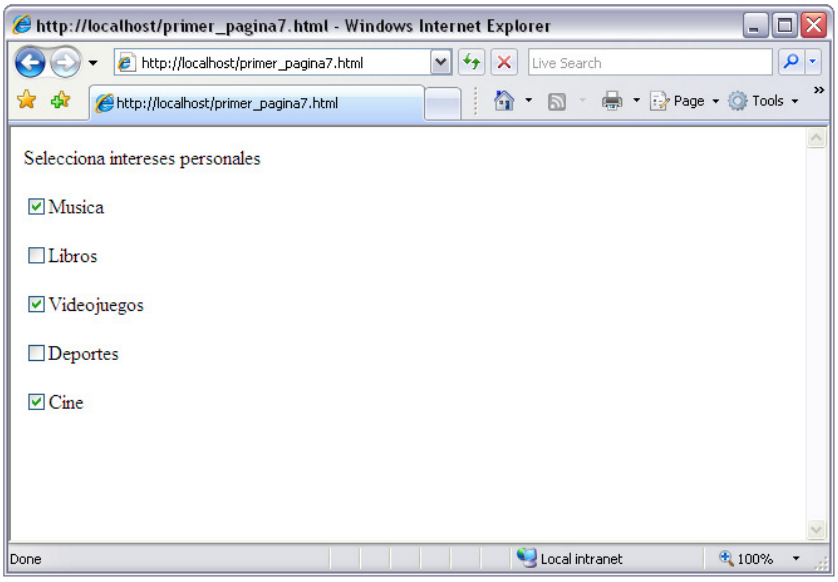
Ejecución archivo primer_pagina6.htm

Para agregar un componente de tipo CheckBox debemos colocar las etiquetas siguientes:

```
1       <html>
2
3       <head>
4       </head>
5
6       <body>
7       <form name="form" method="post" action="mi_otra_pagina.html">
8         <p>Selecciona intereses personales </p>
9         <p><input type="checkbox" name="checkbox1" value="checkbox">Musica</p>
10        <p><input type="checkbox" name="checkbox2" value="checkbox">Libros</p>
11        <p><input type="checkbox" name="checkbox3" value="checkbox">Videojuegos</p>
12        <p><input type="checkbox" name="checkbox4" value="checkbox">Deportes</p>
13        <p><input type="checkbox" name="checkbox5" value="checkbox">Cine</p>
14      </form>
15      </body>
16      </html>
```

Archivo primer_pagina7.html, Página HTML con diferentes Checkbox

En el ejemplo anterior se incluyeron cinco casillas de verificación, cada una de ellas tiene nombre propio y la selección de alguna no interfiere con ninguna otra como ocurría con los Radio Buttons. En pantalla tendremos lo siguiente:



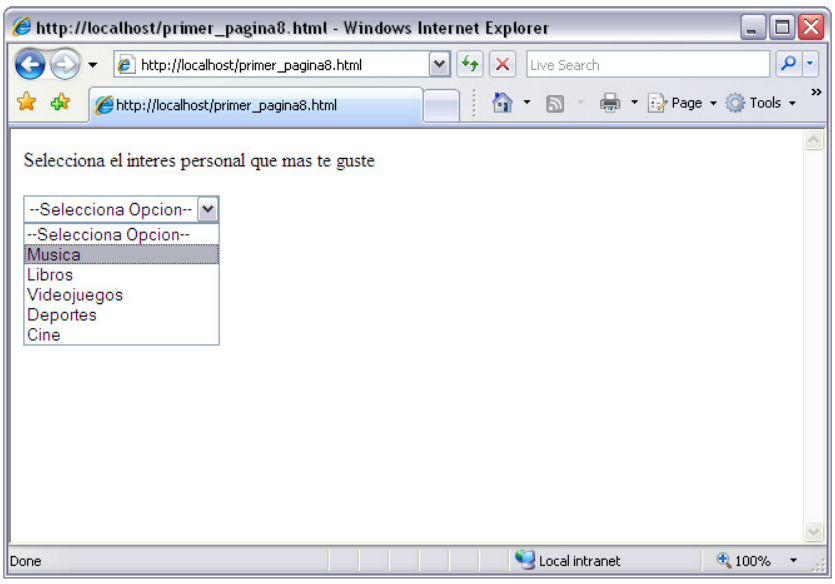
Ejecución archivo primer_pagina7.html

Para agregar una lista desplegable se debe hacer lo siguiente:

```
1        <html>
2
3        <head>
4        </head>
5
6        <body>
7        <form name="form" method="post" action="mi_otra_pagina.html">
8          <p>Selecciona el interes personal que mas te guste </p>
9          <p>
10           <select name="interesesp">
11             <option value="">--Selecciona Opcion--</option> //si se coloca value="" es lo mismo que omitirlo
12             <option value="M">Musica</option>
13             <option value="L">Libros</option>
14             <option value="V">Videojuegos</option>
15             <option value="D">Deportes</option>
16             <option value="C">Cine</option>
17           </select>
18         </p>
19       </form>
20       </body>
21       </html>
```

Archivo primer_pagina8.html, Página HTML con lista desplegable

En el ejemplo anterior se incluyeron las mismas cinco opciones que con las casillas de verificación, cada una de ellas tiene un valor propio excepto por la primera opción que corresponde a la opción de "--Selecciona Opcion--", lo cual es muy útil en las validaciones vistas posteriormente. En pantalla tendremos lo siguiente:



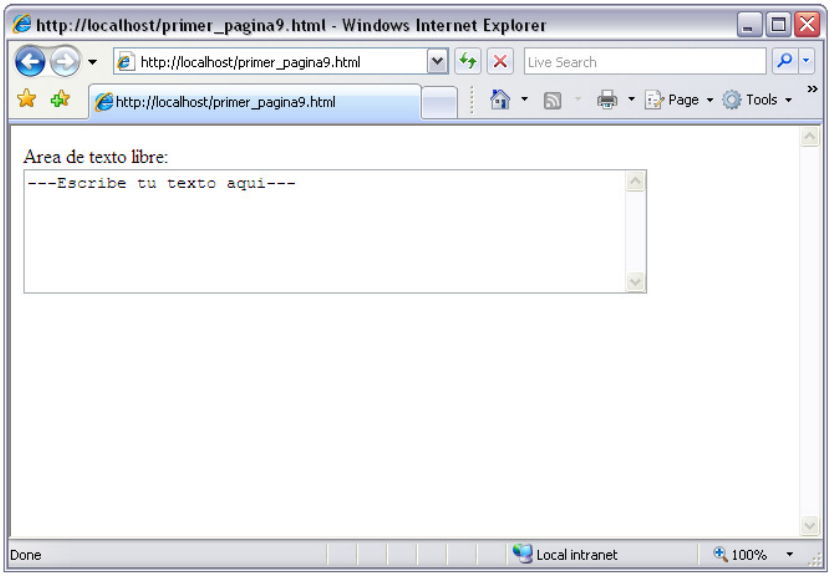
Ejecución archivo primer_pagina9.html

Para agregar un área de texto libre se debe incluir en nuestra página HTML lo siguiente:

```
1       <html>
2
3       <head>
4       </head>
5
6       <body>
7       <form name="form" method="post" action="mi_otra_pagina.html">
8         <p>Area de texto libre:
9             <textarea id="mi_area" name="mi_area" cols="60" rows="6">---Escribe tu texto aqui---</textarea>
10        </p>
11      </form>
12      </body>
13      </html>
```

Archivo primer_pagina9.html, Página HTML con un área de texto

En el ejemplo anterior se incluye un área de texto libre con una anchura de 60 columnas y 6 líneas, una vez rebasado este límite aparece una barra de desplazamiento vertical para moverse entre líneas. En pantalla tendremos lo siguiente:



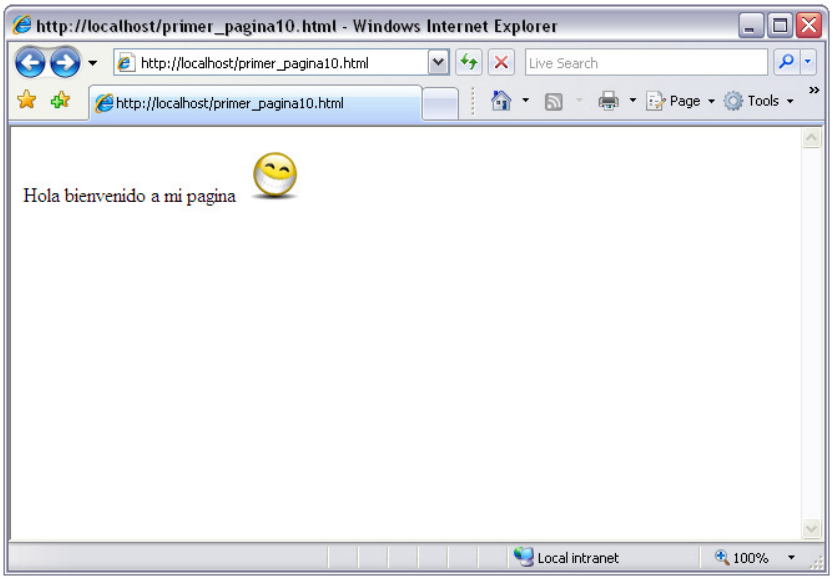
Ejecución archivo primer_pagina9.html

Para agregar una imagen a nuestra página debemos incluir la etiqueta siguiente:

```
1       <html>
2
3       <head>
4       </head>
5
6       <body>
7       <form name="form" method="post" action="mi_otra_pagina.html">
8         <p> Hola bienvenido a mi pagina
9             <input type="image" name="imageField" src="sonrisa.gif">
10        </p>
11      </form>
12      </body>
13      </html>
```

Archivo primer_pagina10.html, Página HTML con imagen

Con lo que incluimos una imagen llamada “sonrisa.gif” en nuestra página, como la imagen se encuentra las etiquetas **<form>** y **</form>**, la imagen se convierte en un vinculo y al hacer Click sobre ella nos redireccionará a la página llamada “mi_otra_pagina.html”, si no queremos eso, basta con incluir la etiqueta que incluye a la imagen afuera de las etiquetas **form**. En pantalla tendremos lo siguiente:



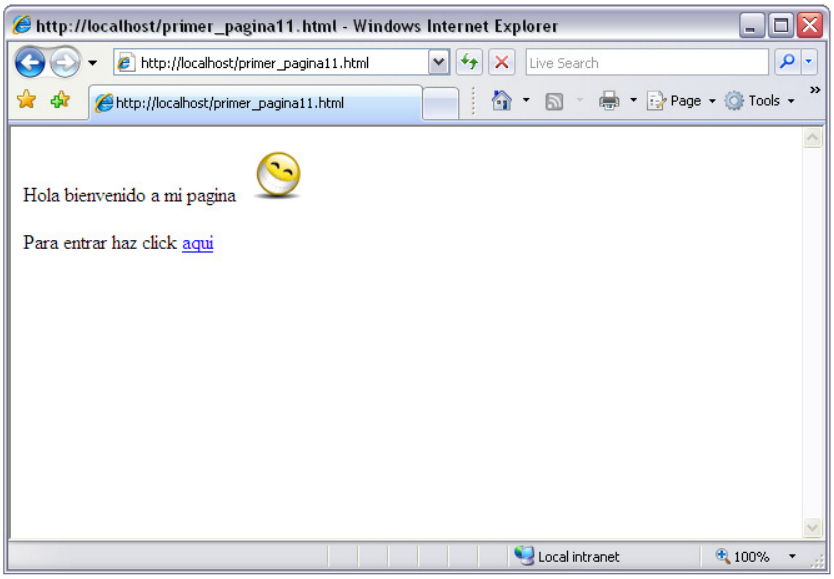
Ejecución archivo primer_pagina10.html

Para tener una liga a otra página y quitar el vínculo a nuestra imagen debemos hacer lo siguiente:

```
1       <html>
2
3       <head>
4       </head>
5
6       <body>
7
8         <p>
9                 Hola bienvenido a mi pagina
10          <input type="image" name="imageField" src="sonrisa.gif">
11        </p>
12        <p>Para entrar haz click <a href="mi_otra_pagina.html">aqui</a></p>
13      </body>
14      </html>
```

Archivo primer_pagina11.html, Página HTML con una imagen y una liga

Como se explicó anteriormente, para evitar que al hacer Click sobre la imagen se haga el redireccionamiento a la página "mi_otra_pagina.html" la etiqueta de la imagen debe estar afuera de las etiquetas **form** y la liga se agrego a la palabra "aquí" que nos redireccionará a dicha página si hacemos Click sobre el vínculo, con el código anterior tenemos lo siguiente:



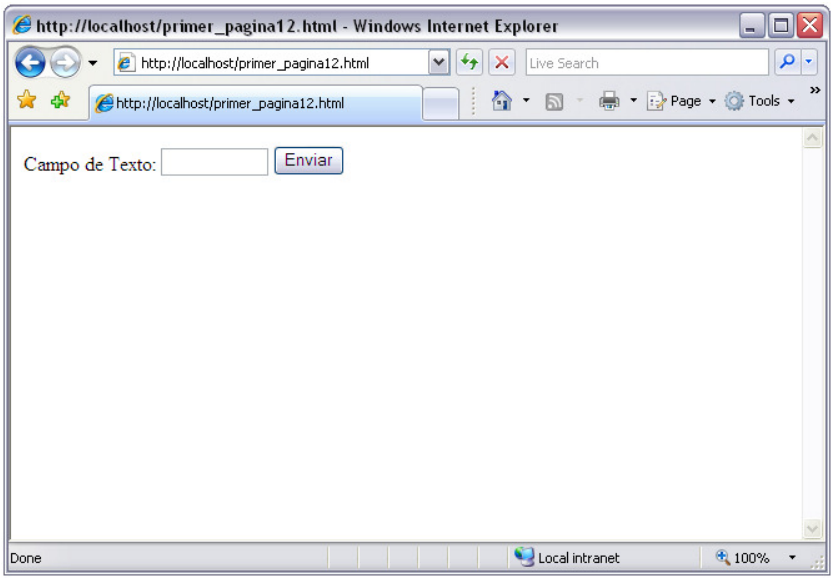
Ejecución archivo primer_pagina11.html

Para enviar el contenido de lo que desde ahora llamaremos **formularios** y que son el conjunto de campos de texto, de imagen, de selección, listas desplegables, etc. dentro de un mismo par de etiquetas **<form>** y **</form>** haremos uso de un botón de envio el cual se agrega de la siguiente manera:

```
1       <html>
2
3       <head>
4       </head>
5
6       <body>
7       <form name="form" method="post" action="mi_otra_pagina.html">
8         Campo de Texto: <input type="text" id="mi_campo" name="mi_campo" size="10" maxlength="50">
9         <input type="submit" name="Submit" value="Enviar">
10      </form>
11      </body>
12      </html>
```

Archivo primer_pagina12.html, Página HTML con botón de envio de formulario

Ahora bien, al hacer Click sobre el botón “Enviar” todos los valores de los componentes que se encuentren entre las etiquetas **<form>** y **</form>** del código anterior (en este caso sólo el campo de texto llamado “mi_campo”), serán enviados a la página llamada “mi_otra_pagina.html”, debemos notar que el código correspondiente al botón de envío también se encuentra dentro de dichas etiquetas, el resultado es el siguiente:



Ejecución archivo primer_pagina12.html

Tal vez el envío de formularios entre páginas HTML no sea de gran utilidad en este momento debido a que con HTML no se pueden hacer validaciones o manipulaciones sobre los valores que se introducen en los campos de texto, así que, aunque parezca que no sirve de nada, se debe comprender muy bien el concepto del envío y la recepción de formularios ya que son la base para realizar las validaciones y manipulaciones de datos con lenguajes que si lo permiten.

En el último ejemplo enviábamos el formulario a una página llamada "mi_otra_pagina.html" pero para enviarlo a una página de PHP solo debemos cambiar a "mi_otra_pagina.**php**" la cual si puede hacer validaciones de campos, cifrado de los datos introducidos, interactuar con un Sistema Manejador de Bases de Datos (SMBD), enviar el contenido del campo por e-mail, etc. pero eso será descrito en los capítulos siguientes.

Los ejemplos vistos en el presente capítulo se encuentran disponibles en D:\Capitulol\.

CAPITULO II.- JAVASCRIPT, LENGUAJE WEB DEL CLIENTE

Javascript es un lenguaje de programación utilizado para crear pequeños programas encargados de realizar acciones dentro del ámbito de una página web.

Se trata de un lenguaje de programación del lado del cliente, porque es el navegador el que soporta la carga de procesamiento. Gracias a su compatibilidad con la mayoría de los navegadores modernos, es el lenguaje de programación del lado del cliente más utilizado.

Validación de formularios con Javascript

La validación de los datos de un formulario mediante Javascript no sustituye a la validación que debe realizarse, por motivos de seguridad, en la aplicación del servidor que recibe la información. Sin embargo, al añadir una validación local con Javascript, la experiencia de usuario mejora notablemente, al no ser necesario enviar los datos al servidor y esperar su respuesta para obtener sólo un mensaje informando de la incorrección de la información suministrada. Resulta frustrante completar un formulario, pulsar el botón enviar, y esperar 20 o 30 segundos para saber que hemos introducido mal, los datos de un campo.

La validación de campos de formulario se basa en interceptar el momento en que el usuario realiza el envío de los datos del formulario (es decir, hace Click sobre el botón de **enviar**). Como es sabido, el botón de envío de datos se codifica con HTML mediante un tipo especial de objeto de formulario, llamado **submit**. Una vez terminada la ejecución de la(s) función(es) de Javascript se procede a hacer el envío de los datos del formulario a la página definida por la etiqueta **action**.

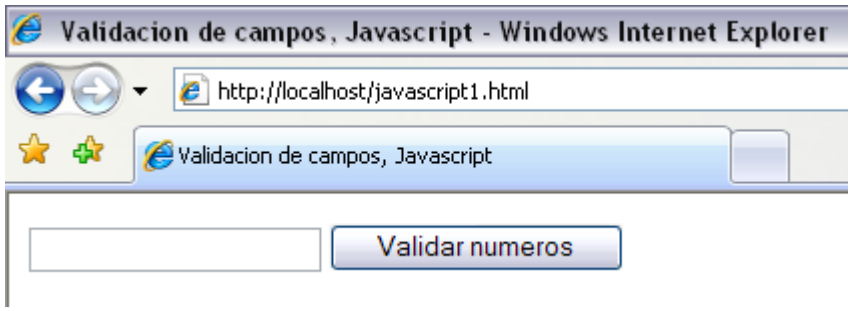
Validación de campos con valores numéricos

La validación de campos de tipo numérico se realiza de la siguiente manera:

```
1   <html>
2   <head>
3   <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
4   <title>Validacion de campos, Javascript</title>
5   <script>
6           function validar(){
7                   if(isNaN(document.getElementById('campo').value)){
8                           alert('El valor introducido no es numerico');
9                           return false;
10                  }
11                  return true;
12          }
13  </script>
14  </head>
15  <body>
16  <form name="form" method="post" action="mi_pagina.html" onSubmit="return validar()"> //function de javascript
17          <input name="campo" id="campo" type="text" />
18          <input type="submit" name="Submit" value="Enviar" />
19  </form>
20  </body>
21  </html>
```

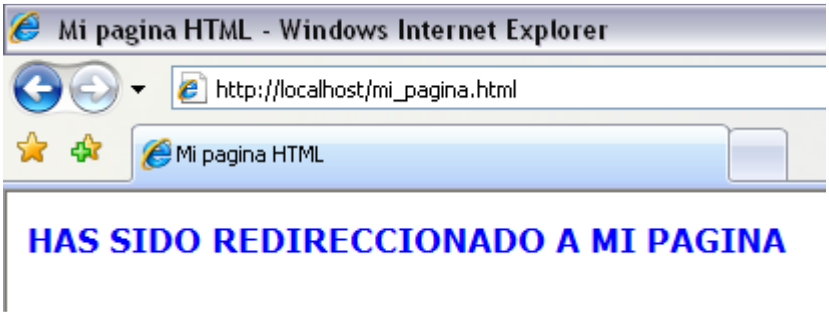
Archivo javascript1.html, Validación de campos numéricos

El resultado en pantalla sería el siguiente:



Ejecución archivo javascript1.html

Si al campo de texto se le introduce un valor numérico (45, 532, -2, 3.1416), la validación se hará de manera correcta y se hará el redireccionamiento a la página indicada en la etiqueta **action** que en este caso es **mi_pagina.html**.



Archivo mi_pagina.html, Todos los archivos de validación redireccionan a esta página si la validación fue correcta

En caso de error muestra un mensaje de alerta informando el fallo de la validación y el redireccionamiento no se realiza.

Archivo javascript1.html, Mensaje de error

Validación de campos no vacíos

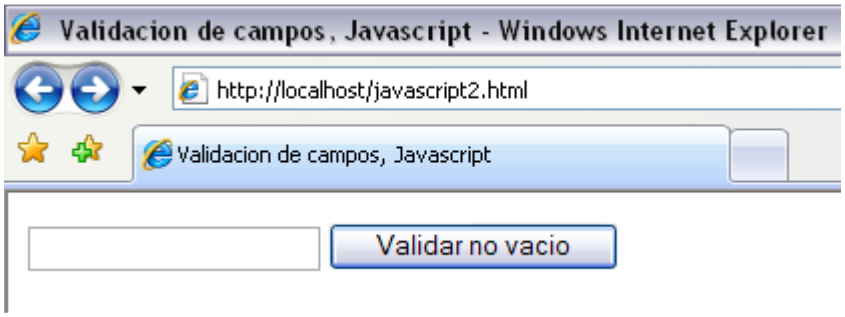
La validación de campos no vacíos se realiza de la siguiente manera:

```
1       <html>
2       <head>
3       <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
4       <title>Validacion de campos, Javascript</title>
5       <script>
6               function validar(){
7                       if(document.getElementById('campo').value.length == 0){ // o tambien puede ser
document.getElementById('campo').value == “”
```

```
8                                       alert('El campo no puede ir vacio');
9                                       return false;
10                              }
11                              return true;
12                      }
13      </script>
14      </head>
15      <body>
16      <form name="form" method="post" action="mi_pagina.html" onSubmit="return validar()">
17              <input name="campo" id="campo" type="text" />
18              <input type="submit" name="Submit" value="Enviar" />
19      </form>
20      </body>
21      </html>
```

Archivo javascript2.html, Validación de campos no vacíos

El resultado en pantalla sería el siguiente:



Ejecución archivo javascript2.html

Si al campo de texto se le introduce algún valor, la validación se hará de manera correcta y se hará el redireccionamiento a la página indicada en la etiqueta **action** que en este caso es **mi_pagina.html**.

En caso de error muestra un mensaje de alerta informando el fallo de la validación y el redireccionamiento no se realiza.

Archivo javascript2.html, Mensaje de error

Validación de campos de tamaño fijo

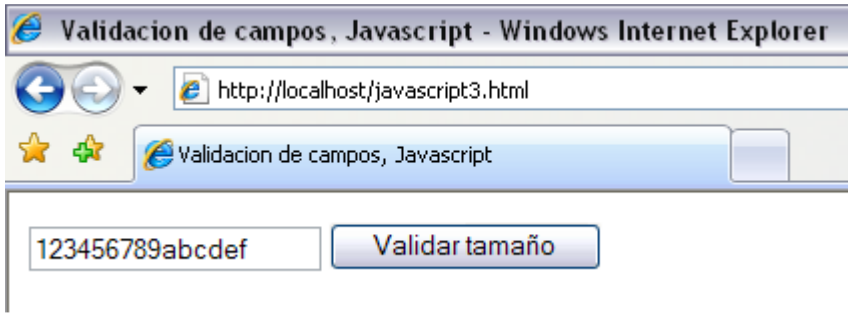
La validación de campos de tamaño fijo se realiza de la siguiente manera:

```
1       <html>
2       <head>
3       <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
4       <title>Validacion de campos, Javascript</title>
5       <script>
6               function validar(){
```

```
7                               if(document.getElementById('campo').value.length != 15){
8                                       alert('El campo debe tener 15 caracteres');
9                                       return false;
10                              }
11                              return true;
12                      }
13      </script>
14      </head>
15      <body>
16      <form name="form" method="post" action="mi_pagina.html" onSubmit="return validar()">
17              <input name="campo" id="campo" type="text" maxlength="15" /> <!--maxlength = “15” para que el campo
no admita más de 15 caracteres -->
18              <input type="submit" name="Submit" value="Enviar" />
19      </form>
20      </body>
21      </html>
```

Archivo javascript3.html, Validación de campos de tamaño fijo

El resultado en pantalla sería el siguiente:



Ejecución archivo javascript3.html

Si al campo de texto se le introduce algún valor con 15 caracteres, la validación se hará de manera correcta y se hará el redireccionamiento a la página indicada en la etiqueta **action** que en este caso es **mi_pagina.html**.

En caso de error muestra un mensaje de alerta informando el fallo de la validación y el redireccionamiento no se realiza.

Archivo javascript3.html, Mensaje de error

Validación de campos de fecha

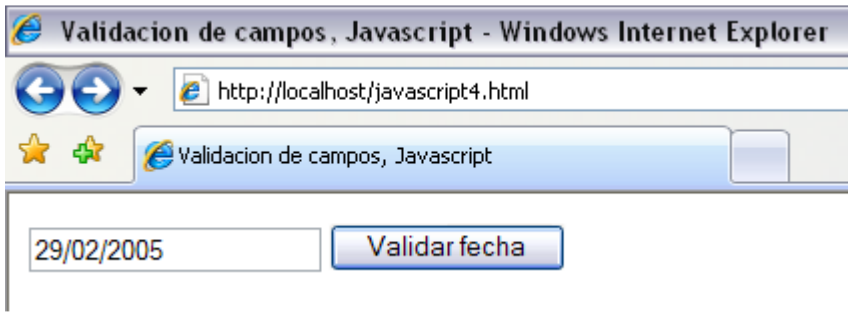
La validación de campos de fecha se realiza de la siguiente manera:

```
1       <html>
2       <head>
3       <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
```

```
4       <title>Validacion de campos, Javascript</title>
5       <script>
6               function validar(){
7                       var fecha = document.getElementById('campo').value;
8                       var fechaFormato = /^(\d{1,2})(\/|-)(\d{1,2})\2(\d{2}|\d{4})$/;
9                       var fechaArreglo = fecha.match(fechaFormato);
10
11                      if(fechaArreglo == null) {
12                              alert('La fecha introducida no es valida');
13                              return false;
14                      }
15
16                      mes = fechaArreglo[3]; //1
17                      dia = fechaArreglo[1]; //3
18                      anio = fechaArreglo[4];
19
20                      if (mes < 1 || mes > 12) { // verifica el rango de meses
21                              alert('La fecha introducida no es valida');
22                              return false;
23                      }
24                      if (dia < 1 || dia > 31) {
25                              alert('La fecha introducida no es valida');
26                              return false;
27                      }
28                      if ((mes==4 || mes==6 || mes==9 || mes==11) && dia==31) {
29                              alert('La fecha introducida no es valida');
30                              return false
31                      }
32                      if (mes == 2) { // valida el 29 de Febrero
33                              var bisiesto = (anio % 4 == 0 && (anio % 100 != 0 || anio % 400 == 0));
34                              if (dia>29 || (dia==29 && !bisiesto)) {
35                                      alert('La fecha introducida no es valida');
36                                      return false;
37                              }
38                      }
39                      if(anio>2500 || anio<1900){
40                              alert('La fecha introducida no es valida');
41                              return false;
42                      }
43                      return true;  // la fecha es valida
44              }
45      </script>
46      </head>
47      <body>
48      <form name="form" method="post" action="mi_pagina.html" onSubmit="return validar()">
49              <input name="campo" id="campo" type="text" />
50              <input type="submit" name="Submit" value="Validar fecha" />
51      </form>
52      </body>
53      </html>
```

Archivo javascript4.html, Validación de fechas

El resultado en pantalla sería el siguiente:



Ejecución archivo javascript4.html

Si al campo de texto se le introduce algún valor de fecha con formato dd-mm-aaaa ó dd/mm/aaaa, la validación se hará de manera correcta y se hará el redireccionamiento a la página indicada en la etiqueta **action** que en este caso es **mi_pagina.html**.

En caso de error ya sea porque la fecha introducida no es válida o porque no se trata de una fecha se muestra un mensaje de alerta informando el fallo de la validación y el redireccionamiento no se realiza.

Archivo javascript4.html, Mensaje de error

Validación de campos de correo electrónico

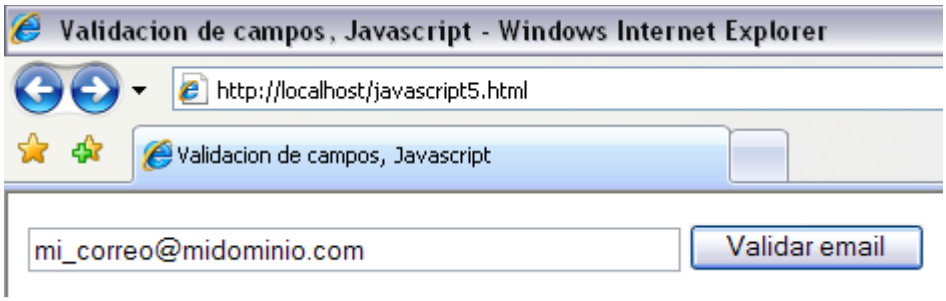
La validación de campos de correo electrónico se realiza de la siguiente manera:

```
1	<html>
2	<head>
3	<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
4	<title>Validacion de campos, Javascript</title>
5	<script>
6		function validar(){
7			var email_introducido = document.getElementById('campo').value;
8			var email  = /^([a-zA-Z0-9_\.\-])+\@(([a-zA-Z0-9\-])+\.)+([a-zA-Z0-9]{2,4})+$/;
9
10			if (!email.test(email_introducido)){
11				alert('La direccion de correo electronico no es valida');
12				return false;
13			}
14			return true;
15		}
16	</script>
17	</head>
18	<body>
19	<form name="form" method="post" action="mi_pagina.html" onSubmit="return validar()">
20		<input name="campo" id="campo" type="text" size=”50” /><!--size = “50” para que el campo tenga un ancho de 50 caracteres -->

21		<input type="submit" name="Submit" value="Validar email" />
22	</form>
23	</body>
24	</html>
```

Archivo javascript5.html, Validación de campos de correo electrónico

El resultado en pantalla sería el siguiente:



Ejecución archivo javascript5.html

Si al campo de texto se le introduce una dirección de correo electrónico con el formato adecuado, la validación se hará de manera correcta y se hará el redireccionamiento a la página indicada en la etiqueta **action** que en este caso es **mi_pagina.html**.

En caso de error muestra un mensaje de alerta informando el fallo de la validación y el redireccionamiento no se realiza.

Archivo javascript5.html, Mensaje de error

Validación de selección de un valor con listas desplegables

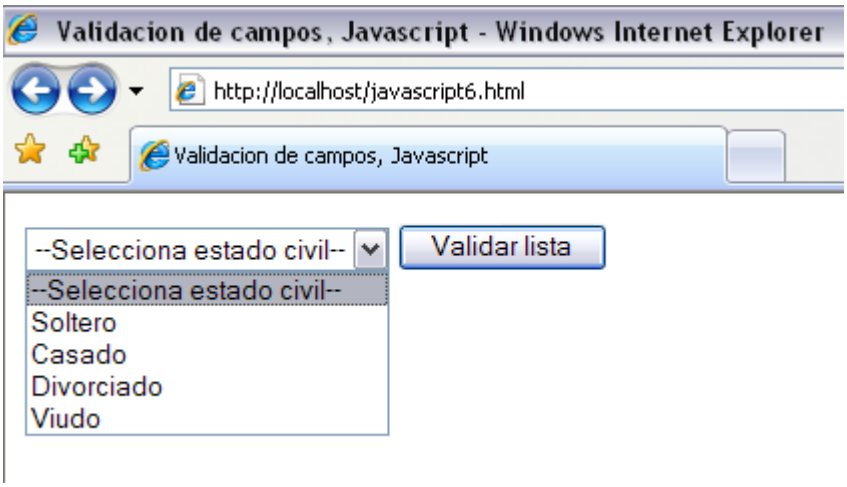
La validación de listas desplegables para saber si se ha seleccionado un valor se realiza de la siguiente manera:

```
1	<html>
2	<head>
3	<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
4	<title>Validacion de campos, Javascript</title>
5	<script>
6		function validar(){
7			if (document.getElementById('lista_desplegable').value==""){
8				alert('Debes seleccionar un estado civil de la lista desplegable');
9			return false;
10			}
11			else{
12				var totalElementos  = document.getElementById('lista_desplegable').length;
13				var elementoElegido = document.getElementById('lista_desplegable').selectedIndex;
14				var valorElementoElegido =
document.getElementById('lista_desplegable').options[elementoElegido].text;
15
16				alert('El total de elementos de la lista es \'" + totalElementos + '\'");
17
18				alert('El numero de elemento que elegiste es el \'" + elementoElegido + '\'");
19
20				alert('El valor del estado civil \'" + valorElementoElegido + '\" es \'" +
document.getElementById('lista_desplegable').value + '\'");
21
22				return true;
```

```
23                          }
24                  }
25          </script>
26          </head>
27          <body>
28          <form name="form" method="post" action="mi_pagina.html" onSubmit="return validar()">
29                  <select name="lista_desplegable" id="lista_desplegable">
30                    <option value="">--Selecciona estado civil--</option>
31                    <option value="S">Soltero</option>
32                    <option value="C">Casado</option>
33                    <option value="D">Divorciado</option>
34                    <option value="V">Viudo</option>
35             </select>
36                  <input type="submit" name="Submit" value="Validar lista" />
37          </form>
38          </body>
39          </html>
```

Archivo javascript6.html, Validación de listas desplegables

El resultado en pantalla sería el siguiente:



Ejecución archivo javascript6.html

Si se selecciona un valor de la lista desplegable distinto a la opción de --Selecciona estado civil--", la validación se hará de manera correcta y se hará el redireccionamiento a la página indicada en la etiqueta **action** que en este caso es **mi_pagina.html**. Además se muestran una seria de parámetros que contienen las listas desplegables como son el total de elementos de la lista, el valor de lista elegido (número entero que es 0 para el primer elemento de la lista), y el tanto el texto como el valor de la opción elegida.

Archivo javascript6.html, Mensajes con propiedades de listas desplegables

En caso de error muestra un mensaje de alerta informando el fallo de la validación y el redireccionamiento no se realiza.

Archivo javascript6.html, Mensaje de error

Validación de selección de un valor con Radio Buttons

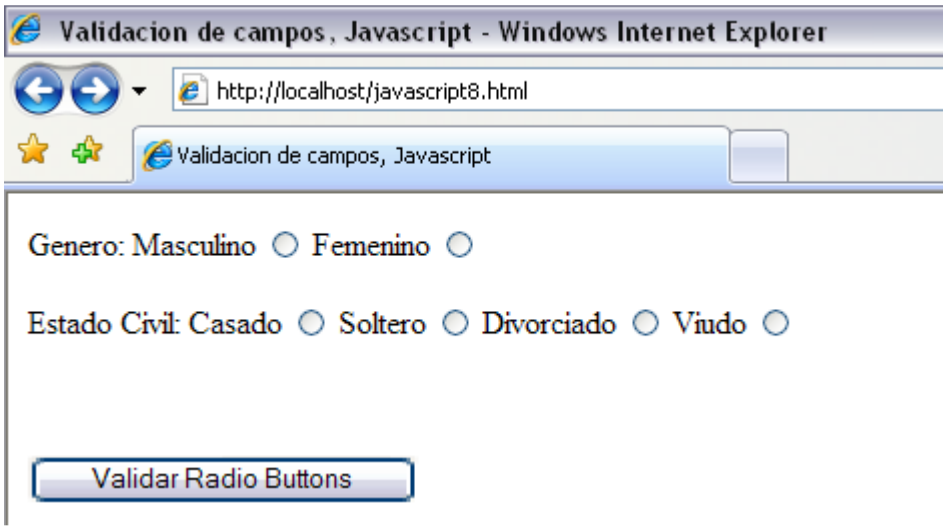
La validación de radio buttons para saber si se ha seleccionado un valor se realiza de la siguiente manera:

```
1       <html>
2       <head>
3       <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
4       <title>Validacion de campos, Javascript</title>
5       <script>
6               function validar(){
7                       if (!document.form.genero[0].checked && !document.form.genero[1].checked){
8                               alert('Debes seleccionar un genero');
9                               return false;
10                      }
11                      if(!document.form.edocivil[0].checked && !document.form.edocivil[1].checked && !document.form.edocivil[2].checked && !document.form.edocivil[3].checked){
12                        alert('Debes seleccionar un estado civil');
13                        return false;
14                      }
15                              return true;
16              }
17      </script>
```

```
18        </head>
19        <body>
20        <form name="form" method="post" action="mi_pagina.html" onSubmit="return validar()">
21          <p>Genero:
22            Masculino
23            <input name="genero" id="genero" type="radio" value="M">
24            Femenino
25            <input name="genero" id="genero" type="radio" value="F">
26          </p>
27          <p>Estado Civil:
28            Casado
29            <input name="edocivil" id="edocivil" type="radio" value="M">
30            Soltero
31            <input name="edocivil" id="edocivil" type="radio" value="F">
32            Divorciado
33            <input name="edocivil" id="edocivil" type="radio" value="F">
34            Viudo
35            <input name="edocivil" id="edocivil" type="radio" value="F">
36          </p>
37          <p> </p>
38          <p><input type="submit" name="Submit" value="Validar Radio Buttons" /></p>
39        </form>
40        </body>
41        </html>
```

Archivo javascript7.html, Validación de Radio Buttons

El resultado en pantalla sería el siguiente:



Ejecución archivo javascript7.html

Si se selecciona un valor de cada conjunto de botones (genero y estado civil) de selección, la validación se hará de manera correcta y se hará el redireccionamiento a la página indicada en la etiqueta **action** que en este caso es **mi_pagina.html**.

En caso de error (si no se seleccionó un género o no se seleccionó un estado civil) muestra un mensaje de alerta informando el fallo de la validación y el redireccionamiento no se realiza.

Archivo javascript7.html, Mensajes de error

Validación de selección de un valor con CheckBox

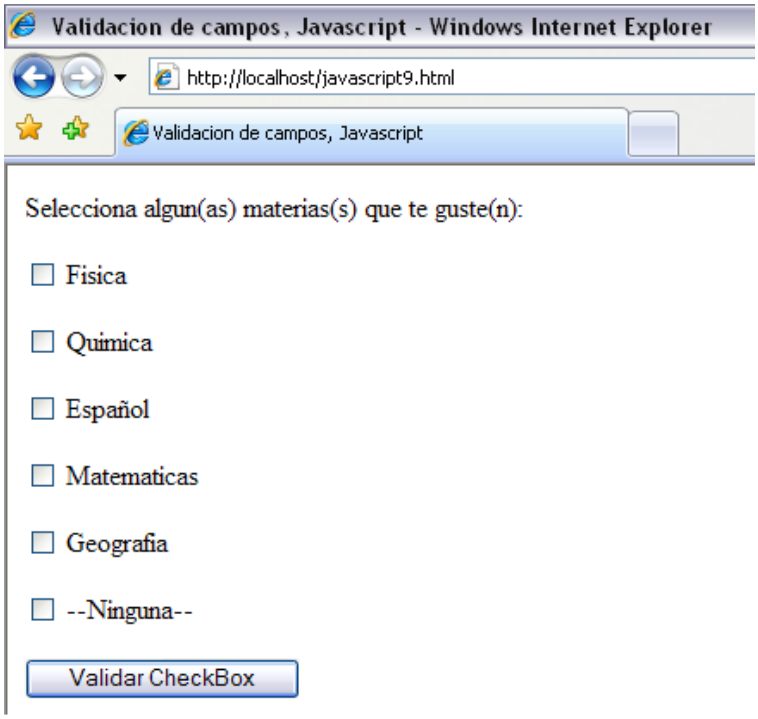
La validación de los CheckBox para saber si se ha seleccionado un valor se realiza de la siguiente manera:

```
1	<html>
2	<head>
3	<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
4	<title>Validacion de campos, Javascript</title>
5	<script>
6		function validar(){
7			if(!document.form.materias[0].checked && !document.form.materias[1].checked && !document.form.materias[2].checked && !document.form.materias[3].checked && !document.form.materias[4].checked && !document.form.materias[5].checked){
8			  alert('Debes seleccionar una opcion');
9		  return false;
10			}
11			return true;
12		}
13
14		function modifica_seleccion(){
15			if (document.form.materias[5].checked == true){
16			  document.form.materias[0].checked = false;
17			  document.form.materias[1].checked = false;
18			  document.form.materias[2].checked = false;
19			  document.form.materias[3].checked = false;
20			  document.form.materias[4].checked = false;
21			}
22		}
23	</script>
24	</head>
25	<body>
26	<form name="form" method="post" action="mi_pagina.html" onSubmit="return validar()">
27	  <p>Selecciona algun(as) materias(s) que te guste(n):  </p>
28	  <p><input type="checkbox" name="materias" value="checkbox">
29	    Fisica
30	  <p><input type="checkbox" name="materias" value="checkbox">
31	  Quimica
32	  <p><input type="checkbox" name="materias" value="checkbox">
33	  Espa&ntilde;ol
34	  <p><input type="checkbox" name="materias" value="checkbox">
35	  Matematicas
```

```
36          <p><input type="checkbox" name="materias" value="checkbox">
37          Geografia
38          <p><input type="checkbox" name="materias" value="checkbox" onClick="modifica_seleccion()">
39          --Ninguna--
40          <p><input type="submit" name="Submit" value="Validar CheckBox" /></p>
41        </form>
42        </body>
43        </html>
```

Archivo javascript8.html, Validación de Checkbox

El resultado en pantalla sería el siguiente:



Ejecución archivo javascript8.html

Si se selecciona un valor de alguna casilla de verificación, la validación se hará de manera correcta y se hará el redireccionamiento a la página indicada en la etiqueta **action** que en este caso es **mi_pagina.html**. Si se selecciona la casilla de verificación correspondiente a la opción --Ninguna--", se ejecuta la función **modifica_seleccion** de la etiqueta **onClick**, dicha función verifica si la casilla ha sido marcada para desmarcar todas las demás casillas.

En caso de error (si no se seleccionó ninguna opción) muestra un mensaje de alerta informando el fallo de la validación y el redireccionamiento no se realiza.

Archivo javascript8.html, Mensaje de error

En el disco de anexos (D:\CapituloII\) se muestran y describen ejemplos prácticos de validación de formularios con Javascript (campos de tipo entero, flotante, texto, fecha, hora, e-mail, url's, no vacios), que pueden servir para entender mejor los conceptos explicados anteriormente así como los ejemplos vistos en el presente capítulo.

CAPITULO III.- LENGUAJES WEB DEL SERVIDOR: PHP, JSP Y SERVLETS

Cuando la información es manipulada en el servidor no existen las limitantes del manejo de datos que se tienen del lado del Cliente. Con scripts del servidor se puede limitar el acceso al sistema a los usuarios, manejar sesiones, cifrar datos, interactuar con los recursos del equipo servidor (como archivos, carpetas y programas), interactuar con el SMBD, etc.

Los mecanismos de seguridad que se pueden implementar en el servidor son los siguientes:

- Validar los datos introducidos por el cliente
- Proteger Archivos utilizando contraseñas
- Cifrar el contenido de nuestros scripts
- Cifrar el contenido antes de ser descargado
- Cifrar el contenido antes de ser almacenado en BD
- Controlar el acceso al sistema analizando la validez y los tiempos de sesión

PHP, Hypertext Preprocesor

PHP es el acrónimo de Hipertext Preprocesor. Es un lenguaje de programación del lado del servidor gratuito e independiente de plataforma, rápido, con una gran librería de funciones y mucha documentación.

PHP se escribe dentro del código HTML, lo que lo hace realmente fácil de utilizar. Es independiente de la plataforma, puesto que existe un módulo de PHP para casi cualquier servidor web. Esto hace que cualquier sistema pueda ser compatible con el lenguaje y significa una ventaja importante, ya que permite portar el sitio desarrollado en PHP de un sistema a otro sin prácticamente ningún trabajo.

PHP, en el caso de estar montado sobre un servidor Linux u Unix, es más rápido que ASP, dado que se ejecuta en un único espacio de memoria y esto evita las comunicaciones entre componentes COM que se realizan entre todas las tecnologías implicadas en una página ASP.

Por último señalamos la seguridad, en este punto también es importante el hecho de que en muchas ocasiones PHP se encuentra instalado sobre servidores Unix o Linux, que son de sobra conocidos como más veloces y seguros que el sistema operativo donde se ejecuta las ASP, Windows NT.

Este lenguaje de programación está preparado para realizar muchos tipos de aplicaciones web gracias a la extensa librería de funciones con la que está dotado. La librería de funciones cubre desde cálculos matemáticos complejos hasta tratamiento de conexiones de red, por poner ejemplos.

Algunas de las más importantes capacidades de PHP son: compatibilidad con las bases de datos más comunes, como MySQL, mSQL, Oracle, Informix, y ODBC, por ejemplo. Incluye funciones para el envío de correo electrónico, carga de archivos, crear dinámicamente en el servidor imágenes en formato GIF, incluso animadas, etc.

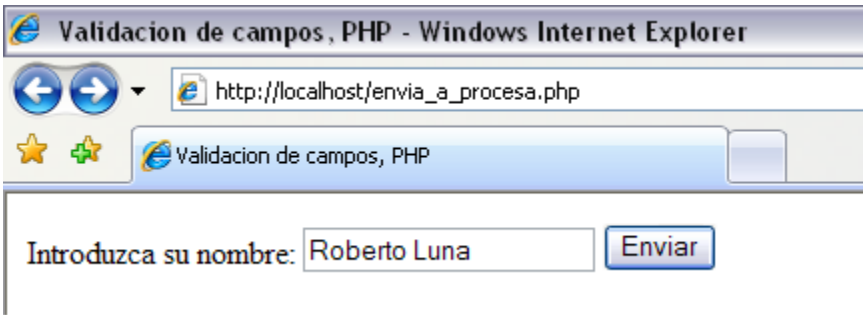
Validación de formularios con PHP

El lenguaje PHP nos proporciona una manera sencilla de manejar formularios, permitiéndonos de esta manera procesar la información que el usuario ha introducido.

Al diseñar un formulario debemos indicar la página PHP que procesará el formulario, así como el método por el que se le pasará la información a la página.

```
1       <html>
2       <head>
3         <title> Validacion de campos, PHP </title>
4       </head>
5       <body>
6       introduzca su nombre:
7       <form action="procesa.php" method="post">
8       <input type="text" name="nombre" id="nombre"><br>
9       <input type="submit" value="Enviar">
10      </form>
11      </body>
12      </html>
```

Archivo envia_a_procesa.php, envio de formularios con POST



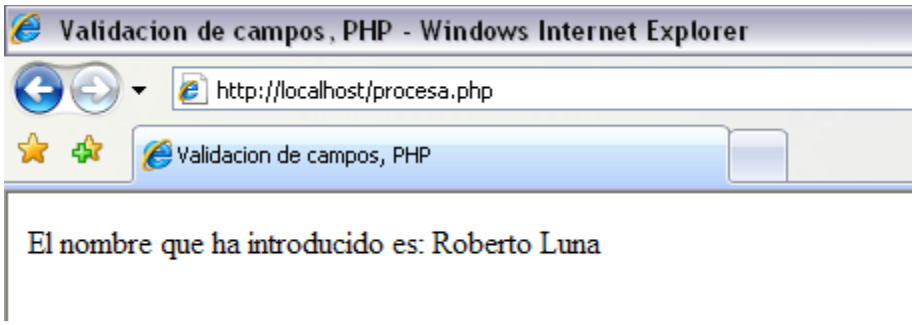
Ejecución archivo envia_a_procesa.php

Al hacer click en el botón "Enviar" el contenido del cuadro de texto es enviado a la página que indicamos en el atributo **action** de la etiqueta **form** (línea 8) llamado "procesa.php".

Para acceder a las variables del formulario hay que usar el array de parámetros $_POST[] o $_GET[] dependiendo del método usado para enviar los parámetros (en el ejemplo anterior el método utilizado fue POST, línea 7). Es por eso que en el archivo "procesa.php" haremos referencia al valor del campo de texto **nombre**, de la siguiente manera:

```
1       <html>
2       <head>
3         <title> Validacion de campos, PHP </title>
4       </head>
5       <body>
6       El nombre que ha introducido es: <?php echo $_POST['nombre']." ". $_POST['apellidos']?>
7       </body>
8       </html>
```

Archivo procesa.php, mostrar campos de un formulario con POST



Ejecución archivo procesa.php

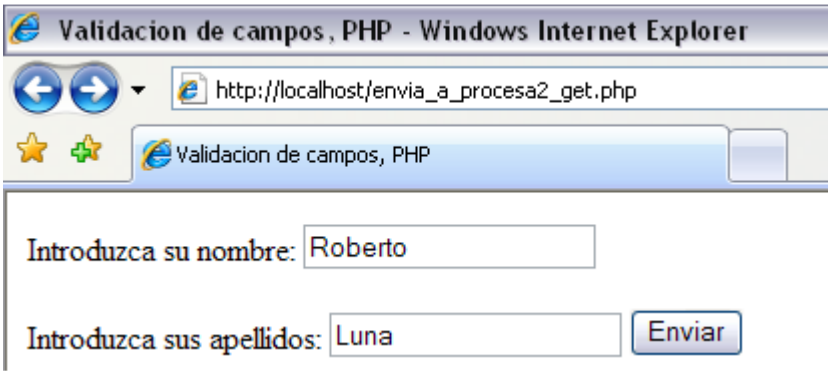
Métodos GET y POST

En la página anterior hemos comentado que los datos de un formulario se envían mediante el método indicado en el atributo METHOD de la etiqueta FORM, los dos métodos posibles son GET y POST.

La diferencia entre estos dos métodos radica en la forma de enviar los datos a la página, mientras que el método GET envía los datos usando la URL, el método POST los envía por la entrada estándar STDIO.

```
1       <html>
2       <head>
3         <title> Validacion de campos, PHP </title>
4       </head>
5       <body>
6       <h1>Ejemplo de manejo de formularios con GET</h1>
7       <form action="procesa2.php" method="get">
8       introduzca su nombre:<input type="text" name="nombre"><br>
9       introduzca sus apellidos:<input type="text" name="apellidos"><br>
10      <input type="submit" value="enviar">
11      </form>
12      </body>
13      </html>
```

Archivo envia_a_procesa2_get.php, envio de formularios con GET

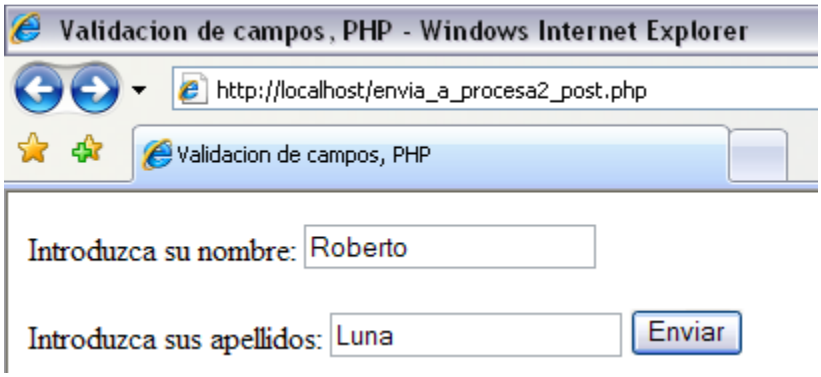


Ejecución archivo envia_a_procesa2_get.php

```
1       <html>
2       <head>
3         <title> Validacion de campos, PHP </title>
4       </head>
5       <body>
6       <h1>Ejemplo de manejo de formularios con POST</h1>
```

```
7       <form action="procesa2.php" method="post">
8       introduzca su nombre:<input type="text" name="nombre"><br>
9       introduzca sus apellidos:<input type="text" name="apellidos"><br>
10      <input type="submit" value="enviar">
11      </form>
12      </body>
13      </html>
```

Archivo envia_a_procesa2_post.php, manejo de formularios con POST

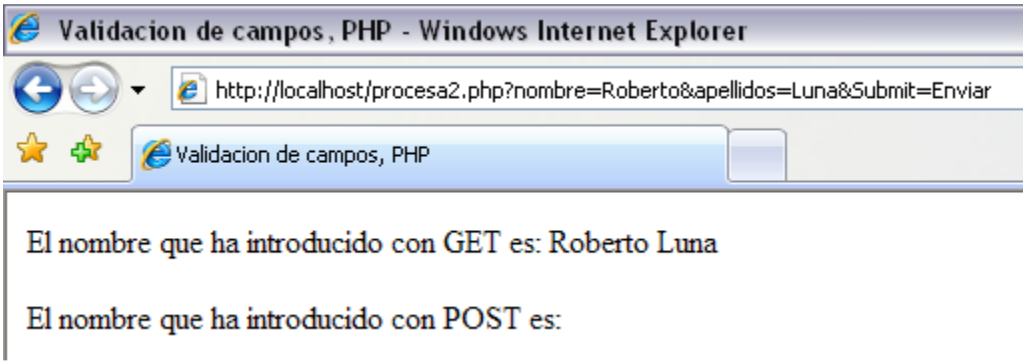


Ejecución archivo envia_a_procesa2_post.php

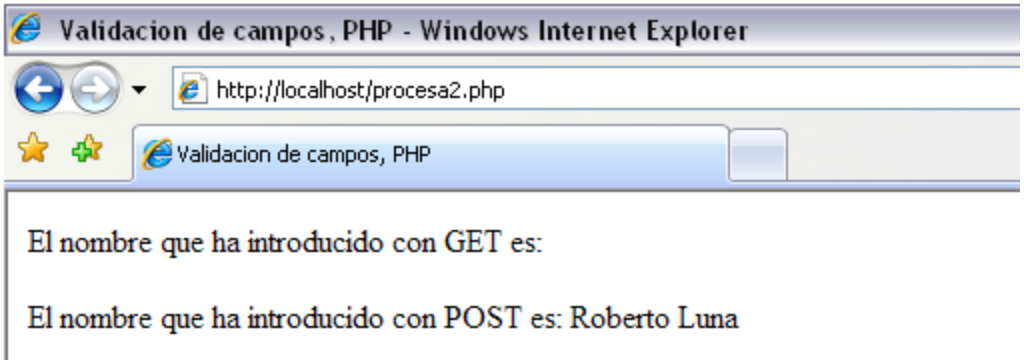
```
1       <html>
2       <head>
3         <title>Ejemplo de PHP</title>
4       </head>
5       <body>
6       El nombre que ha introducido por GET es: <?php echo $_GET['nombre']," ",$_GET['apellidos'] ?><br>
7       El nombre que ha introducido por POST es: <?php echo $_POST['nombre']," ",$_POST['apellidos'] ?>
8       <br>
9       </body>
10      </html>
```

Archivo procesa2.php, envio de formularios con GET

El resultado final es el mismo, solo que con el método GET podemos ver los parámetros pasados ya que están codificados en la URL.



Archivo procesa2.php, resultado del envío de parámetros con GET



Ejecución archivo procesa2.php

Validación de distintos tipos de campos en PHP

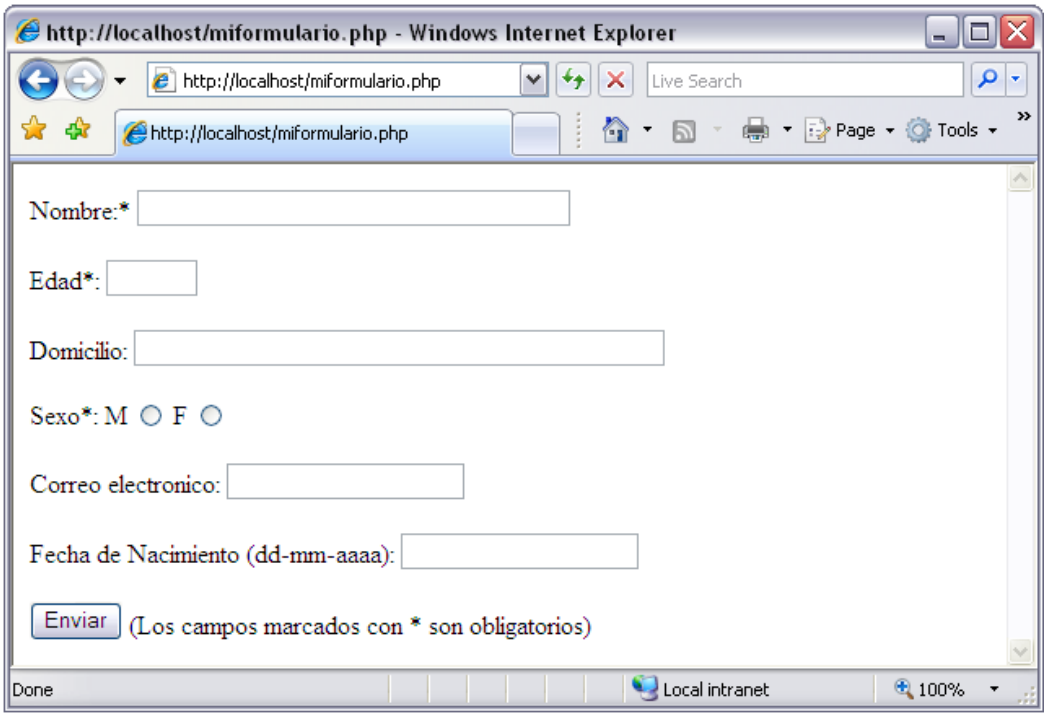
Con Javascript se analizó la forma de validar campos de tipo numérico, no vacios, de fecha, etc., ahora veremos la forma de realizar las mismas validaciones con el lenguaje PHP, dichas validaciones siempre se deben hacer a pesar de haberlas realizado con Javascript porque, como se mencionó anteriormente, la ejecución del lenguaje Javascript puede estar deshabilitada en algunos navegadores de internet lo que podría dar como resultado datos incorrectos en nuestros sistemas.

Tenemos el siguiente código:

```
1       <html>
2       <head>
3       <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
4       </head>
5
6       <body>
7       <form id="form" name="form" method="post" action="validacion.php">
8
9       <p>Nombre:* <input name="nombre" type="text" id="nombre" size="40" maxlength="40" /></p>
10      <p>Edad*:
11        <input name="edad" type="text" id="edad" size="5" /></p>
12      <p>Domicilio: <input name="domicilio" type="text" id="domicilio" size="50" maxlength="70" /></p>
13      <p>Sexo*:
14              M <input name="sexo" type="radio" value="M" />
15              F <input name="sexo" type="radio" value="F" />
16      </p>
17      <p>Correo electronico: <input name="email" type="text" id="email" /></p>
18      <p>Fecha Nacimiento (dd-mm-aaaa): <input name="fnacimiento" type="text" id="fnacimiento" /></p>
19
20        <input type="submit" name="Submit" value="Enviar" />
21        (Los campos marcados con * son obligatorios)
22      </form>
23      </body>
24      </html>
```

Archivo miformulario.php, un formulario a ser validado con PHP

Que nos genera en pantalla lo siguiente:



Ejecución archivo miformulario.php

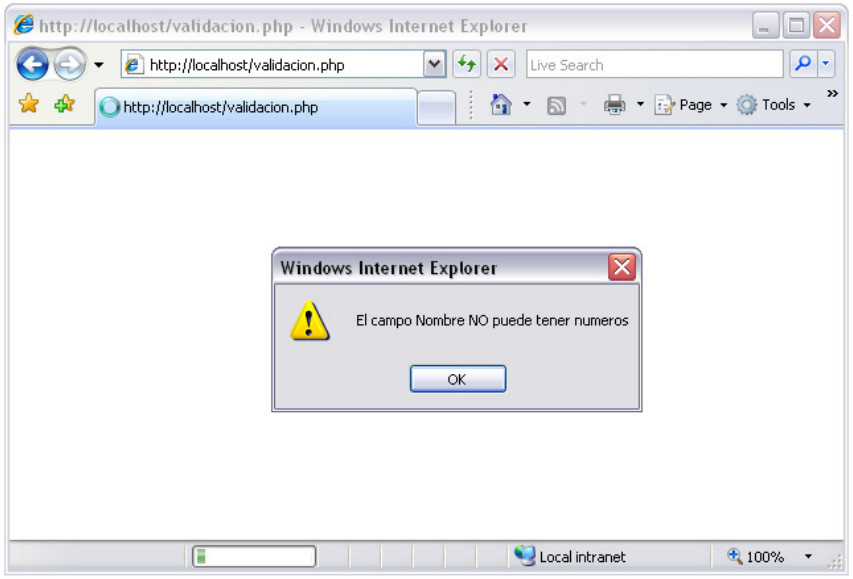
El formulario anterior debe ser validado por el archivo "validacion.php" tomando ciertas consideraciones, la primera de ellas es que los campos **Nombre**, **Edad** y **Sexo** no deben de ser vacios, el **Correo electrónico**, en caso de no ser vacio, debe tener una cadena con formato: "micorreo@midominio.algoonada.com" y la **Fecha de nacimiento** debe tener formato: "dd-mm-aaaa o dd/mm/aaaa" y aparte ser fecha válida, el código del archivo "validacion.php" para validar el formulario anterior es el siguiente:

```
1       <?php
2       include("validaciones.php");
3       //validamos que ni el nombre ni la edad sean vacios
4               if(strlen($_POST['nombre']) == 0 || strlen($_POST['edad']) == 0){
5                       echo "<script>";
6                       echo "alert('Favor de llenar los campos marcados con *');";
7                       echo "window.history.back();";
8                       echo "</script>";
9                       exit;
10              }
11      //validamos que el nombre tenga solo letras
12              if(ctype_digit($_POST['nombre'])==true){
13                      echo "<script>";
14                      echo "alert('El campo Nombre NO puede tener numeros');";
15                      echo "window.history.back();";
16                      echo "</script>";
17                      exit;
18              }
19      //validamos que la edad tenga solo numeros
20              if(ctype_digit($_POST['edad'])==false){
21                      echo "<script>";
22                      echo "alert('El campo Edad NO puede tener letras');";
23                      echo "window.history.back();";
24                      echo "</script>";
25                      exit;
26              }
```

```
27      //validamos que se haya seleccionado el sexo de la persona
28              if($_POST['sexo'] == ""){
29                      echo "<script>";
30                      echo "alert('Seleccione el sexo de la persona');";
31                      echo "window.history.back();";
32                      echo "</script>";
33                      exit;
34              }
35              if($_POST['fnacimiento'] != ""){  //validamos que el campo fecha de nacimiento no sea vacio
36      //si no es vacio validamos que se haya introducido una fecha con formato dd-mm-aaaa o dd/mm/aaaa
37                      $fecha = split("-", $_POST['fnacimiento']);
38
39                      if(sizeof($fecha)==1)
40                              $fecha = split("/", $_POST['fnacimiento']);
41
42                      $band = 0;
43
44                      if($fecha[1]=='01'  ||  $fecha[1]=='03'  ||  $fecha[1]=='05'  ||  $fecha[1]=='07'  ||
$fecha[1]=='08' || $fecha[1]=='10' || $fecha[1]=='12'){
45                              if($fecha[0] > '31' || $fecha[0] < '01')
46                                      $band = 1;
47                      }
48                      else if($fecha[1]=='04' || $fecha[1]=='06' || $fecha[1]=='09' || $fecha[1]=='11'){
49                              if($fecha[0] > '30' || $fecha[0] < '01')
50                                      $band = 1;
51                      }
52                      else if($fecha[1]=='02'){
53                              if($fecha[2] % 4 > 0 && ($fecha[0] > '28' || $fecha[0] < '01'))
54                                      $band = 1;
55                              else if($fecha[2] % 4 == 0 && ($fecha[0] > '29' || $fecha[0] < '01'))
56                                      $band = 1;
57                      }
58                      else
59                              $band = 1;
60
61                      if($band == 1 || strlen($fecha[2] != 4 || ctype_digit($fecha[2])==false)){
62                              echo "<script>";
63                              echo "alert('La fecha de nacimiento no es correcta');";
64                              echo "window.history.back();";
65                              echo "</script>";
66                              exit;
67                      }
68              }
69      ?>
```

Archivo validacion.php, validación de formularios con PHP

En caso de algún error, aparecerá en pantalla el mensaje correspondiente la ejecución regresará a la pantalla del formulario para reintroducir los datos.



Ejecución archivo validacion.php

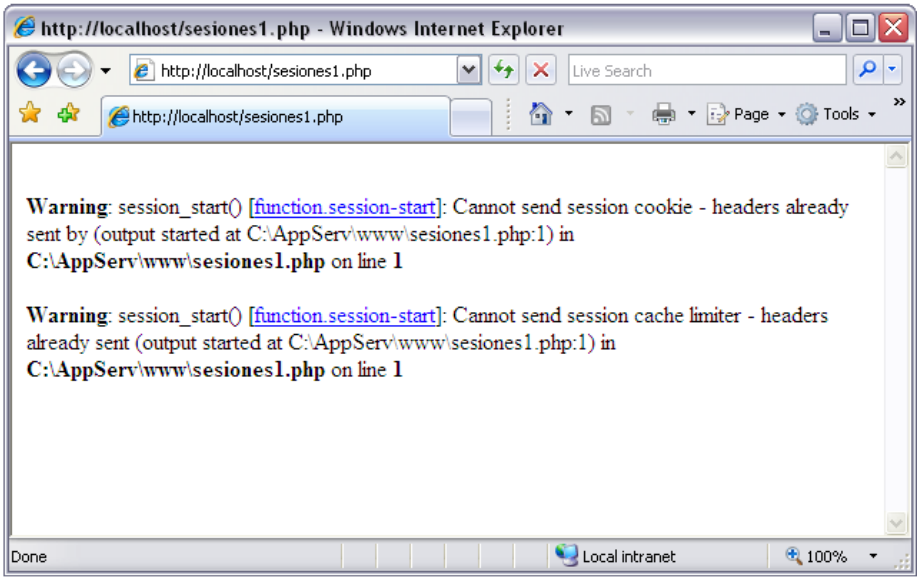
Manejo de sesiones con PHP

Para poder hacer uso de las sesiones en PHP lo primero es iniciarlas con la función **session_start()**, esta función debe estar ANTES de cualquier otra cosa.

```
1        <?php sesión_start(); ?>
```

Archivo sesiones1.php, inicializar una sesion

De no colocarla antes de cualquier salida HTML se obtendrán mensajes de error como el de la siguiente figura (un espacio en blanco antes de **<?php** se toma como una salida HTML).



Ejecución archivo sesiones1.php, Mensaje de error

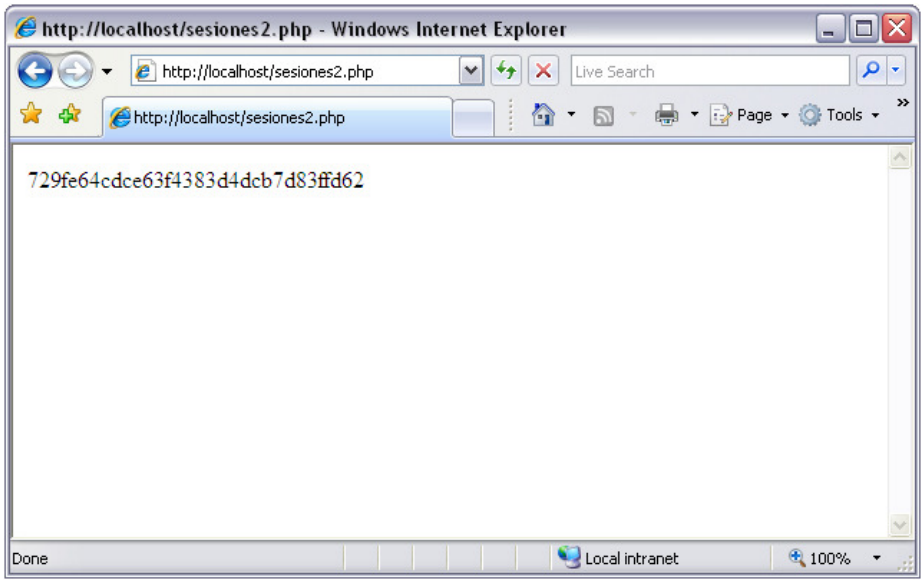
El manejo de las sesiones impide el intercambio de datos entre ellas ya que se trata información específica para cada usuario e incluso si se trata del mismo usuario.

Las sesiones nos permiten almacenar y consultar información sobre un visitante sin necesidad de estar enviándola a través de formularios.

Para obtener la sesión de un usuario se utiliza el método **session_id()** que devuelve un identificador único asociado a una sesión:

```
1       <?php
2       session_start();
3       $sesion=session_id();
4       echo "Identificador de la sesión: ".$sesion;
5       ?>
```

Archivo sesiones2.php, crear una sesion



Ejecución archivo sesiones2.php

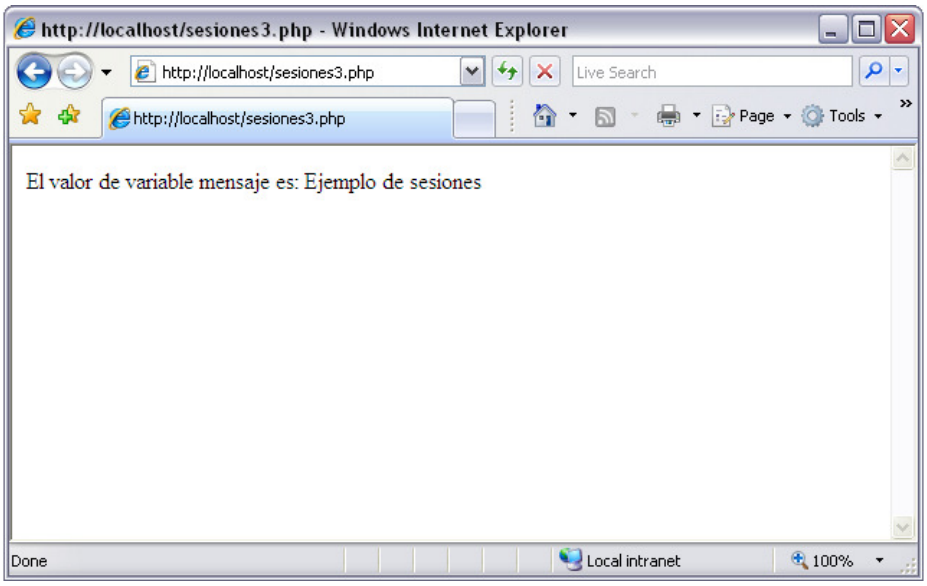
Guardar Objetos en una Sesión

Para guardar un objeto en una variable de sesión se utiliza el arreglo $_SESSION[], que almacena cualquier tipo de dato:

Un ejemplo de cómo guardar una cadena de texto en una variable de sesión es el siguiente:

```
1       <?php
2       session_start();
3       $_SESSION['mensaje']="Ejemplo de sesiones";
4
5       echo "El valor de variable mensaje es: ".$_SESSION['mensaje'];
6
7       ?>
```

Archivo sesiones3.php, guardar objetos en una sesion

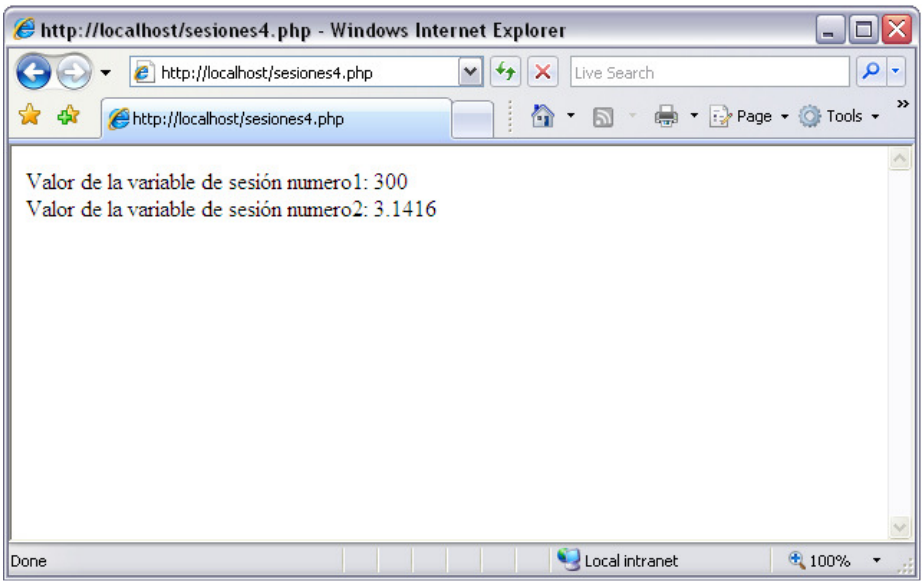


Ejecución archivo sesiones3.php

Lo mismo aplica para variables numéricas:

```
1       <?php
2       session_start();
3       $_SESSION[‘numero1’]=300;
4       $_SESSION[‘numero2’]=3.1416;
5       echo “Valor de la variable de sesión numero1: ”.$_SESSION[‘numero1’];
6       echo “<br>”;
7       echo “Valor de la variable de sesión numero2: ”.$_SESSION[‘numero2’];
8       ?>
```

Archivo sesiones4.php, guardar objetos numéricos en una sesion



Ejecución archivo sesiones4.php

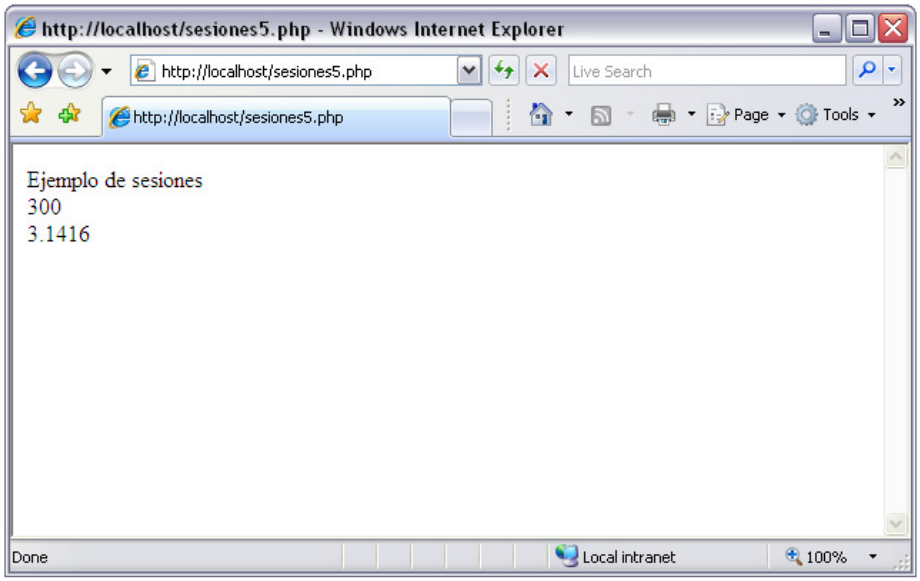
Recuperar objetos de una Sesión

Los datos que se guardan en la sesión permanecen ahí a la espera de ser utilizados. Para ello es necesario realizar el proceso contrario a cuando se graban, comenzando por la recuperación del objeto de la sesión para empezar a ser tratado.

Para poder realizar este paso se utiliza el arreglo $_SESSION[], utilizando como argumento el nombre que identifica al objeto que se quiere recuperar.

```
1       <?php
2       session_start();
3
4       echo $_SESSION[‘mensaje’].”<br>”;
5
6       echo $_SESSION[‘numero1’] .”<br>”;
7       echo $_SESSION[‘numero2’] ;
8       ?>
```

Archivo sesiones5.php, recuperar objetos de una sesion



Ejecución archivo sesiones5.php

Si no existe ningún objeto almacenado en la sesión bajo el identificador que se utiliza en el arreglo **$_SESSION[]**, el valor devuelto será vacío.

Destruir una Sesión

Como ya se ha visto, los datos almacenados por las sesiones pueden destruirse en tres casos:

- El usuario abandona aplicación web (cambia de web o cierra el navegador)
- Se alcanza el tiempo máximo permitido de inactividad de un usuario (timeout).
- El servidor se para o se reinicia.

Pero la situación más probable es querer iniciar las sesiones o dar por finalizada una si se han cumplido una o varias condiciones. En este caso no es necesario esperar a que ocurra alguno de los tres casos

citados anteriormente, ya que mediante el método **sesión_destroy()** es posible destruir una sesión concreta.

```
1       <?php
2       [...]     //líneas de código PHP
30      session_destroy();
31      ?>
```

Archivo sesiones6.php, destruir una sesion

Cifrado de datos en PHP

Cifrado lineal o de un solo sentido

Como se mencionó anteriormente, cada vez más aplicaciones son implementadas sobre tecnología Web, esto, con la finalidad de tener la información más actual, disponible en todo momento desde cualquier parte del mundo. Obviamente, la información que viaja en la red, que utiliza nuestros sistemas y que almacenamos en nuestra base de datos puede ser desde nombres o descripciones de algún artículo que alguna empresa de comercio electrónico quiera vender en línea hasta información personal (nombre, domicilio, nombres de los familiares, etc.) sobre los clientes que tienen enormes cuentas en algún banco, sus saldos, sus números de cuenta o los nombres de usuario y contraseñas que tienen acceso a los sistemas que programamos.

Es por tal motivo que esta información que se guarda en la base de datos debe tener algún tipo de protección. En PHP se utilizan las funciones **md5()** y **sha1()**, que son funciones **irreversibles** (de un sólo sentido), es decir, cifran los datos y es imposible que partiendo desde la cadena encriptada se vuelvan a obtener los datos origen. Por esto mismo no hay problema de que alguien pueda acceder al campo encriptado de la base de datos.

Tal vez a algunas personas esto les parezca ilógico pero en verdad es una funcionalidad muy útil, por ejemplo cuando una página requiere el nombre de usuario y contraseña, esta última se encripta y se guarda en la Base de datos. Como en la base de datos se guarda la contraseña encriptada, cuando un usuario quiere acceder, habrá que realizar una comparación entre la contraseña que introduce encriptado con MD5 o con SHA1, y lo que tenemos en la base de datos, (que es la contraseña encriptada), y si coincide se le permite el acceso, si no, se rechaza.

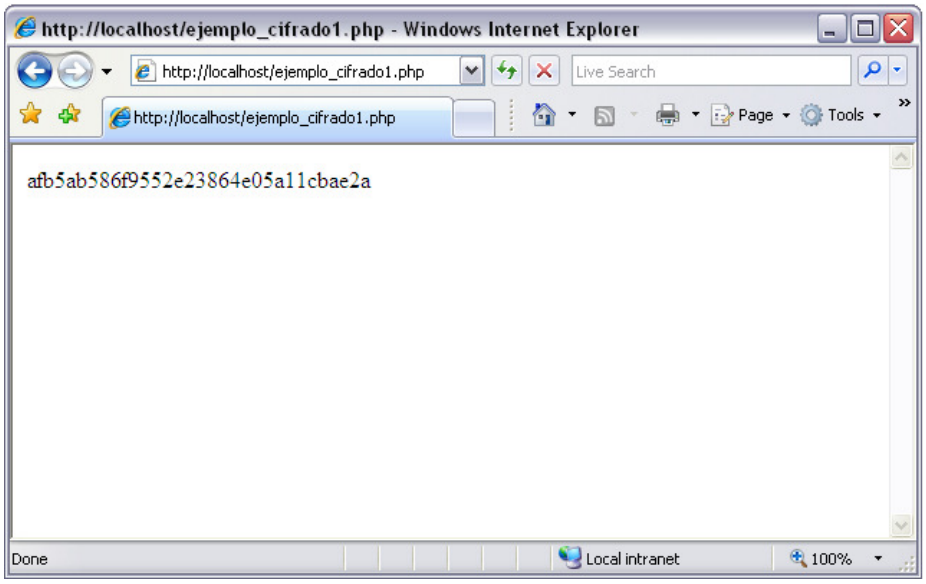
Esto se utiliza también cuando un usuario olvida su contraseña, si quiere recuperar la contraseña se le pide que introduzca por ejemplo el correo, y se le envía un mail con una URL tal que si entra en ella genere una nueva contraseña que se le indica al usuario y se reescribe en la base de datos (borrando la contraseña anterior).

A continuación tenemos un segmento de código que ejemplifica el uso de la función md5() de PHP.

```
1       <?
2       $contrasena = md5 ("mi contrasena");
3       echo $contrasena;
4       ?>
```

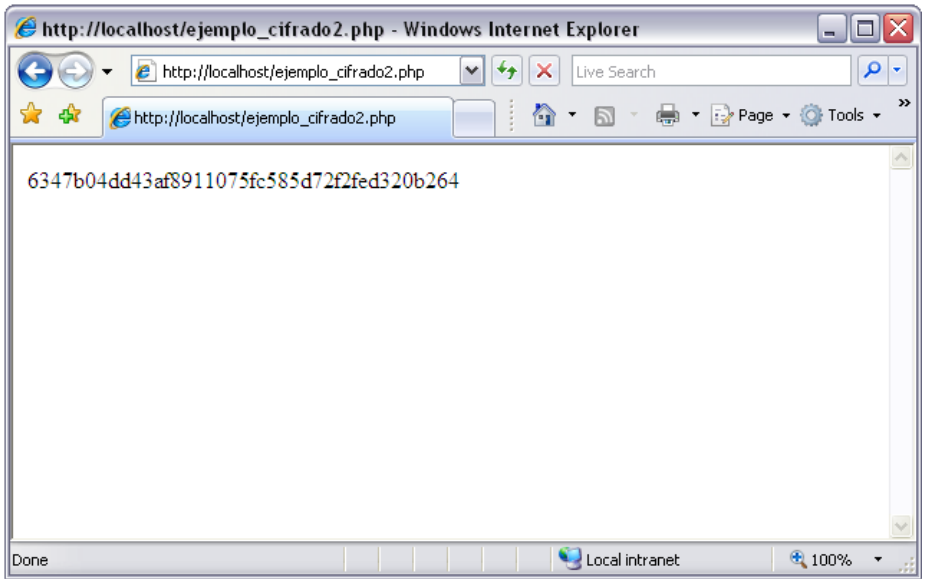
Archivo ejemplo_cifrado1.php, cifrado de datos con MD5

La ejecución del código anterior mostrará en pantalla lo siguiente:



Ejecución archivo ejemplo_cifrado1.php

Lo anterior es el equivalente cifrado en MD5 de las palabras “mi contrasena” que se introdujeron como parámetro a la función. Si en el código del ejemplo anterior se sustituye la función **md5()** por **sha1()** la salida en pantalla sería la siguiente:



Archivo ejemplo_cifrado2.php

Lo que sería el equivalente cifrado en SHA1 de las palabras “mi contrasena” que se introdujeron como parámetro a la función.

Cifrado con DES y 3DES

Las funciones nativas de PHP para el cifrado no permiten o soportan el respectivo proceso inverso, sin embargo, mucha de la información que manipularán nuestros sistemas Web requerirán el descifrado de la información, por ejemplo, si almacenamos información de los clientes de nuestros sistemas en

un servidor pero dicho servidor es contratado con un proveedor, es muy probable que la persona encargada de administrar el servidor "hurgue" la información que nuestra aplicación almacene en la BD haciéndose de información confidencial sobre los usuarios y los servicios de nuestro sistema.

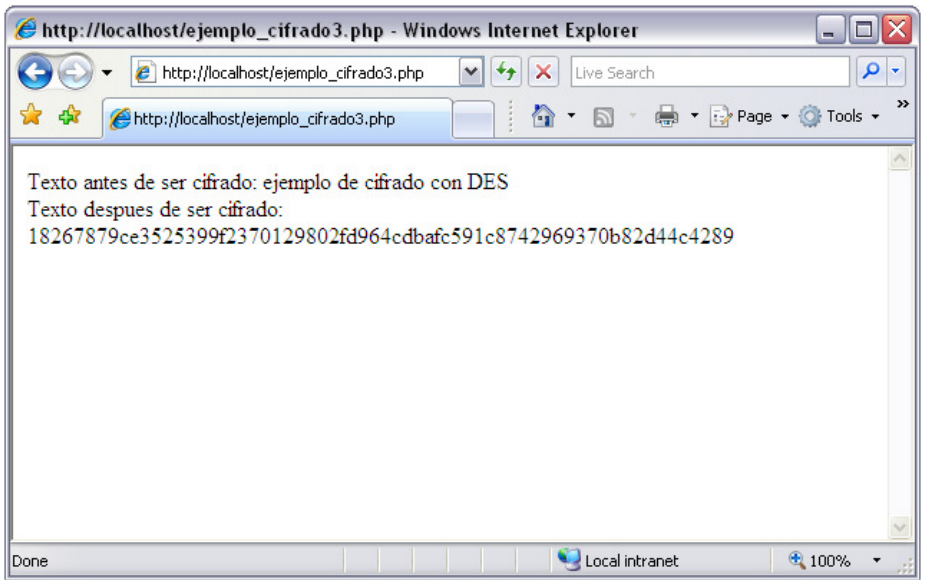
Debido a lo anterior, es necesario implementar funciones de cifrado y descifrado de datos. Para el sistema SIBAINES fue necesario el desarrollo de una función que implemente el algoritmo DES (o también 3DES), dicha función, recibe 4 parámetros:

- El primero de ellos es la llave que se empleará para realizar el cifrado del texto.
- El segundo parámetro es la cadena de texto a cifrar o descifrar.
- El tercer parámetro es una bandera: 1 para cifrar, 0 para descifrar.
- El cuarto y último parámetro es otra bandera: 1 para activar el cifrado utilizando CBC y 0 para cifrar sin CBC.

Un ejemplo del uso del algoritmo de cifrado es el siguiente:

```
1       <?
2       include("des.php");
3
4       $llave = "8bytekey8bytekey8bytekey";
5       $mensaje = "ejemplo de cifrado con DES";
6
7       $texto_cifrado = bin2hex(des($llave, $mensaje, 1, 0));
8       echo $texto_cifrado;
9       ?>
```

Archivo ejemplo_cifrado3.php, descifrado de datos con DES



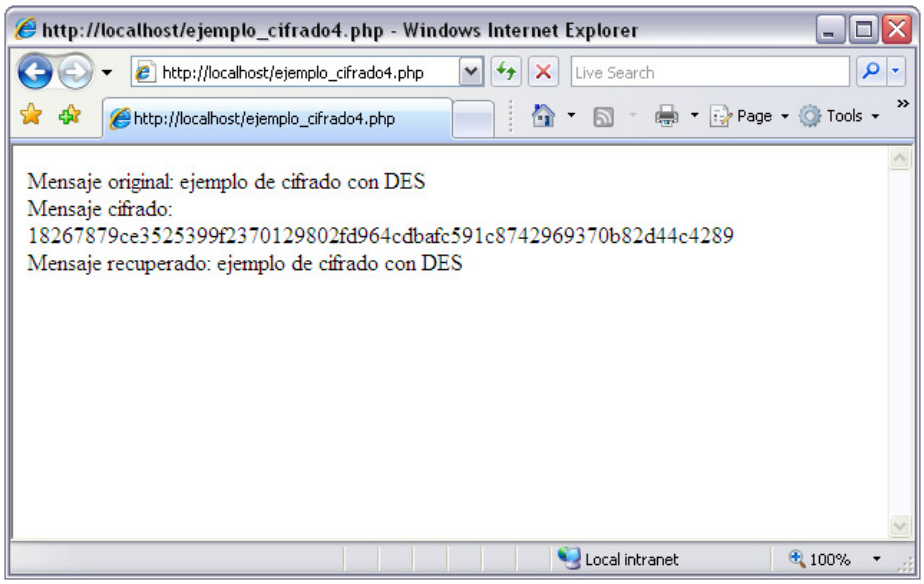
Ejecución archivo ejemplo_cifrado3.php

El proceso de descifrado de texto es el siguiente.

```
1       <?
2       include("des.php");
3
4       $llave = "8bytekey8bytekey8bytekey";
```

```
5       $mensaje = "ejemplo de cifrado con DES";
6       echo "Mensaje original: ".$mensaje."<br>";
7       $texto_cifrado = bin2hex(des($llave, $mensaje, 1, 0));
8       echo "Mensaje cifrado: ".$texto_cifrado."<br>";      //imprimimos el texto cifrado en DES
9
10      $mensaje_original = des($llave, pack("H*", $texto_cifrado), 0, 0);
11      echo ."Mensaje Recuperado: ".$mensaje_original."<br>";
12      ?>
```

Archivo ejemplo_cifrado4.php, cifrado y descifrado de datos con DES



Ejecución archivo ejemplo_cifrado4.php

Hay que tener en cuenta que esto no es 100% seguro, puesto que la contraseña se encripta en el servidor, entonces al enviar la contraseña desde el cliente al servidor podría ser interceptada. Por tal motivo es importante hacer uso de algún certificado digital o de conexiones seguras, por ejemplo utilizando SSL.

El archivo que contiene la función de cifrado en DES descrita anteriormente se encuentra en la carpeta D:\CapituloIII\cifrado php\.

JSP, Java Server Pages

JSP es un acrónimo de Java Server Pages, la cual es una tecnología orientada a crear páginas web con programación en Java.

Con JSP podemos crear aplicaciones web que se ejecuten en variados servidores web, de múltiples plataformas, ya que Java es en esencia un lenguaje multiplataforma. Las páginas JSP están compuestas de código HTML/XML mezclado con etiquetas especiales para programar scripts de servidor en sintaxis Java. Por tanto, las JSP podremos escribirlas con nuestro editor HTML/XML habitual. El motor de las páginas JSP está basado en los servlets de Java programos en Java destinados a ejecutarse en el servidor, aunque el número de desarrolladores que pueden afrontar la programación de JSP es mucho mayor, dado que resulta mucho más sencillo aprender que los servlets.

En JSP creamos páginas de manera parecida a como se crean en ASP o PHP. Generamos archivos con extensión .jsp que incluyen, dentro de la estructura de etiquetas HTML, las sentencias Java a ejecutar en el servidor. Antes de que sean funcionales los archivos, el motor JSP lleva a cabo una fase de traducción de esa página en un servlet, implementado en un archivo class (Byte codes de Java). Esta fase de traducción se lleva a cabo habitualmente cuando se recibe la primera solicitud de la página .jsp, aunque existe la opción de precompilar en código para evitar ese tiempo de espera la primera vez que un cliente solicita la página.

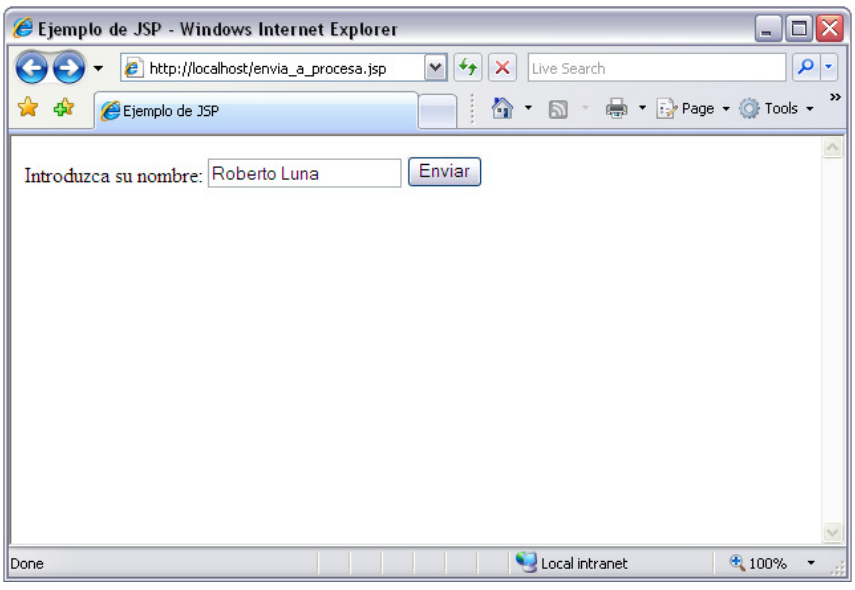
Validación de Formularios con JSP

El lenguaje JSP nos proporciona una manera sencilla de manejar formularios, permitiendo de esta manera procesar la información que el usuario ha introducido.

Al diseñar un formulario debemos indicar la página JSP que procesará el formulario, así como en método por el que se le pasará la información a la página.

```
1       <%@ page contentType="text/html; charset=iso-8859-1" language="java" import="java.sql.*"
errorPage="" %>
2       <html>
3       <head>
4         <title>Ejemplo de JSP</title>
5       </head>
6       <body>
7       introduzca su nombre:
8       <form action="procesa.jsp" method="get">
9       <input type="text" name="nombre"><br>
10      <input type="submit" value="Enviar">
11      </form>
12      </body>
13      </html>
```

Archivo envia_a_procesa.jsp, envio de formularios con GET



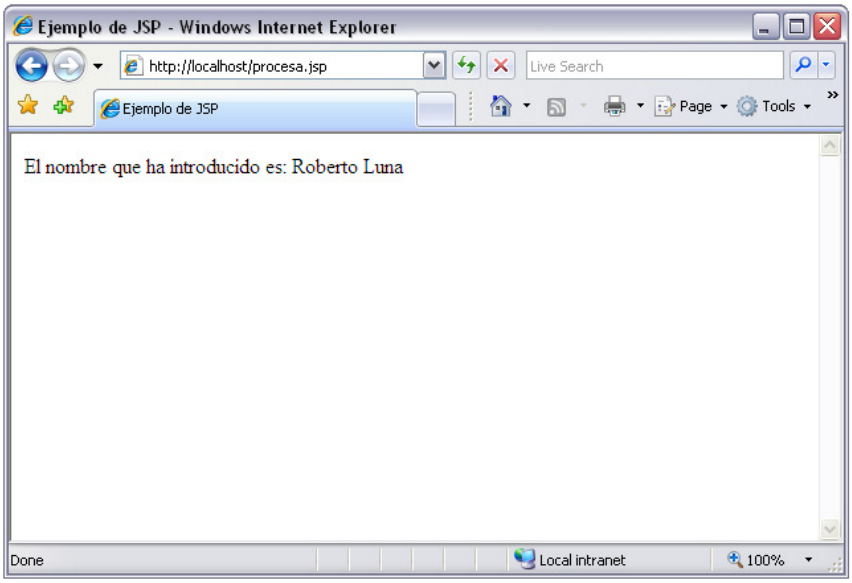
Ejecución archivo envia_a_procesa.jsp

Al hacer Click sobre el botón Enviar el contenido de cuadro de texto es enviado a la página que indicamos en el atributo **action** de la etiqueta **form**.

Para recoger los valores que han sido pasados a través de un formulario, tenemos que usar el objeto del servidor **Request** y sin importar cómo hayan sido pasados, si por **GET** o por **POST**, usaremos **request.getParameter("Nombre variable")**.

```
1       <%@ page contentType="text/html; charset=iso-8859-1" language="java" import="java.sql.*"
errorPage="" %>
2       <html>
3       <head>
4         <title>Ejemplo de JSP</title>
5       </head>
6       <body>
7               El nombre que ha introducido es: <% out.print(request.getParameter("nombre"));%>
8       </body>
9       </html>
```

Archivo procesa.jsp, mostrar datos de un formulario



Ejecución archivo envia_a_procesa.jsp

Validación de distintos tipos de campos en JSP

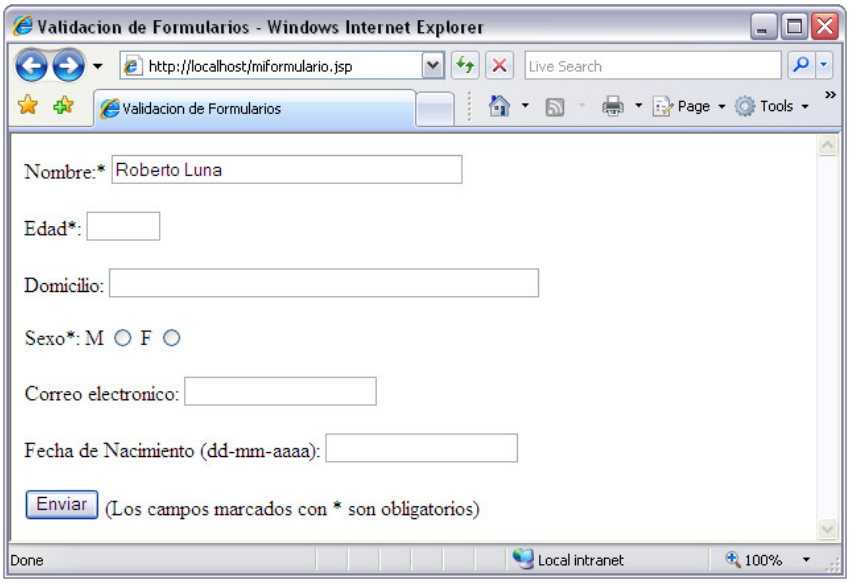
Anteriormente se analizó la forma de validar varios tipos de campos con PHP, ahora analizaremos la forma de validar los mismos tipos de campos con JSP. Tenemos el siguiente código:

```
1       <%@ page contentType="text/html; charset=iso-8859-1" language="java" import="java.sql.*"
errorPage="" %>
        <html>
2       <head>
3       </head>
```

```
5
6       <body>
7       <form id="form" name="form" method="post" action="validacion.jsp">
8
9       <p>Nombre:* <input name="nombre" type="text" id="nombre" size="40" maxlength="40" /></p>
10      <p>Edad*:
11        <input name="edad" type="text" id="edad" size="5" /></p>
12      <p>Domicilio: <input name="domicilio" type="text" id="domicilio" size="50" maxlength="70" /></p>
13      <p>Sexo*:
14                M <input name="sexo" type="radio" value="M" />
15                F <input name="sexo" type="radio" value="F" />
16      </p>
17      <p>Correo electronico: <input name="email" type="text" id="email" /></p>
18      <p>Fecha Nacimiento (dd-mm-aaaa): <input name="fnacimiento" type="text" id="fnacimiento" /></p>
19
20        <input type="submit" name="Submit" value="Enviar" />
21        (Los campos marcados con * son obligatorios)
22      </form>
23      </body>
24      </html>
```

Archivo miformulario.jsp, un formulario a ser validado

Que nos genera en pantalla lo siguiente:



Ejecución archivo miformulario.jsp

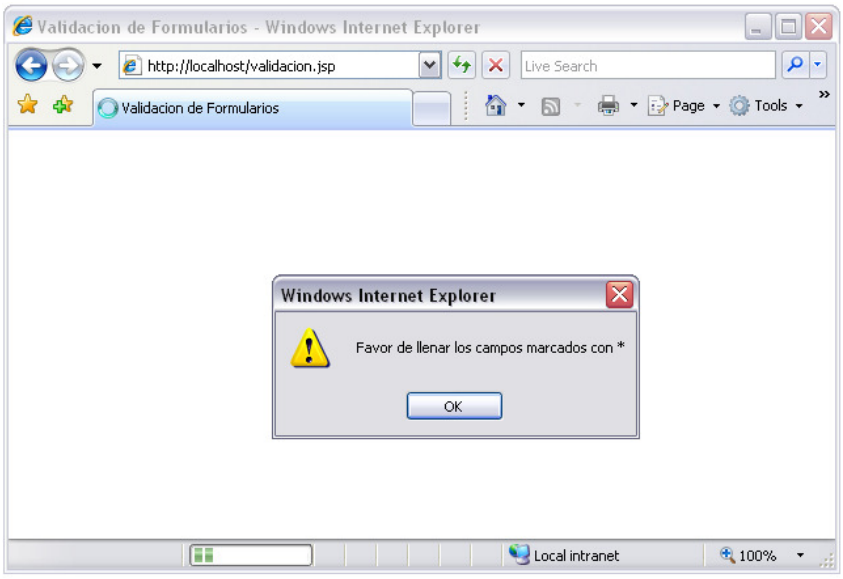
El formulario anterior debe ser validado por el archivo "validacion.jsp" tomando ciertas consideraciones, la primera de ellas es que los campos **Nombre**, **Edad** y **Sexo** no deben de ser vacios, el **Correo electrónico**, en caso de no ser vacio, debe tener una cadena con formato: "micorreo@midominio.algoonada.com" y la **Fecha de nacimiento** debe tener formato: "dd-mm-aaaa o dd/mm/aaaa" y aparte ser fecha válida, el código del archivo "validacion.php" para validar el formulario anterior es el siguiente:

```
1       <%
2               //validamos que ni el nombre ni la edad sean vacios
3               if(request.getParameter("nombre").compareTo("") == 0 ||
request.getParameter("edad").compareTo("") == 0){
4                       out.print("<script>");
5                       out.print("alert('Favor de llenar los campos marcados con *');");
6                       out.print("window.history.back();");
```

```
7                       out.print("</script>");
8               }
9
10      //validamos que el nombre tenga solo letras
11              try {
12                      int nnombre = Integer.parseInt(getParameter("nombre"));
13                      out.print("<script>");
14                      out.print("alert('El campo Nombre NO puede tener letras');");
15                      out.print("window.history.back();");
16                      out.print("</script>");
17              } catch (Exception e) {
18
19              }
20
21      //validamos que la edad tenga solo numeros
22              try {
23                      int nedad = Integer.parseInt(getParameter("edad"));
24              } catch (Exception e) {
25                      out.print("<script>");
26                      out.print("alert('El campo Edad NO puede tener letras');");
27                      out.print("window.history.back();");
28                      out.print("</script>");
29              }
30
31      //validamos que se haya seleccionado el sexo de la persona
32              if(request.getParameter("sexo").compareTo("")==0){
33                      out.print("<script>");
34                      out.print("alert('Seleccione el sexo de la persona');");
35                      out.print("window.history.back();");
36                      out.print("</script>");
37              }
38
39              if(request.getParameter("fnacimiento").length() != 0){ //validamos que el campo fecha de
nacimiento no sea vacia
40                      String dateFormat = "MM/dd/yyyy";
41                      int dateFormatLength = dateFormat.length();
42                      try {
43                              if (inDate.length() != dateFormatLength)
44                                      throw new Exception();
45                              else {
46                                      SimpleDateFormat format = new SimpleDateFormat(dateFormat);
47                                      format.setLenient(false);
48                                      Date theDate = new Date();
49                                      theDate = format.parse(inDate);
50                                      //si llega aquí la fecha es correcta
51                              }
52                      }
53                      catch(Exception e){
54                              out.print("<script>");
55                              out.print("alert('La fecha de nacimiento no es valida');");
56                              out.print("window.history.back();");
57                              out.print("</script>");
58                      }
59
60              }
61      %>
```

Archivo validacion.jsp, validación de formularios con JSP

En caso de algún error, aparecerá en pantalla el mensaje correspondiente la ejecución regresará a la pantalla del formulario para reintroducir los datos.



Ejecución archivo validacion.jsp

Servlets

Los Servlets son una serie de aplicaciones programadas en Java que se ejecutan completamente en un servidor. Un servlet va a aceptar una petición de un cliente a través del Web Server, hará su tarea y devolverá al cliente una respuesta.

Los Servlets son el sustituto de los antiguos CGI (Common Gateway Interface), puesto que los CGI estaban escritos en C ó Perl y los Servlets estarán escritos en Java, aportando este lenguaje la independencia de plataforma. Algunas ventajas de los servlets frente a CGI son:

- Persistencia de los servlets: Los servlets se cargan una sola vez por el Web Server y pueden mantener la conexión entre varias peticiones.
- Rapidez de los Servlets: puesto que sólo se cargan una vez.
- Independencia de plataforma.
- Extensibilidad de los Servlets. Como están escritos en Java, aportan todos los beneficios de este lenguaje. Java es un lenguaje robusto y orientado a objetos, por lo que es fácilmente extensible a nuestras necesidades.
- Seguridad de los Servlets: La única forma de invocar un servlet es a través de un Web Server. Esto da un alto nivel de seguridad, especialmente si el Web Server está protegido por un firewall. Esto significa que el cliente no puede borrar ni modificar nada del propio servidor. Para ampliar la seguridad, se pueden definir usuarios y grupos de usuarios. Por último decir que se pueden usar características nativas de seguridad, como el encriptamiento de mensajes.
- Los servlets pueden ser usados por cualquier número de clientes.

Validación de Formularios con Servlets

Ahora veremos cómo se recuperan los datos de un formulario típico de cualquier página web. Los servlets normalmente reciben los datos de entrada a través de las peticiones POST o GET. Los

métodos que se usan para recuperar los datos son los mismos en cada caso. Los tres métodos usados para recuperar los parametros son **getParameterNames()**, **getParameter()** y **getParameterValues()** y tienen la siguiente definición:

- public Enumeration ServletRequest.getParameterNames();
- public String ServletRequest.getParameter(String nombre);
- public String ServletRequest.getParameterValues(String nombre);

getParameterNames() devuelve los nombres de los parámetros de la petición como una enumeración de cadenas, o una enumeración vacía si no hay parámetros en la petición. Se usa como un método soportado de **getParameter()**. Cuando tienes una enumeración de la lista de los nombres de los parámetros, puedes iterar con ellos, llamando al método **getParameter()** con cada uno de los nombres de la lista.

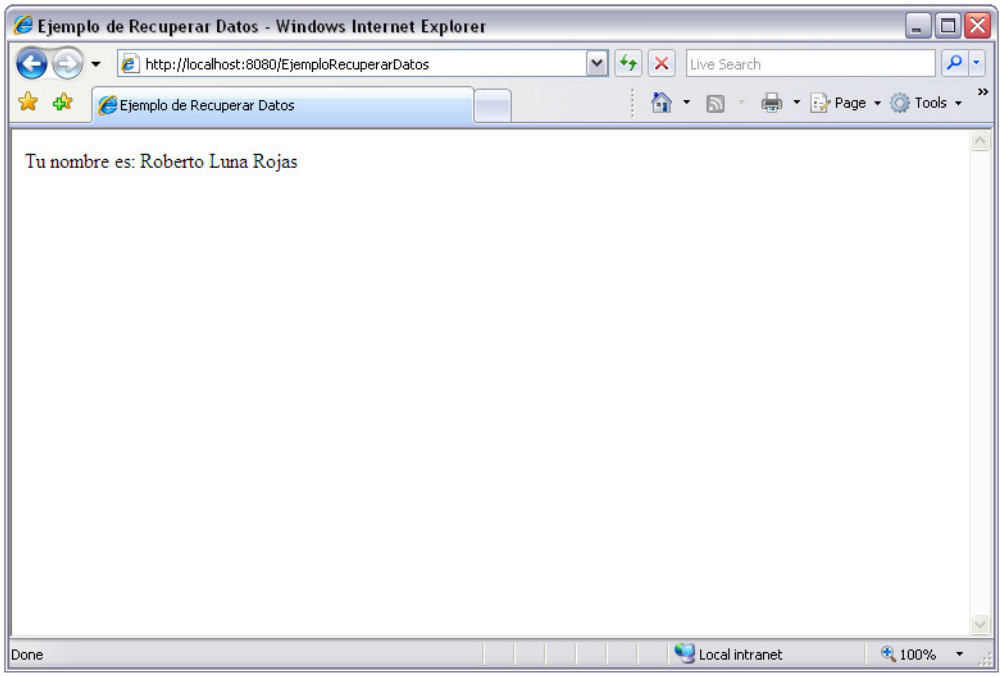
getParameter() devuelve una cadena que contiene el valor simple del parámetro especificado, o null si el parámetro no existe en la petición. Este método, aunque es el más utilizado, sólo debería ser usado si estás totalmente seguro de que existe el parámetro que se va a recoger. Si el parámetro tiene múltiples valores debe utilizarse **getParameterValues()**.

getParameterValues() devuelve los valores de los parámetros especificados como una array de caracteres, o **null** si no existe el parámetro en la petición.

Vamos a ver un ejemplo de un servlet que implementa los casos anteriores:

```
1	import javax.servlet.*;
2	import javax.servlet.http.*;
3	import java.io.*;
4	import java.util.*;
5	public class EjemploRecuperarDatos extends HttpServlet{
6		public void init(ServletConfig config) throws SerlvetException {
7			// Siempre se pasa el objet ServletConfig a la superclase
8			super.init(config);
9		}
10		// Proceso HTTP Get de la petición
11		public void doGet(HttpSerlvetRequest peticion,HttpserlvetResponse respuesta)
throws SerlvetException, IOException {
12			doPost(peticion, respuesta);
13		}
14		// Proceso HTTP Post de la petición
15	public void doPost(HttpSerlvetRequest peticion,HttpServletResponse respuesta) throws
ServletException,IOException {
16			respuesta.setContentType("text/html");
17			PrintWriter out = respuesta.getWriter();
18			out.println("<html>");
19			out.println("<head><title>Ejemplo de Recuperar Datos</title></head>");
20			out.println("<body>");
21			out.println("Tu nombre es: ");
22			out.println(peticion.getParameter("nombre"));
23			out.println(peticion.getParameter("apaterno"));
24			out.println(peticion.getParameter("amaterno"));
25			out.println("</body></html>");
26			out.close();
27		}
28		// Devuelve la información del Servlet
29		public String getServletInfo(){
30			return "Información del Servlet de Recuperación de Parametros";
31		}
32	}
```

Archivo EjemploRecuperarDatos.java, recuperación de datos de un formulario



Ejecución archivo EjemploRecuperarDatos.java

En el servlet anterior podemos ver dos funciones importantes, doGet (línea 11) y doPost (línea 17). Para ser un servlet, la clase debe extender (o heredar) de la clase HttpServlet y sobreescribir doGet o doPost (o ambos), dependiendo de si los datos están siendo enviados mediante GET o POST. Estos métodos toman dos argumentos: un HttpServletRequest y un HttpServletResponse.

El HttpServletRequest tiene métodos que nos permiten encontrar información entrante como datos de un FORM, cabeceras de petición HTTP, etc. El HttpServletResponse tiene métodos que nos permiten especificar líneas de respuesta HTTP (200, 404, etc.), cabeceras de respuesta (Content-Type, Set-Cookie, etc.) y, todavía más importante, nos permiten obtener un PrintWriter usado para envíar la salida de vuelta al cliente.

Para servlets sencillos, la mayoría del esfuerzo se gasta en sentencias println que generan la página deseada. Observamos que doGet y doPost lanzan dos excepciones, por eso es necesario incluirlas en la declaración. También observamos que tenemos que importar las clases de los paquetes java.io (para PrintWriter), javax.servlet (para HttpServlet), y javax.servlet.http (para HttpServletRequest y HttpServletResponse). Finalmente, observamos que doGet y doPost son llamados por el método service, y algunas veces queremos sobreescribir directamente el método service, por ejemplo, para un servlet que maneje tanto peticiones GET como POST.

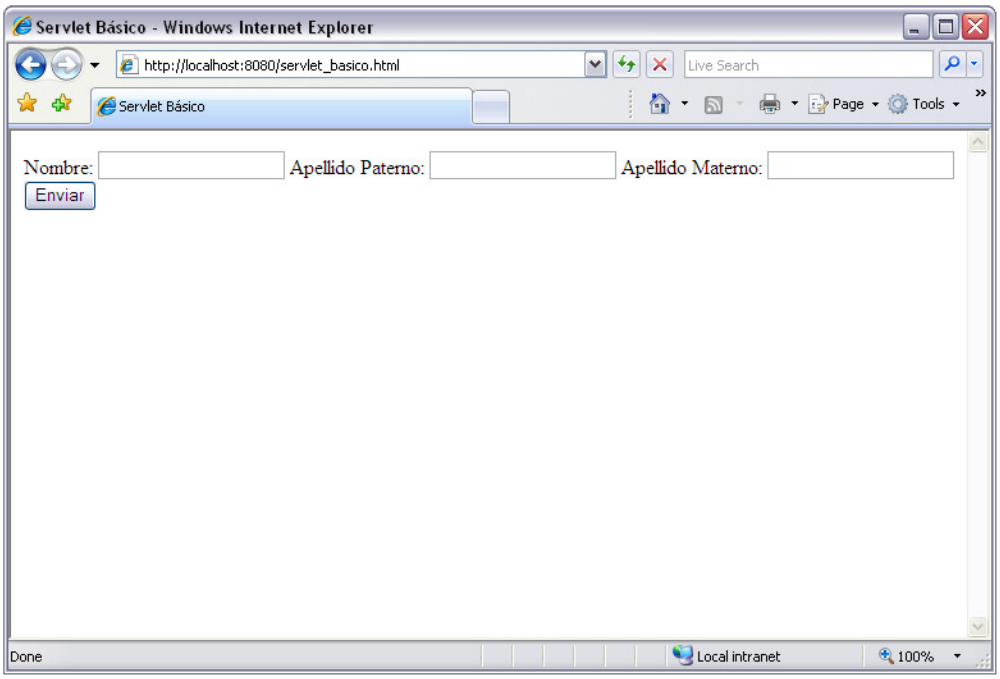
Este serlvet, también debe ser compilado y colocado en la carpeta adecuada para que Java Web Server pueda encontrarlo. Ahora veremos cómo hacer la página web que contiene el formulario con los datos y cómo llamar al servlet anterior.

```
1       <html>
2       <head>
3       <title>Servlet Básico</title>
4       </head>
5       <body>
6       <form action="http://localhost:8080/servlet/EjemploRecuperarDatos"method="POST">
```

```
7       Nombre: <input name="nombre" type="text">
8       Apellido Paterno: <input name="apaterno" type="text">
9       Apellido Materno: <input name="amaterno" type="text">
10      <input type="submit" value="Enviar" name="Enviar">
11      </form>
12      </body>
13      </html>
```

Archivo servlet_basico.html, formulario a ser validado con Servlets

Después de esto sólo nos queda probar la llamada al servlet, y la ejecución del mismo.



Ejecución archivo servlet_basico.html

Todas las operaciones sobre los elementos HTML realizadas con JSP son equivalentes para los Servlets, la única diferencia es que el código de los Servlets siempre debe estar dentro de una clase Java y que los valores de los elementos HTML se deben obtener ya sea con el método **doGet** o con **doPost** como se explicó anteriormente. Además, en un Servlet no debemos colocar los simbolos “<”, “>”, ”%”, “@”, ni la directiva **page**, ya que todos esos símbolos y códigos pertenecen a JSP.

Manejo de sesiones con JSP / Servlets

Para poder hacer uso de las sesiones se debe poner el atributo session de la directiva **page** a true, de esta forma se notifica al contenedor que la página interviene en un proceso que utiliza las sesiones del protocolo HTTP:

```
1       <%@page import="java.util.*" session=’true’%> //no necesario en Servlets
```

En JSP y en los Servlets, las acciones que se pueden realizar sobre las sesiones se llevan a cabo mediante la interface **HttpSession** y los métodos que implementa. Esta interfaz está incluida dentro del paquete **javax.servlet.http** y es utilizada por el contenedor de páginas JSP para crear una sesión entre el servidor y el cliente.

Para obtener la sesión de un usuario se utiliza el método **getSession()** que devuelve una interfaz de tipo **HttpSession**.

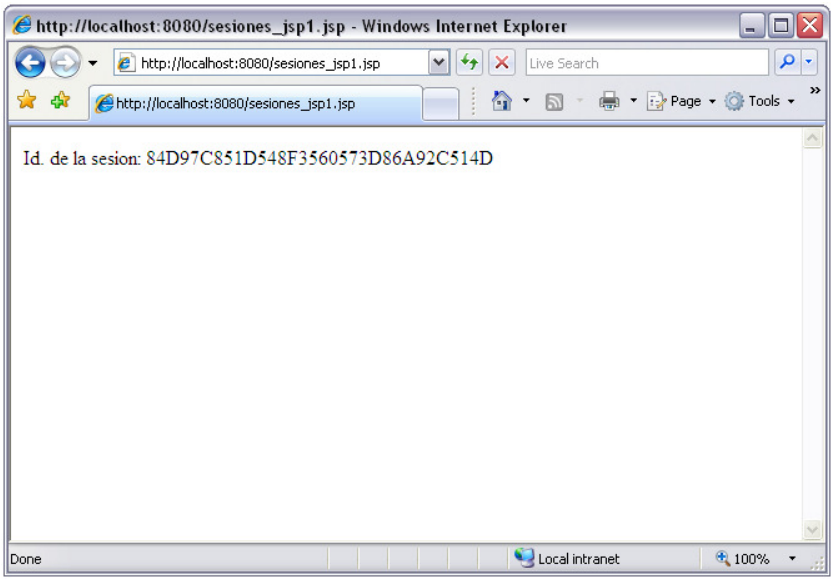
```
1       <%@page import="java.util.*" session='true'%>  //no necesario en Servlets
2       <%                                              //no necesario en Servlets
3       HttpSession sesion=request.getSession();
4       %>                                              //no necesario en Servlets
```

Una vez creado el objeto de tipo sesión es posible acceder a una serie de datos sobre la misma. Uno de estos datos es **idSession** que devuelve un identificador único asociado a una sesión:

```
1       <%@page import="java.util.*" session='true'%>
2       <%
3       HttpSession session = request.getSession();
4       out.print("Id. de la Sesion: "+sesion.getId());
5       %>
```

Archivo sesiones_jsp1.jsp, inicio de sesiones con JSP / Servlets

El ejemplo en marcha es el siguiente:



Ejecución archivo sesiones_jsp1.jsp

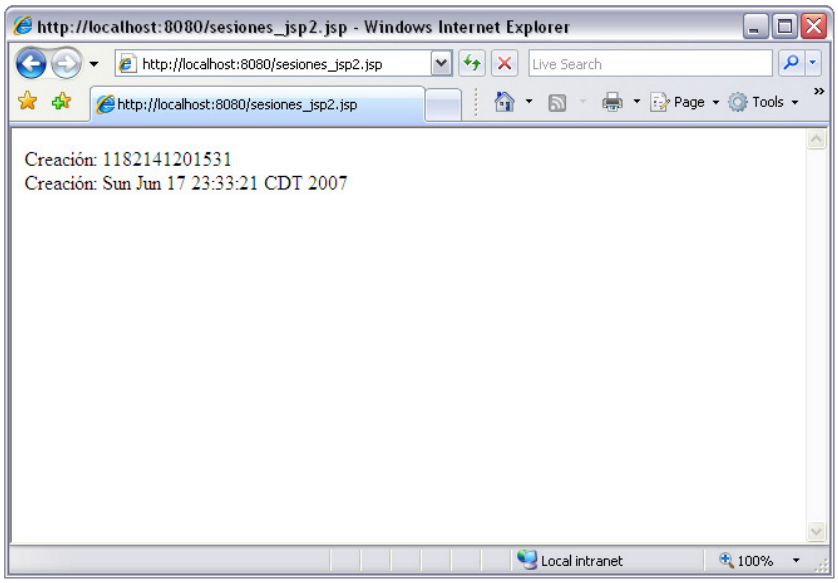
Es posible conocer el momento en el que se creó la sesión:

```
1       <%@page import="java.util.*" session="true"%>
2       <%
```

```
3       HttpSession sesion=request.getSession();
4       out.println("Creación: "+sesion.getCreationTime());
5       Date momento=new Date(sesion.getCreationTime());
6       out.println("<BR>Creación: "+momento);
7       %>
```

Archivo sesiones_jsp2.jsp, funciones de sesiones con JSP / Servlets

El código anterior nos genera:



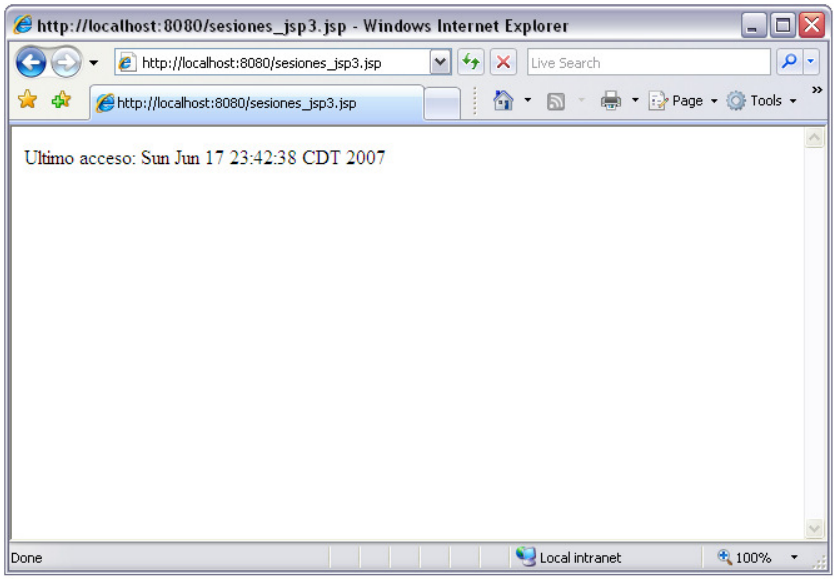
Ejecución archivo sesiones_jsp2.jsp

En el primer caso se muestra el dato tal cual lo devuelve el método **getCreationTime()**, que es una fecha en formato long, mientras que en el segundo caso se le da formato para que tenga un aspecto más común.

También se puede conocer la fecha y hora de la última vez que el cliente accedió al servidor con el que se creó la sesión, utilizando el método **getLastAccesedTime()**:

```
1       <%@page import="java.util.*" session='true'%>
2       <%
3       HttpSession sesion=request.getSession();
4       Date acceso=new Date(sesion.getLastAccessedTime());
5       out.println("Último acceso: "+acceso+"<br>");
6       %>
```

Archivo sesiones_jsp3.jsp, funciones de sesiones con JSP / Servlets

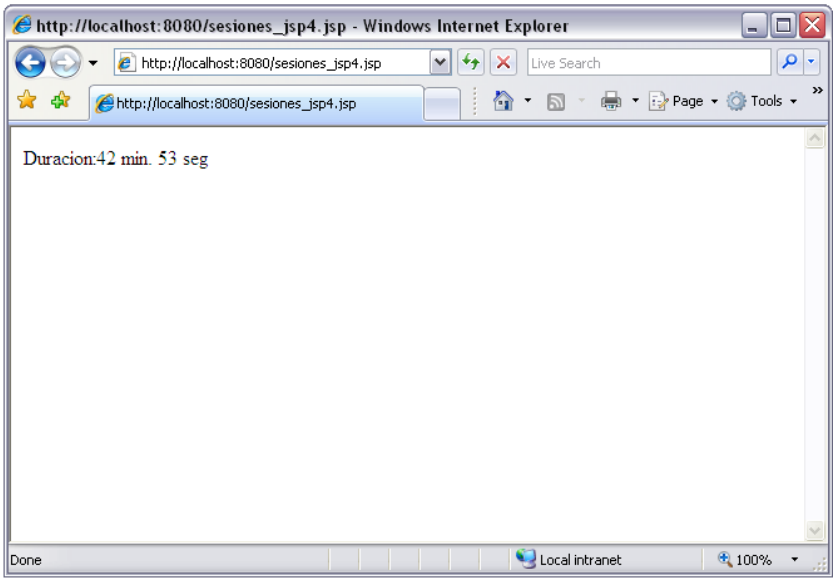


Archivo sesiones_jsp3.jsp

Teniendo en cuenta el momento en el que ser creó la sesión y la última vez que se accedió al servidor, se puede conocer el tiempo que lleva el cliente conectado al servidor, o lo que es lo mismo el tiempo que lleva el usuario navegando por las páginas JSP:

```
1       <%@page import="java.util.*" session='true'%>
2       <%
3       HttpSession sesion=request.getSession();
4       long longDuracion=sesion.getLastAccessedTime();
5       sesion.getCreationTime();
6       Date duracion=new Date(longDuracion);
7       out.println("Duracion:"+duracion.getMinutes()+"min."+duracion.getSeconds()+"seg");
8       %>
```

Archivo sesiones_jsp4.jsp, funciones de sesiones con JSP / Servlets



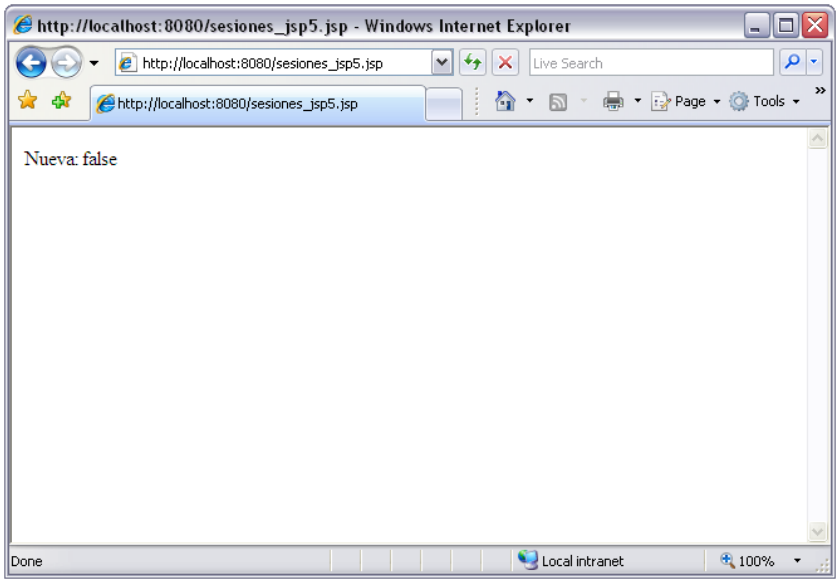
Ejecución archivo sesiones_jsp4.jsp

La interfaz HttpSession ofrece el método isNew() mediante el cual es posible saber si la sesión creada es nueva o se está tomando de una previamente creada:

```
1       <%@page import="java.util.*" session='true'%>
2       <%
3       HttpSession sesion=request.getSession();
4       out.println("nueva: "+sesion.isNew());
5       %>
```

Archivo sesiones_jsp5.jsp, funciones de sesiones con JSP / Servlets

Si se ejecuta el ejemplo la primera vez el método devolverá **true**, ya que previamente no había ninguna sesión y ha sido creada en ese instante. Si se recarga la página devolverá **false** ya que la sesión ya ha sido creada.



Ejecución archivo sesiones_jsp5.jsp

Las sesiones no necesitan ningún tipo de mantenimiento, una vez creadas no es necesario utilizarlas de forma obligatoria o tener que refrescar los datos asociados a las mismas, se deben ver como una variable más con la diferencia que pueden ser utilizadas en cualquier página independientemente del lugar en el que hayan sido creadas.

Guardar Objetos en una Sesión

Para guardar un objeto en una sesión se utiliza el método **setAttribute()**, que ha sustituido al método **putValue()**. Este método utiliza dos argumentos:

- El primero es el nombre que identificará a esa variable.
- El segundo es el dato que se va a guardar.

```
SetAttribute(java.lang.String name, java.lang.Object value)
```

Un ejemplo de cómo guardar una cadena de texto en la sesión:

```
1       <%@page import="java.util.*" session="true" %>
2       <%
3       HttpSession sesion=request.getSession();
4       sesion.setAttribute("trabajo","Sesiones en JSP");
5       %>
```

Si se quiere pasar un parámetro que no sea un objeto es necesario realizar una conversión:

```
1       <%@page import="java.util.*" session="true" %>
2       <%
3       HttpSession sesion=request.getSession();
4       Integer edad=new Integer(26);
5       sesion.setAttribute("edad",edad);
6       %>
```

Si se hubiera utilizado el valor entero en vez del objeto Integer, el resultado habría sido un error en el momento de ejecutar el script.

En el primer ejemplo este no sucedería puesto que una cadena es un objeto de tipo String, no así un entero. Así habría sido igual si en el primer caso ponemos:

```
1       <%@page import="java.util.*" session="true" %>
2       <%
3       HttpSession sesion=request.getSession();
4       String nombre=new String("Sesiones en JSP");
5       sesion.setAttribute("trabajo",nombre);
6       %>
```

En caso de tratarse objeto de tipo Vector (parecido a un array con dos diferencias: la primera es que puede almacenar todo tipo de objetos, y la segunda es que no es necesario establecer de forma previa el tamaño que va a tener) que almacene los 7 días de la semana. El código sería el siguiente:

```
1       <%@page import="java.util.*" session="true" %>
2       <%
3       HttpSession sesion=request.getSession();
4       Vector v=new Vector();
5       v.addElement(new String("Lunes"));
6       v.addElement(new String("Martes"));
7       v.addElement(new String("Miercoles"));
8       v.addElement(new String("Jueves"));
9       v.addElement(new String("Viernes"));
10      v.addElement(new String("Sábado"));
11      v.addElement(new String("Domingo"));
12      sesion.setAttribute("diasSemana",v);
13      %>
```

Recuperar objetos de una Sesión

Los datos que se guardan en la sesión permanecen ahí a la espera de ser utilizados. Para ello es necesario realizar el proceso contrario a cuando se graban, comenzando por la recuperación del objeto de la sesión para empezar a ser tratado.

Para poder realizar este paso se utiliza el método **getAttribute()** (anteriormente se utilizaba el método **getValue()**, pero este método se encuentra en desuso), utilizando como argumento el nombre que identifica al objeto que se quiere recuperar.

```
getAttribute(java.lang,String nombre)
```

Un ejemplo de recuperación de objetos almacenados en la sesión:

```
1       <%@page import="java.util.*" session="true" %>
2       <%
3       HttpSession sesion=request.getSession();
4       Sesion.getAttribute("nombre");
5       %>
```

Cuando este método devuelve el objeto no establece en ningún momento de qué tipo de objeto se trata, por ello si se conoce previamente el tipo de objeto que puede devolver tras ser recuperado de la sesión es necesario realizar un cast, para convertir el objeto de tipo genérico al objeto exacto que se va a usar. Para realizar esta operación se añade el tipo de objeto al lado de tipo **HttpSession** que utiliza el método **getAttribute()** para obtener el objeto que devuelve:

```
1       <%@page import="java.util.*" session="true" %>
2       <%
3       HttpSession sesion=request.getSession();
4       String nombre=(String)sesion.getAttribute("nombre");
5       out.println("Contenido de nombre: "+nombre);
6       %>
```

Si no existe ningún objeto almacenado en la sesión bajo el identificador que se utiliza en el método **getAttribute()**, el valor devuelto será null. Por ello habrá que prestar especial atención ya que si se realiza el **cast** de un valor **null** el contenedor JSP devolverá un error. Lo mejor en estos casos es adelantarse a los posibles errores que pueda haber.

```
1       <%@page import="java.util.*" session="true" %>
2       <%
3       if(sesion.getAttribute("nombre")!=null){
4               String nombre=(String)sesion.getAttribute("nombre");
5               out.println("Contenido de nombre: "+nombre);
6       }
7       %>
```

En el caso de tipos primitivos deben de ser convertidos a objetos previamente a su integración sesión de tal forma que su proceso de extracción viene a ser similar:

```
1       <%@page import="java.util.*" session="true" %>
2       <%
3       HttpSession sesion=request.getSession();
4       Integer edad=(Integer)sesion.getAttribute("edad");
5       out.println("Edad: "+edad.intValue());
6       %>
```

En esta ocasión el objeto devuelto y convertido a Integer no es del todo válido ya que es necesario obtener de él el valor entero. Para ello se utiliza el método intValue() que devuelve el valor que realmente se había guardado previamente en la sesión. Por último, el ejemplo del vector guardado en la sesión tiene un tratamiento similar al de los casos anteriores. El primer paso es recuperar el objeto de la sesión:

```
1       <%@page import="java.util.*" session="true" %>
2       <%
```

```
3       HttpSession sesion=request.getSession();
4       sesion.getAttribute("diasSemana");
5       %>
```

Como se sabe que el objeto es de tipo Vector se puede recuperar y convertir en un solo paso:

```
Vector v= (Vector) sesion.getAttribute("diasSemana");
```

A partir de este momento se puede acceder a los elementos del vector independientemente de si venía de una sesión o ha sido creado. Para ello se utiliza el método size() que devuelve el tamaño del vector para ir leyendo cada uno de sus elementos:

```
1       <%
2       for(int i=0; i<v.size(); i++){
3               out.println("<b>Dia: </b>"+(String)v.get(i)+"<br>");
4       }
5       %>
```

Se ha realizado otro proceso de conversión que sin ser necesario, ayuda a entender mejor el funcionamiento. Cuando se recupera un elemento de un vector (que se trata de un objeto) es necesario realizar el casting y convertirlo a su tipo de objeto definitivo. El resultado sería el siguiente:
Para recuperar todos los objetos de una sesión se puede hacer uso también del método **getAttributeNames()** de la interfaz **HttpSession**. Para recoger todos los objetos almacenados en la sesión se recorre el objeto **Enumeration** que contiene el nombre de todos los objetos que contiene la sesión y que ha sido devuelto por el método **getAttributeNames()**. Cada nombre de atributo de la sesión se utiliza en la llamada a cada método **getAttribute()**, que devolverá el objeto correspondiente.

```
1       <%@page session="true" language="java" import="java.util.*" %>
2       <%
3       HttpSession sesion=request.getSession();
4       String nombre="Sesiones con JSP";
5       Vector v=new Vector();
6       v.addElement(new String("Lunes"));
7       v.addElement(new String("Martes"));
8       v.addElement(new String("Miercoles"));
9       v.addElement(new String("Jueves"));
10      v.addElement(new String("Viernes"));
11      v.addElement(new String("Sábado"));
12      v.addElement(new String("Domingo"));
13      sesion.setAttribute("diasSemana",v);
14      sesion.setAttribute("nombre",nombre);
15      %>
```

Con el siguiente código se recuperan los objetos:

```
1       <%@page import="java.util.*" session="true" %>
2       <%
3       HttpSession sesion=request.getSession();
4       Enumeration enum=sesion.getAttributeNames();
5       String nombreAtributo;
6       while (enum.hasMoreElements()){
7               nombreAtributo=(String)enum.nextElement();
8               out.println("<b>Atributo:</b>"+nombreAtributo);
9               if(nombreAtributo.equals("diasSemana")){
10              Vector v= (Vector)
11              sesion.getAttribute("diasSemana");
12              for(int i=0; i<v.size(); i++){
13                      out.println("<br><b>Dia:
```

```
14                          </b>"+(String)v.get(i));
15                  }
16          }
17          else
18                  out.println("<b>"+sesion.getAttribute(nombreAtributo)+"</b><BR><BR>");
19          }
20          %>
```

Destruir una Sesión

Como ya se ha visto, los datos almacenados por las sesiones pueden destruirse en tres casos:

- El usuario abandona aplicación web (cambia de web o cierra el navegador)
- Se alcanza el tiempo máximo permitido de inactividad de un usuario (timeout).
- El servidor se para o se reincia.

Pero la situación más probable es querer iniciar las sesiones o dar por finalizada una si se han cumplido una o varias condiciones. En este caso no es necesario esperar a que ocurra alguno de los tres casos citados anteriormente, ya que mediante el método **invalidate()** es posible destruir una sesión concreta.

En el siguiente caso la sesión "sesión" se destruye al invocar el método **invalidate()**; y por lo tanto el valor u objeto que está asociado a la misma.

```
1           <%
2           [...]
30          sesion.invalidate();
31          %>
```

Manejo de Cookies con JSP / Servlets

Las sesiones vistas anteriormente basan su funcionamiento en los cookies. Cuando se hace uso de la interfaz **HttpSession** de forma interna y totalmente transparente al programador se está haciendo uso de los **cookies**. De hecho cuando a través de una página JSP o de un Servlet se comienza una sesión, se crea una cookie llamada **JSSESIONID**. La diferencia es que ésta es temporal y durará el tiempo que permanezca el navegador ejecutándose, siendo borrada cuando el usuario cierre el navegador.

El objetivo de utilizar cookies es poder reconocer al usuario en el momento en el que se conecta al servidor. Una de las páginas que recoge la petición del usuario puede comprobar si existe una cookie que ha dejado anteriormente, si es así, sabe que ese usuario ya ha visitado ese sitio y por lo tanto puede leer valores que le identifiquen. Otro de los usos de los cookies es ofrecer una personalización al usuario. En muchos sitos web es posible elegir el color de fondo, el tipo de letra utilizado, etc. Estos valores pueden ser almacenados en cookies de forma que cuando acceda de nuevo al sitio y se compruebe la existencia de esos valores, serán recuperados para utilizarlos en la personalización de la página tal y como el usuario estableció en su momento. Un ejemplo que se ha podido encontrar en muchas páginas es en el momento de realizar un registro o solicitar el alta en un área restringida, ya que en muchas ocasiones existe un checkbox que cuando se selecciona permite recordar el nombre de usuario a falta de que sólo se escriba la clave.
Para trabajar con cookies se utiliza la clase **Cookie** que está disponible en paquete **javax.servlet.http**. Por medio de esta clase se pueden crear cookies, establecer sus valores y nombres, alguna de sus propiedades, eliminarlas, leer valores que almacenan, etc.

Crear una cookie

Una cookie almacenada en la PC de un usuario está compuesta por un nombre y un valor asociado al mismo. Además, asociada a esta cookie pueden existir una serie de atributos que definen datos como su tiempo de vida, alcance, dominio, etc.

Cabe reseñar que las cookies, no son más que archivos de texto, que no pueden superar un tamaño de 4Kb, además los navegadores tan sólo pueden aceptar 20 cookies de un mismo servidor web (300 cookies en total).

Para crear un objeto de tipo Cookie se utiliza el constructor de la clase Cookie que requiere su nombre y el valor a guardar. El siguiente ejemplo crearía un objeto Cookie que contiene el nombre "nombre" y el valor "objetos".

```
1       <%
2       Cookie miCookie=new Cookie("nombre","objetos");
3       %>
```

También es posible crear cookies con contenido que se genere de forma dinámica. El siguiente código muestra un cookie que guarda un texto que está concatenado a la fecha/hora en ese momento:

```
1       <%@page contentType="text/html; charset=iso-8859-1"
2       session="true" language="java" import="java.util.*" %>
3       <%
4       Cookie miCookie=null;
5       Date fecha=new Date();
6       String texto= "Este es el texto que vamos a guardar en la cookie"+fecha;
7       miCookie=new Cookie("nombre",texto);
8       %>
```

En esta ocasión el contenido del valor a guardar en la cookie está en la variable "texto". También se pueden guardar valores o datos que provengan de páginas anteriores y que hayan sido introducidas a través de un formulario:

```
1       <%
2       Cookie miCookie=null;
3       String ciudad= request.getParameter("formCiudad");
4       miCookie=new Cookie("ciudadFavorita",ciudad);
5       %>
```

Una vez que se ha creado una cookie, es necesario establecer una serie de atributos para poder ser utilizada. El primero de esos atributos es el que se conoce como tiempo de vida. Por defecto, cuando creamos una cookie, se mantiene mientras dura la ejecución del navegador. Si el usuario cierra el navegador, las cookies que no tengan establecido un tiempo de vida serán destruidas. Por tanto, si se quiere que una cookie dure más tiempo y esté disponible para otras situaciones es necesario establecer un valor de tiempo (en segundos) que será la duración o tiempo de vida de la cookie. Para establecer este atributo se utiliza el método **setMaxAge()**. El siguiente ejemplo establece un tiempo de 31 días de vida para el cookie "unaCookie":

```
1       <%
2       unaCookie.setMaxAge(60*60*24*31);
3       %>
```

Si se utiliza un valor positivo, la cookie será destruida después de haber pasado ese tiempo, si el valor es negativo la cookie no será almacenada y se borrará cuando el usuario cierre el navegador. Por último si el valor que se establece como tiempo es cero, la cookie será borrada.

Otros de los atributos que se incluye cuando se crea una cookie es el path desde el que será visto, es decir, si el valor del path es "/" (raíz), quiere decir que en todo el sitio se podrá utilizar esa cookie, pero si el valor es "/datos" quiere decir que el valor de la cookie sólo será visible dentro del subdominio "datos". Este atributo se establece mediante el método **setPath()**.

```
1       <%
2       unaCookie.setPath("/");
3       %>
```

Para conocer el valor de path, se puede utilizar el método **getPath()**.

```
1       <%
2       out.println("cookie visible en: "+unaCookie.getPath());
3       %>
```

Existe un método dentro de la clase Cookie que permite establecer el dominio desde el cual se ha generado la cookie. Este método tiene su significado porque un navegador sólo envía al servidor las cookies que coinciden con el dominio del servidor que los envió. Si en alguna ocasión se requiere que estén disponibles desde otros subdominios se especifica con el método **setDomain()**.

Si no se establece la propiedad domain se entiende que la cookie será vista sólo desde el dominio que la creó, pero sin embardo si se especifica un nombre de dominio se entenderá que la cookie será visto en aquellos dominios que contengan el nombre especificado.

En el siguiente ejemplo hace que la cookie definida en el objeto "unaCookie" esté disponible para todos los dominios que contengan el nombre ".midominio.com". Un nombre de dominio debe comenzar por un punto.

```
1       <%
2       unaCookie.setDomain(".midominio.com");
3       %>
```

Igualmente, para conocer el dominio sobre el que actúa la cookie, basta con utilizar el método **getDomain()** para obtener esa información.

Una vez que se ha creado el objeto Cookie, y se han establecido todos los atributos necesarios es el momento de crear realmente, ya que hasta ahora sólo se tenía un objeto que representa esa cookie. Para crear el archivo cookie real, se utiliza el método **addCookie()** de la interfaz **HttpServletResponse**:

```
1       <%
2       response.addCookie(unaCookie);
3       %>
```

Una vez ejecutada esta línea es cuando la cookie existe en el equipo del cliente que ha accedido a la página. Es importante señalar que si no se ejecuta esta última línea la cookie no habrá sido grabada, y por lo tanto, cualquier aplicación o página que espere encontrar dicha cookie no lo encontrará.

Recuperar una cookie

El proceso de recuperar una cookie determinada puede parecer algo complejo, ya que no hay una forma de poder acceder a una cookie de forma directa. Por este motivo es necesario recoger todos las cookies que existen hasta ese momento e ir buscando aquella que se requiera, y que al menos, se conoce su nombre.

Para recoger todos las cookies que tenga el usuario guardadas se crea un array de tipo Cookie, y se utiliza el método **getCookies()** de la interfaz **HttpServletRequest** para recuperarlos:

```
1       <%
2       Cookie [] todasLasCookies=request.getCookies();
3       /* El siguiente paso es crear un ciclo que vaya leyendo todas las cookies. */
4
5       for(int i=0;i<todasLasCookies.length;i++){
6               Cookie unaCookie=todasLasCookies[i];

7       /* A continuación se comparan los nombres de cada una de las cookies con el que se está buscando. Si se
encuentra una cookie con ese nombre se ha encontrado la que se andaba buscando, de forma que se sale del ciclo
mediante un break. */
8
9       if(unaCookie.getName().equals("nombre"))
10              break;
11      }
12
13      /* Una vez localizado tan sólo queda utilizar los métodos apropiados para obtener la información necesaria
que contiene. */
14
15      out.println("Nombre: "+unaCookie.getName()+"<BR>");
16      out.println("Valor: "+unaCookie.getValue()+"<BR>");
17      out.println("Path: "+unaCookie.getPath()+"<BR>");
18      out.println("Tiempo de vida:"+unaCookie.getMaxAge()+"<BR>");
19      out.println("Dominio: "+unaCookie.getDomain()+"<BR>");
20      %>
```

Los ejemplos analizados en el capítulo se encuentran disponibles en D:\CapituloIII\.

CAPITULO IV.- CONECTIVIDAD CON EL SMBD

Es importante aclarar un detalle antes de comenzar con las formas de conexión con el SMBD, en general, los lenguajes de programación que interactúan con un sistema gestor de bases de datos manejan dos tipos de objetos, los que regresan datos y los que no.

Los que regresan datos son llamados también conjuntos de registros y algunos tienen funcionalidades propias para poder obtener los datos contenidos en ellos, estos generalmente corresponden a las instrucciones SELECT de SQL.

Los que no regresan datos sólo regresan un status de la consulta SQL ejecutada que indica si se realizó o no de manera satisfactoria, generalmente corresponden a las instrucciones INSERT, UPDATE, DELETE, CREATE, etc.

Es por tal motivo que en el presente trabajo para cada lenguaje de programación Web se realiza sólo un ejemplo de una consulta que regresa datos y otro que no, debido a que las demás instrucciones SQL que no regresan datos son manipuladas de igual forma todas. Los ejemplos del capítulo se encuentran disponibles en D:\CapituloIV.

Conectar PHP con MySQL

Requerimientos y forma de conexión

Afortunadamente, PHP no tiene ningún requerimiento especial (salvo obviamente el SMBD instalado y funcional) para realizar conexiones con los SMBD MySQL y PostgreSQL. Haciendo uso de algunas funciones de PHP podemos conectarnos con la BD que queramos en el servidor que le indiquemos.

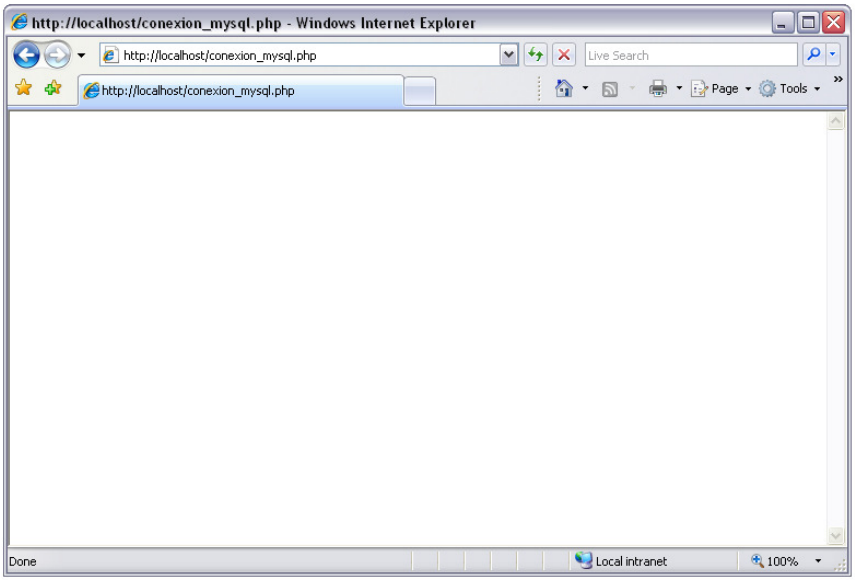
Una vez que tenemos creada la base de datos en nuestro servidor, el siguiente paso es conectarnos a la misma desde una página PHP. Para ello PHP nos proporciona una serie de instrucciones para acceder a bases de datos MySQL. El siguiente es un ejemplo de un archivo de configuración para acceder a MySQL. De las líneas 2 a la 5 se definen los parámetros de la conexión, el primero de ellos es la dirección o nombre del servidor en el cual se aloja la base de datos. El segundo es el nombre de usuario con el cual tenemos acceso a la base de datos. El tercero es la contraseña de acceso a la base de datos y el cuarto es el nombre de la base de datos con la que deseamos trabajar. En la línea 6 tenemos la función que realiza la conexión con la BD la cual recibe como parámetros la dirección del host del SMBD, el nombre de usuario y la contraseña, dicha función regresa un objeto el cual contiene la información del SMBD en caso de que la conexión haya sido satisfactoria, si no pudo realizar la conexión regresa un valor nulo, es por esto que en la línea 7 se verifica el valor que haya regresado la función, en dicha línea se verifica el valor del apuntador, si es nulo se despliega el mensaje de error de la línea 8 y se termina la ejecución del archivo con el **exit** de la línea 9, si la conexión se realizó de manera exitosa la ejecución continúa en la línea 11 en donde se selecciona la base de datos que le hayamos indicado anteriormente, la función recibe como parámetros el nombre de la base de datos y el apuntador de la conexión que regreso la función **mysql_connect**.

```
1       <?php
2       $dbhost = "localhost";   //host, normalmente localhost
3       $dbuser = "root";        //usuarios de la base de datos
4       $dbpass = "root";        //contraseña de la base de datos
5       $db = "nombre_bd";       //nombre de la base de datos
6       $conectar = mysql_connect($dbhost,$dbuser,$dbpass);
7       if(!$conectar){
8           echo"Error, no es posible establecer la conexión con la Base de datos";
9           exit;
```

```
10          }
11      mysql_select_db($db,$conectar);
12      ?>
```

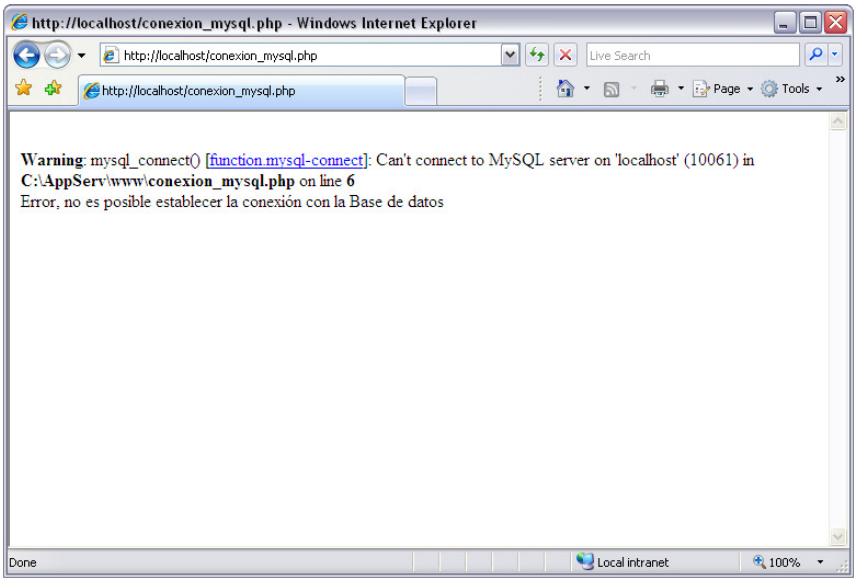
Archivo conexion_mysql.php, archivo de conexión con MySQL

Si la conexión se realiza de manera satisfactoria, al ejecutar el código del ejemplo anterior tendremos una imagen como la siguiente:



Ejecución archivo conexion_mysql.php

Debemos recordar que se trata sólo del archivo de conexión con el SMBD por lo tanto si no aparece un mensaje como el siguiente:



Ejecución archivo conexion_mysql.php, Mensaje de error

Significa que la conexión con el SMBD no se ha realizado. Lo mismo aplica con los archivos de conexión de los diferentes SMBD que se describen a continuación y los respectivos lenguajes con los que se esté trabajando.

Ejecución de Consultas

Una vez que nos hemos conectado con el SMBD, ya podemos realizar consultas a las tablas de la base de datos.

Para facilitar la programación debemos separar la función de conexión creada en el subtema anterior (**conexión_mysql.php)** en un archivo aparte, con la finalidad de simplemente incluir dicho archivo en lugar de volver a programar las líneas para la conexión cada vez que hagamos manipulación de datos. El código para realizar un **SELECT** de una tabla de MySQL es el siguiente:

```
1       <html>
2       <head>
3         <title>Ejemplo de PHP</title>
4       </head>
5       <body>
6       <h1>Ejemplo de uso de bases de datos con PHP y MySQL</h1>
7       <?php
8         include("conexion_mysql.php");
9
10        $consulta = mysql_query("SELECT campo1, campo2, campo3 FROM nombre_tabla");
11      ?>
12        <table border=1 cellspacing=1 cellpadding=1>
13          <tr>
14              <td><strong>Campo 1</strong></td>
15              <td><strong>Campo 2</strong></td>
16              <td><strong>Campo 3</strong></td>
17          </tr>
18      <?php
15        while($datos = mysql_fetch_array($consulta)) {
16              echo "<tr>";
17                      echo "<td>$datos[campo1]</td>";
18                      echo "<td>$datos[campo2']</td>";
19                      echo "<td>$datos[campo3]</td>";
20              echo "</tr>";
21        }
22        mysql_free_result($consulta);
23        mysql_close($conexion);
24      ?>
25      </table>
26      </body>
27      </html>
```

Archivo select_mysql.php, ejecucion de consulta SELECT con MySQL

En el ejemplo anterior tenemos como primer instrucción relevante la inclusión del archivo de conexión con la BD generado en el subtema anterior (línea 8), como sabemos, si no despliega ningún mensaje de error que hayamos colocado la conexión se ha realizado con éxito. Enseguida tenemos la instrucción **mysql_query()**, esta función recibe como parámetro una cadena de texto correspondiente a una consulta SQL y regresa el conjunto de datos (llamado record set) correspondientes a la instrucción colocada, en este caso el record set es asignado a una variable llamada **$consulta**, después de eso (líneas 12 a 17) se imprime el comienzo de una tabla en HTML (no olvidemos que podemos mezclar los códigos PHP y HTML a voluntad) la cual contiene los encabezados en negritas de los campos que obtendremos de la consulta ejecutada. Después es necesario recorrer el record set que nos regresó la función mysql_query, esto se hace colocando dentro de un ciclo la instrucción **mysql_fetch_array()** (línea 15) que recibe como parámetro la variable a la cual se asignó el record

set (**$consulta**) y regresa uno a uno el conjunto de filas de campos que se indicaron en la consulta SQL, a este conjunto de filas le llamamos **$datos**. Posteriormente, se imprimen (líneas 16 a 20), con la ayuda de la instrucción echo, los códigos HTML correspondientes a filas (<tr>) y celdas (
) de tablas y dentro de cada celda se imprime el contenido de uno de los tres campos que elegimos en la consulta SQL. Para hacer referencia a un campo se debe colocar el nombre de la variable que está haciendo el recorrido por el record set y entre corchetes y comillas sencillas el nombre del campo deseado, por ejemplo, **$datos[‘campo1’]**.

Lo último que queda por hacer es liberar el record set en caso de que esa misma instrucción no se ocupe más y finalmente cerrar la conexión con la BD (líneas 22 y 23 respectivamente).

A continuación veremos el código de ejemplo para realizar una instrucción **UPDATE** de una tabla MySQL:

```
1       <html>
2       <head>
3         <title>Ejemplo de PHP</title>
4       </head>
5       <body>
6       <h1>Ejemplo de uso de bases de datos con PHP y MySQL</h1>
7       <?php
8         include("conexion_mysql.php");
9
10        $consulta = mysql_query("UPDATE nombre_tabla SET campo2 = ‘valor2’, campo3 = ‘valor3’ WHERE
campo1 = ‘valor1’");
11      if(!$consulta){
12              echo "Error al ejecutar la consulta";
13              exit;
14      }
15        mysql_close($conexion);
16      ?>
17      </table>
18      </body>
19      </html>
```

Archivo update_mysql.php, ejecucion de consulta UPDATE con MySQL

En el ejemplo anterior nuevamente incluimos el archivo de conexión con la BD (línea 8). Enseguida tenemos la instrucción **mysql_query()**, que en este caso recibe como parámetro una cadena de texto correspondiente a una consulta de actualización SQL (línea 10), en este caso la función solo regresa un valor **true** en caso de una ejecución satisfactoria, por tal motivo comprobamos su valor (línea 11), en caso de error se manda el mensaje correspondiente y termina la ejecución si no existe error continúa la ejecución del script y se cierra la conexión con la BD (línea 15). Lo mismo aplica cuando se trata de una consulta SQL de eliminación de campos: **DELETE** o inserción de registros: **INSERT**.

Conectar PHP con PostgreSQL

Requerimientos y forma de conexión

Al igual que con MySQL, haciendo uso de algunas funciones de PHP podemos conectarnos con la BD que queramos en el servidor que le indiquemos sin mayores requerimientos.

Una vez que tenemos creada la base de datos en nuestro servidor, el siguiente paso es conectarnos a la misma desde una página PHP. Para ello PHP nos proporciona una serie de instrucciones para acceder a bases de datos PostgreSQL.

El siguiente es un ejemplo de un archivo de configuración para conectarnos con PostgreSQL. En la línea 2 se ejecuta la función para realizar la conexión con el SMBD, esta función debe recibir los parámetros como cadena dentro la función misma, en algunas versiones de PHP si se definen estos valores en variables como se hizo con la conexión a MySQL no se realiza la conexión por tal motivo es mejor manejar la conexión de esta forma, el primer parámetro de la función es nuevamente la dirección o nombre del servidor en el cual se aloja la base de datos. El segundo es el nombre de la BD con la cual nos queremos conectar. El tercero es el nombre de usuario con el cual tenemos acceso a la base de datos y el cuarto es la contraseña de acceso a la base de datos.

Al igual que con MySQL, la función regresa un objeto el cual contiene la información del SMBD en caso de que la conexión haya sido satisfactoria, si no pudo realizar la conexión regresa un valor nulo. Para PostgreSQL existe la función **pg_ErrorMessage** que nos sirve para capturar cualquier error que haya ocurrido con el SMBD, es por eso que en la línea 3 se manda llamar dicha función para validar la conexión, en caso de error se manda el mensaje correspondiente en la línea 4 y se termina la ejecución del archivo con el **exit** de la línea 5.

```
1       <?php
2       $conexion = pg_connect("host=localhost dbname=nombre_bd user=root password=root");
3       if(pg_ErrorMessage($conexion)){
4            echo "Error, no es posible establecer la conexión con la Base de datos";
5            exit;
6          }
7       ?>
```

Archivo conexion_postgres.php, archivo de conexión con POSTGRESQL

Ejecución de Consultas

Una vez que nos hemos conectado con el SMBD, ya podemos realizar consultas a las tablas de la base de datos.

Para facilitar la programación debemos separar la función de conexión creada en el subtema anterior (**conexión_postgres.php)** en un archivo aparte, con la finalidad de simplemente incluir dicho archivo en lugar de volver a programar las líneas para la conexión cada vez que hagamos manipulación de datos. El código para realizar un **SELECT** de una tabla de Postgres es el siguiente:

```
1       <html>
2       <head>
3         <title>Ejemplo de PHP</title>
4       </head>
5       <body>
6       <h1>Ejemplo de uso de bases de datos con PHP y MySQL</h1>
7       <?php
8         include("conexion_postgres.php");
9
10        $consulta = pg_exec("SELECT campo1, campo2, campo3 FROM nombre_tabla");
11      ?>
12        <table border=1 cellspacing=1 cellpadding=1>
13          <tr>
14               <td><strong>Campo 1</strong></td>
15               <td><strong>Campo 2</strong></td>
16               <td><strong>Campo 3</strong></td>
17          </tr>
18      <?php
15        for($cont=0; $cont < pg_numrows($consulta); $cont++) {
16               echo "<tr>";
17                        echo "<td>".pg_result($consulta, $cont, 0)."</td>";
18                        echo "<td>".pg_result($consulta, $cont, 1)."</td>";
19                        echo "<td>".pg_result($consulta, $cont, 2)."</td>";
20               echo "</tr>";
```

```
21          }
22          pg_FreeResult($resultado);
23        ?>
24        </table>
25        </body>
26        </html>
```

Archivo select_postgres.php, ejecucion de consulta SELECT con POSTGRESQL

En el ejemplo anterior tenemos como primer instrucción relevante la inclusión del archivo de conexión con la BD generado en el subtema anterior (línea 8), como sabemos, si no despliega ningún mensaje de error que hayamos colocado la conexión se ha realizado con éxito. Enseguida tenemos la instrucción **pg_exec()**, esta función recibe como parámetro una cadena de texto correspondiente a una consulta SQL y regresa el conjunto de datos (llamado record set) correspondientes a la instrucción colocada, en este caso el record set es asignado a una variable llamada **$consulta**, después de eso (líneas 12 a 17) se imprime el comienzo de una tabla en HTML la cual contiene los encabezados en negritas de los campos que obtendremos de la consulta ejecutada.

En Postgres no existe una instrucción para recorrer el record set línea a línea como sucede con MySQL, es por tal motivo que primero se debe obtener el número de líneas que regresó la consulta SQL, esto se logra con la instrucción **pg_numrows()**, que recibe como parámetro la variable que contiene al record set, después se inicia un ciclo que va desde cero hasta el número de líneas que regresó la función pg_numrows (línea 15),el objetivo de esto es el siguiente, la forma en la cual se obtienen los datos del record set es con la instrucción **pg_result()**, pero esta instrucción recibe tres parámetros, el primero de ellos es la variable a la cual se asignó el record set, el segundo es el número de fila del record set que se quiere obtener y el tercer parámetro es el número de campo de la consulta que se quiere leer, es por ello que si observamos el código entre las líneas 17 y 19 podremos observar que estamos leyendo fila por fila y campo por campo los valores del record set. La variable **$cont** sirve como índice contador del número de fila que estamos obteniendo y las constantes 0, 1 y 2 son los números de campos de la consulta SQL que estamos leyendo y corresponden a campo1, campo2 y campo3 respectivamente.

Lo último que queda por hacer es liberar el record set en caso de que esa misma instrucción no se ocupe más (línea 22).

A continuación veremos el código de ejemplo para realizar una instrucción **UPDATE** de una tabla Postgres:

```
1         <html>
2         <head>
3           <title>Ejemplo de PHP</title>
4         </head>
5         <body>
6         <h1>Ejemplo de uso de bases de datos con PHP y MySQL</h1>
7         <?php
8           include("conexion_postgres.php");
9
10          $consulta = pg_exec("UPDATE nombre_tabla SET campo2 = 'valor2', campo3 = 'valor3' WHERE campo1
= 'valor1'");
11        if(!$consulta){
12                  echo "Error al ejecutar la consulta";
13                  exit;
14        }
15        pg_FreeResult ($consulta);
16        ?>
17        </table>
18        </body>
19        </html>
```

Archivo update_postgres.php, ejecucion de consulta UPDATE con POSTGRESQL

En el ejemplo anterior nuevamente incluimos el archivo de conexión con la BD (línea 8). Enseguida tenemos la instrucción **pg_exec()**, que en este caso recibe como parámetro una cadena de texto correspondiente a una consulta de actualización SQL (línea 10), en este caso la función solo regresa un valor **true** en caso de una ejecución satisfactoria, por tal motivo comprobamos su valor (línea 11), en caso de error se manda el mensaje correspondiente y termina la ejecución si no existe error continúa la ejecución del script y se cierra la conexión con la BD (línea 15). Lo mismo aplica cuando se trata de una consulta SQL de eliminación de campos: **DELETE** o inserción de registros: **INSERT**.

Conectar JSP / Servlets con MySQL

Requerimientos y forma de conexión

Tanto el lenguaje JSP como los Servlets requieren la integración de un JDBC para realizar las conexiones con bases de datos, también requiere de una versión de JSDK, dicha versión ya debe estar instalada en nuestro equipo servidor.

JDBC es una API pura de Java que se usa para ejecutar comandos de SQL. Suministra una serie de clases e interfaces que permiten al desarrollador de Web escribir aplicaciones que gestionen Bases de Datos. El JDBC que se utiliza en el presente ejemplo está disponible en D:\CapituloIV\JSP Servlets MySQL\, el JDBC también es conocido como conector.

El primer paso entonces será colocar el archivo del conector en la carpeta **$_RUTA_DE_INSTALACION\Apache Software Foundation\Tomcat 6.0\common\lib** algunas versiones del Tomcat no tienen la carpeta common por lo tanto el archivo deberá de ser copiado en la siguiente ruta **$_RUTA_DE_INSTALACION\Apache Software Foundation\Tomcat 6.0\lib**.

El segundo paso es registrar las variables de entorno de Java y de Tomcat, esto se realiza de la siguiente manera, se debe hacer click derecho sobre el ícono de MiPC y seleccionar la opción de propiedades. En la ventana que aparezca hacemos click sobre la pestaña **Avanzado**:

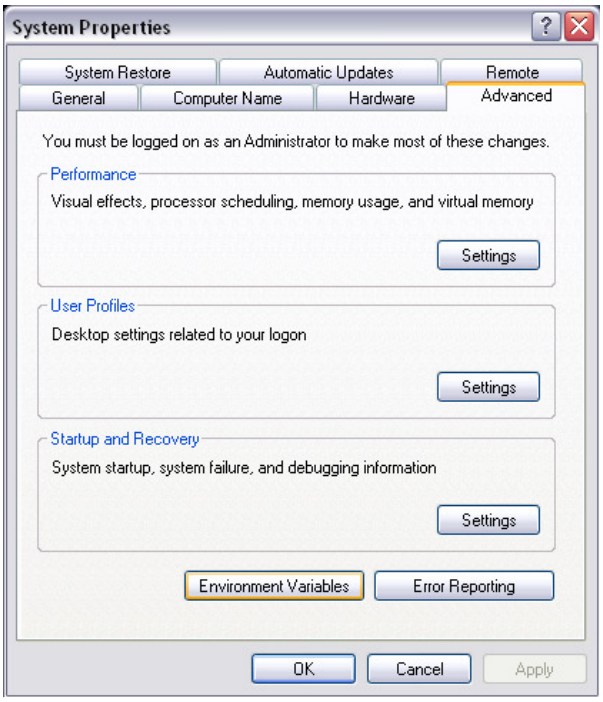


Figura 4.1 Opciones Avanzadas, Propiedades de MiPC

Para posteriormente hacer Click sobre el botón de **Variables de Entorno** para que aparezca otra ventana dividida en 2, si queremos que las variables queden registradas solo para la sesión de Windows actual debemos hacer Click en el primer botón (de arriba a abajo) de Agregar, si queremos registrar la variable de entorno para todos los usuarios del sistema hacemos Click en el segundo botón de Agregar.

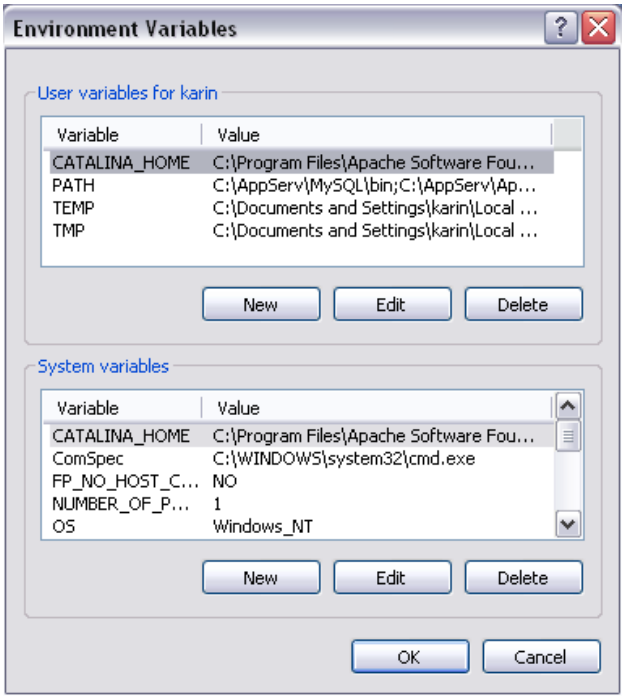


Figura 4.2 Variables de Entorno, Propiedades de MiPC

Sea cual sea la opción aparece otra ventana más en la cual se indica, primero el nombre de la variable de entorno y segundo, su valor o ubicación.

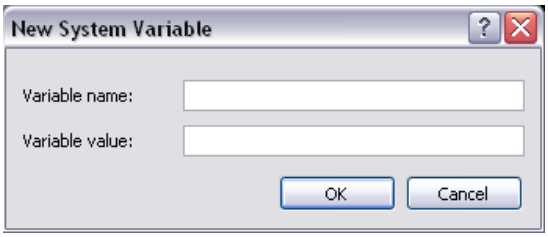


Figura 4.3 Agregar Variables de Entorno, Propiedades de MiPC

Las variables a registrar en este caso son dos:

- **JAVA_HOME** con el valor de **$_RUTA_DE_INSTALACION\Java\$_version_del_JSDK**
- **CATALINA_HOME** con el valor de **$_RUTA_DE_INSTALACION\Apache Software Foundation\Tomcat 6.0**

Una vez realizados estos pasos es momento de generar el archivo de conexión con la BD. En las líneas 3, 4, 6 y 7 definimos las variables necesarias para realizar la conexión, la primera de ellas es el objeto con el que haremos referencia a la conexión con la BD, la segunda es "la interfaz" (por decirlo de alguna manera) que enviará o recibirá los datos del SMBD a través del objeto de la conexión, la tercer variable es una cadena que identifica al JDBC que estamos usando esta cadena puede ser **org.gjt.mm.mysql.Driver** o **com.myscql.jdbc.Driver**, la diferencia radica en el driver que se ocupe para la conexión, el archivo que copiamos anteriormente en la carpeta **lib** es un archivo comprimido que contiene a ambos drivers por eso funciona perfectamente en cualquiera de los dos casos, esto se debe especificar cuando ese archivo es descomprimido y su contenido colocado por separado. Finalmente, la cuarta variable que definimos es la URL de la conexión a la BD, en ella especificamos el driver utilizado en este caso es un JDBC, el SMBD en este caso es MySQL, la dirección o nombre del servidor en el cual se aloja la base de datos, el nombre de la base de datos.

```
1       <%@page import = "java.sql.*"%>
2       <%
3               Connection conexion  = null;
4               Statement  statement = null;
5
6               String  driver = "org.gjt.mm.mysql.Driver";
7               String  dbconexion = "jdbc:mysql://localhost/nombre_bd";
8
9                       try  {
10                      Class.forName(driver);
11                      conexion = DriverManager.getConnection(dbconexion,"usuario","contrasena");
12                      statement = conexion.createStatement();
13                      }catch (Exception error)   {
14                              out.print("Error, no es posible establecer la conexión con la Base de datos");
15                      }
16      %>
```

Archivo conexion_mysql.jsp, archivo de conexión con MySQL

La conexión se realiza dentro de un **try** y un **catch** para capturar de manera adecuada el error o excepción arrojados por Java (líneas 9 a 15), lo primero que se hace es instanciar la clase del JDBC (línea 10) ya que como mencionamos es una API de Java que se usa para ejecutar comandos de SQL, después se crea la conexión a la base de datos con la función **DriverManager.getConnection** la cual recibe como parámetros la URL de la conexión que hayamos definido, el nombre de usuario de la BD y la contraseña, ahora debemos asignar el objeto de la conexión a la interfaz que habíamos

mencionado anteriormente (línea 12), con esta variable es con la que se ejecutarán todas las consultas en la BD.

Con a la captura de excepciones, si ocurre algún error en cualquiera de estos pasos la ejecución se interrumpirá y se mandará el mensaje de error.

Ejecución de Consultas

Con JSP no es posible generar un archivo que contenga los datos de la conexión con el SMBD y simplemente incluirlo como con PHP, esto se debe a que las consultas en JSP requieren el apuntador del objeto que contiene a la conexión es por esto que en cada página que se ejecuten consultas se tendrá que codificar la parte correspondiente a conexión con la BD.

El código para realizar un **SELECT** de una tabla de MySQL es el siguiente:

```
1       <%@page import = "java.sql.*"%>
2       <%
3               Connection conexion  = null;
4               Statement  statement = null;
5
6               String  driver = "org.gjt.mm.mysql.Driver";
7               String  dbconexion = "jdbc:mysql://localhost/nombre_bd";
8
9               try  {
10                      Class.forName(driver);
11                      conexion = DriverManager.getConnection(dbconexion,"usuario","contrasena");
12                      statement = conexion.createStatement();
13                      ResultSet  record = null;
14                      record     =     statement.executeQuery("SELECT     campo1,campo2,campo3     FROM
nombre_tabla");
15                      out.print("<table border=1 cellspacing=1 cellpadding=1>");
16                      out.print("<tr>");
17                      out.print("<td><strong>Campo 1</strong></td>");
18                      out.print("<td><strong>Campo 2</strong></td>");
19                      out.print("<td><strong>Campo 3</strong></td>");
20                      out.print("</tr>");
21
22                      while (record.next()){
23                              out.print("<tr>");
24                              out.print("<td>" + record.getString(1) + "</td>");
25                              out.print("<td>" + record.getString(2) + "</td>");
26                              out.print("<td>" + record.getString(3) + "</td>");
27                              out.print("<tr>");
28                      }
29                      conexion.close();
30              }catch (Exception error)   {
31                      out.print(error);
32              }
33      %>
```

Archivo select_mysql.jsp, ejecucion de consulta SELECT con MySQL

Si observamos detenidamente, el código de las líneas 1 a la 12 es el correspondiente al archivo de conexión a la BD descrito en el subtema anterior. Posteriormente se declara una variable de tipo ResultSet (línea 13) que nos sirve para almacenar el conjunto de registros que regrese la instrucción SQL ejecutada, luego de eso, se ejecuta la consulta SQL con ayuda de la instrucción **executeQuery()** que recibe como parámetro una cadena de texto correspondiente a la consulta, si observamos bien, la instrucción se ejecuta a partir del objeto que describimos como la interfaz entre nuestra aplicación y el SMBD por lo tanto aparece como **statement.executeQuery("consulta SQL")**. Tal función regresa el conjunto de registros a nuestra variable record que es la que va a contener los datos regresados.

De las líneas 15 a la 20, al igual que en los ejemplos anteriores, imprimimos los encabezados de un tabla HTML con los títulos de los campos, una vez impresos, se recorre fila a fila el conjunto de registros que ahora están en nuestra variable **record** con la ayuda de la función **next()** (línea 22), y como queremos que se impriman todos los registros colocamos dicha función dentro de un ciclo **while**, en este ciclo imprimimos fila por fila y uno a uno los campos que obtuvimos de la consulta SQL. Finalizamos como siempre cerrando la conexión con la BD (línea 29).

A continuación veremos el código de ejemplo para realizar una instrucción **UPDATE** de una tabla MySQL:

```
1       <%@page import = "java.sql.*"%>
2       <%
3               Connection conexion  = null;
4               Statement  statement = null;
5
6               String  driver = "org.gjt.mm.mysql.Driver";
7               String  dbconexion = "jdbc:mysql://localhost/nombre_bd";
8
9               try  {
10                      Class.forName(driver);
11                      conexion = DriverManager.getConnection(dbconexion,"usuario","contrasena");
12                      statement = conexion.createStatement();
13
14                      statement.executeUpdate("UPDATE nombre_tabla SET campo2 = 'valor2', campo3 =
'valor3' WHERE campo1 = 'valor1'");
15
16                      conexion.close();
17              }catch (Exception error)   {
18                      out.print(error);
19              }
20      %>
```

Archivo update_mysql.jsp, ejecucion de consulta UPDATE con MySQL

En el ejemplo anterior nuevamente colocamos el código correspondiente a la conexión con la BD (líneas 1 a 12). Enseguida se ejecuta la consulta SQL con ayuda de la instrucción **executeUpdate()** que recibe como parámetro una cadena de texto correspondiente a la consulta, si observamos bien, la instrucción se ejecuta a partir del objeto que describimos como la interfaz entre nuestra aplicación y el SMBD por lo tanto aparece como **statement.executeUpdate("consulta SQL")**. Lo mismo aplica cuando se trata de una consulta SQL de eliminación de campos: **DELETE** o inserción de registros: **INSERT**.

Es importante destacar la diferencia entre los comandos **executeQuery** y **executeUpdate**, el primero de ellos se utiliza para todas aquellas consultas que regresan un conjunto de información como resultado después de una ejecución de consulta, dicho resultado se debe asignar a una variable de tipo ResultSet, la segunda se utiliza solo para consultas que realizan actualizaciones a la BD y no regresan resultado alguno, en caso de fallo ambas pasarían a ejecutar el código programado en el **catch**.

Manejo de Errores del SMBD

Existen ocasiones en las que es necesario ejecutar más de una consulta que agregue o modifique los contenidos de las tablas de nuestra base de datos, sin embargo, la naturaleza misma de las conexiones de red o eléctricas propician la generación de errores que pueden intervenir con las actualizaciones que estemos realizando.

Por ejemplo, supongamos que tenemos un formulario el cual contiene información personal de algunos contactos, tal información puede ser nombre, edad, sexo, domicilio, teléfono, ciudad, estado

y que por alguna razón la información anterior se almacena en dos tablas dentro de una base de datos. Para almacenar contenidos en dos tablas obviamente se deben ejecutar dos consultas INSERT en SQL.

Entonces primero almacenamos los datos correspondientes a la persona en una tabla y en la otra tabla almacenamos los datos correspondientes a su domicilio, entonces dichas consultas SQL quedarían así:

- INSERT INTO tabla1 (clavep, nombre, edad, sexo) VALUES ('valor_clavep', 'valor_nombre', 'valor_edad', 'valor_sexo')
- INSERT INTO tabla2 (clavep, domicilio, telefono, ciudad, estado) VALUES ('valor_clavep', 'valor_domicilio', 'valor_telefono', 'valor_ciudad', 'valor_estado')

Obviamente el valor del campo **clavep** debe ser único en estas y en todas las demás tablas donde hagamos referencia a esta persona. Ahora bien, en nuestro script tendremos las dos consultas anteriores una seguida de la otra, sin embargo, no se realizan al mismo tiempo, el SMBD primero recibe una consulta, la ejecuta, manda el mensaje de error o de éxito y continúa con la siguiente consulta, supongamos que por alguna razón la primer consulta se envió y ejecutó con éxito, y la segunda consulta no se mando correctamente por lo tanto su ejecución no fue correcta, entonces tendremos la mitad de los datos de una persona en la **tabla1** pero nada en la **tabla2**.

En los ejemplos de los temas anteriores se analizó la manera de identificar cuando ocurría un error con la consulta SQL, sin embargo, solo se hace una notificación al usuario de que algo falló mas no se hizo ningún proceso de recuperación de errores. Afortunadamente, los SMBD utilizan archivos especiales llamados bitácoras los cuales nos ayudan a identificar la última transacción correcta ejecutada para que, en caso de fallo, deshagamos esa última transacción.

Esto funciona de la siguiente manera, regresando al ejemplo anterior, antes de ejecutar la primer consulta iniciamos una transacción, la primer consulta se ejecutó correctamente por lo tanto se almacena la actualización que se realizó en la bitácora de la BD, la segunda consulta no se ejecutó de manera correcta por lo tanto deshacemos las operaciones que se hayan hecho hasta el momento en el que iniciamos la transacción el cual fue antes de ejecutar la primer consulta, de esta manera no tenemos el registro sólo en una tabla con lo que se evitan inconsistencias en la BD.

Esto se logra haciendo uso de tres instrucciones SQL: **BEGIN**, **COMMIT** Y **ROLLBACK**

La instrucción BEGIN indica el inicio de una transacción (bloque de consultas SQL), justo antes de ejecutar más de una consulta que modifique los registros de la BD se debe ejecutar esta instrucción.

La instrucción COMMIT indica el término de un bloque de consultas, cuando se ejecuta esta instrucción, los registros en la bitácora son eliminados con lo que se indica al SMBD que la transacción se ejecutó de manera correcta.

La instrucción ROLLBACK deshace todas las consultas que se hayan ejecutado hasta antes de haber ejecutado la instrucción BEGIN. Cabe señalar, que si la instrucción COMMIT no se ejecuta después de cierto tiempo el SMBD lo interpreta como una desconexión por parte del cliente y la transacción se tomará como no terminada, por lo tanto, ejecutará la instrucción ROLLBACK automáticamente.

Las transacciones son soportadas por cualquier tipo de tabla en Postgres pero en MySQL sólo son soportadas por tablas de tipo InnoBD, por lo tanto, al crear las tablas de su BD es importante fijar el tipo de tabla como InnoDB (recordemos que al momento de instalar nuestro servidor AppServ se seleccionó la casilla de verificación en la instalación de MySQL para soportar tablas tipo InnoDB).

Manejo de errores con PHP

Tomando el ejemplo descrito anteriormente en el cual se realizan dos consultas a dos diferentes tablas de una BD se ha desarrollado el código correspondiente en PHP.

```
1       <html>
2       <head>
3         <title>Ejemplo de PHP</title>
4       </head>
5       <body>
6       <h1>Ejemplo de uso de bases de datos con PHP y MySQL</h1>
7       <?php
8         include("conexion_mysql.php");
9         mysql_query("BEGIN");
10        $consulta1 = mysql_query("INSERT INTO tabla1 (clavep, nombre, edad, sexo) VALUES ('valor_clavep', 'valor_nombre', 'valor_edad', 'valor_sexo')");
11        $consulta2 = mysql_query("INSERT INTO tabla2 (clavep, domicilio, telefono, ciudad, estado) VALUES ('valor_clavep', 'valor_domicilio', 'valor_telefono', 'valor_ciudad', 'valor_estado')");
12      if(!$consulta1 || !$consulta2){
12              mysql_query("ROLLBACK");
13              exit;
14      }
15      mysql_query("COMMIT");
16      mysql_close($conexion);
17      ?>
18      </table>
18      </body>
19      </html>
```

Archivo insert_mysqlt.php, ejecucion consulta INSERT MySQL con transacciones

En el ejemplo anterior podemos observar la ejecución de las tres instrucciones descritas anteriormente, como se mencionó, la instrucción BEGIN debe estar antes de la transacción o conjunto de consultas, posteriormente se ejecutan ambas consultas y se verifica el status que regresa la instrucción mysql_query (línea 12) para cada una de las consultas, si alguna de ellas falló se ejecuta el ROLLBACK para deshacer las modificaciones realizadas a la BD hasta donde se encuentra la instrucción BEGIN, si no ocurrió ningún error, se continúa la ejecución del script hasta la instrucción COMMIT (línea 15) que indica al SMBD que la transacción fue exitosa.

El ejemplo anterior también funciona para Postgres, sólo que en lugar de incluir el archivo de conexión para MySQL (línea 8) se debe incluir el archivo de conexión para Postgres, la instrucción **mysq_query** debe ser reemplazada por **pg_exec** y la instrucción **mysql_close** deberá ser reemplazada por **pg_FreeResult**.

Manejo de errores con JSP y Servlets

De nuevo tomamos el ejemplo descrito anteriormente en el cual se realizan dos consultas a dos diferentes tablas de una BD, a continuación tenemos el código correspondiente en JSP.

```
1       <%@page import = "java.sql.*"%>
2       <%
3               Connection conexion  = null;
4               Statement  statement = null;
5
6               String  driver = "org.gjt.mm.mysql.Driver";
7               String  dbconexion = "jdbc:mysql://localhost/nombre_bd";
8
9               try  {
10                      Class.forName(driver);
11                      conexion = DriverManager.getConnection(dbconexion,"usuario","contrasena");
```

```
12                          statement = conexion.createStatement();
13                          statement.executeUpdate("BEGIN");
14                          statement.executeUpdate("INSERT INTO tabla1 (clavep, nombre, edad, sexo) VALUES
("+valor_clavep+", "+valor_nombre+", "+valor_edad+", "+valor_sexo+")");
15                          statement.executeUpdate("INSERT INTO tabla2 (clavep, domicilio, telefono, ciudad,
estado)    VALUES    ("+valor_clavep+",    "+valor_domicilio+",    "+valor_telefono+",    "+valor_ciudad+",
"+valor_estado+")");
16                          conexion.close();
17                  }catch (Exception error)  {
18                          statement.executeUpdate("ROLLBACK");
19                          out.print(error);
19                  }
20                  statement.executeUpdate("COMMIT");
21          %>
```

Archivo insert_mysqlt.jsp, ejecucion consulta INSERT MySQL con transacciones

En el ejemplo anterior podemos observar la ejecución de las tres instrucciones descritas anteriormente, como se mencionó, la instrucción BEGIN debe estar antes de la transacción o conjunto de consultas (línea 13), como son consultas que no regresan ningún tipo de registros no debemos hacer uso de las variables de tipo **ResultSet**, posteriormente se ejecutan las consultas, debemos recordar que si ocurre un error en cualquier parte del código dentro de una sección **try** inmediatamente se ejecutará el código de la sección **catch(Exception error)**, por lo tanto si ocurre un error en alguna de las consultas se ejecutará el ROLLBACK y deshará las operaciones realizadas. Caso contrario se llega hasta la ejecución del COMMIT para dar por terminada la transacción.
También se puede dar el caso de que el error ocurra durante la conexión con el SMBD, de ser así sabemos que el código de la sección **catch** se ejecutará y como consecuencia también se ejecutará la instrucción ROLLBACK, pero al no haberse ejecutado nunca la instrucción BEGIN no existe bitácora alguna para analizar y deshacer los cambios por lo tanto no ocurre nada.

Los ejemplos y archivos de conexión vistos anteriormente se encuentran disponibles en D:\CapituloIV\.

CAPITULO V.- SEGURIDAD EN EL SERVIDOR

SSL protocolo de certificación para envío seguro de datos

Combina elementos de un protocolo de transmisión de Internet con técnicas de encriptación y autenticación. El resultado es un entorno seguro de comunicación de transacciones en una red pública. SSL implementa encriptación RSA para garantizar comunicaciones privadas y autentificadas por Internet.
Este protocolo permite confidencialidad y autenticación en Internet. SSL opera como una capa adicional entre Internet y las aplicaciones, esto permite que el protocolo sea independiente de la aplicación, siendo posible utilizar FTP, Telnet y otras aplicaciones además de HTTP.
El protocolo SSL ofrece tres propiedades fundamentales:

- Criptografía simétrica para garantizar la intimidad de la conexión. La encriptación se activa tras un intercambio inicial de protocolos para definir un código con clave secreta.
- Criptografía asimétrica para autentificar la conexión.
- Conexiones fiables en las que una función hash segura, como MD5, verifican la integridad del mensaje.

Para establecer una comunicación segura utilizando SSL se tienen que realizar una serie de pasos. Primero se debe hacer una solicitud de seguridad. Después de haberla hecho, se deben establecer los parámetros que se utilizarán para SSL. Esta parte se conoce como SSL Handshake. Una vez se haya establecido una comunicación segura, se deben hacer verificaciones periódicas para garantizar que la comunicación sigue siendo segura a medida que se transmiten datos. Luego que la transacción ha sido completada, se termina SSL.

Solicitud de SSL

Antes de que se establezca SSL, se debe hacer una solicitud. Típicamente esto implica un cliente haciendo una solicitud de una URL a un servidor que soporte SSL. SSL acepta solicitudes por un puerto diferente al utilizado normalmente para ese servicio.
Una vez se ha hecho la solicitud, el cliente y el servidor empiezan a negociar la conexión SSL, es decir, hacen el SSL Handshake.

SSL Handshake

Durante el handshake se cumplen varios propósitos. Se hace autenticación del servidor y opcionalmente del cliente, se determina que algoritmos de criptografía serán utilizados y se genera una llave secreta para ser utilizada durante el intercambio de mensajes subsiguientes durante la comunicación SSL.

Los pasos que se siguen son los siguientes:

1. Client Hello: El saludo de cliente tiene por objetivo informar al servidor que algoritmos de criptografía puede utilizar y solicita una verificación de la identidad del servidor. El cliente envía el conjunto de algoritmos de criptografía y compresión que soporta y un número aleatorio. El propósito del número aleatorio es para que en caso de que el servidor no posea un certificado para comprobar su identidad, aún se pueda establecer una comunicación segura utilizando un conjunto distinto de algoritmos. Dentro de los protocolos de criptografía hay un protocolo de intercambio de llave que define como cliente y servidor van a intercambiar la información, los algoritmos de llave secreta que definen que métodos pueden utilizar y un algoritmo de hash de una sola vía. Hasta ahora no se ha intercambiado información secreta, solo una lista de opciones.
2. Server Hello: El servidor responde enviando su identificador digital el cual incluye su llave pública, el conjunto de algoritmos criptográficos y de compresión y otro número aleatorio. La decisión de que algoritmos serán utilizados está basada en el más fuerte que tanto cliente como

servidor soporten. En algunas situaciones el servidor también puede solicitar al cliente que se identifique solicitando un identificador digital.

3. Aprobación del Cliente: El cliente verifica la validez del identificador digital o certificado enviado por el servidor. Esto se lleva a cabo descifrando el certificado utilizando la llave pública del emisor y determinando si este proviene de una entidad certificadora de confianza. Después se hace una serie de verificaciones sobre el certificado, tales como fecha, URL del servidor, etc. Una vez se ha verificado la autenticidad de la identidad del servidor. El cliente genera una llave aleatoria y la encripta utilizando la llave pública del servidor y el algoritmo criptográfico y de compresión seleccionado anteriormente. Esta llave se le envía al servidor y en caso de que el handshake tenga éxito será utilizada en el envío de futuros mensajes durante la sesión.
4. Verificación: En este punto ambas partes conocen la llave secreta, el cliente porque la generó y el servidor porque le fue enviada utilizando su llave pública, siendo la única forma posible de descifrarla utilizando la llave privada del servidor. Se hace una última verificación para comprobar si la información transmitida hasta el momento no ha sido alterada. Ambas partes se envían una copia de las anteriores transacciones encriptadas con la llave secreta. Si ambas partes confirman la validez de las transacciones, el handshake se completa, de otra forma se reinicia el proceso.

Ahora ambas partes están listas para intercambiar información de manera segura utilizando la llave secreta acordada y los algoritmos criptográficos y de compresión. El handshake se realiza solo una vez y se utiliza una llave secreta por sesión.

Intercambio de datos

Ahora que se ha establecido un canal de transmisión seguro SSL, es posible el intercambio de datos. Cuando el servidor o el cliente desean enviar un mensaje al otro, se genera un digest (utilizando un algoritmo de hash de una vía acordado durante el handshake), encriptan el mensaje y el digest y se envían, cada mensaje es verificado utilizando el digest.

Terminación de una sesión SSL

Cuando el cliente deja una sesión SSL, generalmente la aplicación presenta un mensaje advirtiendo que la comunicación no es segura y confirma que el cliente efectivamente desea abandonar la sesión SSL

Instalar SSL en un servidor Windows con AppServ

Añadir soporte SSL a una web puede parecer complicado, pero nada más lejos de la realidad. Es fácil implementar una solución de conectividad segura a Apache bajo Linux. Sin embargo es complicado hacer eso mismo bajo Windows, y por norma, quienes optan por esta plataforma, suelen decantarse por IIS para extender certificados SSL por su facilidad.
A continuación vamos a construir un servidor web con SSL bajo Apache Windows, utilizando para ello herramientas abiertas (open source).

Lo primero que se necesita es OpenSSL para Windows. Descargamos entonces la versión 0.9.8e para Windows de la siguiente página: http://www.openssl.org/related/binaries.html o tambien se encuentra disponible en D:\CapituloV\SSL.

Instalado éste, accedemos a una ventana MS-DOS, y desde la carpeta Openssl/bin, realizamos los siguientes pasos.

1. Generamos la pareja de claves para la CA con passphrase "test": openssl genrsa -out CA_TEST.key -passout pass:test 2048

2. Generamos un certificado autofirmado con la clave pública de la CA, con la passphrase "test": **openssl req -new -x509 -out CA_TEST.crt -subj /DC=TEST/L=Mexico/O=TEST/CN=CA_TEST -days 365 -sha1 -config openssl.cnf -extensions v3_ca**
3. Generando una pareja de claves para el servidor: **openssl genrsa -out TEST.key -passout pass:test 1024**
4. Generando la petición de certificado para el servidor: **openssl req -new -out TEST.csr -key TEST.key -sha1 -subj /L=Mexico/O=TEST/OU=Test/CN=TEST -config openssl.cnf**
5. Generando el certificado de la clave pública del servidor firmado por la CA: **openssl x509 -req -in TEST.csr -CAkey CA_TEST.key -sha1 -days 365 -out TEST.crt -signkey TEST.key -extfile openssl.cnf -extensions v3_ca**

Aquí se está utilizando una supuesta empresa llamada TEST con contraseña "test". Es evidente que el usuario deberá sustituir estos parámetros por la empresa de su elección con su contraseña preferida.

Ahora, mediante un editor procedemos a buscar y editar el archivo httpd.conf que se encuentra en C:\appserv\apache\conf

Hay que comentar la línea de # Port 80 en el archivo httpd.conf y añadir las siguientes dos líneas:

```
Listen 80
Listen 443
```

Los puertos por los que queremos que escuche, siendo el 443 el correspondiente a SSL. Reiniciamos el servidor Apache, mediante el comando Restart del grupo de programas Appserv. Se realiza una prueba de conexión a ambos puertos, viéndose una página de prueba satisfactoria. Ejemplo: http://localhost:443

Creamos una carpeta llamada "modssl" dentro de la carpeta de Apache. Cuando se instala Openssl, al descomprimir el binario de esta distribución, se copian automáticamente los archivos ssleay32.dll y libeay32.dll a la carpeta System32 de nuestro Windows, ya sea C:\windows\system32\ para Windows 9x/ME/XP o C\:winnt\system32\ para Windows NT y Windows 2000. Estos archivos son imprescindibles para el buen funcionamiento del sistema.

Copiamos entonces los archivos TEST.crt y TEST.key, generados por OpenSSL, a la carpeta modssl de Apache. Editamos de nuevo el archivo httpd.conf y añadimos la siguiente línea después de las otras directivas:

```
LoadModule:
LoadModule ssl_module modules/mod_ssl.so
```

Hay que comprobar que el módulo se encuentra en la carpeta propiamente dicha. En AppServ este módulo no se incluye por defecto; así que lo vamos a tener que localizar por Internet. Para ello descargamos una versión no oficial de Apache 1.3.3 + mod_ssl 2.8.22 + OpenSSL 0.9.7e desde la siguiente dirección: http://hunter.campbus.com/ (tambien disponible en D:\CapituloV\SSL).

Esta versión hay que descomprimirla sobre la carpeta Apache de Appserv, sobrescribiendo los archivos existentes. Antes de proceder con ello hay que hacer una copia del archivo httpd.conf y, una vez instalado, volver a copiar este archivo en su ubicación original. Añadimos entonces también esta línea al archivo httpd.conf:

```
AddModule mod_ssl.c
```

Junto a las directivas similares en el archivo de configuración httpd.conf. Por último, añadimos todo lo siguiente al final del archivo de configuración:

```
SSLMutex sem
SSLRandomSeed startup builtin
SSLSessionCache none
SSLLog logs/SSL.log
SSLLogLevel info
<VirtualHost localhost:443>
SSLEngine On
SSLCertificateFile conf/modssl/test.crt
SSLCertificateKeyFile conf/modssl/test.key
</VirtualHost>
```

Guardamos los cambios y reiniciamos el servidor Apache. Si desde el mismo navegador donde está instalado el servidor tecleamos https://localhost aparecerá una advertencia de que estamos a punto de conectarnos a una conexión segura.

Protección de Scripts

Los archivos contenidos en un servidor web son almacenados en texto plano, es decir, el código de estos archivos puede ser leído fácilmente por cualquier persona que tenga acceso al servidor y así las rutinas que hemos programado o inclusive el sistema entero pueden ser copiados. Desafortunadamente, no se puede emplear un mecanismo de cifrado que altere la información binaria de los archivos porque cuando un usuario ingrese al sistema se deberán de descifrar los archivos y cuando termine de utilizarlo de deberán de volver a cifrar o eliminar, además se deberán descifrar los archivos una vez por cada usuario que ingrese al sistema y almacenar los archivos ya descifrados de manera temporal para cada usuario lo que implica una gran cantidad de memoria y de espacio para el servidor.

La solución a este problema es implementar un mecanismo que trabaje sobre el texto contenido en los scripts del sistema, dicha solución se encuentra disponible en D:\CapituloV\cifrado scripts\, disponible para archivos HTML, Javascript y PHP. El archivo que se debe ejecutar es "cifrador_scrips.php", una vez ejecutado el script aparecerá una pantalla como la siguiente:

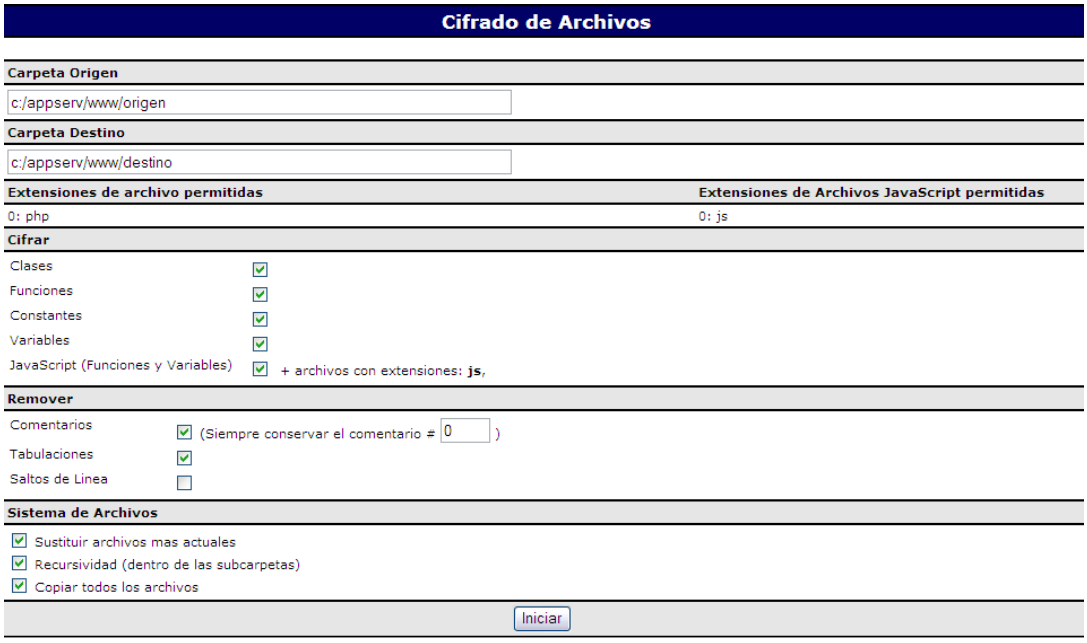


Figura 5.1 Cifrador de archivos PHP

Dicha aplicación solicita la hubicación de la **carpeta origen** que es de la cual se tomarán los scripts para ser cifrados, después solicita la hubicación de la **carpeta destino** que es la carpeta en la cual se almacenarán los archivos cifrados, después se pide especificar los **elementos que serán cifrados** como son clases, funciones, constantes, variables, seguido de eso, se pide especificar los elementos que serán **removidos** de los scripts de la carpeta origen, elementos tales como saltos de línea, tabulaciones y comentarios, finalmente, se pide especificar una seria de parámetros sobre el sistema de archivos como son la sustitución de los archivos mas nuevos solamente, la recursividad para reemplazar los contenidos de los scripts dentro de las subcarptetas de la carpeta origen, y finalmente indicar si se deben sustituir todos los archivos.

Lo que hace la aplicación es tomar archivo por archivo y remplazar los nombres de las funciones y variables que se encuentran en los códigos de los Scripts por sus mismos nombres pero cifrados con el algoritmo MD5 usando una clave de 8 bytes, también se eliminan los comentarios que agrega el programador y los saltos de línea lo que hace casi imposible la interpretación del código contenido en los archivos por cualquier persona. Se eligió el cifrado MD5 de 8 bytes ya que el cifrado de 32 bytes (que es el estándar) incrementaría el tamaño de los códigos de los archivos.

Este proceso se divide en dos partes principales.

- La primera de ellas es abrir los archivos de la carpeta origen, al abrir cada archivo se hace una lista de las variables definidas por el usuario así como las funciones o constantes que se encuentran en el código fuente.
- La segunda consiste en buscar y reemplazar las variables, funciones y constantes de las listas generadas anteriormente para así comenzar a escribir los nuevos archivos en la carpeta destino.

Lo que obtenemos es un archivo no legible por ninguna persona pero si legible por el servidor ya que a éste no le importan los nombres de variables o funciones, los saltos de línea, tabulaciones o los comentarios ya que sólo interpreta el código contenido en los archivos.

Los nombres dados a las variables de sesión, las cookies y las variables de archivo (FILE) no se modifican ya que las mismas variables pueden ser utilizadas por diferentes archivos.

De esta manera tenemos un mecanismo que cifra hasta cierto punto la información y que no requiere de ningún proceso o recurso extra haciéndolo transparente al servidor y al usuario.

Replicación de Bases de Datos

La replicación de información consiste en tener un respaldo exacto de los datos contenidos en un servidor con la finalidad de seguir teniendo disponibles los datos a pesar de algún fallo en éste. Generalmente se utiliza cuando la relevancia de los datos o del sistema es alta, cuando requieren de un proceso muy arduo o largo de captura o simplemente cuando se tienen datos cuya acumulación se ha ido haciendo desde hace mucho tiempo atrás.

La replicación funciona con archivos de bitácora que manejan y administran los SMBD llamados “log files”, una vez configurada la opción de replica en un servidor se comienzan a generar estos archivos que después de cierto tiempo se intercambian con el servidor replica para que éste tome las actualizaciones que se han hecho y las haga él también.

Los sistemas que programamos deben tener la capacidad de identificar cuando el servidor maestro o primario o principal está activo, para que en el momento en el que no se tenga el acceso a él, las consultas se envíen al servidor secundario o replicado ya que éste deberá tener los datos actualizados del sistema. Cuando el servidor secundario es el encargado del funcionamiento del sistema se debe

tener la capacidad de identificar el momento en el cual el servidor primario esté activo nuevamente y se deben enviar las consultas realizadas para continuar con el funcionamiento normal del sistema y mantener la integridad de la información ya que si esto último no se hace, existe una gran posibilidad de que cuando el servidor maestro se recupere, el servidor secundario tendrá datos más actuales.

Para evitar este inconveniente se realiza la replicación doble o en espejo donde cada uno de los dos servidores hace a la vez de maestro y esclavo, la detección de la disponibilidad de los servidores no depende de estos ya que si alguno de ellos deja de funcionar o falla estos no tienen la capacidad de enviar un mensaje de error al usuario o al sistema, la detección de dicha disponibilidad depende al 100% de la forma en la cual programamos las aplicaciones, la mejor forma es intentar conectarse al servidor principal, si no se puede, intentar nuevamente e inclusive una tercera vez para posteriormente realizar los intentos de conexión al servidor secundario, ellos se encargarán de la actualización de los datos una vez hecha la correcta configuración del SMBD para realizar la replicación.

Hay varios SMBD que soportan la replicación de sus BD, sin embargo, algunos de ellos requieren de programas o scripts especiales para poder replicar, también depende de la versión del SMBD que se esté utilizando ya que algunas versiones más antiguas de algunos de ellos no soportan la replicación y algunas otras versiones difieren un poco unas de otras en la forma de configurar sus archivos o programas para poder habilitar la replica.

Sabemos que existe una amplia gama de SMBD, a continuación se mencionan algunos de los más utilizados así como el programa adicional o software que se debe emplear para que dicho SMBD pueda replicar su información:

SMBD	**Programas utilizados para replicar**
MySQL	Forma nativa
PostgreSQL	Slony-I PGReplicate eRserver
Oracle	Forma nativa versiones más nuevas
SQL Server	Forma nativa con ayuda de otros servidores incluidos en su CD de instalación

Sin embargo, no se puede garantizar que estos SMBD soporten la doble replicación ya que nuestra finalidad con este apartado es dar a conocer un poco las herramientas disponibles para poder replicar pero nuestro proyecto se centra en la replicación en el SMBD MySQL el cual sí soporta la doble replicación, en caso de requerir este tipo de información se deberá de ingresar en la página de Internet de dichos programas o SMBD e investigar si soportan la doble replicación donde podrá ver que inclusive algunos de ellos soportan la múltiple replicación (un servidor maestro y varios esclavos) cómo es el caso de Slony-I para PostgreSQL.

Replicación Lineal en MySQL

Esta funcionalidad es soportada desde MySQL versión 4 y para lograrla se hace lo siguiente:

a) Para poder replicar en MySQL, necesitamos que en los dos sitios se encuentre la misma base de datos, con el mismo nombre, con los mismos campos y los mismos datos.

b) Después en el maestro se va a crear un usuario que va a permitir compartir la información con el esclavo, así que dentro de la terminal de MySQL en el servidor maestro y usando nuestra base de datos a compartir ponemos:

```
mysql> GRANT FILE ON nombre_bd.* TO 'maestro1'@'192.168.1.66' IDENTIFIED BY 'maestro1';
```

En el caso anterior la dirección ip que estamos poniendo es la perteneciente al esclavo.

c) Poner en el archivo /etc/my.cnf del maestro en el apartado de [mysqld]

```
log-bin      #Para generar bitácoras

server-id=1  #Identificador  de la máquina, puede ser cualquier valor entero, pero debe ser diferente al de el esclavo.
```

d) En el archivo my.cnf del esclavo en el apartado de [mysqld] ponemos las siguientes líneas:

```
master-host=192.168.1.69        #Ponemos la dirección ip del nuestro maestro

master-user=maestro1      #Nombre del usuario que creamos en nuestro maestro.

master-password=maestro1     #Contraseña del usuario que creamos en nuestro maestro.

master-connect-retry=30         #Tiempo en segundos para reintentar conectarse
```

Después de haber modificado el archivs my.cnf debemos reiniciar el servicio de mysql con:

```
service mysqld restart;
```

Se vuelve a entrar a mysql en el servidor maestro y se verifica que las bitácoras coincidan mediante la instrucción:

```
SHOW SLAVE STATUS \G;
```

Con esta instrucción deben aparecer los datos del servidor esclavo en la pantalla como la dirección IP, el nombre del archivo de la bitácora, si no aparecen ejecute la instrucción siguiente:

```
RESET SLAVE;
```

Después de eso, se intenta de nuevo con:

```
SHOW SLAVE STATUS \G;
```

Replicación en Espejo en MySQL

Para lograrla se hace lo siguiente:

a) Para poder replicar en ambos sentidos en MySQL, necesitamos que en los dos sitios se encuentre la misma base de datos, con el mismo nombre, con los mismos campos y los mismos datos.

b) Después en el maestro se va a crear un usuario que va a permitir compartir la información con el esclavo, así que dentro de la terminal de MySQL en el servidor maestro y usando nuestra base de datos a compartir ponemos:

```
mysql> GRANT FILE ON nombre_bd.* TO 'maestro1'@'192.168.1.66' IDENTIFIED BY 'maestro1';
```

En el caso anterior la dirección ip que estamos poniendo es la perteneciente al esclavo.

c) Poner en el archivo /etc/my.cnf del maestro en el apartado de [mysqld]

log-bin #Para generar bitácoras

server-id=1 #Identificador de la máquina, puede ser cualquier valor entero, pero debe ser diferente al de el esclavo.

d) En el archivo my.cnf del esclavo en el apartado de [mysqld] ponemos las siguientes líneas:

master-host=192.168.1.69 #Ponemos la dirección ip del nuestro maestro

master-user=maestro1 #Nombre del usuario que creamos en nuestro maestro.

master-password=maestro1 #Contraseña del usuario que creamos en nuestro maestro.

master-connect-retry=30 #Tiempo en segundos para reintentar conectarse

log-slave-updates #Esto es para que las actualizaciones también se hagan en el maestro.

Como queremos que la replicación se haga en ambos sentidos, necesitamos que nuestro maestro sea también esclavo y nuestro esclavo sea también maestro, entonces lo que hacemos es:

a) Desde nuestro primer esclavo (el cual ahora queremos que sea maestro), vamos a meternos en una términal de mysql y vamos a poner:

mysql> GRANT FILE ON nombre_bd.* TO 'maestro2'@'192.168.1.69' IDENTIFIED BY 'maestro2';

b) Agregamos al archivo my.cnf del primer esclavo en la parte [mysqld], sin quitar lo que habíamos puesto.

log-bin
server-id= 2

c) Agregamos al archivo my.cnf del que ahora sera nuestro esclavo(anteriormente maestro) en el apartado [mysqld]

master-host=192.168.1.66 #ponemos la dirección ip del nuestro maestro

master-user=maestro2 #Nombre del usuario que creamos en nuestro maestro

master-password=maestro2 #Contraseña del usuario que creamos en nuestro maestro

master-connect-retry=30 #Tiempo en segundos para intentar reconectarse después de una caída del servidor maestro

Después de haber modificado los archivos my.cnf debemos reiniciar el servicio de mysql con:

service mysqld restart;

Se vuelve a entrar a mysql tanto en esclavo como en maestro y se verifica que las bitácoras coincidan mediante la instrucción:

SHOW MASTER STATUS;
SHOW SLAVE STATUS \G;

La información que veas como MAESTRO INICIAL lo debes de ver en el ESCLAVO INICIAL y visceversa (nombre de archivo, número y posición)

Si no coinciden, poner en ambas computadoras (dentro del mysql):

```
RESET MASTER;
RESET SLAVE;
```

Después de eso, se intenta de nuevo con:

```
SHOW MASTER STATUS;
SHOW SLAVE STATUS \G;
```

En ocasiones si no se hace la replicación en ambos sentidos se tiene que cambiar de nuevo el server-id por otro en el archivo my.cnf de los dos archivos, y cambiar los nombres de usuarios en ambas máquinas de nueva cuenta de los dos archivos y volver a darle permisos utilizando el comando GRANT como se mostró anteriormente y finalmente ejecutar reset al esclavo y al maestro.

De esta forma queda replicada la BD y cuando el servidor maestro no esté disponible, el sistema, dentro de su script de conexión a la BD tiene la capacidad de detectar cuando el servidor maestro no esté funcionando y envía las consultas al servidor esclavo, cuando el servidor maestro se reactiva, MySQL actualiza la BD de forma automática.

Replicación Multilineal en MySQL

La replicación multilineal no es muy diferente de la replicación en espejo, basta con definir correctamente los servidores que serán replicados y los sentidos de la replicación.

a) Para poder replicar en varios sentidos en MySQL, necesitamos que en todos los sitios se encuentren las mismas tablas de la base de datos que queremos replicar, con el mismo nombre, con los mismos campos y los mismos datos.

b) Después en cada maestro se va a crear un usuario que va a permitir compartir la información con los demás esclavos, así que dentro de la terminal de MySQL en el servidor maestro y usando nuestra base de datos a compartir ponemos:

```
mysql> GRANT FILE ON nombre_bd.* TO 'maestro1'@'192.168.1.66' IDENTIFIED BY 'maestro1';
mysql> GRANT FILE ON nombre_bd.* TO 'maestro3'@'192.168.1.67' IDENTIFIED BY 'maestro3';
```

En el caso anterior las direcciones IP que estamos poniendo son las pertenecientes a los esclavos.

c) Poner en el archivo /etc/my.cnf del maestro en el apartado de [mysqld]

```
log-bin      #Para generar bitácoras

server-id=1  #Identificador  de la máquina, puede ser cualquier valor entero, pero debe ser diferente al de cualquier esclavo.
```

d) En el archivo my.cnf de cada esclavo en el apartado de [mysqld] ponemos las siguientes líneas para el **esclavo 1**:

```
master-host=192.168.1.69        #Ponemos la dirección ip del nuestro maestro

master-user=maestro1      #Nombre del usuario que creamos en nuestro maestro.
```

master-password=maestro1 #Contraseña del usuario que creamos en nuestro maestro.

master-connect-retry=30 #Tiempo en segundos para reintentar conectarse

Y las siguientes líneas para el **esclavo 3**:

master-host=192.168.1.69 #Ponemos la dirección ip del nuestro maestro

master-user=maestro3 #Nombre del usuario que creamos en nuestro maestro.

master-password=maestro3 #Contraseña del usuario que creamos en nuestro maestro.

master-connect-retry=30 #Tiempo en segundos para reintentar conectarse

De esta forma cuando se haga un cambio en la BD del maestro los cambios se verán reflejados en ambos esclavos. Posteriormente, en cada esclavo que quiera ser replicado se deberán de modificar de igual manera sus archivos de configuración.

CAPITULO VI.- XML

Introducción a XML

XML es un metalenguaje usado para definir documentos que contienen datos estructurados. Las características y beneficios del XML se pueden agrupar en estas áreas principales:

- Extensibilidad: como metalenguaje, XML puede usarse para crear sus propios lenguajes de marcas. Hoy en día existen muchos lenguajes de marcas basados en XML, incluyendo "Wireless Markup Language" (WML).

- Estructura precisa: HTML sufre de una pobre estructura que hace dificil procesar eficientemente documentos HTML. Por otro lado, los documentos XML están bien estructurados, cada documento tiene un elemento raíz y todos los demás elementos deben estar anidados dentro de otros elementos.

- Dos tipos de documentos: hay dos tipos principales de documentos XML.

    - **Documentos Válidos**: un documento XML válido está definido por una "Document Type Definition" (DTD), que es la gramática del documento que define qué tipos de elementos, atributos y entidades podría haber en el documento. El DTD define el orden y también la ocurrencia de elementos.

    - **Documentos Bien-Formateados**: un documento XML bien formateado no tiene que adherirse a una DTD. Pero debe seguir dos reglas: 1) todo elemento debe tener una etiqueta de apertura y otra de cierre. 2) debe haber un elemento raíz que contenga todos los otros elementos.

- Extensión Poderosa: Como se mencionó anteriormente, XML sólo se usa para definir la síntaxis. En otras palabras, se usa para definir contenido. Para definir la semántica, el estilo o la presentación, necesitamos usar "Extensible Stylesheet Language" (XSL). Observa que un documento podría tener múltiples hojas de estilo que podrían ser usadas por diferentes usuarios.

XML y PHP

Quizás no seamos conscientes del potencial de estas dos tecnologías juntas, pero si nos fijamos bien, podemos darnos cuenta de que XML y PHP pueden funcionar de una forma muy similar a como puede trabajar PHP con una base de datos.

Aunque utilizar estas dos tecnologías juntas no excluye usar bases de datos (eso es lo mejor de todo).

Para empezar vamos a crear nuestro archivo XML de ejemplo, al que llamaremos “noticias.xml”:

```
1       <?xml version=”1.0” encoding=”ISO-8859-1”?>
2       <bloque>
3       <noticia>
4       <titulo>Hola Caracola </titulo>
```

```
5       <autor>KaoS</autor>
6       <cuerpo>Olla Kaitos a Luisete</cuerpo>
7       </noticia>
8       <noticia>
9       <titulo>Nuevo articulo en desarrollo </titulo>
10      <autor>Raul</autor>
11      <cuerpo>Jeje hola, aqui estamos </cuerpo>
12      </noticia>
13      </bloque>
```

Archivo noticias.xml, archivo XML de un portal de noticias ficticio

Bueno ya tenemos creado nuestro archivo XML, que como ya sabemos nos permite crear nuestras propias etiquetas. Ahora necesitamos crear un archivo PHP que lea nuestro archivo "noticias.xml".

Lo primero que tenemos que hacer es leer el archivo, para ello utilizaremos la función fopen. Da igual si el archivo se encuentra en nuestro servidor o no, por lo que si nos interesa podriamos crear un archivo PHP que funcionara igualmente en un servidor remoto que leyese las noticias de nuestra web.

```
//$ruta_archivo="http://www.dominio.com/noticias.xml";
$ruta_archivo="noticias.xml";

$contenido = "";
if($da = fopen($ruta_archivo,"r"))
{
        while ($aux= fgets($da,1024))
        {
                $contenido.=$aux;
        }
        fclose($da);
}
else
{
        echo "Error: no se ha podido leer el archivo <strong>$ruta_archivo</strong>";
}
```

Si todo ha ido correctamente ahora tendremos nuestro archivo XML cargado en nuestra variable **$contenidoAhora**, antes un detalle, debido a que nuestras noticias podrían tener caracteres especiales, para evitar fallos le meteremos un pequeño filtro, que en este caso por ejemplo vamos a sustituir las tildes y las eñes en el caso de que las hubiesen, para ello utilizaremos la función **ereg_replace**.

```
1       $contenido=ereg_replace("á","a",$contenido);
2       $contenido=ereg_replace("é","e",$contenido);
3       $contenido=ereg_replace("í","i",$contenido);
4       $contenido=ereg_replace("ó","o",$contenido);
5       $contenido=ereg_replace("ú","u",$contenido);
6       $contenido=ereg_replace("Á","A",$contenido);
7       $contenido=ereg_replace("É","E",$contenido);
8       $contenido=ereg_replace("Í","I",$contenido);
9       $contenido=ereg_replace("Ó","O",$contenido);
10      $contenido=ereg_replace("Ú","U",$contenido);
11      $contenido=ereg_replace("Ñ","NI",$contenido);
12      $contenido=ereg_replace("ñ","ni",$contenido);
```

El siguiente paso es cargar nuestro archivo XML en una estructura que podamos trabajar con PHP de forma cómoda, para esta tarea vamos a utilizar las funciones **dom** que vienen implementadas a partir de la versión 4 de PHP. Concretamente usaremos:

- domxml_open_mem: Crea un objeto DOM desde un documento XML
- document_element: Crear un nuevo nodo de tipo elemento
- get_elements_by_tagname: Obtiene elementos por el nombre de etiqueta
- get_content: Obtiene el contenido del nodo

```
1       $tagnames = array ("titulo","autor","cuerpo");
2
3       if (!$xml = domxml_open_mem($contenido)) {
4               echo "Ha ocurrido un error con el archivo<strong> \"$ruta_archivo\"</strong> a XML <br>";
5               exit;
6       }
7       else {
8               $raiz = $xml->document_element();
9
10              $tam=sizeof($tagnames);
11
12              for($i=0; $i<$tam; $i++) {
13                      $nodo = $raiz->get_elements_by_tagname($tagnames[$i]);
14                      $j=0;
15                      foreach ($nodo as $etiqueta) {
16                              $matriz[$j][$tagnames[$i]]=$etiqueta->get_content();
17                              $j++;
18                      }
19              }
20      }
```

Veamos más detenidamente este último segmento de código para analizar qué es lo que realmente hace. Para empezar hemos creado un arreglo con los campos que contiene cada noticia en la variable "$tagnames". A continuación cargamos la variable contenido en un objeto **DOM**, en el caso de que todo haya ido bien extraemos el nodo raiz, en nuetro caso "bloque". El siguiente paso es calcular el número de campos que obtendremos de cada noticia, para ello utilizamos la función **sizeof** que nos devuelve el tamaño del arreglo.

Es ahora cuando extraemos la información del documento XML. Esta información la vamos a introducir en una matriz para que nos sea más simple trabajar con los datos. De forma que matriz quede así:

indice / Nombre Columna	Titulo	Autor	Cuerpo
0	Hola Caracola	KaoS	Olla Kaitos a Luisete
1	Nuevo articulo	Roberto Luna	Hola!!!, aqui estamos

El primer ciclo extrae las etiquetas de los nodos (primero titulo, despues autor y luego cuerpo).

El **foreach** se encarga de sacar una a una las etiquetas de cada una de las noticias, por lo que primero extrae "Hola Caracola" y en la segunda iteración "Nuevo articulo en desarrollo". De este modo vamos guardando en nuestra matriz los datos extraidos.

En la segunda iteración del ciclo **for** recogeremos la etiqueta “autor”, y en el **foreach** extraeremos los valores para introducirlos en la matriz, y así hasta terminar. Lo mejor de todo es que de esto se encarga el propio ciclo, solo tendremos que preocuparlos de declarar el arreglo de etiquetas.

Bueno para que nos sea más cómodo podemos crear una función a la que le pasaremos el archivo XML que queremos que nos lea y nos devuelva una matriz con los datos, haciendo así nuestro trabajo más limpio y eficiente. El código resultante sería:

```
1       function CargarXML($ruta_archivo) {
2               $contenido = "";
3               if($da = fopen($ruta_archivo,"r"))
4               {
5                       while ($aux= fgets($da,1024))
6                       {
7                               $contenido.=$aux;
8                       }
9                       fclose($da);
10              }
11              else
12              {
13                      echo "Error: no se ha podido leer el archivo <strong>$ruta_archivo</strong>";
14              }
15
16              $contenido=ereg_replace("á","a",$contenido);
17              $contenido=ereg_replace("é","e",$contenido);
18              $contenido=ereg_replace("í","i",$contenido);
19              $contenido=ereg_replace("ó","o",$contenido);
20              $contenido=ereg_replace("ú","u",$contenido);
21              $contenido=ereg_replace("Á","A",$contenido);
22              $contenido=ereg_replace("É","E",$contenido);
23              $contenido=ereg_replace("Í","I",$contenido);
24              $contenido=ereg_replace("Ó","O",$contenido);
25              $contenido=ereg_replace("Ú","U",$contenido);
26              $contenido=ereg_replace("Ñ","NI",$contenido);
27              $contenido=ereg_replace("ñ","ni",$contenido);
28
29              $tagnames = array ("titulo","autor","cuerpo");
30
31              if (!$xml = domxml_open_mem($contenido))
32              {
33                      echo "Ha ocurrido un error con el archivo<strong> \"$ruta_archivo\"</strong> a XML
<br>";
34                      exit;
35              }
36              else
37              {
38                      $raiz = $xml->document_element();
39
40                      $tam=sizeof($tagnames);
41
42                      for($i=0; $i<$tam; $i++)
43                      {
44                              $nodo = $raiz->get_elements_by_tagname($tagnames[$i]);
45                              $j=0;
46                              foreach ($nodo as $etiqueta)
47                              {
48                                      $matriz[$j][$tagnames[$i]]=$etiqueta->get_content();
49                                      $j++;
50                              }
51                      }
52
53                      return $matriz;
54              }
```

```
55      }
```

Ya hemos cargado una matriz con el contenido de un archivo XML, por lo que ahora solo nos queda mostrar la información que queramos. Vamos a ver en un pequeño código como hacerlo.

```
1       $matriz=CargarXML("noticias.xml");
2
3       $num_noticias=sizeof($matriz);
4       for($i=0;$i<$num_noticias;$i++)
5       {
6               echo '
7                       <table border=1>
8                       q<tr><td align=center>'.$matriz[$i]["titulo"].'</td></tr>
9                       <tr><td>'.$matriz[$i]["cuerpo"].'</td></tr>
10                      <tr><td align=right >'.$matriz[$i]["autor"].'</td></tr>
11                      </table><br>
12              ';
13      }
```

XML y JSP / Servlets

El ejemplo siguiente es un documento de ejemplo XML. Como podemos ver, hay un elemento raíz (portfolio), y cada elemento tiene una etiqueta de inicio y una etiqueta de cierre.

```
1       <?xml version="1.0" encoding="UTF-8"?>
2       <portfolio>
3       <stock>
4       <symbol>SUNW</symbol>
5       <name>Sun Microsystems</name>
6       <price>17.1</price>
7       </stock>
8       <stock>
9       <symbol>AOL</symbol>
10      <name>America Online</name>
11      <price>51.05</price>
12      </stock>
13      <stock>
14      <symbol>IBM</symbol>
15      <name>International Business Machines</name>
16      <price>116.10</price>
17      </stock>
18      <stock>
19      <symbol>MOT</symbol>
20      <name>MOTOROLA</name>
21      <price>15.20</price>
22      </stock>
23      </portfolio>
```

Archivo stocks1.xml, archivo de inventarios ficticio

La primera línea indica el número de versión de XML, que es 1; y le permite al procesador conocer qué esquema de códificación se va a utilizar "UTF-8".

Aunque es un documento sencillo contiene información útil que puede ser procesada fácilmente porque está bien estructurada. Por ejemplo, el servidor de stocks podría querer ordenar el portfolio, en un orden específico, basándose en el precio de los stocks.

Presentar Documentos XML

Los documentos XML contienen datos portables. Esto significa que el ejemplo stocks1.xml puede procesarse como salida para diferentes navegadores (navegadores de escritorio para PC, micronavegadores para dispositivos móviles). En otras palabras, un documento XML puede ser transformado en HTML o WML o cualquier otro lenguaje de marcas.

Si cargamos el documento stocks1.xml en un navegador que soporte XML (como IE de Microsoft), veríamos algo similar a la figura siguiente:

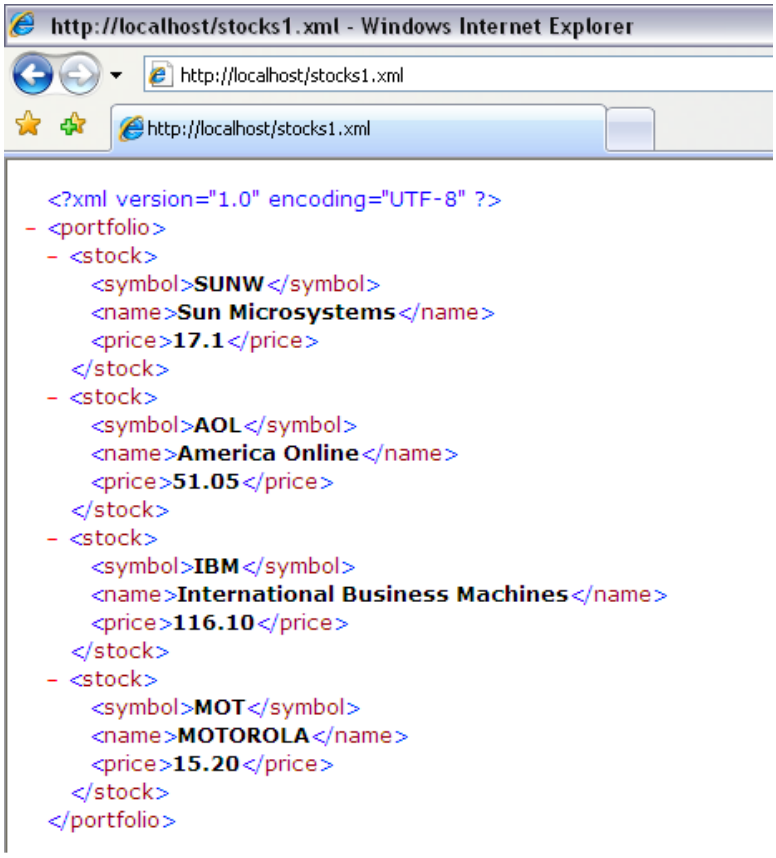


Figura 6.1, Ejecución archivo stocks1.xml

Básicamente, el navegador ha analizado el documento y lo ha mostrado de una forma estructurada. Pero, esto no es realmente útil desde el punto de vista de un usuario. Los usuarios desean ver los datos mostrados como información útil de una forma que sea fácil de navegar.

Una forma agradable para mostrar documentos XML es aplicar una transformación sobre el documento XML, para extraer los datos o para crear un nuevo formato (como transformar datos XML a HTML). Esta transformación se puede hacer usando un lenguaje de transformación como "Extensible Stylesheet Language Transformation" (XSLT), que forma parte de XSL. XSL permite que escribamos el vocabulario XML para especificar la semántica del formato. Es decir hay dos partes de XSL, que son

parte de la actividad de estilo del World Wide Web Consortium (W3C).

- Lenguaje de Transformación (XSLT)
- Lenguaje de Formateo (objetos de formateo XSL)

La hoja de estilos XSL del ejemplo siguiente realiza la transformación requerida para el elemento portfolio. Genera marcas HTML y extrae datos de los elementos del documento stocks1.xml.

```
1       <?xml version="1.0"?>
2
3       <xsl:stylesheet version="1.0" xmlns:xsl= "http://www.w3.org/TR/WD-xsl">
4
5       <xsl:template match="/">
6       <html>
7       <head>
8       <title>Stocks</title>
9       </head>
10      <body bgcolor="#ffffcc" text="#0000ff">
11      <xsl:apply-templates/>
12      </body>
13      </html>
14      </xsl:template>
15
16      <xsl:template match="portfolio">
17
18      <table border="2" width="50%">
19      <tr>
20      <th>Stock Symbol</th>
21      <th>Company Name</th>
22      <th>Price</th>
23      </tr>
24      <xsl:for-each select="stock">
25      <tr>
26      <td>
27      <i><xsl:value-of select= "symbol"/></i>
28      </td>
29      <td>
30      <xsl:value-of select="name"/>
31      </td>
32      <td>
33      <xsl:value-of select="price"/>
34      </td>
35
36      </tr>
37      </xsl:for-each>
38      </table>
39      </xsl:template>
40
41      </xsl:stylesheet>
```

Archivo stocks2.xml, archivo de inventarios ficticio

Al principio parece un poco complejo pero una vez que se conozca la síntaxis XSL es bastante sencillo. Aquí lo tenemos todo sobre la síntaxis de arriba:

- xsl:stylesheet: elemento raíz
- xsl:template: cómo transformar los nodos seleccionados

- match: atributo para seleccionar un nodo
- "/": nodo raíz del documento XML de entrada
- xsl:apply-templates: aplica las plantillas a los hijos del nodo seleccionado
- xsl:value-of: nodo seleccionados (extrae datos de los nodos seleccionados)

Ahora la cuestión es ¿cómo usar esta hoja de estilos XSL con el documento stocks.xml? La respuesta es sencilla, necesitamos modificar la primera línea del documento stocks.xml del ejemplo stocks1.xml para usar la hoja de estilos stocks.xsl para su representación. La primera línea del ejemplo stocks1.xml ahora debería ser:

```
1       <?xml:stylesheet type="text/xsl" href="stocks.xsl" version="1.0" encoding="UTF-8"?>
```

Esto dice básicamente que cuando cargamos stocks1.xml en un navegador (será analizado como un árbol de nodos), se debe utilizar la hoja de estilos correspondiente para extraer datos de los nodos del árbol. Cuando cargamos el archivo stocks.xml modificado en un navegador que soporta XML y XSL (como IE de Microsoft), veremos algo similar a la figura 6.2:

Stock Symbol	Company Name	Price
SUNW	Sun Microsystems	17.1
AOL	America Online	51.05
IBM	International Business Machines	116.10
MOT	MOTOROLA	15.20

Figura 6.2: Usando una hoja de estilo, archivo stocks3.xsl

Generar XML desde JSP

Se puede usar la tecnología JSP para generar documentos XML. Una página JSP podría generar fácilmente una respuesta conteniendo el documento stocks1.xml, como se ve en el siguiente ejemplo. El principal requerimiento para generar XML es que la página JSP seleccione el tipo de contenido de la página de forma apropiada. Como podemos ver a partir del siguiente ejemplo, el tipo de contenido se selecciona a text/xml. Se puede usar la misma técnica para generar otros lenguajes de marcas (como WML).

```
1       <%@ page contentType="text/xml" %>
2       <?xml version="1.0" encoding="UTF-8"?>
3       <portfolio>
4       <stock>
5       <symbol>SUNW</symbol>
6       <name>Sun Microsystems</name>
7       <price>17.1</price>
8       </stock>
9       <stock>
10      <symbol>AOL</symbol>
11      <name>America Online</name>
12      <price>51.05</price>
```

```
13      </stock>
14      <stock>
15      <symbol>IBM</symbol>
16      <name>International Business
17      Machines</name>
18      <price>116.10</price>
19      </stock>
20      <stock>
21      <symbol>MOT</symbol>
22      <name>MOTOROLA</name>
23      <price>15.20</price>
24      </stock>
25      </portfolio>
```

Archivo genXML.jsp, generar archivos XML desde JSP

Si solicitamos la página genXML.jsp desde un servidor Web (como Tomcat), veríamos algo similar a la figura 6.1.

Generar XML desde JSP y JavaBeans

Los datos para XML también pueden recuperarse desde un componente JavaBean. Por ejemplo, el siguiente componente JavaBean, PortfolioBean, define un bean con datos de stocks:

```
1       package stocks;
2
3       import java.util.*;
4
5       public class PortfolioBean implements java.io.Serializable {
6               private Vector portfolio = new Vector();
7
8               public PortfolioBean() {
9                       portfolio.addElement(new Stock("SUNW", "Sun Microsystems", (float) 17.1));
10                      portfolio.addElement(new Stock("AOL", "America Online", (float) 51.05));
11                      portfolio.addElement(new Stock("IBM", "International Business Machines",(float)
        116.10));
12                      portfolio.addElement(new Stock("MOT", "MOTOROLA", (float) 15.20));
13              }
14
15              public Iterator getPortfolio() {
16                      return portfolio.iterator();
17              }
18      }
```

Archivo PortfolioBean.java

La clase PortfolioBean usa la clase Stock que se muestra en el ejemplo siguiente:

```
1       package stocks;
2
3       public class Stock implements java.io.Serializable {
4               private String symbol; private String name; private float price;
5
6               public Stock(String symbol, String name, float price) {
7                       this.symbol = symbol; this.name = name; this.price = price;
8               }
9
10              public String getSymbol() {
11                      return symbol;
```

```
12              }
13
14              public String getName() {
15                      return name;
16              }
17
18              public float getPrice() {
19                      return price;
20              }
21      }
```

Archivo Stock.jsp

Ahora, podemos escribir una página JSP para generar un documento XML donde los datos son recuperados desde el PortfolioBean, como se ve en el ejemplo siguiente.

```
1       <%@ page contentType="text/xml" %>
2       <%@ page import="stocks.*" %>
3
4       <jsp:useBean id="portfolio" class="stocks.PortfolioBean" />
5
6       <%
7               java.util.Iterator folio = portfolio.getPortfolio(); Stock stock = null;
8       %>
9
10      <?xml version="1.0" encoding="UTF-8"?>
11      <portfolio>
12      <% while (folio.hasNext()) { %>
13      <% stock = (Stock)folio.next(); %>
14      <stock>
15      <symbol<>%=stock.getSymbol() %></symbol>
16      <name<>%=stock.getName() %></name>
17      <price<>%=stock.getPrice() %></price>
18      </stock>
19      <% } %>
20      </portfolio>
```

Archivo stocks3.jsp

Si solicitamos la página stocks.jsp desde un navegador Web que soporte XML, obtendríamos algo similar a la Figura 6.3.

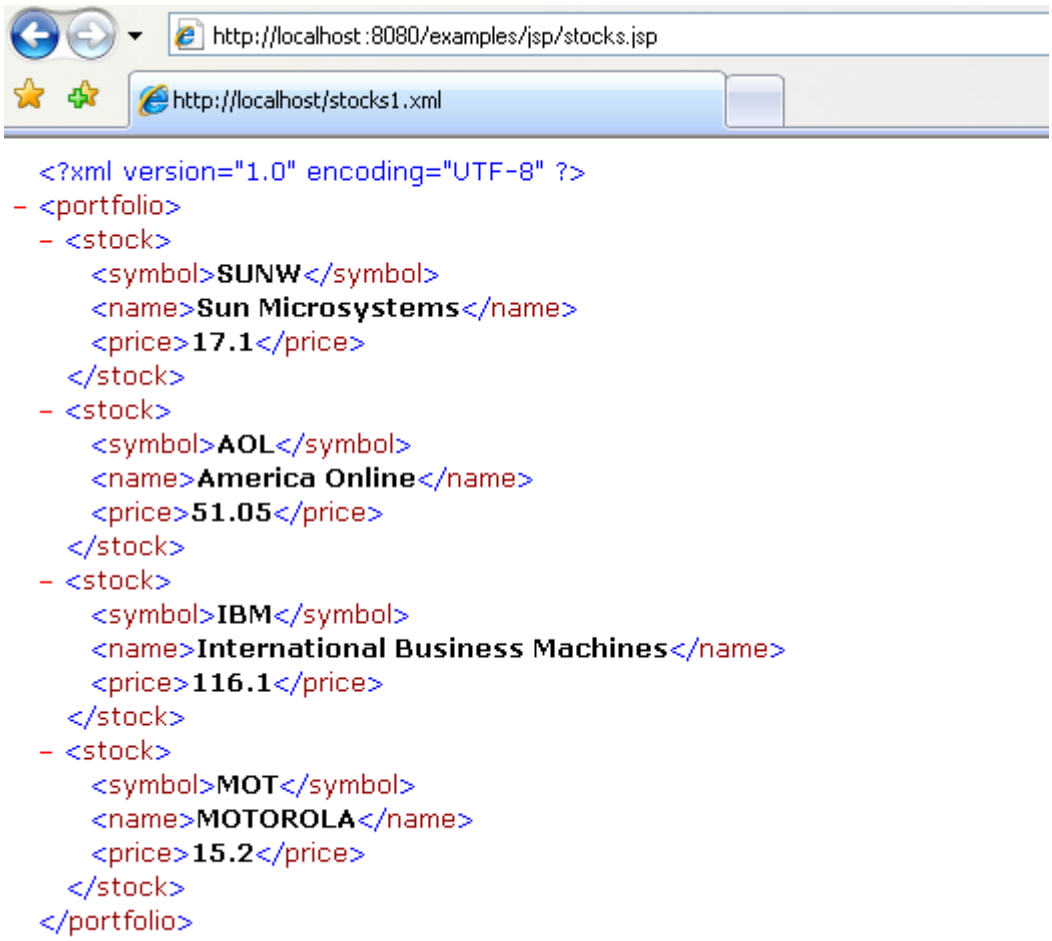


Figura 6.3: Generar XML desde JSP y JavaBeans

Si reemplazamos la línea:

```
<?xml version="1.0" encoding="UTF-8"?>
```

Con una línea que especifique una hoja de estilos:

```
<?xml:stylesheet type="text/xsl" href="stocks.xsl" version="1.0"encoding="UTF-8"?>
```

El documento XML será generado y se aplicará la hoja de estilo XSL y veremos algo similar a la figura 6.2 anterior.

Convertir XML a Objetos del Lado del Servidor

Hemos visto cómo generar XML, así que la pregunta ahora es cómo consumirlo (o usarlo) en aplicaciones. Para poder hacer todo lo que necesitamos para convertir XML en objetos del lado del servidor y extraer las propiedades del objeto. La conversión no es automática; tenemos que analizar manualmente un documento XML, y encapsularlo dentro de un componente JavaBeans. En el futuro sin embargo, la tecnología de unión XML/Javaautomatizará este proceso pues permitirá compilar un

esquema XML en clases Java.

Para analizar se pueden usar dos interfaces:

- Simple API for XML (SAX)
- Document Object Model (DOM)

Antes de introducirnos en esta técnica de análisis, primero describiremos el entorno de software.

El Entorno de Software

El entorno de software que usaremos para analizar es el API para Procesamiento de XML (JAXP) versión 1.1 que soporta SAX, DOM level 2, y XSL transformations. JAXP puede ser descargado desde la dirección http://java.sun.com/xml/download.html.

Para instalarlo, lo descomprimimos en un directorio de nuestro elección, y actualizamos el classpath para incluir el árbol de archivo JAR del JAXP.

- crimson.jar: el analizador XML por defecto, que fue derivado del analizador "Java Project X " de Sun
- xalan.jar: El motor XSLT por defecto
- jaxp.jar: los APIs

Alternativamente, podemos instalar estos archivos JAR como extensiones de Java 2 simplemente copiándolos en el directorio JAVA_HOME/jre/lib/ext, donde JAVA_HOME es el directorio donde instalamos el JDK (por ejemplo c:\jdk1.3). Instalar los archivos JAR como extensiones de Java 2 elimina la necesidad de modificar las variables de entorno.

API Simple para XML (SAX)

SAX es un sencillo API para XML. No es un analizador. Simplemente es un interface estándar, implementado por muchos y diferentes analizadores XML, que fue desarrollado por los miembros de la XML-DEV mailing list, actualmente hospedada por OASIS.

La ventaja principal de SAX es que es ligero y rápido. Esto es principalmente porque es un API basado en eventos, lo que significa que reporta eventos de análisis (como el comienzo y el final de los elementos) directamente a la aplicación usando servicios repetidos, según lo mostrado en la Figura 6.4. Por lo tanto, la aplicación implementa manejadores para ocuparse de los diversos eventos, como el manejo de eventos en un interface gráfico de usuario.

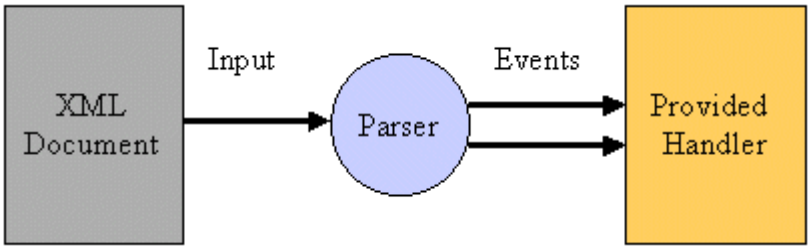


Figura 6.4: SAX usa retrollamadas para notificar a los manejadores las cosas de interés

Usar SAX implica los siguientes pasos, mostrados en la figura 6.7.

- Implementar uno o más manejadores (en este ejemplo se implementa el ContentHandler)
- Crear un XMLReader
- Crear un InputSource
- Llamar a parse sobre la fuente de entrada

Observa que MySAXParserBean sobreescribe los métodos startElement, endElement, y characters, todos los cuales están definidos por el interface ContentHandler. El analizador llama al método startElement al principio de cada elemento del documento XML, y llama al método characters para reportar cada dato de caracter, y finalmente llama al método endElement al final de cada elemento del documento XML.

```
1       package saxbean;
2
3       import java.io.*;
4       import java.util.*;
5       import org.xml.sax.*;
6       import org.xml.sax.helpers.DefaultHandler;
7       import javax.xml.parsers.SAXParserFactory;
8       import javax.xml.parsers.ParserConfigurationException;
9       import javax.xml.parsers.SAXParser;
10
11      public class MySAXParserBean extends
12              DefaultHandler implements java.io.Serializable {
13              private String text;
14              private Vector vector = new Vector();
15              private MyElement current = null;
16
17              public MySAXParserBean() {
18              }
19
20              public Vector parse(String filename) throws
21                      Exception { SAXParserFactory spf = SAXParserFactory.newInstance();
22                      spf.setValidating(false);
23                      SAXParser saxParser = spf.newSAXParser();
24                      // create an XML reader
25                      XMLReader reader = saxParser.getXMLReader(); FileReader file = new
                        FileReader(filename);
26                      // set handler reader.setContentHandler(this);
27                      // call parse on an input source
28                      reader.parse(new InputSource(file));
29                      return vector;
30              }
31
32              // receive notification of the beginning of an element
33              public void startElement (String uri, String name, String qName, Attributes atts) {
34                      current = new MyElement( uri, name, qName, atts);
```

```
35                      vector.addElement(current); text = new String();
36              }
37
38              // recive notificacion del fin de un elemento
39              public void endElement (String uri, String name, String qName) {
40                      if(current != null && text != null) {
41                              current.setValue(text.trim());
42                      }
43                      current = null;
44              }
45
46              // recive notificacion de los datos de los caracteres
47              public void characters (char ch[], int start, int length) {
48                      if(current != null && text != null) {
49                              String value = new String(ch, start, length);
50                              text += value;
51                      }
52              }
53      }
```

Archivo MySAXParserBean.java

MySAXParserBean está usando la clase MyElement, que se define en el ejemplo siguiente.

```
1       package saxbean;
2
3       import org.xml.sax.Attributes;
4
5       public class MyElement implements java.io.Serializable {
6               String uri;
7               String localName; String qName; String value=null;
8               Attributes attributes;
9
10              public MyElement(String uri, String localName, String qName, Attributes attributes) {
11                      this.uri = uri; this.localName = localName; this.qName = qName;
12                      this.attributes = attributes;
13              }
14
15              public String getUri() {
16                      return uri;
17              }
18
19              public String getLocalName() {
20                      return localName;
21              }
22
23              public String getQname() {
24                      return qName;
25              }
26
27              public Attributes getAttributes() {
28                      return attributes;
29              }
30              public String getValue() {
31                      return value;
32              }
33
34              public void setValue(String value) {
35                      this.value = value;
36              }
37      }
```

Archivo MyElement.java

Ahora, si queremos, podemos probar el MySAXParserBean desde la línea de comando para asegurarnos de que funciona. El ejemplo siguiente muestra una sencilla clase de prueba:

```
1   import java.io.*;
2   import java.util.*;
3
4   public class MyTestDriver {
5
6       public static void main(String argv[]) {
7               String file = new String(argv[0]);
8               MySAXParserBean p = new MySAXParserBean();
9               String str = null;
10              try {
11                      Collection v = p.parse(file);
12                      Iterator it = v.iterator();
13                      while(it.hasNext()) {
14                              MyElement element = (MyElement)it.next();
15                              String tag = element.getLocalName();
16
17                              if(tag.equals("symbol")) {
18                                      System.out.println("Symbol. " + element.getValue());
19                              }
20                              else if(tag.equals("name")) {
21                                      System.out.println("Name: "+element.getValue());
22                              }
23                              else if(tag.equals("price")) {
24                                      System.out.println("Price: "+element.getValue());
25                              }
26                      }
27              }
28              catch (Exception e) {
29              }
30      }
31  }
```

Archivo MyTestDriver.java

Si ejecutamos MyTestDriver proporcionándole el documento XML del ejemplo "stocks1.xml", veremos algo similar a la Figura 6.5.

```
C:\jaxp-1.1>java MyTestDriver stocks.xml
Symbol: SUNW
Name: Sun Microsystems
Price: 17.1
Symbol: AOL
Name: America Online
Price: 51.05
Symbol: IBM
Name: International Business Machines
Price: 116.10
Symbol: MOT
Name: MOTOROLA
Price: 15.20
```

Figura 6.5: Probar el analizador XML desde la línea de comandos

Ahora, veamos cómo usar el MySAXParserBean desde una página JSP. Como podemos ver desde el siguiente ejemplo, es sencillo. LLamamos al método parse de MySAXParserBean sobre el documento XML (stocks.xml), y luego iteramos sobre los stocks para extraer los datos y formatearlos en una tabla HTML.

```
1       <html>
2       <head>
3       <title>sax parser</title>
4       <%@ page import="java.util.*" %>
5       <%@ page import="saxbean.*" %>
6       </head>
7
8       <body bgcolor="#ffffcc">
9
10      <jsp:useBean id="saxparser" class="saxbean.MySAXParserBean" />
11
12      <%
13              Collection stocks = saxparser.parse("c:/stocks/stocks.xml"); Iterator ir = stocks.iterator();
14      %>
15
16      <center>
17      <h3>My Portfolio</h3>
18      <table border="2" width="50%">
19      <tr>
20      <th>Stock Symbol</th>
21      <th>Company Name</th>
22      <th>Price</th>
23      </tr>
24      <tr>
25
26      <%
27      while(ir.hasNext()) {
28              MyElement element = (MyElement) ir.next();
29              String tag = element.getLocalName();
30              if(tag.equals("symbol")) { %>
31                      <td><%= element.getValue() %></td>
32                      <% } else if (tag.equals("name")) { %>
33                      <td><%= element.getValue() %></td>
34                      <% } else if (tag.equals("price")) { %>
35                      <td><%= element.getValue() %></td></tr><tr>
36              <% } %>
37      <% } %>
38
39      </body>
40      </html>
```

Archivo saxstocks.jsp

Si solicitamos saxstocks.jsp veremos algo similar a la Figura 6.6.

My Portfolio

Stock Symbol	Company Name	Price
SUNW	Sun Microsystems	17.1
AOL	America Online	51.05
IBM	International Business Machines	116.10
MOT	MOTOROLA	15.20

Figura 6.6: Usar MySAXParserBean desdeJSP

Document Object Model (DOM)

DOM es un interface neutral a la plataforma - y al lenguaje- para acceder y actualizar documentos. Sin embargo, al contrario que SAX, DOM accede a un documento XML a través de una estructura arborescente, compuesta por nodos elemento y nodos de texto, según lo mostrado en la figura 6.7.

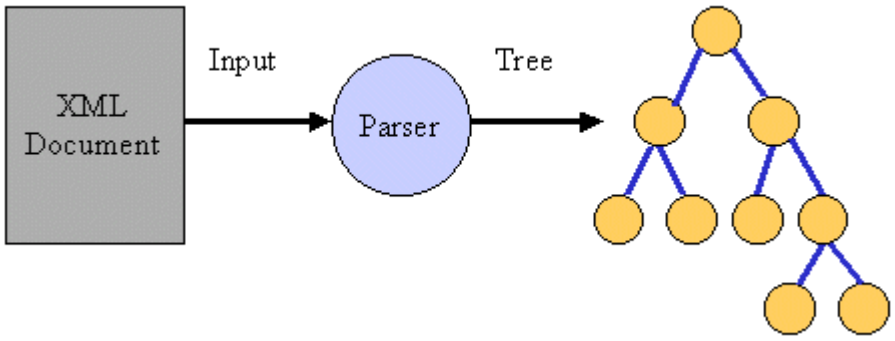


Figura 6.7: DOM crea un árbol desde un documento XML.

El árbol será construido en memoria y por lo tanto se consumen grandes requisitos de memoria al usar DOM. Sin embargo, la ventaja de DOM es que es más simple de programar que SAX.

Como podemos ver desde el siguiente ejemplo, un documento XML se puede convertir en un árbol con tres líneas de código. Una vez que tengamos un árbol, podemos recorrerlo o atravesarlo hacia adelante y hacia atrás.

```
1       package dombean;
2
3       import javax.xml.parsers.*;
4       import org.w3c.dom.*;
5
6       import java.io.*;
7
8       public class MyDOMParserBean implements java.io.Serializable {
9               public MyDOMParserBean() {
10              }
11
12              public static Document getDocument(String file) throws Exception {
13                      DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();
14                      DocumentBuilder db = dbf.newDocumentBuilder();
15                      Document doc = db.parse(new File(file));
16                      return doc;
17              }
18      }
```

Archivo MyDOMParserBean.java

Esto producirá un árbol, por eso necesitamos atraversarlo, Esto se hace usando el método traverseTree en la página JSP, domstocks.jsp, en el ejemplo siguiente.

```
1       <html>
2       <head>
3       <title>dom parser</title>
4       <%@ page import="javax.xml.parsers.*" %>
5       <%@ page import="org.w3c.dom.*" %>
```

```
6       <%@ page import="dombean.*" %>
7       </head>
8
9       <body bgcolor="#ffffcc">
10
11      <center>
12      <h3>My Portfolio</h3>
13      <table border="2" width="50%">
14      <tr>
15      <th>Stock Symbol</th>
16      <th>Company Name</th>
17      <th>Price</th>
18      </tr>
19
20      <jsp:useBean id="domparser" class="dombean.MyDOMParserBean" />
21
22      <%
23      Document doc = domparser.getDocument("c:/stocks/stocks.xml");
24      traverseTree(doc, out);
25      %>
26
27      <%! private void traverseTree(Node node, JspWriter out) throws Exception {
28              if(node == null) {
29                      return;
30              }
31              int type = node.getNodeType();
32
33              switch (type) {
34                      out.println("<tr>"); traverseTree(((Document)node).getDocumentElement(), out);
35                      break;
36              }
37              String elementName = node.getNodeName();
38              if(elementName.equals("stock")) {
39                      out.println("</tr><tr>");
40              }
41              NodeList childNodes = node.getChildNodes();
42              if(childNodes != null) {
43                      int length = childNodes.getLength();
44                      for (int loopIndex = 0; loopIndex < length ; loopIndex++){
45                              traverseTree(childNodes.item(loopIndex),out);
46                      }
47              }
48              break;
49      }
50      String data = node.getNodeValue().trim();
51
52              if((data.indexOf("\n")  <0) && (data.length() > 0)) {
53                      out.println("<td>"+ data+"</td>");
54              }
55      }
56
57      %>
58
59      </body>
60      </html>
```

Archivo domstocks.jsp

Si solicitamos domstocks.jsp desde un navegador, veríamos algo similar a la figura 6.6 anterior.

Transformar XML

Como que los navegadores están llegando a estar disponibles en más dispositivos, la habilidad de

transformar datos de Internet en múltiples formatos, como XML, HTML, WML, o XHTML, se está convirtiendo cada vez en más importante. El XSLT se puede utilizar para transformar XML en cualquier formato deseado. Un motor de transformación o un procesador tomaría un documento XML como entrada y utilizaría la hoja de estilo de transformación XSL para crear un nuevo formato de documento, según lo mostrado en la figura 6.8.

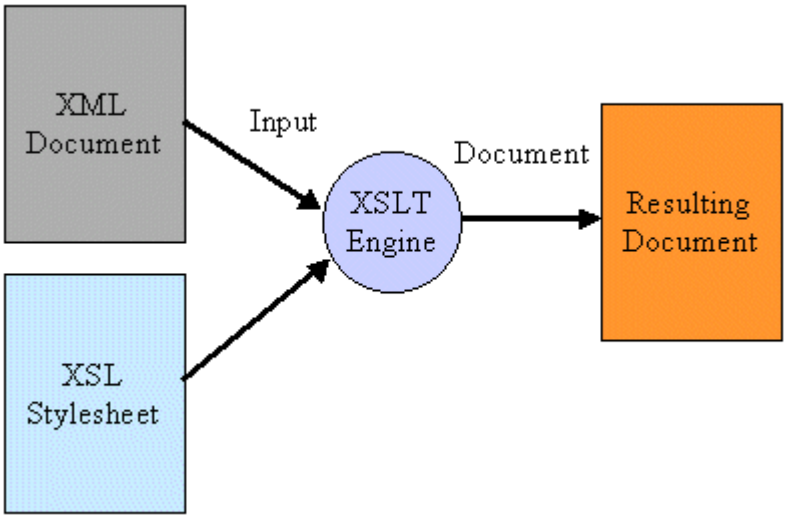


Figura 6.8: Usar un motor XSLT

El ejemplo siguiente, xml2html, muestra cómo transformar el documento XML, stocks1.xml, en un documento HTML, stocks1.html. Como podemos ver, una vez que tengamos el documento XML y la hoja de estilo XSL a aplicar, toda la transformación se hace en tres líneas.

```
1       import javax.xml.transform.*;
2       import javax.xml.transform.stream.*;
3       import java.io.*;
4
5       public class xml2html {
6               public static void main(String[] args) throws TransformerException,
                        TransformerConfigurationException, FileNotFoundException, IOException{
7                       TransformerFactory tFactory = TransformerFactory.newInstance();
8                       Transformer transformer = tFactory.newTransformer(new StreamSource("stocks.xsl"));
9                       transformer.transform(new StreamSource(args[0]), new StreamResult(new
                        FileOutputStream(args[1])));
10                      System.out.println("** The output is written in "+ args[1]+" **");
11              }
12      }
```

Archivo xml2html.java

Para ejecutar este ejemplo, usamos el comando:

C:> java xml2html stocks.xml stocks.html

Stocks1.xml es el archivo de entrada, y será transformado en stocks1.html basándose en la hoja de estilos stocks1.xsl. Aquí estamos utilizando la hoja de estilo anterior, pero hemos agregado un par de nuevas líneas para especificar el método de salida (HTML en este caso) según lo mostrado en el ejemplo siguiente.

```
1       <?xml version="1.0"?>
2       <xsl:stylesheet xmlns:xsl= "http://www.w3.org/1999/XSL/Transform" version= "1.0">
3
4       <xsl:output method="html" indent="yes"/>
5
6       <xsl:template match="/">
7       <html>
8       <body>
9       <xsl:apply-templates/>
10      </body>
11      </html>
12      </xsl:template>
13
14      <xsl:template match="portfolio">
15      <table border="2" width="50%">
16      <xsl:for-each select="stock">
17      <tr>
18      <td>
19      <i><xsl:value-of select= "symbol"/></i>
20      </td>
21      <td>
22      <xsl:value-of select="name"/>
23      </td>
24      <td>
25      <xsl:value-of select="price"/>
26      </td>
27
28      </tr>
29      </xsl:for-each>
30      </table>
31      </xsl:template>
32      </xsl:stylesheet>
```

Archivo stocks2.xml

Se generarán las etiquetas HTML como se muestra en la Figura 6.9:

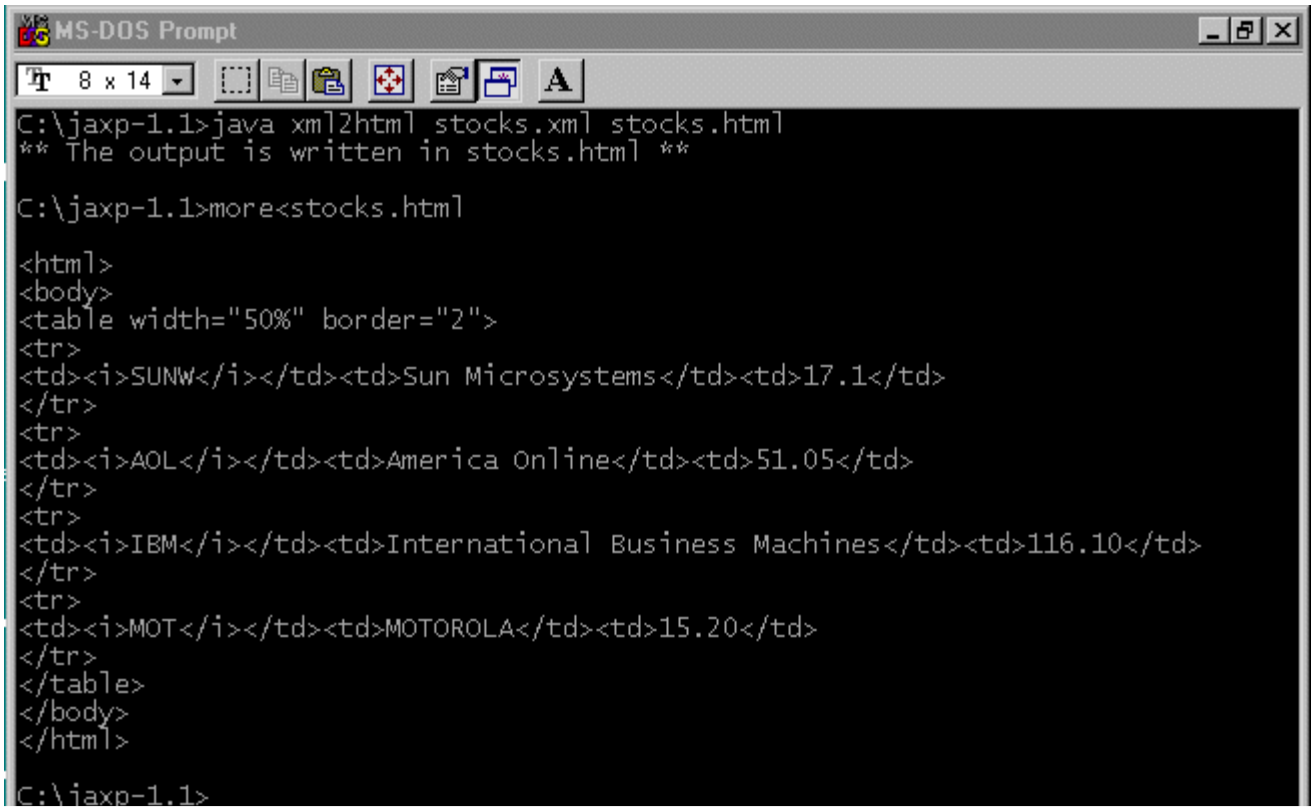


Figura 6.9: Transformación xml2html

Los ejemplos vistos en el presente capítulo se encuentran disponibles en D:\CapituloVI\

CASO DE USO SIBAINES

SIBAINES maneja la estadística a nivel estatal de los colegios pertenecientes al CECyTEM, por tratarse de información confidencial, el acceso a los datos no puede ser llevado a cabo por cualquier persona, debido a esto, se creó en la Base de datos una tabla de usuarios que contiene la siguiente estructura:

Tabla: Usuarios

Campos:

- nusuario: nombre de usuario (clave del plantel)
- password: contraseña del usuario
- nombre: Nombre propio del usuario

En la tabla anterior se almacenan los datos de las personas que tienen acceso al sistema, un ejemplo de datos contenidos en ella sería:

nusuario	password	nombre
16ETC0001G	1234567	Roberto Luna
16ETC0002B	Abcdefg	David Ornelas

Tabla C.1 Estructura de la tabla usuarios, SIBAINES

En la tabla anterior se aprecia que los nombres de usuario son las claves de los planteles o centros de trabajo, esto se hace por dos razones, la primera es que la clave del plantel es la llave primaria de muchas tablas de la BD por lo tanto se debe almacenar su valor en una variable de sesión para utilizarla cuando sea en lugar de estrla requiriendo al usuario a cada rato, la segunda es para validar la autenticación del usuario como se describirá más adelante.

Entonces, cuando un usuario hace click en alguna de las opciones que modifican los datos del sistema (CAPTURA, CONSULTA o REPORTE), aparece una pantalla como la siguiente:

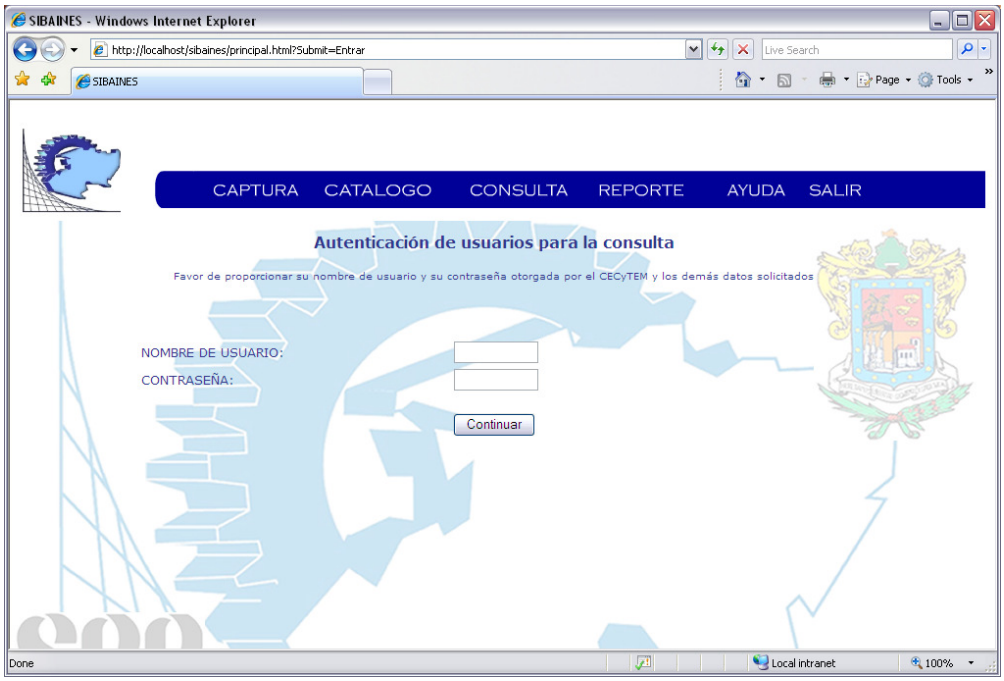


Figura C.1 Pantalla de autenticación, SIBAINES

Que no es más que un formulario con un campo de texto y un campo de contraseña, quitando los estilos de texto y las fuentes el código sería el siguiente:

```
1       <html>
2
3       <head>
4       </head>
5
6       <body>
7       <form name="form" method="post" action="valida_captura_inicial.php">
8         Nombre de Usuario: <input type="text" id="usuario" name="usuario"><br>
9         Contrase&ntilde;a: <input type="password" id="contrasena" name="contrasena">
10      <input type="submit" name="Submit" value="Continuar">
11      </form>
12      </body>
13
14      </html>
```

Archivo captura_inicial.php, SIBAINES

Al hacer Click en el boton “Continuar”, los datos del formulario serán enviados a la página “calida_captura_inicial.php” que se encarga de crear la sesión del usuario si el nombre de usuario y la contraseña son correctos, sino manda el correspondiente mensaje de error y regresa a la página de petición de nombre de usuario y contraseña.

```
1       <?
2       session_start();            //se inicializa la sesion antes de cualquier codigo HTML
3
4       include("config.php") ;  //incluimos el config.php que contiene la conexion a la BD
5
```

```
6       $con = mysql_query("SELECT nombre FROM usuarios WHERE nusuario='$usuario' AND password= '$contrasena'") ;
7
8       if(mysql_num_rows($con) > 0) {
9           $_SESSION["autenticado"]=$usuario;//si nombre de usuario y contraseña son correctos se crea la variable
10                                            //de sesion y se redirecciona a la siguiente página del sistema
11          header("Location: inicio_captura.php");
12          exit;
13         }
14      else {
15          echo " <script languaje=\"javascript\"> alert('Nombre de usuario o contraseña incorrectos'); </script> ";
16          echo " <script languaje=\"javascript\"> location.href = \"javascript:history.back()\"; </script> ";
17         }
18
19      }
20      ?>
```

Archivo valida_captura_inicial.php, SIBAINES

Con el código anterior nos sercioramos de que el usuario esté registrado en la BD del sistema, pero si el usuario no ingresó a la página de autenticación anterior y en su dirección del navegador coloca la dirección “inicio_captura.php”, tendrá acceso a dicha página evitando la autenticación de usuarios, por tal motivo se crea la variable de sesión llamada **autenticado** a la cual le asignamos el valor del nombre de usuario y ahora debemos hacer que en cada página del sistema se verifique dicha variable de sesión y si su valor es vacio, significa que el usuario no se autenticó de manera correcta.

Por tratarse de datos estadísticos, la información capturada debe ser de tipo numérico salvo contadas excepciones en las cuales se admiten cadenas (por ejemplo el nombre del colegio, su dirección, nombre del director, etc.). Pongamos como ejemplo la siguiente pantalla del sistema que sería el archivo “captura_inicial.php” al cual fuimos redireccionados después de la autenticación anterior:

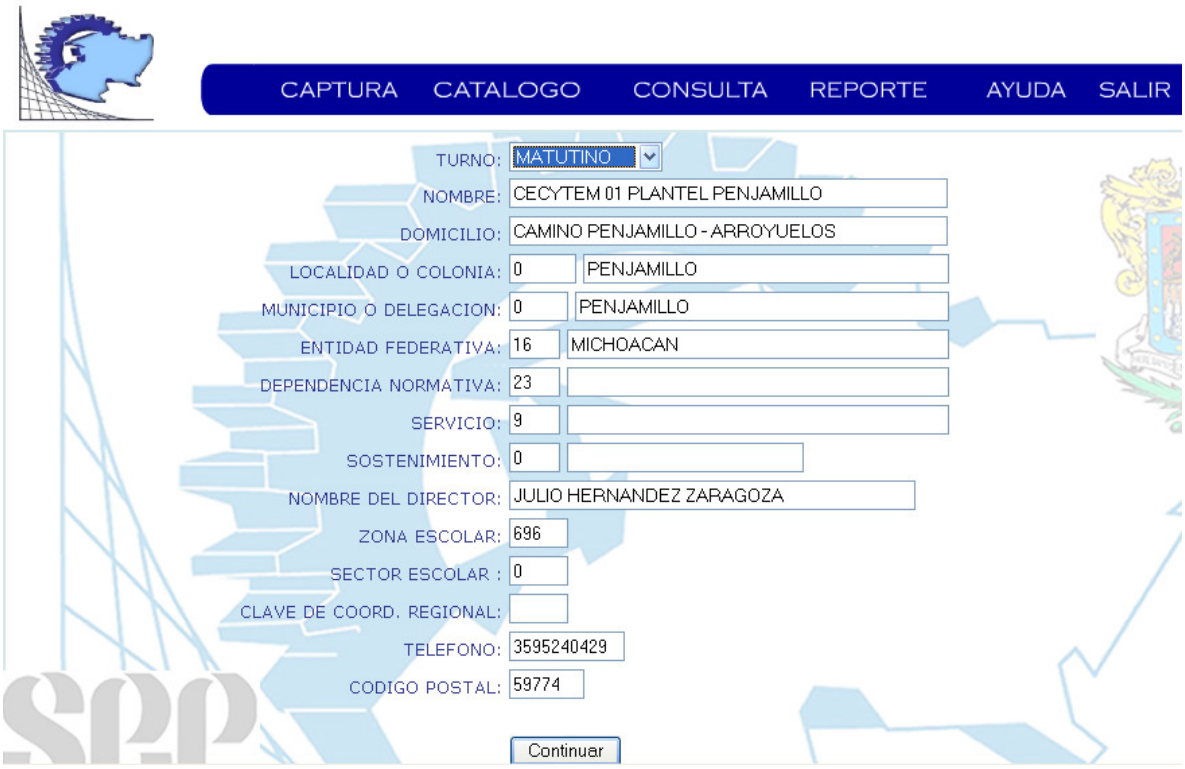


Figura C.2 Pantalla de Inicio de Captura, SIBAINES

En la pantalla anterior se puede apreciar que varios campos requieren información numérica y otro tanto admiten cadenas de texto; desde el campo **Localidad o colonia** hasta el campo **Sostenimiento** existen "parejas" de campos de las cuales el campo de lado izquierdo debe ser de tipo numérico al igual que los campos **Zona escolar**, **Sector Escolar**, **Clave de Coord. Regional** y **Código Postal**, este último inclusive tiene la limitante de que forzosamente debe tener 5 dígitos, ni más ni menos.

Al momento de hacer Click sobre el botón de enviar se mandará llamar la función de "valida_formulario" a la cual le enviamos todo el formulario con ayuda del objeto **this** para hacer las validaciones correspondientes. A continuación se muestra el script de la página de la imagen anterior incluyendo el código Javascript que realiza las validaciones antes mencionadas; para simplificar y acortar un poco el código del ejemplo se han eliminado los estilos de párrafo y de texto así como algunos campos cuya validación se realiza de igual manera que otros.

```
1   <?
2   session_start();                    //se inicializa la sesion antes de cualquier codigo HTML
3   if($_SESSION['autenticado']==""){
4       echo " <script languaje=\"javascript\"> alert('Usuario no autenticado'); </script> ";
5       echo " <script languaje=\"javascript\"> location.href = \"javascript:history.back()\"; </script> ";
6       exit;
7   }
8   include("config.php") ; //incluimos el config.php que contiene la conexion a la BD
9
10  $consulta = mysql_query("SELECT * FROM datos_iniciales WHERE id_plantel='$_SESSION[autenticado]'") ;
11  $datos = mysql_fetch_array($consulta);
12  ?>
```

```
13	<html>
14	<head>
15	<title>SIBAINES Datos de Identificación del Plantel</title>
16	<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
17	<script>
18	function valida_formulario(form){
19		//validamos que la clave no sea vacia
20		if(form.clave.value.length == 0){
21			alert('El campo Clave no puede ir vacio');
22				return false;
23		}
24
25		//validamos la lista desplegable paracomprobar que se haya seleccionado un turno
26		if(form.turno.value == ''){
27			alert('Debes seleccionar un turno');
28			return false;
29		}
30
31		//validamos la longitud de los campos
32		if(form.nombre.value.length > 40){
33			alert('El campo Nombre no debe tener mas de 40 caracteres');
34			return false;
35		}
36
37		... //mas validaciones similares
38
39		//validamos los campos numericos
40		if(isNaN(form.nlocalidad.value) && form.nlocalidad.value.length > 0){
41			alert('El campo No. de Localidad no puede tener letras');
42			return false;
43		}
44		if(isNaN(form.cp.value) && form.cp.value.length > 0){
45			alert('El campo Codigo Postal no puede tener letras');
46			return false;
47		}
48		else if(form.cp.value.length > 0 && form.cp.value.length != 5){
49			alert('El campo Codigo Postal debe ser de 5 digitos');
50			return false;
51		}
52
53		...  //mas validaciones similares
54
55	}
56	</script>
57	</head>
58	<body>
59	<form name="form" id="form" onSubmit="return valida_formulario(this)" action="bd_datos_iniciales.php">
60	<div align="center">Datos de Identificaci&oacute;n del Centro de Trabajo </div>
61	<table align="center">
62	<tr>
63	  <td>CLAVE DEL CENTRO DE TRABAJO:</td>
64	  <td><input name="clave" type="text" value="<?echo $_SESSION['autenticado'];?>"></td>
65	</tr>
66	<tr>
67	  <td>TURNO:</td>
68	  <td>
69	    <select name="turno">
70	      <option value="">--Selecciona Turno--</option>
71	      <option value="1">MATUTINO</option>
72	      <option value="2">VESPERTINO</option>
73	    </select>
74	  </td>
75	</tr>
76	<tr>
77	  <td>NOMBRE:</td>
78	  <td><input type="text" name="nombre" size="50" value="<?echo $datos['nombre'];?>"></td>
79	</tr>
80	 <tr>
81	  <td>DOMICILIO:</td>
```

```
82          <td><input type="text" name="domicilio" size="50" value="<?echo $datos['domicilio'];?>"></td>
83        </tr>
84        <tr>
85          <td>LOCALIDAD O COLONIA:</td>
86          <td>
87            <input type="text" name="nlocalidad" size="5" value="<?echo $datos['nlocalidad'];?>">
88            <input type="text" name="localidad" size="40" value="<?echo $datos['localidad'];?>">
89          </td>
90        </tr>
91        <tr>
92          <td>MUNICIPIO O DELEGACION:</td>
93          <td>
94            <input type="text" name="nmunicipio" size="5" value="<?echo $datos['nmunicpio'];?>">
95            <input type="text" name="municipio" size="40" value="<?echo $datos['municipio'];?>">
96          </td>
97        </tr>
98         <tr>
99          <td>NOMBRE DEL DIRECTOR:</td>
100         <td><input type="text" name="director" size="46" value="<?echo $datos['director'];?>"></td>
101       </tr>
102       <tr>
103         <td>ZONA ESCOLAR:</td>
104         <td><input type="text" name="zona" size="5" value="<?echo $datos['zona'];?>"></td>
105       </tr>
106       <tr>
107         <td>TELEFONO:</td>
108         <td><input type="text" name="telefono" size="10" value="<?echo $datos['telefono'];?>"></td>
109       </tr>
110       <tr>
111         <td>CODIGO POSTAL:</td>
112         <td><input type="text" name="cp" size="5" value="<?echo $datos['cp'];?>"></td>
113       </tr>
114       <tr>
115         <td> </td>
116         <td><input name="submit" type="submit" value="Enviar"></td>
117       </tr>
118       </table>
119       </form>
120       </body>
121       </html>
```

Archivo captura_inicial.php, SIBAINES

En el ejemplo anterior, se valida la variable de sesión que contiene la clave del plantel, si la variable de sesión está vacía significa que el usuario no se autenticó y por lo tanto se desplegaría un mensaje de error, caso contrario, como ya se conoce la clave del centro de trabajo porque está guardada en esa variable de sesión, se hace una consulta a la BD para obtener los datos del plantel, para que, en caso de haber alguna modificación en alguno de ellos, el capturista los actualice, si se analiza el código HTML anterior, se dará cuenta de que se hace uso del atributo **value** en los campos de texto, con este atributo hacemos que cada campo de texto tenga un valor por default en lugar de estar vacío, y, como lo que queremos que aparezca por default son los datos que acabamos de obtener de la BD con PHP, simplemente introducimos código PHP dentro de la etiqueta value que imprime el valor del campo indicado.

Una vez realizadas las modificaciones, el usuario debe presionarel botón de “Enviar” el cual ejecuta la función de validación de los campos y si todas las validaciones fueron correctas, el formulario será enviado a la página “bd_datos_iniciales.php”, el cual correspondería al siguiente código.

```
1         <?
2         session_start();                  //se inicializa la sesion antes de cualquier codigo HTML
3         if($_SESSION['autenticado']==""){
```

```
4               echo " <script languaje=\"javascript\"> alert('Usuario no autenticado'); </script> ";
5               echo " <script languaje=\"javascript\"> location.href = \"javascript:history.back()\"; </script> ";
6                       exit;
7       }
8       include("config.php") ; //incluimos el config.php que contiene la conexion a la BD
9
10      $consulta = mysql_query("UPDATE datos_iniciales SET nombre = '$nombre', domicilio = '$domicilio', localidad =
'$localidad', nlocalidad = '$nlocalidad'..... WHERE id_plantel='$_SESSION[autenticado]'") ;
11
12      header("Location: alumnos_por_carrera.php");
13      ?>
```

Archivo bd_datos_iniciales.php, SIBAINES

En el archivo anterior se actualizan los datos de la BD para que, en caso de que alguno de ellos se haya modificado, quede el registro actual en el sistema, los puntos suspensivos de la línea 10 significan que la instrucción SQL continúa asi para todos los demás campos del formulario. Una vez concluida la actualización de la BD de continúa con las demás pantallas del sistema que funcionan de manera similar. Se puede introducir el manejo de errores del SMBD visto en el Capítulo IV para asegurar las operaciones en la BD y no perder información, además de que se pueden incluir las validaciones de los campos de los formularios en PHP para segurar que la información introducida haya sido correcta en caso de que el navegador tenga bloqueada la ejecución del código Javascript, asi mismo, en lugar de pedir el nombre de usuario y la contraseña cada vez que el usuario entre al sistema se puede almacenar una cookie que contenga estos datos para evitar pedírselos nuevamente (aunque esto comprometería la seguridad del sistema a menos que sea un usuario muy confiable se recomienda el uso de cookies para este uso).

Cuando un usuario presione la opción SALIR del menú principal, se debe destruir la variable de sesión para que ningún otro usuario tenga acceso al sistema a menos que se autentique correctamente.

```
1       <?
2       session_start();                //se inicializa la sesion antes de cualquier codigo HTML
3       $_SESSION['autenticado']="";    //se elimina la variable de sesion
4       ?>
```

Archivo bd_datos_iniciales.php, SIBAINES

CONCLUSIONES Y RECOMENDACIONES

En la red existen usuarios que atentan contra la integridad de los sistemas con la finalidad de obtener algún dato relevante para poder lucrar con la información obtenida. El robo de la información que circula en la red es inevitable, pero podemos hacer uso de varias herramientas fáciles de configurar o implementar que retrasen o dificulten la lectura de los datos robados y así evitar que sean útiles una vez robados. Tal y como se vio a lo largo del presente trabajo, es necesaria la protección de los recursos de nuestros sistemas en todo momento tanto del lado del servidor como del lado del cliente, así como también es necesaria la combinación de varios elementos de resguardo para los sistemas web y la información que manejan.

La forma en la cual se empleen los métodos de validación de datos y protección de los sistemas debe ser analizada detenidamente porque a pesar de utilizar una excelente herramienta, método o algoritmo de cifrado o protección puede que no sirva de nada si el momento o el lugar de aplicación no es bueno o no es seguro, esto se recomienda debido a que en SIBAINES los métodos de cifrado se realizaban inicialmente en PHP pero este lenguaje se ejecuta en el servidor por lo que se enviaban los datos sin cifrar por la red y en ese momento no se contaba con SSL para proteger la comunicación, esto hacia que el método de cifrado fuera en cierto punto inútil ya que a final de cuentas, en determinado momento, los datos viajaban en texto plano sin protección alguna.

SIBAINES fue una gran experiencia en la forma de ver los sistemas web ya que generalmente se piensa que éste tipo de sistemas es sencillo de programar debido a que su funcionalidad básica es realizar una consulta a una BD y mostrar resultados, lo que casi nunca se toma en cuenta es el gran tamaño que pueden llegar a tener éste tipo de sistemas o la relevancia y confidencialidad de los datos que se manejan, no es por nada que los bancos cuya información viaja en la red tenga claves de cifrado de 512 o 1024 bits, por esto es importante pensar en varias formas de protección de los datos y pensar en la forma en la cual dicha protección funciona, así mismo es muy importante averiguar las herramientas nativas que tienen los lenguajes de programación o interpretes en los cuales realice sus sistemas ya que pueden tener modulos de seguridad que van a hacer que ahorre tiempo de programación y van a generar el mismo resultado.

La replicación de los datos en los servidores web debe ser contemplada por las empresas que tienen aplicaciones de este tipo, los costos de los servidores se han estado abaratando día con día, en la actualidad el costo promedio de un servidor es de $800 pesos por año, esto es un costo que cualquier empresa que maneje este tipo de sistemas y quiera proteger adecuadamente su información puede pagar para así en caso de un fallo crítico en un servidor, los datos se conserven en el otro.

APÉNDICE

Configuración de un Servidor con AppServ en Windows XP

Para instalar AppServ se debe hacer lo siguiente:

- En primer lugar descargar el paquete de la página del **AppServ Open Project:** http://www.appservnetwork.com/, la versión más reciente es la AppServ 2.5.8 (apache 2.2.4, MySQL 5.0.27, PHP 5.2.1 y phpMyAdmin-2.9.2).
- Una vez descargado el ejecutable (disponible también en el disco de anexos D:\AppServ) procedemos a ejecutar la instalación, recibiendo un mensaje de bienvenida y hacemos Click en Next

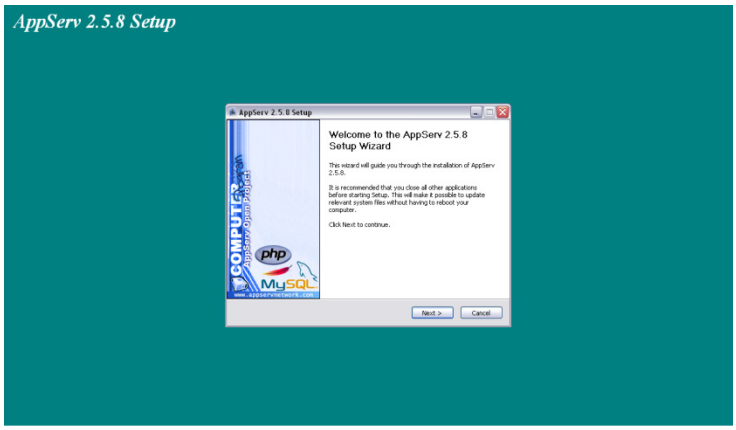


Figura A.1 Pantalla Inicio de Instalación de AppServ

- **Carpeta de Instalación**: en esta pantalla nos solicita la carpeta en el que queremos instalar la aplicación, por defecto nos marca "c:/appserv", lo cambiamos si queremos y pulsamos Next.

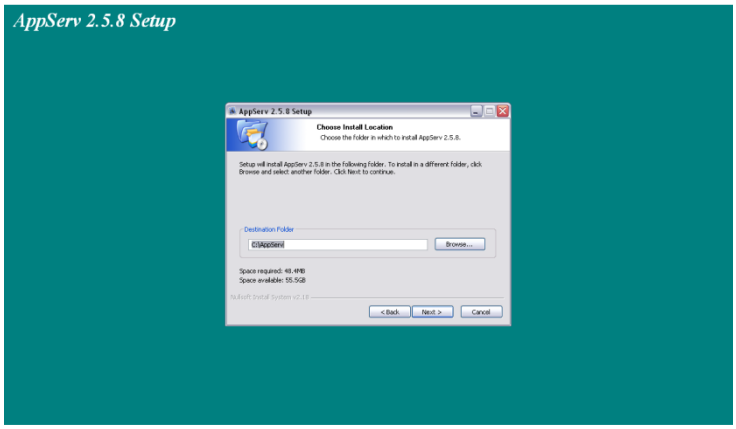


Figura A.2 Carpeta de Instalación de AppServ

- **Componentes que queremos instalar**: Seleccionamos Todos (Apache HTTP Server, MySQL Database, phpHyperText Preprocessor y phpMyAdmin).

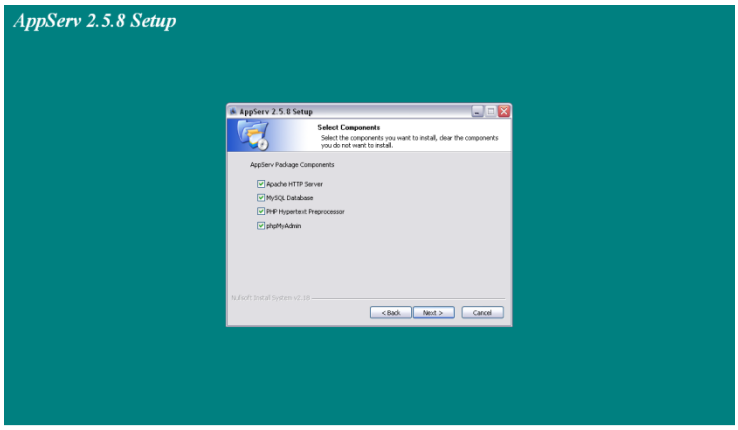


Figura A.3 Selección de Componentes de AppServ

- **Server Information:** La casilla Server Name debe tener "localhost" y en la casilla Administrator E-mail Address introducimos una cuenta de correo que será la del administrador. El campo "HTTP Port" lo dejamos como viene por defecto (80), salvo que queramos que el servidor atienda les peticiones en otro puerto. Pulsamos en Next

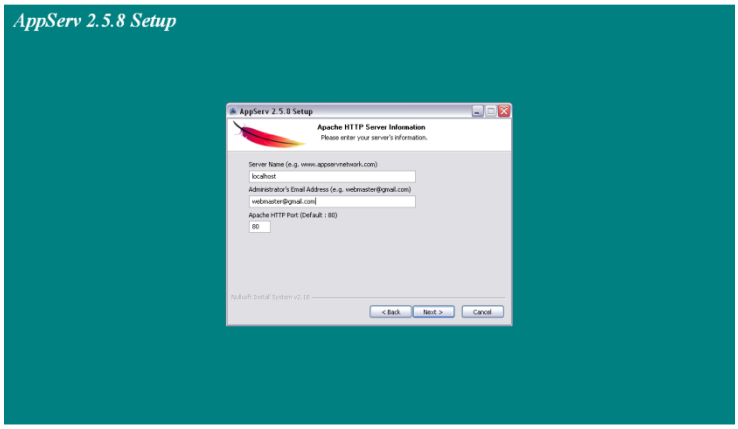


Figura A.4 Información del servidor Apache

- **MySql Server Configuration:** Este es uno de los apartados más importantes ya que empezamos a instalar MySQL. Aquí se solicita la contraseña (y confirmación de la misma) del root o administrador del sistema y no puede dejarse vacía (en nuestro caso la contraseña también será "root" pero posteriormente se puede cambiar con phpMyAdmin). El apartado Charset lo dejamos tal como viene por defecto. Algo muy importante es la casilla que dice Enable InnoDB la cual deberá de estar **seleccionada** ya que si nuestro manejador de bases de datos no soporta tablas tipo InnoDB las transacciones no podrán ser realizadas. Pulsamos en Next

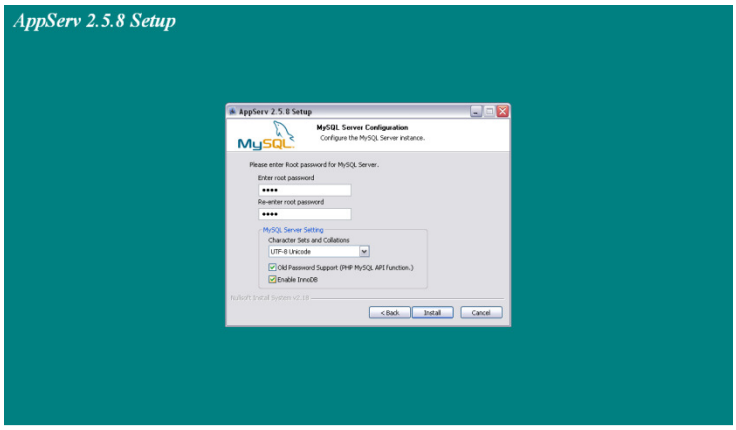


Figura A.5 Configuración de MySQL

- **Progreso de la instalación**: Empieza realmente la instalación mostrando una barra de progreso hasta que aparece una pantalla que nos avisa de que ha finalizado la misma. Finalmente pulsamos en Close.

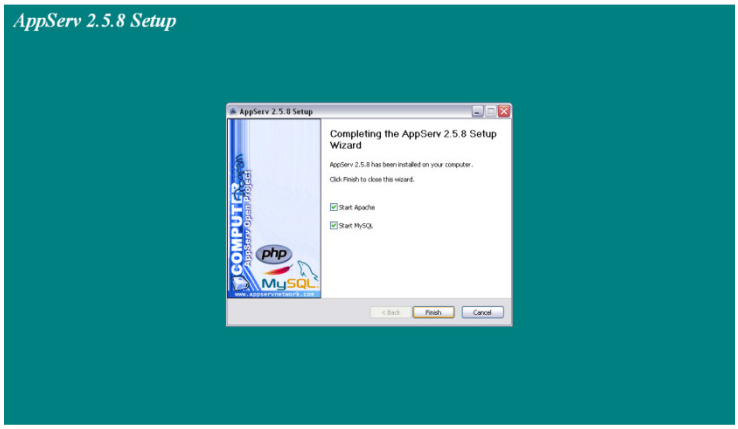


Figura A.6 Finalizando Instalación de AppServ

- **Comprobación de la instalación:** Si la instalación se ha realizado correctamente, al poner en nuestro navegador: http://localhost nos debe aparecer la pantalla siguiente:

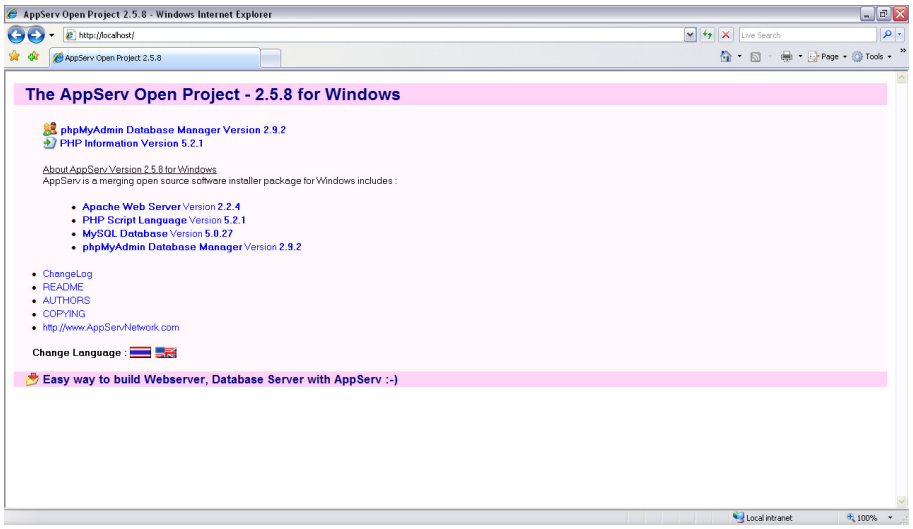


Figura A.7 Comprobando Instalación de AppServ

- Los Scripts del sistema deben ser almacenados en **$_RUTA_DE_INSTALACION\AppServ\www**, donde $_RUTA_DE_INSTALACION es la ruta en la cual se haya realizado la instalación del AppServ, la ruta por defecto es C:\.

Si esta pantalla no aparece puede que bien Apache o MySQL no están funcionado bien o no fueron iniciados los servicios, entonces buscamos en nuestro menú de programas a **AppServ** y después el submenú **Control Server by Service** desde el cual podemos ver el estado de los servicios del AppServ así como detenerlos, iniciarlos o reiniciarlos.

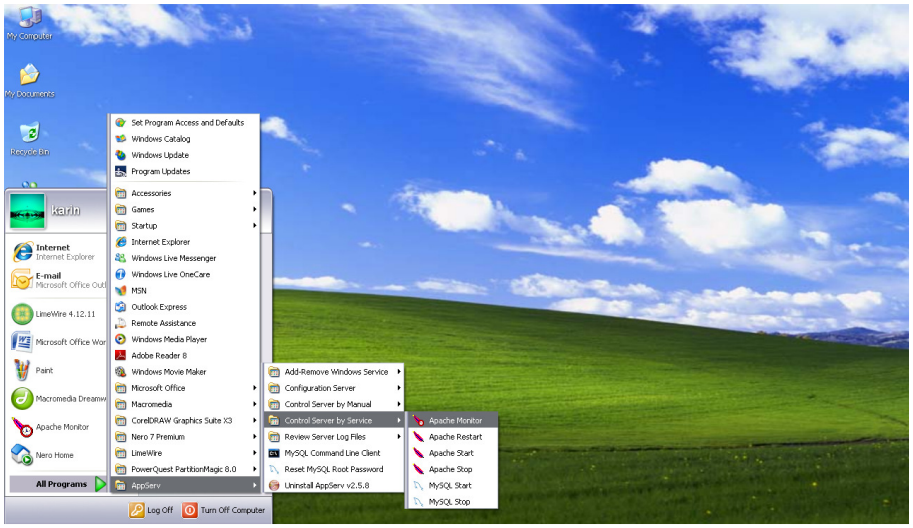


Figura A.8 Menú programas de AppServ

También se debe verificar que el firewall no tenga bloqueados los servicios del AppServ, para ver esto se puede iniciar el administrador de tareas presionando **CTRL + ALT + SUPR** y en el listado de la pestaña de **Procesos** deben aparecer **httpd.exe** y **mysqld-nt.exe**.

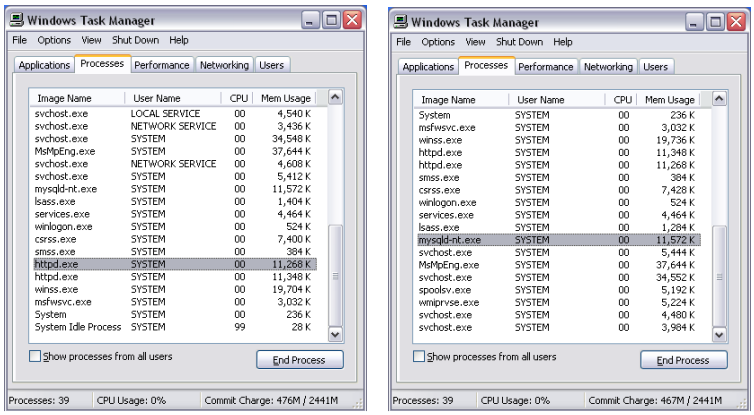


Figura A.9 y A.10 Administrador de tareas, httpd.exe y mysqld-nt.exe

Si los procesos se encuentran en ejecución y aun no puede ver la pantalla al colocar http://localhost en su navegador deberá de instalar nuevamente el AppServ.

Configuración de un Servidor con Tomcat en Windows XP

Para instalar Tomcat se debe hacer lo siguiente:

- En primer lugar descargar el paquete de la página del **AppServ Open Project:** http://www.tomcat.apache.org/, la versión más reciente es la 6.0.10.
- Tener instalado alguna versión del JDK.
- Una vez descargado el ejecutable (disponible también en el disco de anexos D:\Apache Tomcat) procedemos a ejecutar el archivo de instalación, recibiendo un mensaje de bienvenida y hacemos Click en Next

Figura A.17 Pantalla Inicio de Instalación de Tomcat

- **Componentes que queremos instalar**: Seleccionamos el tipo de instalación completa (FULL) para seleccionar de manera automática todos los componentes y presionamos Next.

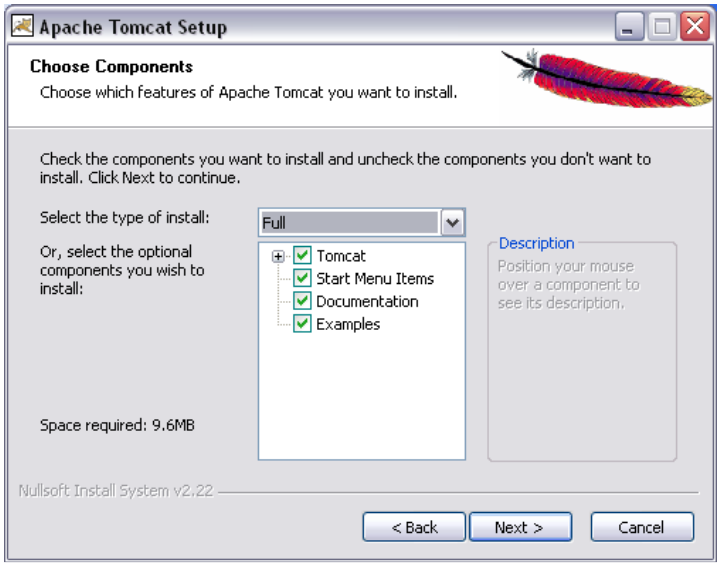


Figura A.18 Selección de Componentes de Tomcat

- **Carpeta de Instalación**: en esta pantalla se solicita la carpeta en la que queremos instalar la aplicación, por defecto nos marca "C:\Program Files\Apache Software Foundation\Tomcat 6.0", lo cambiamos si queremos y pulsamos Next.

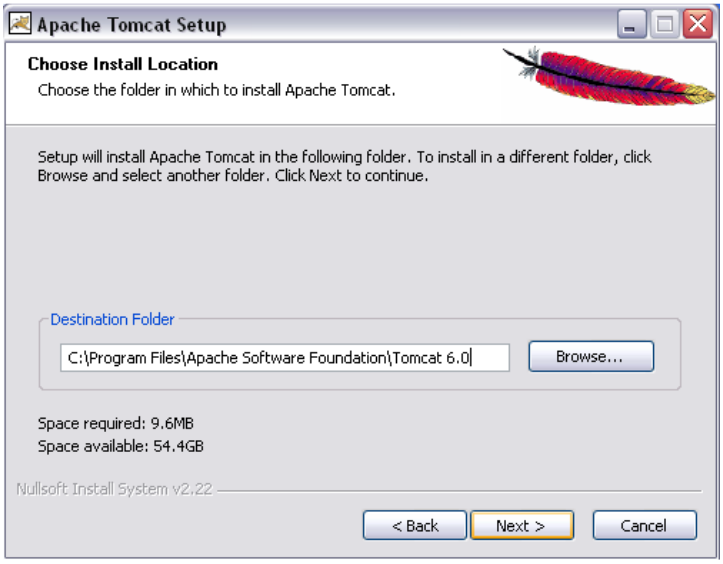


Figura A.19 Carpeta de Instalación de Tomcat

- **Configuration Options:** Aquí se solicita la contraseña del administrador del sistema y no puede dejarse vacía (en nuestro caso la contraseña al igual que el nombre del usuario administrador será "admin"). El campo "HTTP/1.1 Connector Port" lo dejamos como viene por defecto (8080), salvo que queramos que el servidor atienda les peticiones en otro puerto Pulsamos en Next.

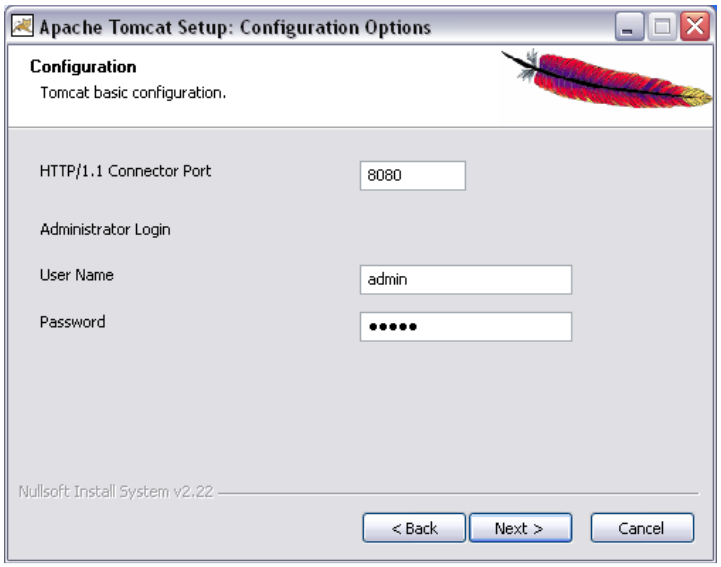


Figura A.20 Configuración de Tomcat

- **Java Virtual Machine path selection:** Aquí se solicita la ubicación de la carpeta raíz de nuestra instalación del JDK (que debimos haber instalado previamente), en caso de que solo tengamos una ésta ya aparece por defecto y solo Pulsamos en Next. Si tenemos más de una y no aparece la deseada basta con buscar la ubicación de la carpeta raíz de la instalación que queramos y hacer Click en next.

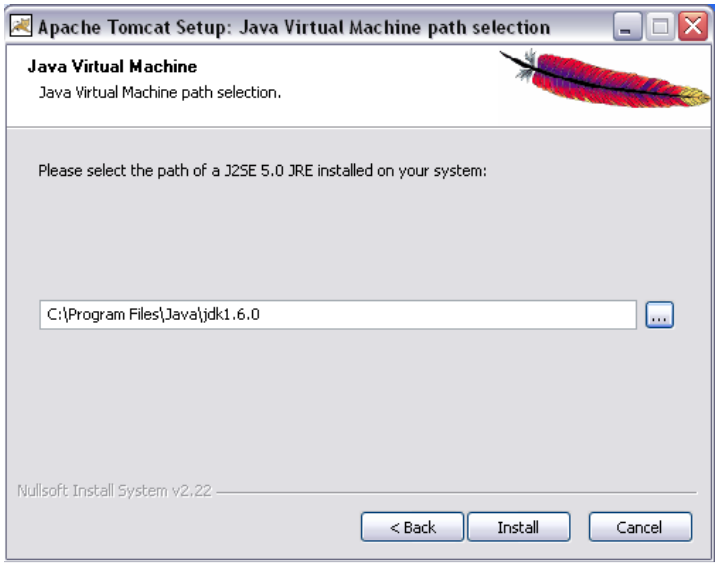


Figura A.21 Configuración Máquina Virtual de Java

- **Progreso de la instalación**: Empieza realmente la instalación mostrandonos una barra de progreso hasta que aparece una pantalla que nos avisa de que ha finalizado la misma, debiendo dejar marcadas las opciones que vienen por defecto. Finalmente pulsamos en Close.

Figura A.22 Finalizando Instalación de Tomcat

- **Comprobación de la instalación:** Si la instalación se ha realizado correctamente, al poner en nuestro navegador: http://localhost:8080 (se agrega el ":8080" porque la instalación la hicimos en dicho puerto, si hubiera sido en el puerto 80, bastaría con poner http://localhost) nos debe la pantalla de bienvenida de Tomcat.

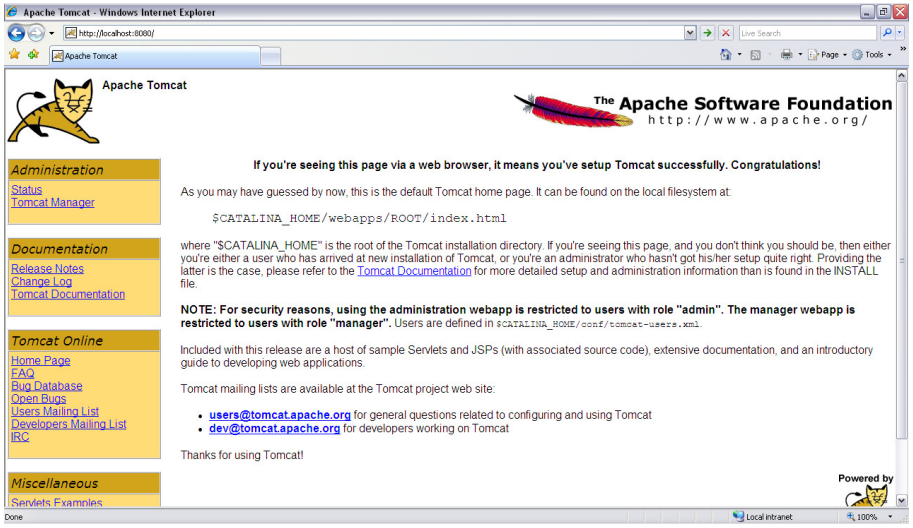


Figura A.23 Comprobando Instalación de Tomcat

- Los Scripts del sistema deben ser almacenados en **$_RUTA_DE_INSTALACION\Apache Software Foundation\Tomcat 6.0\webapps\ROOT**, donde $_RUTA_DE_INSTALACION es la ruta en la cual haya realizado la instalación de Tomcat, la ruta por defecto es C:\Archivos de programa\.

Si esta pantalla no aparece puede que el servicio no esté funcionado bien o no haya sido iniciado, entonces buscamos en nuestro menú de programas **Apache Tomcat x.x** (donde "x.x" es la versión de

Tomcat que se haya instalado, en nuestro caso la 6.0) y después el submenú **Monitor Tomcat** desde el cual podemos ver el estado del servicio de Tomcat así como detenerlo, iniciarlo o reiniciarlo.

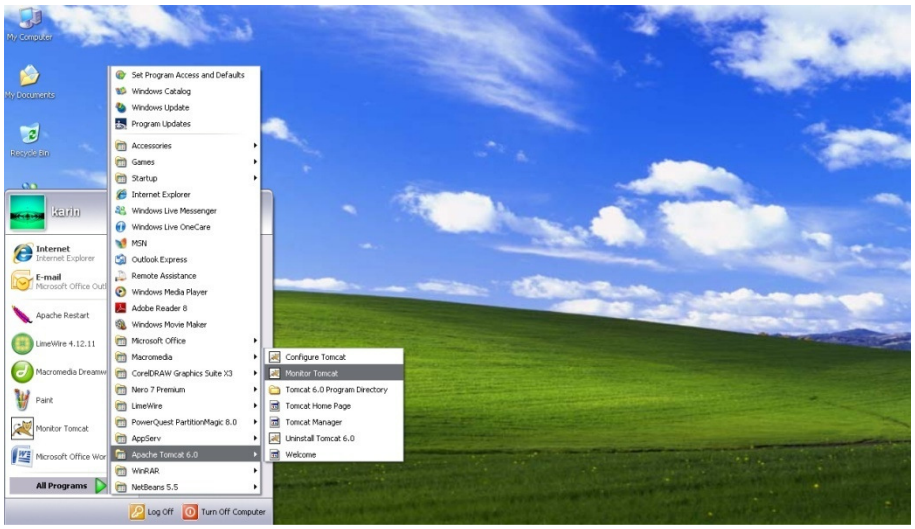


Figura A.24 Menú programas de Tomcat

Si no aparece ninguna ventana al hacer Click en **Monitor Tomcat**, busque el siguiente ícono en la barra de notificaciones (al lado del reloj del sistema) y haga doble Click en él para hacer que aparezca el Monitor del Tomcat.

Figura A.25 Icono del Monitor Tomcat, Barra de notificaciones

También se debe verificar que el firewall no tenga bloqueado el servicio, para ver esto se puede iniciar el administrador de tareas presionando **CTRL + ALT + SUPR** y en el listado de la pestaña de **Procesos** deben aparecer **tomcat6.exe**.

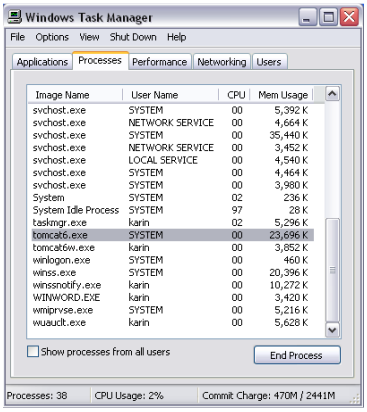


Figura A.26 Administrador de tareas, tomcat.exe

Si el proceso se encuentra en ejecución y aun no puede ver la pantalla al colocar http://localhost:8080 en su navegador deberá de instalar nuevamente el Tomcat.

Configuración de Servicios en Linux

Apache

La **Herramienta de configuración de Apache** permite configurar el archivo de configuración */etc/httpd/conf/httpd.conf* para el servidor Web de Apache.

Se podrán configurar las opciones de Apache tales como hosts virtuales, atributos de registro y número máximo de conexiones a través de la interfaz gráfica.

Sólo se pueden configurar con la **herramienta de configuración de Apache** aquellos módulos que estén incluidos en el paquete de Red Hat Linux.

La **Herramienta de configuración de Apache** requiere el sistema X Window (Es un entorno gráfico, que funciona bajo el modelo cliente/servidor) y el acceso de root. Para iniciar la **Herramienta de configuración de Apache**, use uno de los siguientes métodos:

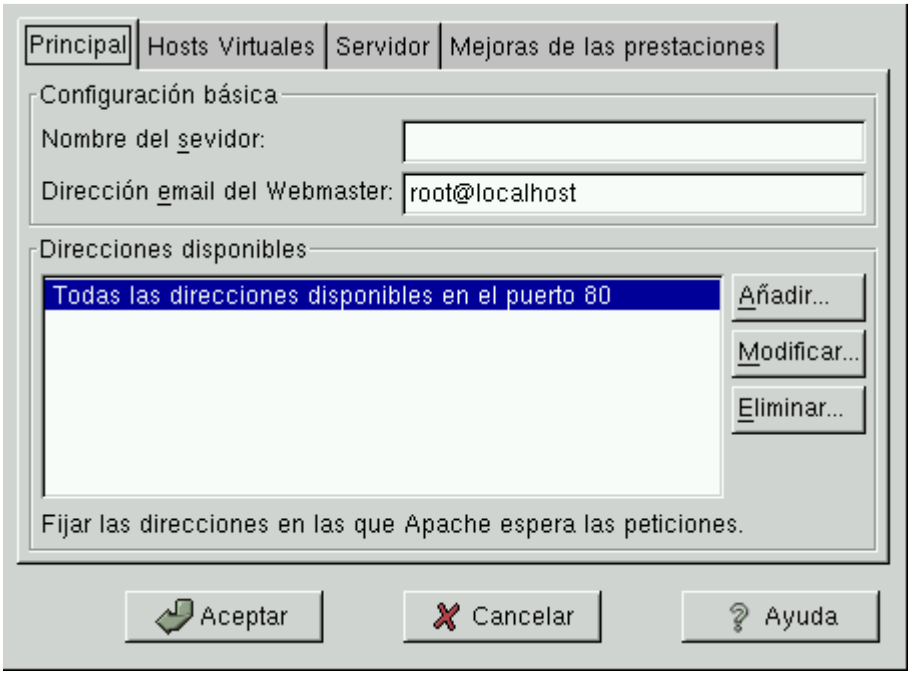
- En el escritorio GNOME,
    - **Menú principal** (en el panel)
    - **Programas**
    - **Sistema**
        - **Configuración de Apache**.
- Tipo de comando apacheconf en el intérprete de comandos de la shell

Los pasos que debe seguir para configurar el servidor Apache con la **herramienta de configuración de Apache** son los siguientes:

1. Configure las posiciones básicas que se encuentran en la pestaña **Principal**.
2. Haga click en **Hosts Virtuales** y configure las opciones predeterminadas.
3. Si desea servir a más de una URL, añada las máquinas virtuales adicionales.
4. Configure las posiciones del servidor que se encuentran en **Servidor**.
5. Configure las conexiones en **Mejoras de las prestaciones**.
6. Copie todos los archivos necesarios a los directorios DocumentRoot y cgi-bin y grabe las posiciones en la **herramienta de configuración de Apache**.

Configuraciones básicas

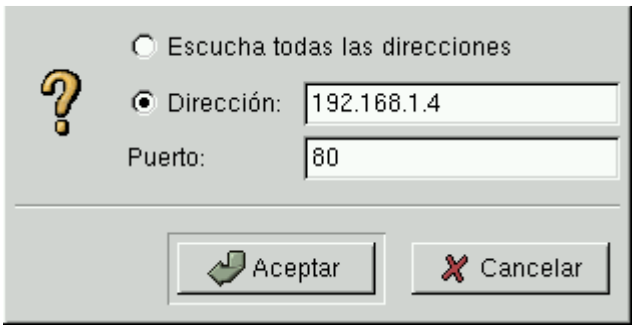
Use ***Principal*** para configurar las posiciones básicas del servidor.



Ingrese el nombre del dominio completo que tenga derecho a usar en **Nombre del servidor**. Esta opción corresponde al directorio servername en httpd.conf. El directorio ServerName establece el nombre de la máquina del servidor de web. Se usa cuando se crean redireccionamientos de URLs. Si usted no introduce el Nombre del servidor, Apache intentará resolverlo desde una dirección IP del sistema. El Nombre del servidor no tiene porqué ser igual al nombre DNS del servidor.

Introduzca la dirección de correo electrónico de la persona que mantiene el servidor web en **Dirección email del Webmaster**. Esta opción corresponde al directorio ServerAdmin en httpd.conf. Si configura la página de errores del servidor para que contenga una dirección de correo electrónico, dicha dirección se usará para que los usuarios puedan expresar el problema que tengan mandando un correo electrónico al administrador del servidor. El valor predeterminado es root@localhost.

Use **Direcciones disponibles** para definir los puertos del servidor Apache. Esta opción corresponde a el *directive listen* en httpd.conf. El valor predeterminado del puerto para Apache es 80, para las comunicaciones Web no-seguras.



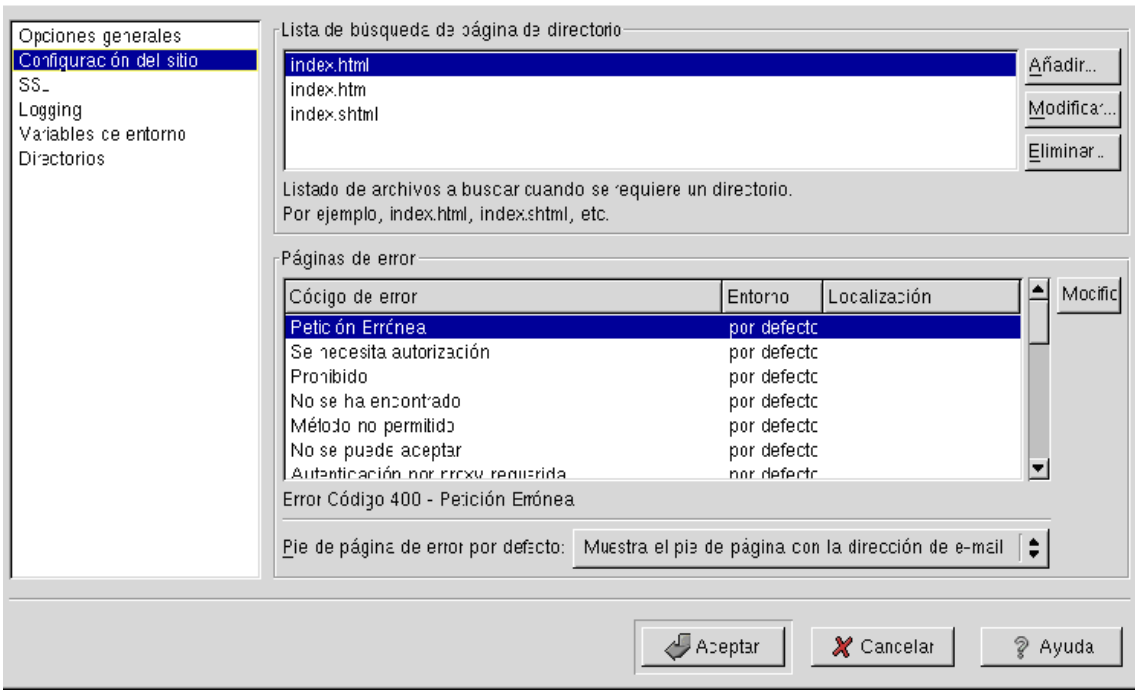
Se puede elegir la opción de 

Escucha todas las direcciones para escuchar todas las direcciones IP del puerto definido o bien especificar la dirección en la que el servidor aceptará las conexiones en el campo **Direcciones** 

. Especifique sólo una dirección IP por número de puerto. Si quiere especificar más de una dirección con el mismo número de puerto, cree una entrada para cada una de ellas. Si esto es posible, utilice una única dirección IP en vez de un nombre de dominio para así evitar que falle la búsqueda del DNS. Si introduce un asterisco (*) en **Direcciones** equivaldrá a elegir la opción **Escucha todas las direcciones**. Haga click en el botón **Modificar** y verá la misma pantalla que si pulsa el botón **Añadir** excepto los campos de la entrada seleccionada. Para borrar una entrada, pulse el botón **Cancelar**.

Configuraciones predeterminadas

Después de haber definido el nombre del servidor, la dirección de correo electrónico del Webmaster y las direcciones disponibles, haga click en **Hosts Virtuales** y en el botón **Modifica las configuraciones por defecto**. Entonces aparecerá la siguiente pantalla



donde debe de configurar las posiciones predeterminadas para su servidor de web. Si añade otra máquina virtual, las posiciones que configure serán para esta máquina. Normalmente, si un directive no está definido en las posiciones de una máquina virtual, se usa el valor predeterminado.

Configuración del sitio

Casi todos los servidores admiten los valores predeterminados de **Lista de búsqueda de página de directorio** y **Páginas de error**.

Las entradas que aparecen en la **Lista de búsqueda de página de directorio** definen la directiva DirectoryIndex, es la página predeterminada que el servidor da a un usuario que pide el índice de un directorio escribiendo la barra inclinada (/) al final del nombre del directorio.

Por ejemplo, cuando un usuario pide la página *http://u.lagos1.cl/index.html*, el servidor busca la página DirectoryIndex si existe o la lista de directorios generada por el servidor. El servidor intentará encontrar uno de los archivos que se encuentran en la lista de la directiva DirectoryIndex y le entregará el primero que encuentre. Si no encuentra ninguno de los archivos y si ese directorio contiene los Índices de las opciones, verá en la pantalla una lista en formato HTML de los subdirectorios y archivos de ese directorio.

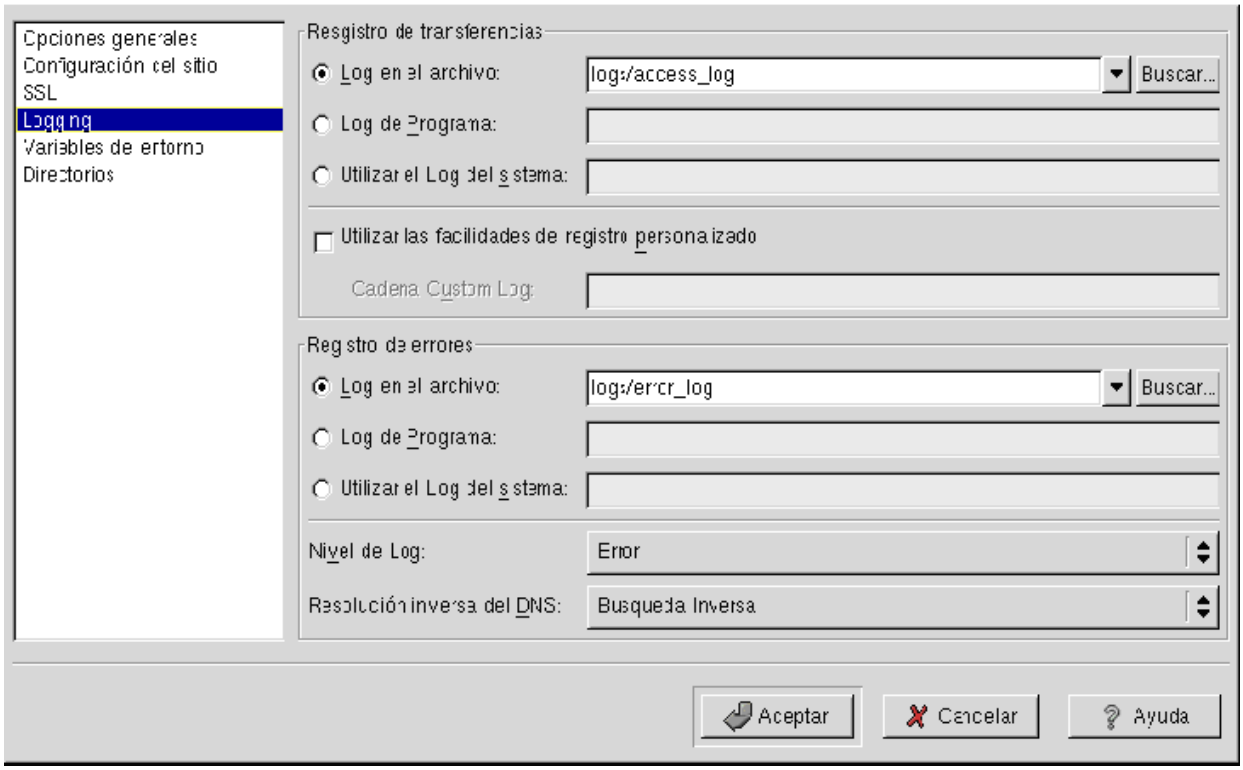
Use la sección **Código de error** para configurar Apache de tal manera que mande al cliente a una URL local o externa si existe algún problema o algún error. Esta opción corresponde a la directiva Error Document. Si ocurriese algún tipo de problema en la conexión al servidor Apache, en la pantalla aparecerá un mensaje de error en la columna **Código de error**. Para anular esta opción seleccione el código de error y haga click en el botón **Modificar**. Elija la opción **Por defecto** para mostrar el mensaje de error. Seleccione **URL** para mandar al cliente a una URL externa e introduzca una URL completa incluyendo http:// en el campo **Localización**. Elija **Archivo** para mandar al cliente a una URL interna e introduzca un archivo en Document Root del servidor de Web. La localización debe de comenzar con la barra (/) y pertenecer al Document Root.

En el menú **Pie de página de error por defecto** escoja una de las siguientes opciones:

- **Muestra el pie de página con la dirección de e-mail** — Esta opción muestra el pie de página predeterminado de Apache en todas las páginas de error junto con las direcciones de correo electrónico del encargado del sitio web especificado por la directiva del ServerAdmin **Muestra el pie de página** — Esta opción le muestra el pie de página predeterminado de Apache en todas las páginas de error.
- **Ningún pie de página** — No muestra el pie de página.

Registro

Por defecto, Apache escribe el registro de transferencias en el archivo /var/log/httpd/access_log y el registro de errores en el archivo /var/log/httpd/error_log.



Este registro contiene la lista de todos los intentos de acceso al servidor web. Graba las direcciones IP del cliente que está intentando conectarse, la fecha y la hora de dicho intento y el archivo del servidor de web que quiere recuperar. Introduzca el camino y el archivo en el que almacenar esta información. Si el nombre de ambos no comienza con (/), entonces se entiende que el recorrido pertenece al directorio raíz del servidor tal y como se configuró. Esta opción corresponde a la directiva TransferLog en http://httpd.apache.org/docs/mod/mod_log_config.html#transferlog.

Puede configurar un registro con formato personalizado usando las **Utilizar las facilidades de registro personalizado** e introduciendo una cadena personalizada en el campo **Cadena de registro personalizada**. Esto configura la directiva LogFormat en http://httpd.apache.org/docs/mod/mod_log_config.html#logformat. Para mayor información sobre los detalles del formato de la directiva consulte http://httpd.apache.org/docs/mod/mod_log_config.html#formats.

El registro de errores contiene la lista de los errores que ocurren en el servidor. Introduzca el nombre del recorrido y del archivo en el que quiera guardar estos datos. Si ambos no comienzan con (/), se entenderá que el recorrido pertenece al directorio raíz del servidor tal y como se configuró. Esta opción corresponde a la directiva ErrorLog en http://httpd.apache.org/docs/mod/core.html#errorlog.

Use el menú **Nivel de Log** para configurar que niveles de mensajes de error pasamos al registro. Se puede establecer (de menor a mayor cantidad de mensajes) dicho nivel para emergencias, alertas, criticos, advertencias, observaciones, informes o depuración. Esta opción equivale a la directiva LogLevel en http://httpd.apache.org/docs/mod/core.html#loglevel.

El valor escogido en el menú **Resolución inversa del DNS** define la directiva HostnameLookups en http://httpd.apache.org/docs/mod/core.html#hostnamelookups. Si escoge **Ninguna búsqueda inversa** el valor desciende, si escoge **Búsqueda inversa** el valor asciende y si escoge **Doble búsqueda inversa** éste se dobla.

Al elegir la opción **Búsqueda inversa**, el servidor resuelve automáticamente la dirección IP para cada conexión que requiera un documento del servidor web. Esto quiere decir que el servidor lleva a cabo más de una conexión a la DNS hasta encontrar el nombre de la máquina a la que le corresponda una dirección IP determinada.

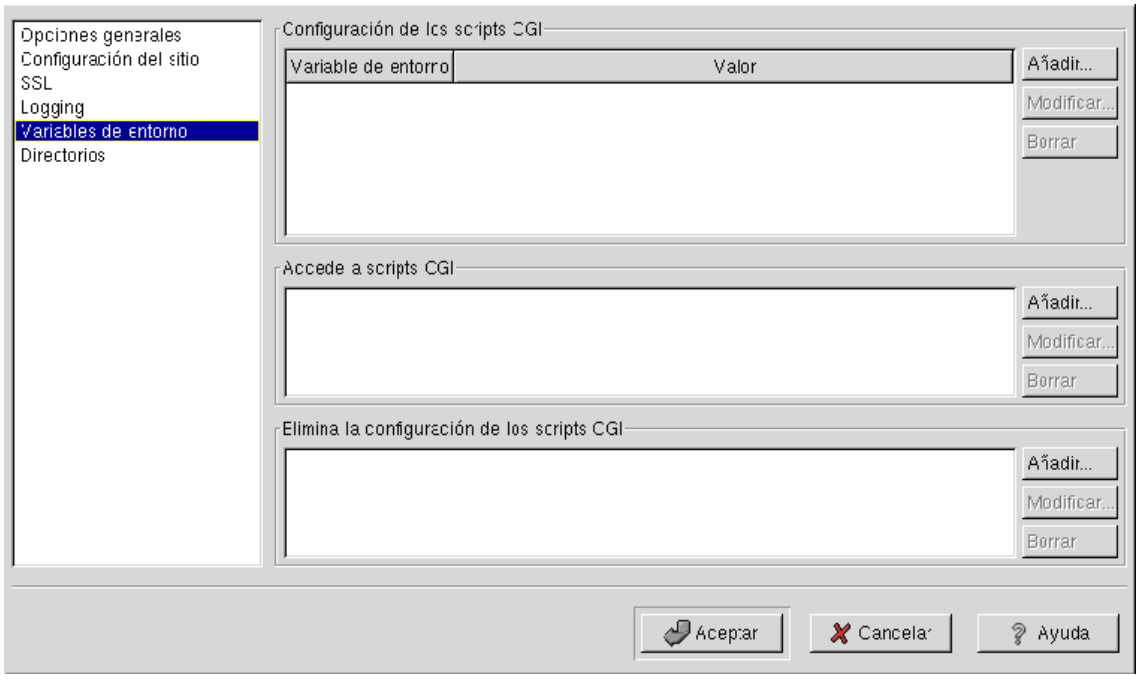
Si elige la opción **Doble búsqueda inversa**, el servidor realizará un DNS inverso doble. En otras palabras, después de una búsqueda inversa, hace una normal al resultado. Al menos una de las direcciones encontrada en esta segunda búsqueda debe coincidir con la primera.

Generalmente, esta opción debería de estar en **Ninguna búsqueda inversa** porque sino se sobrecarga al servidor y disminuye el ritmo de trabajo. Si su servidor tiene mucha carga, este tipo de búsquedas se realizarán más lentamente.

Tanto las búsquedas recíprocas como las dobles también se realizan desde Internet cuando se buscan determinados nombres de las máquinas. Por ello, es mejor si deja esta opción en **Ninguna búsqueda inversa**.

Variables de entorno

Apache usa el módulo mod_env para configurar las variables de entorno que se pasan a los scripts CGI y a las páginas SSI. Use la página **Variables de entorno** para configurar las directivas de este módulo.



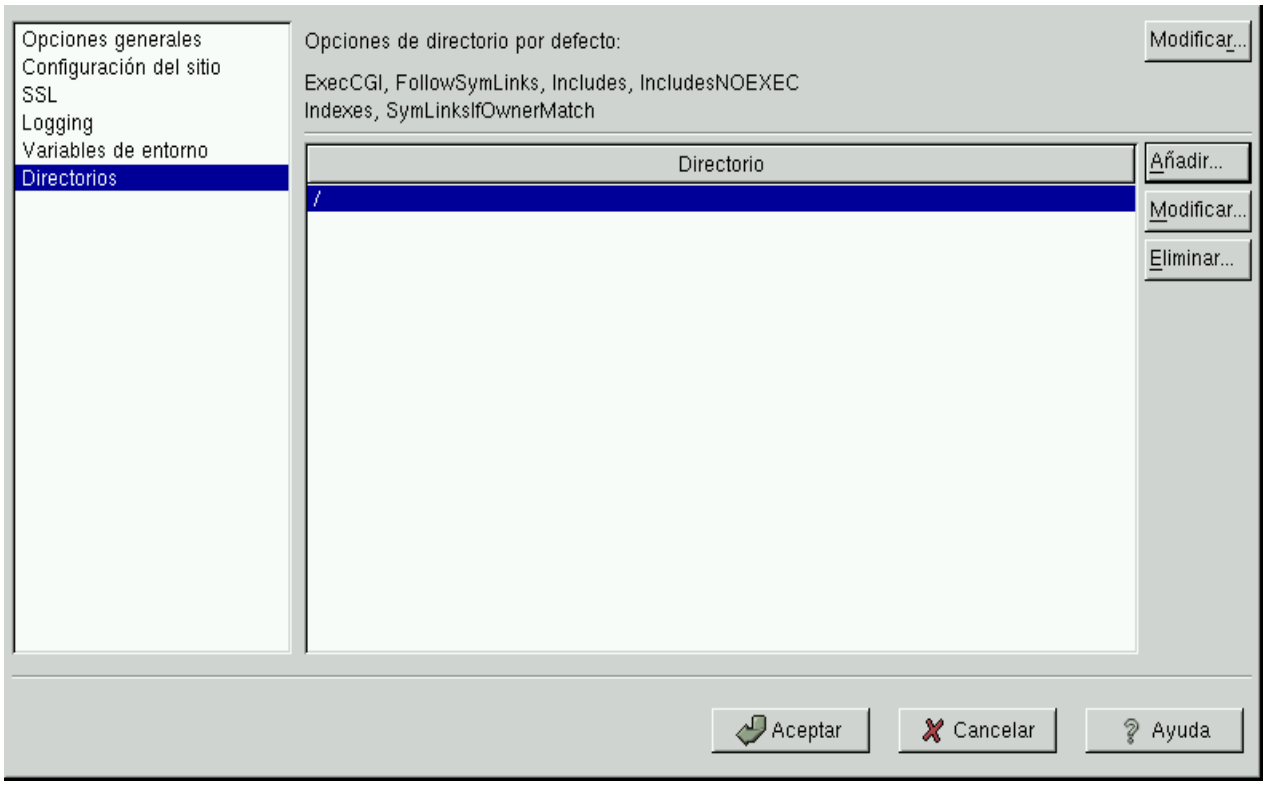
Use la sección **Configuración de los Scripts CGI** para establecer una variable de entorno que se pasa a los scripts CGI y a las páginas SSI.

Use la sección **Acceder a scripts CGI** para pasar el valor de una variable de estado una vez que se haya arrancado el servidor Apache para los scripts CGI. Para ver la variable teclee el comando env en la línea de comandos de la shell. Haga click en **Añadir** en la sección **Acceder a scripts CGIP** e introduzca el nombre de la variable de entorno que aparece en la ventana de diálogo. Después haga click en **OK**. La sección **Acceder a Scripts CGI** configura la directiva SetEnv en http://httpd.apache.org/docs/mod/mod_env.html#passenv.

Si desea eliminar el valor de la variable de entorno para que no pase ni al script CGI ni a la página SSI use la sección **Elimina la configuración de los Scripts CGI**. Luego haga click en **Añadir** en la sección **Elimina la configuración de los Scripts CGI** e introduzca el nombre de la variable de entorno que ha decidido eliminar. Esta opción corresponde a la directiva UnsetEnv en http://httpd.apache.org/docs/mod/mod_env.html#unsetenv.

Directorios

Use la página **Directorios** para configurar las opciones para directorios específicos. Esta opción corresponde a la directiva <Directory> en http://httpd.apache.org/docs/mod/core.html#directory.



Presione en el botón **Modificar** que se encuentra en la esquina superior derecha para configurar las **Opciones de directorio por defecto** para todos los directorios que no están especificados en la lista de **Directorios**. Las opciones que elija se encuentran en la lista de la directiva Opciones http://httpd.apache.org/docs/mod/core.html#options en la directiva <Directory> en http://httpd.apache.org/docs/mod/core.html#directory. Puede configurar las siguientes opciones:

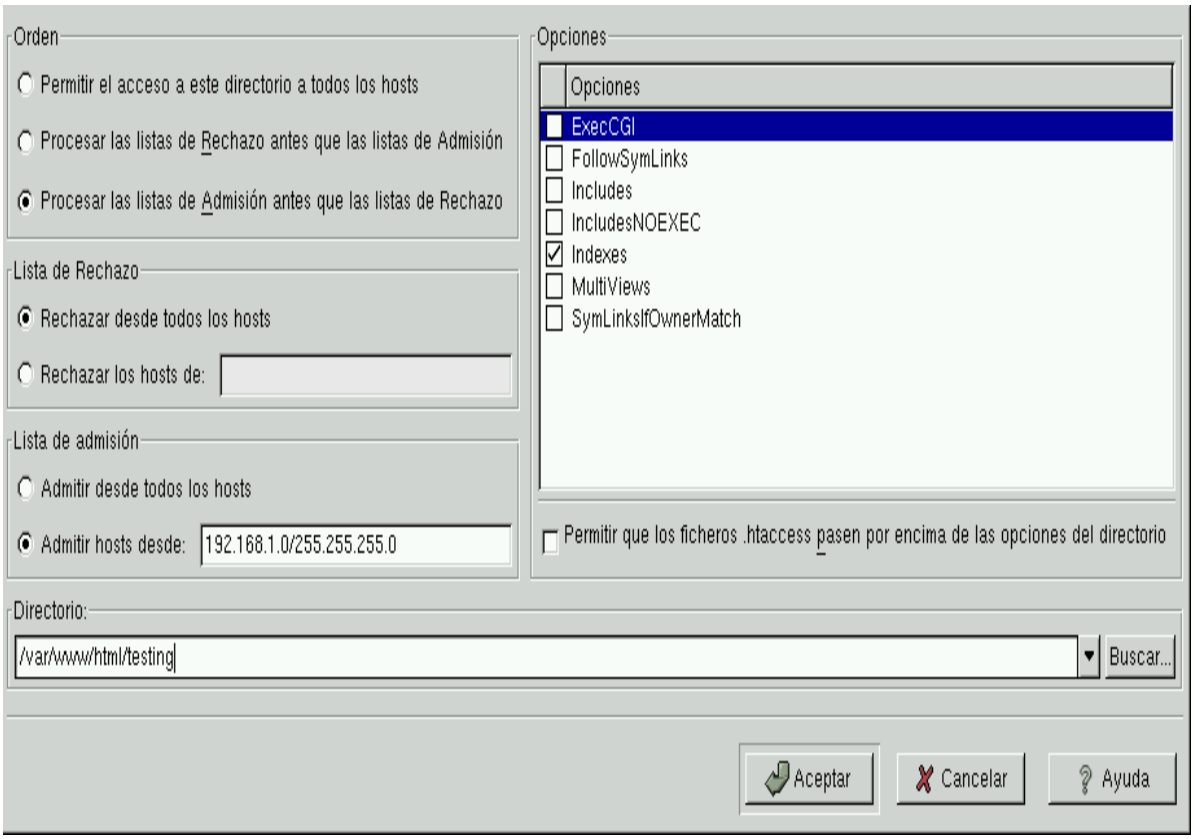
- **ExecCGI** — Permite la ejecución de los scripts CGI. No se ejecutan si no elige esta opción.
- **FollowSymLinks** — Permite que se sigan enlaces simbólicos.
- **Includes** — Permite las inclusiones en el servidor.
- **IncludesNOEXEC** — Permite las inclusiones en el servidor pero anula los comandos #exec y #include en los scripts CGI.
- **Indexes** — Muestra una lista formateada de los contenidos de un directorio si la opción DirectoryIndex (como por ejemplo index.html) existe en el directorio pedido.
- **Multiview** — Soporta las visualizaciones múltiples de los contenidos; esta opción no está activada por defecto.
- **SymLinksIfOwnerMatch** — Permite seguir un enlace simbólico solamente si el archivo o el directorio en cuestión tienen el mismo propietario que el enlace.

Para especificar las opciones para directorios determinados, presione el botón **Añadir** 

que se encuentra al lado de la lista **Directorio**. Introduzca el directorio para configurarlo en el campo **Directorio** que se encuentra en la parte de abajo de la ventana. Seleccione las opciones de la lista de la derecha y configure la directiva Orden en http://httpd.apache.org/docs/mod/mod_access.html#order con las opciones de la izquierda. Esta directiva controla el orden según el cual se permiten o se deniegan las directivas. En los campos **Admitir desde todos los hosts** y **Rechazar los hosts de:**, puede especificar uno de las siguientes:

- Permitir todas las máquinas 

— Teclee **all** para permitir el acceso a todas la máquinas.
- Nombre parcial de dominio — Permite todas las máquinas cuyos nombres coincidan o terminen con una cadena determinado.
- Dirección IP completa — Permite el acceso a una determinada dirección IP.
- Una subred — Como la **192.168.1.0/255.255.255.0**
- Una especificación CIDR de red — como por ejemplo **10.3.0.0/16**



Si controla las **Permitir que los archivos .htaccess pasen por encima de las opciones del directorio**, lo más importante son las directivas de configuración en el archivo .htaccess.

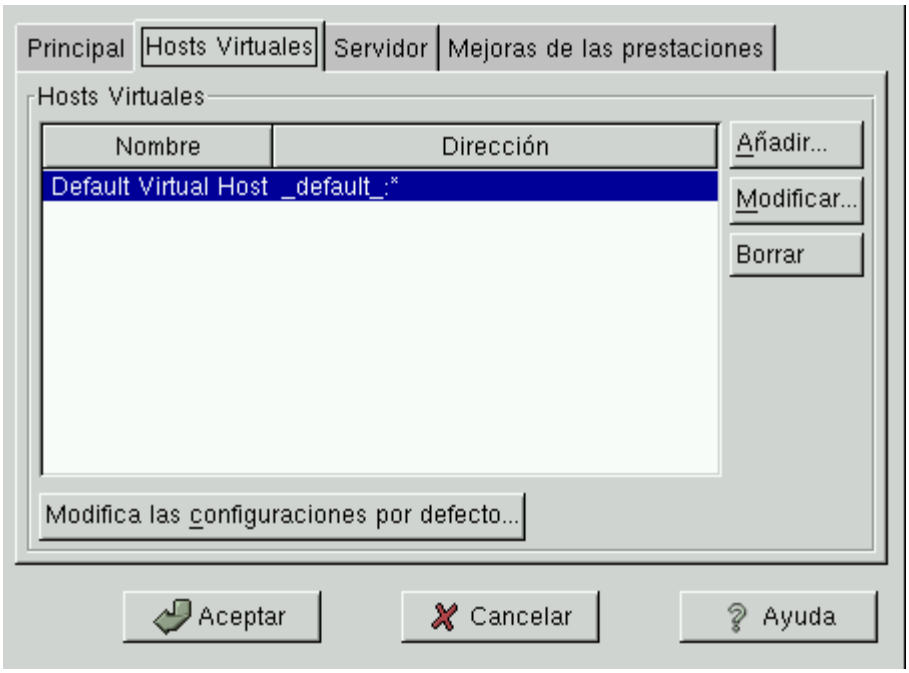
Configuraciones de las máquinas virtuales

Puede usar la **Apache Configuration Tool** para configurar máquinas virtuales. Éstas le permiten ejecutar servidores diferentes para direcciones IP diferentes, nombres de máquinas diferentes o puertos distintos en la misma máquina. Por ejemplo, puede ejecutar http://u.lagos1.cl/index.hmtl y http://u.lagos2.cl/index.hmtl en el mismo servidor Apache utilizando máquinas virtuales. Esta opción corresponde a la directiva <VirtualHost> destinada a la máquina virtual predeterminada y las direcciones IP de éstas. Corresponde a la directiva <NameVirtualHost> http://httpd.apache.org/docs/mod/core.html#namevirtualhost para la máquina virtual basada en el nombre.

Las directivas de Apache establecidas para una máquina virtual son sólo aplicables a ésta. Si se establece una directiva como "otras" usando el botón **Modifica las configuraciones por defecto**

pero no se definen las nuevas posiciones de la máquina virtual, entonces se aplicarán las predeterminadas. Por ejemplo, se puede definir una **Dirección email del webmaster** en **Principal** y no definir las direcciones de correo electrónico para cada una de las máquinas virtuales.

La **Herramienta de Configuración Apache** incluye una máquina virtual predeterminada como se muestra



Máquinas virtuales

- ***Añadir y modificar máquinas virtuales***. Para añadir una máquina virtual, presione en Hosts Virtuales y luego presione el botón Añadir. También puede modificar una máquina virtual seleccionando en la lista y después presione en Modificar.

- **Opciones generales.** Las configuraciones Opciones generales sólo se aplican a la máquina virtual que esté configurando. Escriba el nombre de la máquina virtual en Nombre del Host Virtual. La Herramienta de configuración Apache usa este nombre para distinguirlo de otras máquinas virtuales.

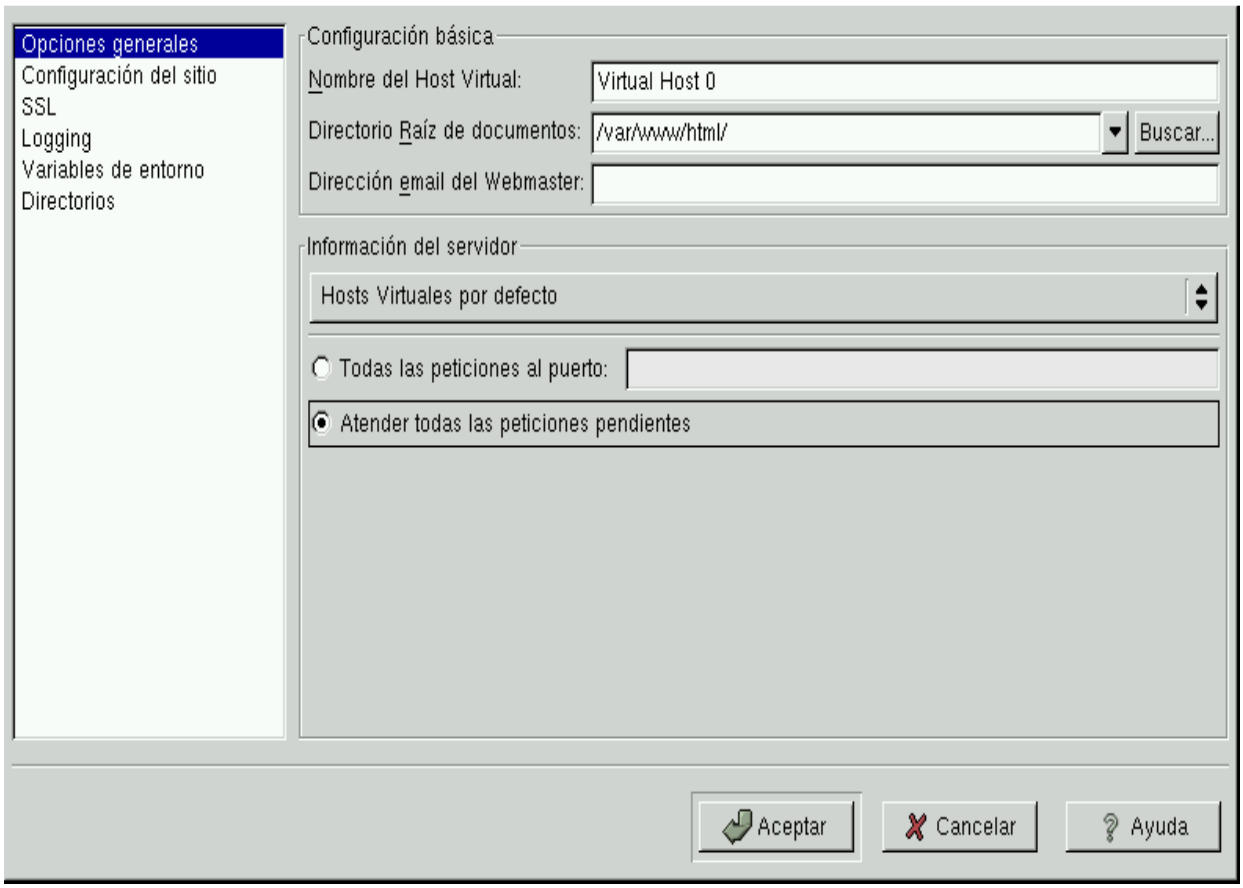
Establezca el valor del **Directorio raíz de documentos** en el directorio que contenga el documento root para la máquina virtual. Esta opción corresponde a la directiva DocumentRoot en la directiva VirtualHost.

La **Dirección email del webmaster** corresponde a la directiva ServerAdmin directiva ServerAdmin en la directiva VirtualHost. Esta dirección de correo electrónico se usa en el pié de página de las páginas de error si selecciona un pié de página con una dirección de correo electrónico en las páginas de error.

En la sección **Host Information** puede elegir entre **Hosts virtuales por defecto**, **Hosts virtuales basados en IP** o **Hosts virtuales basados en nombre**.

Máquinas virtuales predeterminadas

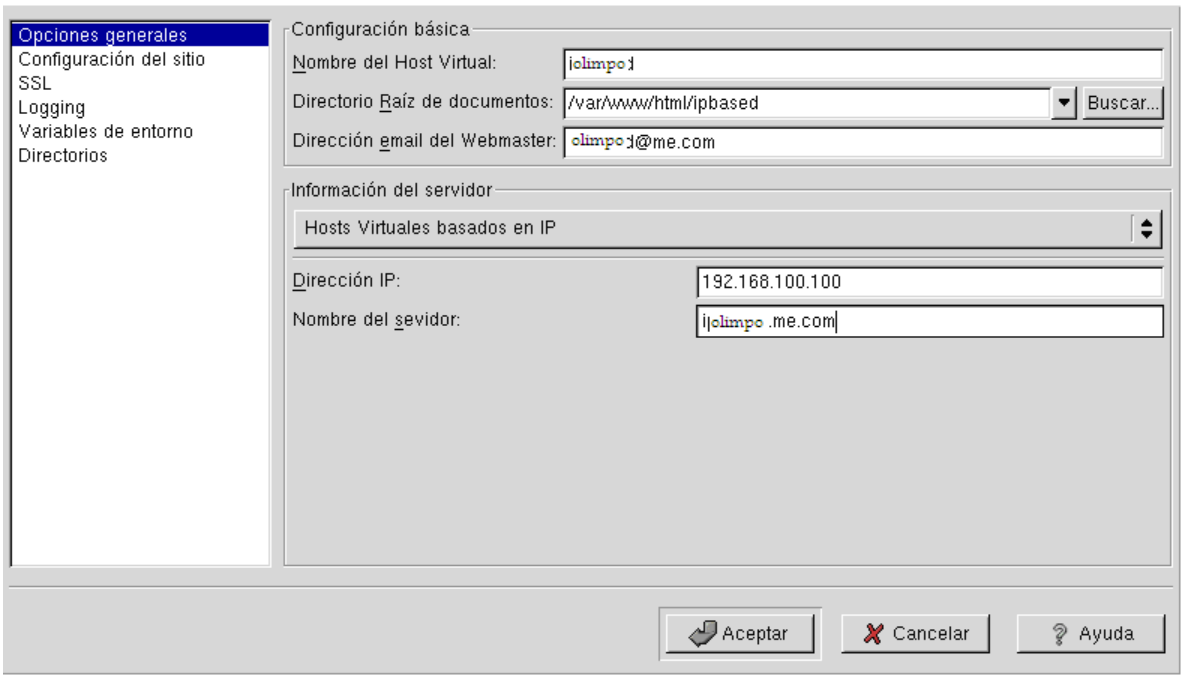
Si elige **Hosts virtuales por defecto**,



Sólo debe configurar una máquina virtual predeterminada. Las posiciones de la máquina virtual predeterminada se usan cuando la dirección IP requerida no aparece explícitamente en la lista de otra máquina virtual. Si no existe ninguna máquina virtual definida, se usan las posiciones principales del servidor.

Máquinas virtuales basadas en la dirección IP

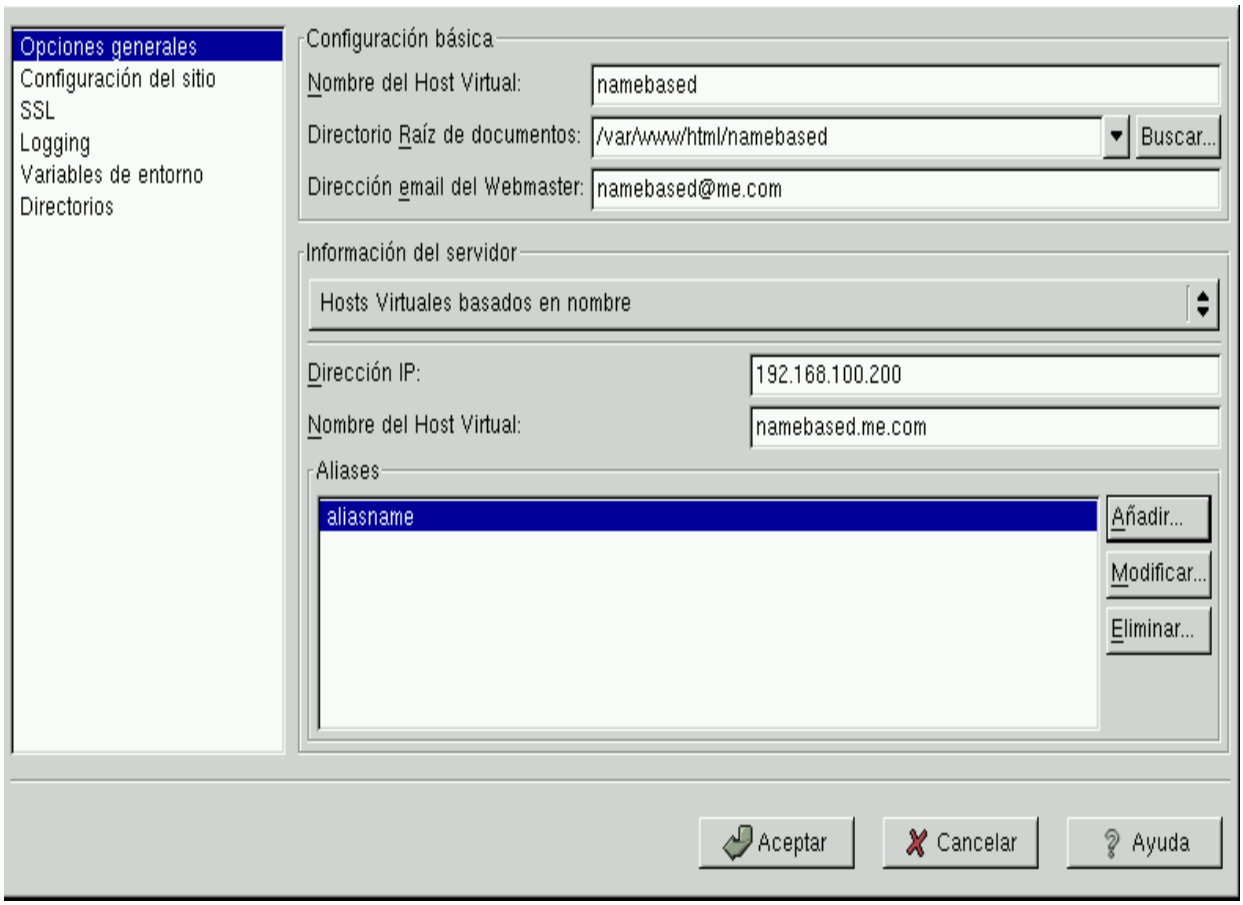
Si elige la opción **Hosts Virtuales basados en IP**, aparecerá una pantalla



para configurar la directiva <VirtualHost> http://httpd.apache.org/docs/mod/core.html#virtualhost basada en la dirección IP del servidor. Especifique la dirección IP en el campo **Drección IP**. Si desea especificar más de una dirección IP sepárelas con un espacio. Utilice la frase *Dirección IP: Puerto*. Use :* para configurar todos los puertos para la dirección IP. Especifique el nombre de la máquina para la máquina virtual en el campo **Nombre del servidor**.

Máquinas virtuales basadas en el nombre

Si escoge la opción **Hosts virtuales basados en nombre**, aparecerá una pantalla como en



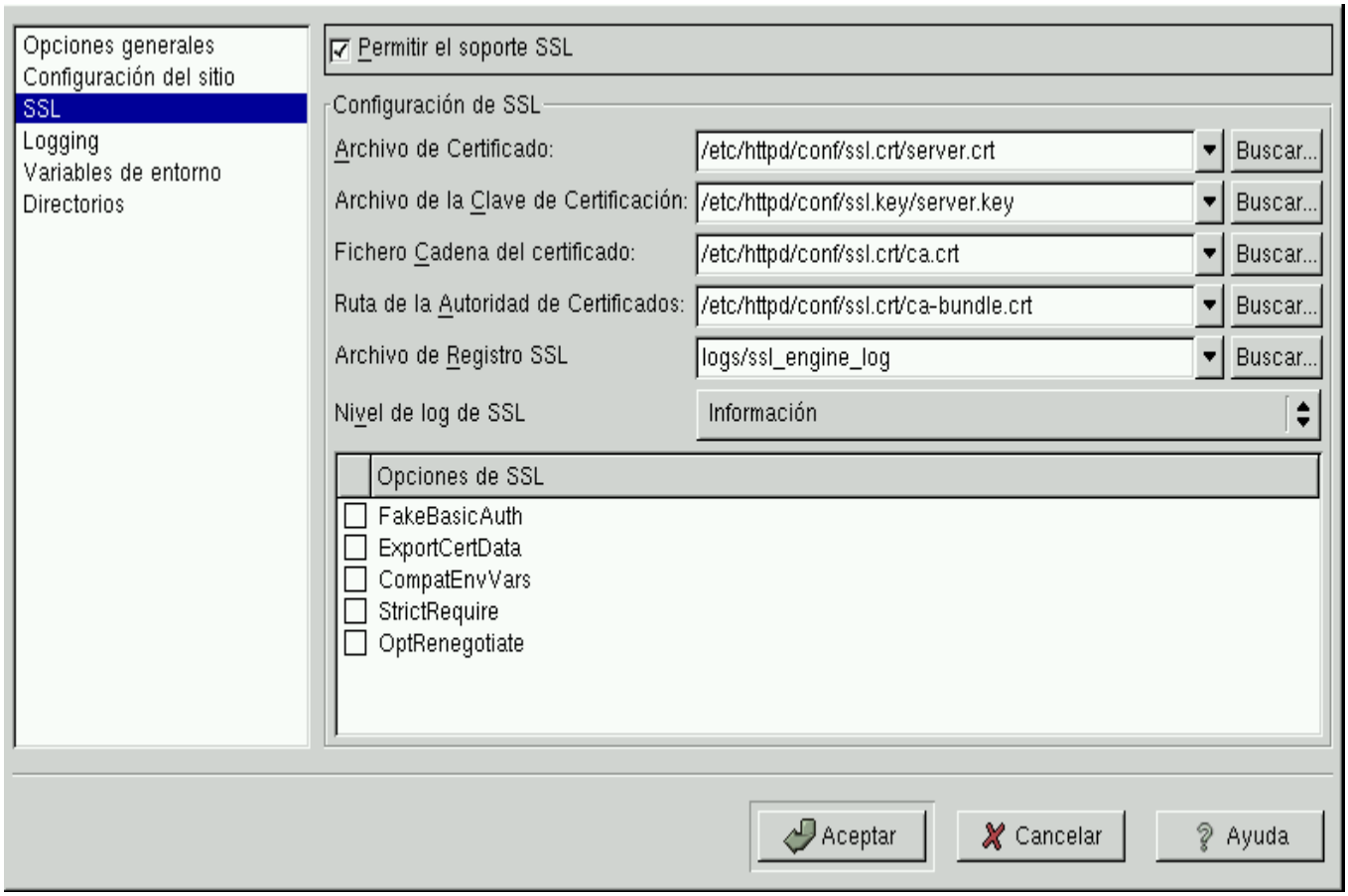
Para configurar la directiva NameVirtualHost basada en el nombre de la máquina del servidor. Especifique la dirección IP en el campo **Dirección IP**. Para especificar más de una dirección IP, sepárelas con espacios. Para un puerto, use la sintaxis *Dirección IP*. Use :* para configurar todos los puertos de esa dirección IP. Especifique el nombre de la máquina para la máquina virtual en el campo **Hosts virtuales basados en nombre**. En la sección **Aliases**, haga click en **Añadir** para añadir un alias del nombre de la máquina. Añadir un alias aquí añade una directiva ServerAlias http://httpd.apache.org/docs/mod/core.html#serveralias en la directiva NameVirtualHost http://httpd.apache.org/docs/mod/core.html#namevirtualhost.

Para poder utilizarlo con la **herramienta de configuración de Apache** tiene que permitir el acceso a través del puerto 443 yendo a

- **Principal**
- **Direcciones disponibles.**

Consulte la sección de nombre *configuración básica* para mayor información. Después, seleccione el nombre de la máquina virtual en **Hosts Virtuales**, presione **Modificar**, elija **SSL** en el menú de la izquierda y busque la opción **Permitir el soporte SSL.**

La sección **Configuración de SSL** está ya configurada con un certificado digital el cual autentica su servidor de web seguro e identifica el servidor seguro para los navegadores de la web.



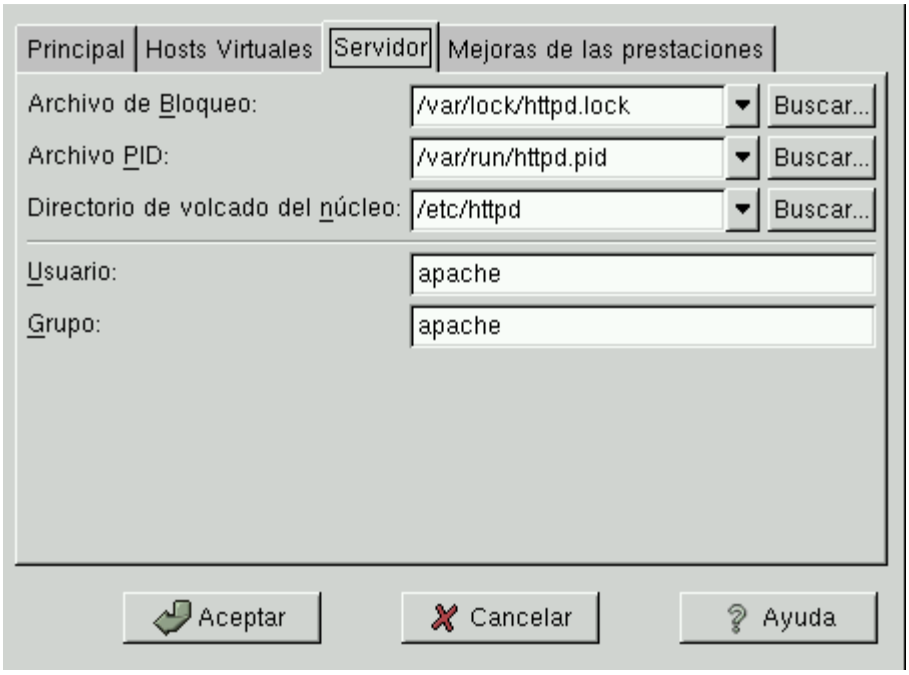
Soporte SSL

Opciones adicionales de las máquinas virtuales

Las opciones **Configuración del sitio**, **Variables de entorno** y **Directorios** para las máquinas virtuales son las mismas directivas que estableció cuando eligió la opción **Modifica las configuraciones por defecto**, excepto que las opciones aquí establecidas son para cada una de las máquinas virtuales que está configurando. Para mayor información, consulte la la sección de nombre *Configuraciones predeterminadas*.

Configuración del servidor

La pestaña **Servidor** le permite configurar la configuración básica del servidor. La configuración básica para estas opciones sirve para la mayoría de las situaciones.



- **Archivo de bloqueo** 

corresponde a la directiva LockFile http://httpd.apache.org/docs/mod/core.html#lockfile. Esta directiva establece el recorrido hacia el archivo de cierre que se utiliza cuando se compila el servidor Apache con USE_FCNTL_SERIALIZED_ACCEPT o con USE_FLOCK_SERIALIZED_ACCEPT. Se debe almacenar en un disco local. Este valor no se debe de cambiar a no ser que el directorio logs esté situado en una partición NFS. Si fuese este el caso, se debería cambiar el valor predeterminado a un disco local y a un directorio que sólo se pueda leer si se es root.

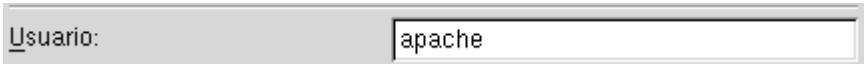
- **Archivo PID** 

corresponde a la directiva PidFile http://httpd.apache.org/docs/mod/core.html#pidfile. Esta directiva establece el archivo en el que se graba el proceso ID (pid). Este archivo se puede leer sólo si se es root. En la mayoría de los casos, se debería de dejar el valor predeterminado.

- **Directorio de volcado del núcleo** 

corresponde a la directiva CoreDumpDirectory http://httpd.apache.org/docs/mod/core.html#coredumpdirectory. El

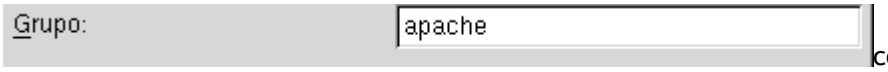
servidor Apache intenta cambiarse a este directorio antes de volcar el núcleo. El valor predeterminado es el ServerRoot. Sin embargo, si la computadora que el servidor está ejecutando no puede escribir en este directorio, no se puede escribir en el buffer del núcleo. Cambie el valor a un directorio en el que el usuario pueda escribir, si desea escribir los vaciados de núcleo en el disco.

- **Usuario** 

corresponde a http://httpd.apache.org/docs/mod/core.html#user. Establece el userid que utiliza el servidor para responder a las preguntas. Las posiciones del usuario determinan el acceso del servidor. Todo archivo al que el usuario no tenga acceso será también inaccesible a los visitantes del sitio web. El valor predeterminado para Usuario es apache.

El usuario debe de tener sólo privilegios de tal manera que pueda acceder a archivos que supuestamente puede ver el resto de los usuarios. También posee todos los procesos CGI del servidor pero no debe ejecutar ningún código cuyo fin no sea responder a las peticiones HTTP.

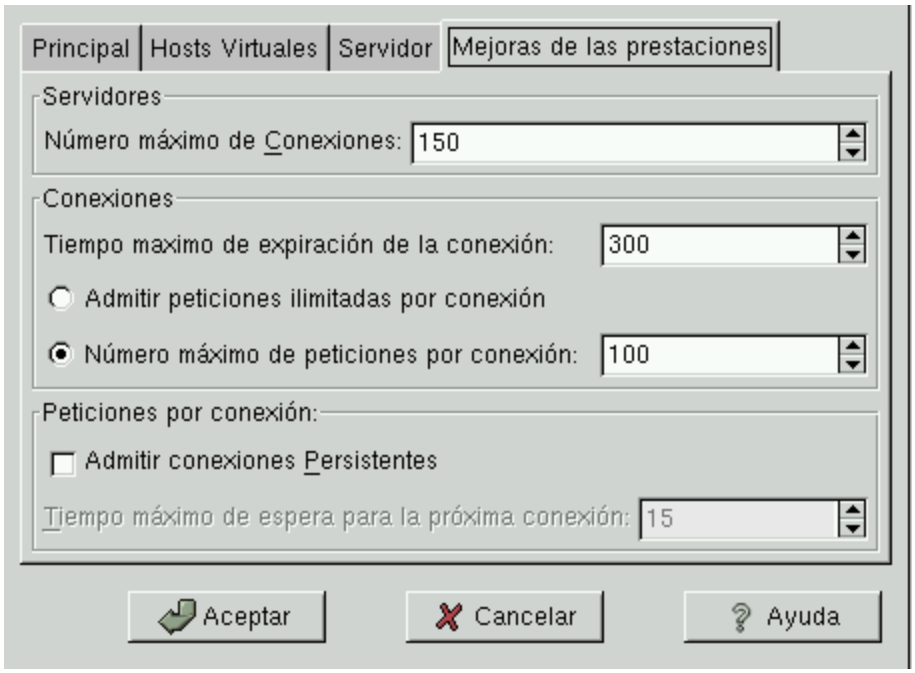
El proceso padre httpd primero se ejecuta como root durante las operaciones normales pero luego pasa a las manos del usuario normal de apache. El servidor debe arrancarse como root porque necesita un puerto cuyo valor sea inferior a 1024. Los puertos con valores inferiores a 1024 están reservados al sistema, por lo tanto no los puede usar cualquiera. Una vez que el servidor se haya conectado a su puerto, el usuario apache se encarga del proceso antes de que responda a las conexiones.

- **Grupo** 

corresponde a la directiva Group http://httpd.apache.org/docs/mod/core.html#group. Esta directiva es similar a la de User. Se encarga de establecer el grupo en el que el servidor contestará a las peticiones. El Group predeterminado también es apache.

Ajuste del rendimiento

Presione en **Mejoras de las prestaciones** para configurar el número máximo de procesos hijo que desee tenga el servidor así como las opciones de Apache para las conexiones del cliente. Las posiciones predeterminadas para estas opciones se pueden aplicar a la mayoría de las situaciones. Si modifica estas posiciones afectará al rendimiento de su servidor web.

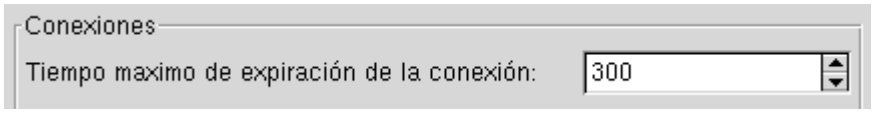


Ajuste del rendimiento

- **Número máximo de conexiones**

en el número máximo de conexiones simultáneas que el servidor llevará a cabo. Para cada conexión se crea un proceso httpd hijo. Una vez que se haya alcanzado este número máximo no se podrán conectar más clientes hasta que se libere el proceso hijo del servidor. Este valor no puede ser superior a 256 a menos que recompile Apache. Esta opción corresponde a la directiva MaxClients http://httpd.apache.org/docs/mod/core.html#maxclients.

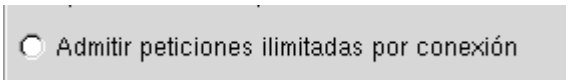
- **Tiempo máximo de expiración de la conexión**



define en segundos el intervalo de tiempo que el servidor espera entre la recepción y la transmisión de datos durante la comunicación. En otras palabras, determina cuánto tiempo espera el servidor para recibir una petición GET, para recibir paquetes POST o PUT y cuánto tiempo espera entre cada ACK que responda a los paquetes TCP. El valor predeterminado es de 300 segundos que se adapta a

todas las situaciones. Esta opción corresponde a la directiva TimeOut http://httpd.apache.org/docs/mod/core.html#timeout.

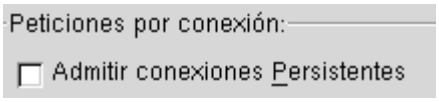
Establezca el número máximo de peticiones permitidas por cada conexión con la opción **Número máximo de peticiones por conexión**. El valor predeterminado es 100 que normalmente se adapta a todas las situaciones. Esta opción corresponde a la directiva MaxRequestsPerChild http://httpd.apache.org/docs/mod/core.html#maxrequestsperchild.

- **Admitir peticiones ilimitadas por conexión** 

el valor de la directiva MaxKeepAliveRequests es 0 lo que significa que se pueden llevar a cabo un número ilimitado de conexiones.

Si no selecciona la opción **Admitir conexiones persistentes** la directiva KeepAlive http://httpd.apache.org/docs/mod/core.html#keepalive aparece como falsa pero si la selecciona aparece como verdadera así como la directiva KeepAliveTimeOut http://httpd.apache.org/docs/mod/core.html#keepalivetimeout le indicará el valor seleccionado para la opción **Tiempo máximo de espera para la próxima conexión**. Esta directiva establece los segundos que el servidor espera entre una petición y otra antes de que se cierre la conexión. Una vez que se ha recibido la petición, se aplica la opción **Tiempo máximo de expiración de la conexión** 

. Esta directiva configura el número de segundos que su servidor esperará, después de que la petición se haya respondido, se aplica el valor **Connection Timeout**.

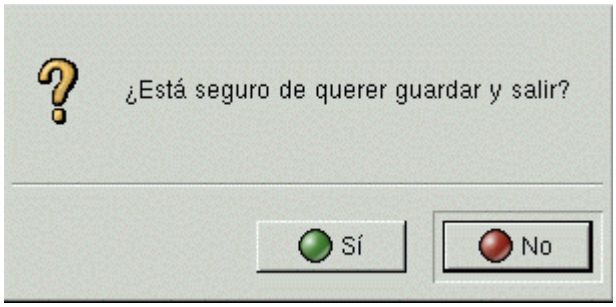
- **Conexiones persistentes** 

es muy alto, el servidor realiza sus tareas más lentamente dependiendo del número de usuarios que estén intentando conectarse en ese momento. Cuanto mayor sea el valor, mayor será el tiempo de espera entre una conexión y otra

Guardar sus configuraciones

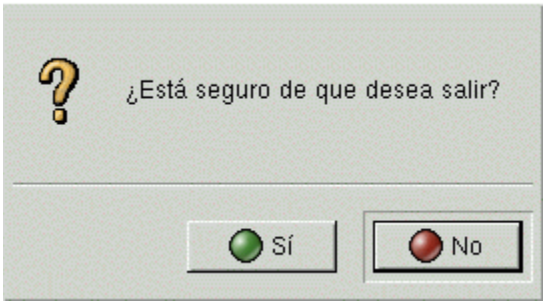
Si no desea guardar la configuración de su servidor Apache, haga click en **Cancelar** que se encuentra en la parte de abajo a la derecha de la ventana de la **herramienta de configuración de Apache**. El sistema le preguntará si desea llevar a cabo esta operación. Si es así, haga click en **Sí**.

En cambio, si desea guardar la configuración de Apache, presione en **OK**. Aparecerá la ventana de diálogo



. Si confirma esta operación, las posiciones de configuración se guardarán en el archivo /etc/httpd/conf/httpd.conf. Recuerde que se sobreescribirá el archivo de configuración original.

Si es la primera vez que ha utilizado la **herramienta de configuración de Apache** , aparecerá una ventana de diálogo



en el se le advertirá que el archivo de configuración se ha modificado manualmente. Si la **herramienta de configuración de Apache** detecta que el archivo de configuración httpd.conf se ha modificado manualmente, guardará el archivo modificado con el nombre /etc/httpd/conf/httpd.conf.bak.

Encendido y apagado del Servidor

Una vez que ya tiene su archivo httpd.config configurado ya puede proceder a encender el servidor.

En /usr/local/apache/bin/ están los binarios de nuestro servidor web. Allí encontraremos un pequeños script llamado apachectl. Este posee, básicamente, tres parámetros:

- start,
- restart
- stop.

El primero enciende el servidor httpd, el segundo lo reinicia y el tercero apaga el servidor. Para poder encender un servidor web, tendremos que ejecutar:

Apachectl start.

MySQL

Para descargar MySQL iremos a http://www.mysql.com, en donde pulsaremos sobre la sección Downloads. En esta sección elegiremos la versión estable más moderna de MySQL (en estos momentos la 3.23). Descargaremos el código fuente de la última distribución (ahora la 3.23.33) en formato .tar.gz (descárguelo directamente si quiere desde http://www.mysql.com/Downloads/MySQL-3.23/mysql-3.23.33.tar.gz).

Descomprimiremos el código fuente (suponiendo que el archivo descargado sea el indicado anteriormente):

```
$tar xvzf mysql-3.23.33.tar.gz
$cd mysql-3.23.33
```

Ahora tenemos que crear un nuevo usuario (mysql) y un nuevo grupo (mysql) que serán los que usen MySQL para ejecutarse:

```
$/usr/sbin/groupadd mysql
$/usr/sbin/useradd -g mysql mysql
```

Ya podemos compilar e instalar el gestor de bases de datos. En nuestro caso vamos a elegir**/usr/local/mysql** como directorio de instalación. Por lo tanto, si elige otro directorio para su instalación tendrá que modificar **/usr/local/mysql** siempre que lo nombremos posteriormente con el directorio donde haya instalado su versión de MySQL.

```
$./configure --prefix=/usr/local/mysql
$make
$make install
```

Ya tenemos instalada nuestro gestor de bases de datos MySQL. Ahora tendremos que instalar las tablas básicas que MySQL necesita para funcionar. Para ello ejecutaremos un script que se nos proporciona:

```
$scripts/mysql_install_db
```

Ahora tendremos que cambiar el usuario y el grupo propietarios del directorio de instalación para que se correspondan con aquellos con los que se ejecutara el demonio **mysqld**:

```
$chown -R mysql /usr/local/mysql
```

```
$chgrp -R mysql /usr/local/mysql
```

Para ejecutar MySQL sólo tendremos que escribir:

```
$/usr/local/mysql/bin/safe_mysqld --user=mysql&
```

Si queremos que MySQL se ejecute cada vez que reiniciamos nuestro sistema Linux tendremos que añadir a uno de los ficheros de inicio la línea:

```
/bin/sh ´cd /usr/local/mysql; ./bin/safe_mysqld --user=mysql&´
```

Tendremos que cambiar el password del adminitrador del gestor de bases de datos, ya que MySQL no pone ninguno por defecto. Para ello, una vez que tengamos MySQL ejecutándose, haremos lo siguiente:

```
$/usr/local/mysql/bin/mysqladmin -u root -p password 'mysql'
```

Una vez que ejecutemos lo anterior, nos pedirá el password, que como en este caso no estará puesto bastará con pulsar <Intro> y quedará fijado mysql como el nuevo password.

Instalando el driver JDBC para MySQL

En la sección Downloads de la página de MySQL, iremos a la sección del API para JDBC. Elegimos que versión queremos descargar (nosotros usaremos la versión 1.2c, que puede descargar directamente desde http://www.mysql.com/Downloads/Contrib/mm.mysql.jdbc-1.2c.tar.gz. Si descarga otra versión tendrá que cambiar los números en los nombres de ficheros y directorios). Descomprimimos el fichero obtenido y nos cambiamos al directorio creado:

```
$tar xvzf mm.mysql.jdbc-1.2.tar.gz
$cd mm.mysql.jdbc-1.2c
```

Veremos que en dicho directorio tenemos un fichero llamado **mysql_comp.jar** que será el que usemos a no ser que nuestra JVM no soporte los ficheros jar comprimidos (tenemos el fichero **mysql_uncomp.jar** en ese caso). Podemos dejar el fichero donde esta o copiarlo a otro directorio, ya que lo único que tenemos que hacer para poder utlizarlo es incluirlo en el CLASSPATH, y para cargarlo desde un programa Java utilizar el siguiente comando:

```
Class.forName("org.gjt.mm.mysql.Driver");
```

Instalando Tomcat

Vamos a descargar antes que nada el código de Tomcat que se encuentra disponible en http://jakarta.apache.org. En este caso descargaremos los binarios, ya que al tratarse de un programa que está escrito en Java los podremos usar en cualquier plataforma con una JVM.

La última versión estable (3.2.1) que es la que usaremos la podéis descargar directamente desde http://jakarta.apache.org/builds/jakarta-tomcat/release/v3.2.1/bin/jakarta-tomcat-3.2.1.tar.gz.

Ahora instalaremos Tomcat en /usr/local/jakarta-tomcat-3.2.1.

```
$cp jakarta-tomcat-3.2.1.tar.gz /usr/local
$tar xvzf /usr/local/jakarta-tomcat-3.2.1.tar.gz
$rm /usr/local/jakarta-tomcat-3.2.1.tar.gz
```

Una vez hecho esto, sólo tenemos que poner una variable llamada TOMCAT_HOME en nuestro entorno y que apunte al directorio anterior, lo que en bash (Bourne Again SHell) se hace así:

```
$TOMCAT_HOME=/usr/local/jakarta-tomcat-3.2.1; export TOMCAT_HOME
```

Recuerde que antes de lanzar el Tomcat debe de tener también una variable de entorno llamada JAVA_HOME que apunte al directorio raíz de la instalación de su JVM, y que además el directorio JAVA_HOME/bin debe estar incluido en la variable de entorno PATH. Ahora ya podemos lanzar nuestro servidor Tomcat con el script:

```
/usr/local/jakarta-tomcat-3.2.1/bin/startup.sh
```

REFERENCIA DE CÓDIGOS DE EJEMPLO UTILIZADOS

BIBLIOGRAFÍA

JAVA HOW TO PROGRAM, Harvey Deitel and Paul Deitel, Fifth Edition

CREACION DE UN PORTAL CON PHP Y MYSQL, Pavon J., Ed. RaMa

FLASH, PHP Y MYSQL CONTENIDOS DINAMICOS, Daniel de la Cruz Heras, Carlos Zumbado Rodríguez Ed. Anaya Multimedia

DESARROLLO WEB CON PHP Y MYSQL, Welling Luke, Thompson Laura, Ed. Anaya Multimedia

http://java.sun.com/developer/technicalArticles/xml/WebAppDev/
http://java.sun.com/developer/technicalArticles/xml/WebAppDev2/
http://www.javaworld.com/jw-03-2000/jw-0331-ssj-jspxml.html
www.php.net/tut.php
www.programacion.net/cursos/php/mysql.htm
www.programacion.com/php/tutorial/php
www.webestilo.com/php/
www.programatium.com/php
www.gamarod.com.ar/recursos/tutoriales/php
http://java.sun.com/products/jsp
http://www.jguru.com/faq/JSP