

Tesis  
QA  
76.73  
.J38  
B37  
2004

UNIVERSIDAD SAN FRANCISCO DE QUITO

COLEGIO POLITECNICO

PROYECTO DE GRADO PREVIO A LA OBTENCION  
DEL TITULO DE:

INGENIERO DE SISTEMAS

USFQ - BIBLIOTECA

14955

TALLER DE JAVA AVANZADO

ELABORADO POR:

GERMAN AUGUSTO BASTIDAS RUIZ

CARLOS ANDRES YONCON CHANGKUON

DIRECTORA:

ING. LORENA BALSECA

QUITO, FEBRERO 2004

USEQ - BIBLIOTECA

d. Autor	
04-00-23	
27 SET 2004	03498

**Universidad San Francisco de Quito**  
Colegio Politécnico

**HOJA DE APROBACION DE TESIS**

**Taller de Java Avanzado**

Germán Augusto Bastidas Ruiz

Carlos Andrés Yoncón Changkuon

Lorena Balseca, Ing.  
Directora de Tesis



---

Lorena Balseca, Ing.  
Miembro del Comité de Tesis



---

Fausto Pasmay, M.S.  
Miembro del Comité de Tesis



---

Santiago Navarro, Ph.D.  
Decano del Colegio Politécnico



---

Quito, febrero del 2004

© Derechos de autor  
Germán Augusto Bastidas Ruiz  
Carlos Andrés Yoncón Changkuon  
2004

# Agradecimiento

Agradecemos a las autoridades y a los profesores de la Universidad San Francisco de Quito, quienes motivaron e impulsaron nuestra formación humanística, científica y profesional, durante cinco años de sacrificio y estudio; a nuestros padres y hermanos, por todos los sabios consejos y el respaldo incondicional y, a todas aquellas personas que siempre estuvieron brindándonos el apoyo necesario para la culminación de este proyecto de grado.

**Germán Bastidas Ruiz**

# Resumen

El **Taller de Java Avanzado** es una herramienta de aprendizaje de conceptos avanzados del lenguaje Java. Este lenguaje es muy utilizado en el desarrollo de aplicaciones para Internet y aplicaciones distribuidas.

El **Taller de Java Avanzado** está dirigido a profesionales o a estudiantes con conocimientos básicos del lenguaje de programación Java y que necesiten reafirmar esos conocimientos y aprender algunos nuevos.

La metodología del taller se basa en presentar un caso de estudio que se desarrollará a lo largo del curso: la construcción de un servidor web. Mediante este desarrollo se pretende introducir conceptos avanzados del lenguaje Java de una forma práctica.

El taller puede ser utilizado para dos formas de enseñanza: una forma guiada, en la cual un instructor maneja el curso; y una forma de auto aprendizaje, en la cual el estudiante puede completar el curso a su propio ritmo.

El taller está dividido en capítulos que introducen de forma estructurada los conceptos y el desarrollo del caso de estudio. Cada capítulo cuenta con una introducción de los conceptos a utilizarse en el mismo, y también, con la debida explicación paso a paso del desarrollo de un segmento de código del servidor web. Además de la estructura en capítulos, se ofrece un glosario de términos y referencias para que el estudiante pueda consultar y ampliar los conceptos.

El taller es presentado en un CD interactivo multimedia, y puede ser accesado a través de un navegador web. Además se incluye en el CD el código fuente del servidor web que servirá al estudiante o instructor como base para poder evaluar el desempeño en el taller.

## Abstract

The **Advanced Java Workshop** is a learning tool of advanced concepts in Java programming language. This language is very useful in the development process of Internet and distributed applications.

The **Advanced Java Workshop** is aimed to professionals or students with basic Java programming language knowledge and who need to reinforce those concepts or gain new ones.

The methodology of this workshop is based in the expose of a study case that will be developed throughout the course: the construction of a web server. By making this development, new concepts of Java programming will be introduced in a practical way.

The workshop can be used in two teaching approaches: one guided approach, in which an instructor guides the course; and a self-learning approach, in which the student can complete the course at his own pace.

The workshop is divided in chapters that introduce, in a structured way, the concepts and the development of the case study. Each chapter includes an introduction to the concepts used on it and then it presents the step-by-step development of a segment of code of the web server. Besides the by-chapters structure, the course offers a glossary of terms and additional reference for the student to consult and gain knowledge.

The workshop is presented in an interactive multimedia CD, and it can be accessed by using a web browser. Also the CD includes the source code of the web server in order to give the chance to students or instructors to track their advances in the course and evaluate their knowledge in the workshop.

# INDICE

Pág.

## 1. ANTECEDENTES

<b>1.1 Introducción al Internet y la WWW</b> .....	1
<b>1.2 Java y sus características principales</b> .....	4
1.2.1 Antecedentes históricos .....	4
1.2.2 Características de Java .....	5
<b>1.3 Descripción básica del entorno de trabajo de los servidores web</b> .....	8
1.3.1 Proceso de comunicación con un servidor web .....	9
1.3.2 Modelos de comunicación cliente-servidor .....	11
<b>1.4 Análisis de requerimientos</b> .....	13
1.4.1 Necesidades .....	14
1.4.2 Descripción de requerimientos .....	14
1.4.3 Definición de entregables .....	15

## 2. MARCO TEORICO

<b>2.1 Protocolo de transferencia de Hipertexto HTTP</b> .....	17
2.1.1 Funcionamiento del protocolo HTTP .....	18
2.1.2 Identificadores de recursos .....	19
2.1.3 Mensajes HTTP .....	22
2.1.3.1 Tipos de mensajes .....	22
2.1.3.2 Encabezado de los mensajes HTTP .....	22
2.1.3.3 Cuerpo del mensaje HTTP .....	24
2.1.3.4 Entidades de los mensajes HTTP .....	28
2.1.4 Mensaje Request .....	28
2.1.4.1 Línea de inicio del mensaje Request .....	29
2.1.4.2 Headers o encabezados del mensaje Request .....	31
2.1.4.3 Cuerpo del mensaje Request .....	32
2.1.4.4 Estructura del mensaje Request .....	32
2.1.5 Mensaje Response .....	33
2.1.5.1 Línea de estado del mensaje Response .....	34
2.1.5.2 Headers o encabezados del mensaje Response .....	37
2.1.5.3 Cuerpo del mensaje Response .....	38
2.1.5.4 Estructura del mensaje Response .....	38
<b>2.2 Análisis del servidor web</b> .....	39
2.2.1 Estados del servidor .....	40
2.2.2 Arquitectura del servidor .....	43
2.2.3 Procesos .....	44
2.2.4 Descripción de los módulos del servidor .....	48



### **3. DESCRIPCION DEL PROYECTO**

<b>3.1 Descripción del servidor</b> .....	51
3.1.1 Alcance de la aplicación .....	52
3.1.2 Características del servidor web .....	55
3.1.3 Requisitos de hardware y software .....	56
3.1.3.1 Requisitos de hardware .....	56
3.1.3.2 Requisitos de software .....	58
3.1.4 Servicios .....	58
3.1.5 Ejecución de la aplicación .....	60
3.1.6 Primer modo de inicio: Proceso .....	63
3.1.7 Segundo modo de inicio: Aplicación GUI .....	65
3.1.7.1 Barra de herramientas .....	66
3.1.7.2 Panel de peticiones .....	69
3.1.7.3 Barra de estado .....	70
<b>3.2 Descripción de currículo del taller</b> .....	71
3.2.1 Metodología .....	71
3.2.2 Beneficios .....	72
3.2.3 Prerrequisitos del estudiante .....	72
3.2.4 Proyección del perfil del instructor .....	74
3.2.5 Estructura general del currículo .....	75
3.2.6 Descripción detallada de los capítulos del currículo .....	75

### **4. CONCLUSIONES Y RECOMENDACIONES** 83

### **5. ANEXOS** 85

### **6. GLOSARIO** 128

### **7. BIBLIOGRAFIA** 131

## INDICE DE FIGURAS

	Pág.
1. Comunicación cliente-servidor .....	9
2. Mensaje request con método POST .....	24
3. Mensaje request con método GET .....	25
4. Mensaje response que contiene un archivo HTML .....	26
5. Mensaje response que contiene un archivo GIF .....	27
6. Estructura del mensaje Request .....	33
7. Estructura del mensaje Response .....	38
8. Arquitectura del servidor .....	43
9. Diagrama de arranque del servidor sin interfase gráfica .....	45
10. Diagrama de arranque del servidor con interfase gráfica (GUI) .....	45
11. Diagrama de atención de petición .....	46
12. Diagrama de cambio de la configuración a través de un archivo de texto .....	47
13. Diagrama de cambio de la configuración a través de aplicación GUI .....	47
14. Petición de un recurso estático .....	53
15. Petición de un recurso dinámico .....	54
16. Estructura del archivo de configuración .....	61
17. Estructura del archivo de configuración después de la modificación .....	62
18. Mensaje de información (Servidor encendido) .....	63
19. Ventana de consola (Servidor encendido) .....	63
20. Administrador de Tareas de Windows .....	64
21. Aplicación GUI (Interfase gráfica del servidor) .....	65
22. Barra de herramientas .....	66
23. Ventana de configuración .....	67
24. Mensaje acerca del servidor .....	68
25. Panel de peticiones .....	69
26. Barra de estado (servidor detenido) .....	70
27. Barra de estado (servidor encendido) .....	70

# Capítulo 1

## ANTECEDENTES

### 1.1 Introducción al Internet y la WWW

Internet es una red mundial para compartir datos y transmitir información entre millones de computadores interconectados entre sí. Esto permite la interacción entre los individuos a través de computadoras y otros dispositivos sin importar las distancias que los separen geográficamente.

El concepto de red mundial se empezó a moldear en los años 60. En tiempos de guerra fría el gobierno de Estados Unidos encargó al Departamento de Defensa (DOD) construir una red militar de comunicación confiable y segura con el objetivo de que, si dejaba de funcionar un segmento de la red, se pudiera acceder a la información desde cualquier parte del país. Para esto la Agencia de Proyectos de Investigación Avanzada del Departamento de Defensa ARPA (Advanced Research Projects Agency) desarrollaría la idea de dividir los mensajes en paquetes, cada uno de los cuales sería direccionado por separado y viajaría individualmente a través de la red hasta llegar al destino final en el que se reagruparían todos los paquetes para formar el mensaje original; no importaba mucho la ruta que tome cada paquete siempre y cuando logre llegar a su destino final, lo que a su vez también aseguraba una forma adicional de seguridad en caso de que alguien interceptara la comunicación.

En 1962 *John Licklider*, científico del MIT, publicó un escrito con un concepto llamado “Red Galáctica” en el que predecía una invención futura en la que las computadoras estarían conectadas entre sí y serían accesibles por todo el mundo. Por las mismas fechas se estaban realizando otras investigaciones dentro del mismo campo:

- Por un lado *Leonard Kleinrock*, también del MIT, estaba trabajando en la idea de una red basada en el intercambio de paquetes y publicó el primer documento sobre esta teoría
- Por otro lado *Paul Baran* de RAND publica un documento sobre Redes de Comunicación Distribuidas
- El Laboratorio Nacional de Física en Inglaterra NPL (National Physical Laboratory) también había estado experimentando con una red que usaba líneas telefónicas de 768 kbps (kilobits por segundo).
- En 1965 ARPA realiza un experimento que resultó en la construcción de la primera red de área amplia WAN (Wide Area Network) al interconectar la Universidad de Berkeley y el MIT usando las líneas de teléfono.

En 1969, una vez que se consiguió reunir toda la investigación realizada por estos diferentes científicos, se construye la Red de la Agencia de Proyectos de Investigación Avanzada (ARPANET). Esta red en un inicio contó solo con cuatro nodos: la Universidad de California en Los Ángeles (UCLA), el Instituto de Investigación de Stanford (SRI), la Universidad de California Santa Barbara (UCSB) y la Universidad de Utah. Posteriormente se empezó a añadir más nodos a esta red, incluyendo universidades en el Reino Unido y Noruega.

Los propósitos militares se desligaron de ARPANET dando lugar a una nueva red denominada MILNET. Luego la Fundación Nacional de Ciencia (NSF) creó su propia red informática llamada NSFNET, que más tarde absorbería a ARPANET, creando así una gran red con propósitos científicos y académicos. Fue tan grande el desarrollo de estas tecnologías que se llegó a crear nuevas redes de libre acceso que más tarde se unirían a NSFNET para dar base a lo que hoy conocemos como INTERNET.

El Internet tal como lo conocemos hoy en día nos brinda varios servicios, entre los cuales contamos con:

- *Correo electrónico (e-mail)*. Nos permite comunicarnos con cualquier persona a través del envío y el recibo de correo electrónico.
- *Telnet o acceso remoto*. Nos permite manejar un computador remoto a través de comandos que damos desde un computador local.
- *Transferencia de archivos*. A través del protocolo de transferencia de archivos FTP (File Transfer Protocol) podemos compartir archivos.
- *WWW (World Wide Web)*. Nos permite localizar y presentar el contenido de una página web como información de varios tipos.

La WWW es una red mundial de servidores de documentos compuestos por texto, imágenes, sonido, animaciones y videos. Para navegar a través de esta red mundial es necesario que estos documentos estén enlazados o vinculados unos con otros a través de hiperlinks<sup>1</sup>, los cuales trabajan en base a localizadores únicos de recursos URL (Uniform Resource Locator). Estos localizadores identifican qué computador dentro de esta gran red tiene el recurso que estamos solicitando, el directorio en el que se encuentra y el nombre de archivo específico del recurso.

La base de la WWW y de la construcción de este tipo de documentos es el lenguaje de etiquetas HTML que nos permite especificar el formato de los distintos elementos contenidos en una página web que van a ser visualizados a través de un navegador o browser. El intercambio de la información que contienen las páginas web entre de dos computadores se realiza a través del protocolo de transferencia de hipertextos HTTP (HyperText Transfer Protocol). Esta transferencia de datos o comunicación entre dos computadores se inicia cuando un computador cliente hace una petición a un computador servidor de páginas web, el cual atiende la petición mediante el envío de la página web solicitada y de todos los elementos contenidos en ella (imágenes, sonidos, animaciones, etc.).

---

<sup>1</sup> Hipervínculos que permiten conectar un documento web con otro recurso en la red, ya sea una página web, imágenes, programas, etc.

## 1.2 Java y sus características principales

### 1.2.1 Antecedentes históricos

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems<sup>2</sup>. Basándose en la sintaxis de C++<sup>3</sup>, su creador James Gosling desarrolló un lenguaje de programación para manejar dispositivos electrónicos inteligentes como microondas y televisión interactiva, al cual lo bautizó con el nombre de Oak en 1991. Era necesario para esto contar con un lenguaje confiable y que permita un fácil desarrollo de aplicaciones. Este nuevo lenguaje presentaba características que mejoraban las deficiencias que Gosling encontró en C y C++ a la hora de desarrollar una aplicación para esta clase de dispositivos. Se cambió el nombre a Java debido a que el nombre propuesto por Gosling ya pertenecía a otro lenguaje de programación usado con anterioridad.

Este proyecto llevado a cabo por Sun Microsystems y bautizado bajo el nombre Green presentó dificultades desde su inicio debido a que el mercado para este tipo de dispositivos inteligentes no estaba evolucionando como la compañía había previsto. Pese a esto, en el año 1993, el proyecto tomó una perspectiva distinta debido al impacto que había causado la aparición de la World Wide Web y el Internet. Entonces se visionó la potencialidad de Java como lenguaje de programación web, lo que hizo resurgir al proyecto Green.

Java permite a los programadores desarrollar páginas web con contenido dinámico e interactivo, desarrollar aplicaciones empresariales de gran escala, mejorar la funcionalidad de los servidores web, proveer aplicaciones para dispositivos electrónicos de consumo como teléfonos celulares y PDA<sup>4</sup>, entre otras utilidades.

---

<sup>2</sup> Empresa norteamericana fundada en 1982 por Andreas Bechtolsheim, Vinod Khosla, y Scott McNeally que se dedica a construir hardware y software para computadoras.

<sup>3</sup> Lenguaje de programación de alto nivel desarrollado por Bjarne Stroustrup en los laboratorios Bell en la década de los 80.

<sup>4</sup> Agenda electrónica personal.

## 1.2.2 Características de Java

Java es un lenguaje de programación que nos brinda varias ventajas entre las que podemos señalar las siguientes:

- Simplicidad

Java ofrece la funcionalidad de un lenguaje potente como lo es C++, pero elimina las características no muy usadas y de difícil implementación de éste. Además cuenta con características muy útiles que no presentan C o C++ como lo es el liberador de memoria (garbage collector) que se ejecuta como un thread<sup>5</sup> de baja prioridad y evita al programador el tener que estar pendiente del uso adecuado de memoria. La simplicidad de Java facilita el desarrollo de sistemas sin que el programador tenga que preocuparse por detalles de bajo nivel.

- POO (Programación Orientada a Objetos)

La programación orientada a objetos es un tipo especial de programación que combina estructuras de datos con funciones para crear objetos reutilizables. Java maneja los conceptos básicos de la programación orientada a objetos. Estos son: el encapsulamiento, la herencia y el polimorfismo. Este método de programación permite crear componentes de software reutilizables que vuelven el proceso de construir software más rápido, correcto y económico.

- Distribuido

Java contiene extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos tales como el HTTP y el FTP, lo que facilita el desarrollo de aplicaciones distribuidas, es decir, que puedan correr mediante la interacción entre módulos que se encuentran dispersos en varias máquinas.

---

<sup>5</sup> Proceso que forma parte de un programa que se ejecuta independientemente.

- Robusto

Java realiza verificaciones del código tanto en tiempo de compilación como en tiempo de ejecución. Además la comprobación de tipos en Java permite detectar errores lo antes posible durante el ciclo de desarrollo. Java también obliga la declaración explícita de métodos y variables para reducir el número de errores. Maneja la memoria de forma transparente para el programador. También implementa arreglos con verificación de límites para evitar la corrupción de datos en memoria. Finalmente realiza una verificación de los códigos de bytes<sup>6</sup> para asegurar el funcionamiento de la aplicación. Todas estas características reducen el tiempo de desarrollo y simplifican el proceso de testing de las aplicaciones.

- Arquitectura neutral

Un programa compilado de Java genera un código de máquina virtual denominado código de bytes, el cual tiene un formato independiente de la arquitectura en la que va a ser ejecutado. Este código de máquina virtual es interpretado por un sistema de tiempo de ejecución que sí depende la arquitectura de la máquina. Este sistema de ejecución se denomina JVM (Java Virtual Machine). “Este concepto de máquina virtual adopta un enfoque de capas y trata el núcleo del sistema operativo y el hardware como si todo fuera hardware”<sup>7</sup>. Esta característica permite desarrollar sistemas multiplataforma.

- Seguro

Java cuenta con características adicionales que evitan errores fatales en el sistema operativo debido a que el programador no tiene la posibilidad de manejar directamente la memoria, además el compilador de Java asegura que los códigos de bytes no contengan fragmentos de código ilegal. Esto permite reducir el número de errores de programación.

---

<sup>6</sup> Código de máquina virtual que es interpretado por el intérprete Java. Este código no es entendido directamente por el hardware de la máquina.

<sup>7</sup> SILBERSCHATZ Abraham, GALVIN Peter, Sistemas Operativos, Prentice Hall, 5ta edición, página 83.



- Portabilidad.

Al ser Java un lenguaje interpretado y de arquitectura neutral permite a los desarrolladores ejecutar sus programas en cualquier plataforma que cuente con la máquina virtual de Java JVM. Actualmente existen sistemas de tiempo de ejecución de Java para Solaris 2.x, SunOs 4.1.x, Windows 95, Windows NT, Linux, Irix, Aix, Mac, Apple.

- Multithread.

A través de los threads Java permite ejecutar varias actividades en paralelo dentro de un solo programa. Los threads o hilos de ejecución son básicamente pequeños procesos independientes de un gran proceso. El beneficio que presenta la programación multithreading consiste en mejorar el rendimiento interactivo y comportamiento en tiempo real de un programa.

- Dinámico.

Java se beneficia todo lo posible de la tecnología orientada a objetos, por eso es que no intenta conectar todos los módulos que comprenden una aplicación hasta el tiempo de ejecución. Esto permite utilizar componentes independientes de código con lo que el tiempo de desarrollo disminuye.

### 1.3 Descripción básica del entorno de trabajo de los servidores web

Los servidores web hacen posible el Internet. Todas las computadoras en el Internet son o bien servidores o bien clientes. Las máquinas que proveen servicios a otras máquinas son servidores. Las máquinas que se conectan a los servidores son clientes. Existen servidores web, de correo electrónico, de transferencia de archivos, y de muchos tipos más que sirven para distintos propósitos.

Cuando nos conectamos a un servidor para leer una página web, el servidor busca la página que queremos y la envía al cliente (browser). Un servidor tiene una dirección IP estática que no cambia muy a menudo. Una computadora que se conecta al Internet con un módem, típicamente tiene una dirección IP que es asignada por el proveedor de Internet cada vez que se conecta al proveedor.

Cualquier servidor provee los servicios haciendo uso de números de puerto. Cada servicio que está disponible en el servidor tiene un número de puerto asignado. Por ejemplo, cuando un servidor tiene disponibles un servidor web y un servidor de transferencia de archivos (FTP), el servicio web típicamente estará asignado al puerto 80, y el servidor FTP estará asignado al puerto 21. Los clientes se conectan a un servicio con una dirección IP específica y a un puerto determinado.

Una vez que el cliente se ha conectado a un servicio en un puerto del servidor, accede a este servicio con un protocolo específico. Los protocolos definen ciertas reglas a través de las cuales el cliente y el servidor pueden tener una conversación. Cada servidor web en el Internet se acoge al protocolo HTTP (Hypertext Transfer Protocol) o protocolo de transferencia de hipertexto.

### 1.3.1 Proceso de comunicación con un servidor web

En el siguiente diagrama podemos ver el proceso general de comunicación entre un cliente y un servidor. El cliente se conecta al Internet para descargar una página web que se encuentra en un servidor remoto.

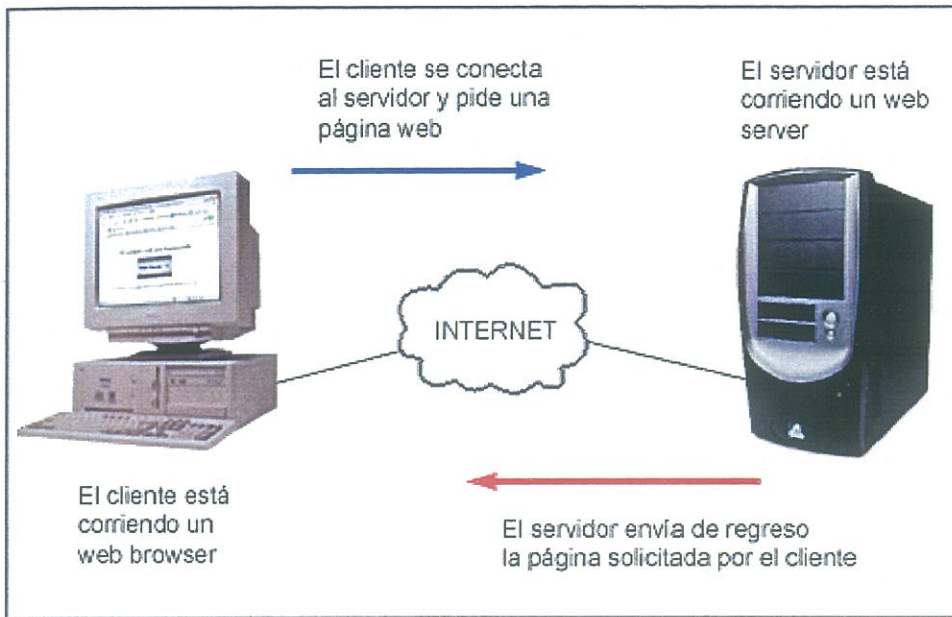


Figura 1. Comunicación cliente-servidor

Un navegador web (por ejemplo Microsoft Internet Explorer o Netscape Navigator) establece una conexión con un servidor web (por ejemplo el servidor IIS de Microsoft o el servidor Apache). Una vez establecida la conexión entre ambos, el cliente pide una página y el servidor responde enviándole la misma. Para conocer todos los por menores que implica esta comunicación entre cliente y servidor daremos un ejemplo. Un estudiante desea conocer el sitio web de la Universidad San Francisco de Quito. Para que él pueda ver el sitio en su navegador web deben darse los siguientes pasos:

1. El estudiante escribe una dirección URL en un campo que provee el navegador web para este propósito, por ejemplo supongamos que la dirección es <http://www.usfq.edu.ec/index.html> (sitio web de la Universidad San Francisco de Quito).

2. El estudiante presiona algún botón del navegador para que éste se conecte con el sitio web de la Universidad San Francisco de Quito.
3. El navegador web separa la dirección URL en tres partes:
  - El protocolo (http)
  - El nombre del servidor (www.usfq.edu.ec)
  - El nombre del archivo (index.html)
4. El navegador web se conecta con un servidor de nombres de dominio (DNS) el cual traduce el nombre del servidor a una dirección IP, la cual se usa para conectarse al servidor web.
5. El navegador entonces crea una conexión con el servidor que alberga el sitio web de la Universidad.
6. Mediante el uso del protocolo HTTP, el navegador construye un mensaje de petición en el cual incluye el nombre del archivo que desea obtener. En este caso sería la página de inicio del sitio web (index.html).
7. Al llegar el mensaje de petición al servidor web, éste lo lee para determinar qué archivo esta siendo solicitado por el cliente.
8. El servidor construye un mensaje de respuesta en el que incluye el contenido (código HTML) del archivo solicitado y lo envía al cliente.
9. El navegador recibe el mensaje de respuesta enviado por el servidor, lo lee y finalmente presenta la página en la pantalla.

La mayoría de los servidores agregan cierto nivel de seguridad al proceso. Por ejemplo, si usted se ha encontrado con páginas que piden un nombre de usuario y contraseña, entonces la página está protegida. El servidor permite que el administrador de

la página mantenga una lista de nombres y contraseñas para aquellas personas que tienen permiso de acceso a la página.

Algunos servidores más avanzados adicionalmente agregan un mayor nivel de seguridad al encriptar el contenido de la comunicación que viaja a través de la conexión hacia el cliente. De esta manera información sensible, como números de tarjetas de crédito, puede ser enviada a través del Internet.

### **1.3.2 Modelos de comunicación cliente-servidor**

Existen distintos procesos que se pueden dar dentro de la comunicación entre un cliente y un servidor. Esto depende del tipo de página que el cliente está solicitando.

Hay dos tipos de páginas que pueden ser procesadas por un servidor web:

- Páginas con contenido estático
- Páginas con contenido dinámico

Las páginas estáticas son aquellas que no pueden cambiar de contenido, a menos que el autor edite la página.

Las páginas dinámicas son aquellas que generan su contenido cada vez que algún cliente las solicita. Este tipo de páginas se usan por ejemplo en situaciones en las que se quiere acceder a la información contenida en una base de datos del servidor, o cuando se realiza una búsqueda en el Internet y se debe generar dinámicamente una página con los resultados de la misma.

De acuerdo a estos dos conceptos podemos distinguir tres modelos de comunicación:

- El primero es aquel en el que un cliente solicita una página estática (archivo HTML) que está almacenada en un computador remoto y que debe ser enviada al cliente por una aplicación servidor que se encarga de darle este recurso.
- El segundo modelo está relacionado con respuestas dinámicas. En este modelo el recurso solicitado por el cliente es un programa o script CGI almacenado en el servidor. Este programa puede ser escrito en C, C++, Perl, Visual Basic, etc. Por ejemplo cuando realizamos una búsqueda de información en el Internet sobre sitios web relacionados con las palabras “curso de inglés”, el cliente envía estas palabras de búsqueda al servidor solicitando que un programa encuentre los resultados adecuados. El servidor inicia el programa CGI y le pasa los datos enviados por el cliente para que éste realice la búsqueda. El programa le devuelve al servidor como resultado una página HTML con los resultados de la búsqueda, y el servidor envía esta página al cliente.
- El tercer modelo también está relacionado con respuestas dinámicas, pero en este caso estas respuestas son generadas por el uso de tecnologías del lado del servidor que incluyen páginas dinámicas. Dos son las características fundamentales de éstas páginas: la primera es que combinan código HTML con otras instrucciones escritas en algún lenguaje de programación, para generar dinámicamente código HTML, y la segunda es que éstas páginas son descifradas por un intérprete que procesa las instrucciones. Algunas de las tecnologías más importantes son:
  - *Active Server Pages (ASP)*: Tecnología Microsoft. Estas páginas son tienen extensión *.asp* y contienen código VBScript.
  - *Personal Home Pages (PHP)*: Tecnología Open Source. Una página PHP puede tener una extensión *.php* o *.php3*.
  - *Java Server Pages (JSP)*: Tecnología Java. Páginas con extensión *.jsp* que contienen código Java.

## 1.4 Análisis de Requerimientos

Como podemos ver las tecnologías mencionadas anteriormente se encuentran íntimamente unidas, siendo Java una de las mejores herramientas para desarrollo de sistemas relacionados o basados en Internet.

La globalización y la necesidad de las empresas de aumentar la productividad de sus empleados y de reducir costos a través de la automatización de procesos y el compartimiento de información, hace que cada día la industria de tecnología o IT requiera un mayor número de personas que cuenten con un conocimiento avanzado en nuevas tecnologías, como Java, que permitan el desarrollo de aplicaciones distribuidas o que manejen información a través de Internet.

A nivel nacional no se cuenta con el suficiente número de profesionales en desarrollo de software que tengan un nivel avanzado de conocimientos en tecnología Java. Además no se cuenta con un método de enseñanza adecuado para proveer este tipo de conocimientos.

Un sistema en el que se pueda aplicar este tipo de conocimientos es un servidor Web modular que facilite el proceso de enseñanza y que además sea capaz de demostrar las ventajas de la tecnología Java de una manera práctica y sencilla en un taller de programación.

Debido a que los profesionales de programación carecen de tiempo para asistir a un curso presencial, es necesario brindar la posibilidad de que el contenido del taller pueda ser entendido mediante un proceso de autoaprendizaje.

## 1.4.1 Necesidades

Las necesidades del taller son:

- Accesar a la información del currículum
- Permitir el autoaprendizaje
- Presentar la información de una manera organizada

Las necesidades del servidor web son:

- Desarrollar mediante lenguaje Java
- Permitir la modularidad
- Facilitar la configuración
- Multiplataforma
- Seguridad
- Sencillo y ligero

## 1.4.2 Descripción de Requerimientos

Los requerimientos del taller son:

- Utilización de herramientas de desarrollo interactivo multimedia para realizar una presentación interactiva del currículum.
- Contenido distribuido por capítulos, es decir que el currículum debe estar organizado en capítulos de complejidad creciente.



Los requerimientos del servidor son:

- Es requisito utilizar el lenguaje de programación orientado a objetos Java ya que es la base de enseñanza del taller. Por esta razón, el servidor web se realizará en Java.
- El servidor debe desarrollarse de una manera modular para facilitar el proceso de enseñanza.
- El servidor debe contar con una interfase gráfica que permita controlar su funcionamiento y facilite la comprensión del proceso de atención de peticiones.
- El servidor debe contar con un módulo que nos permita configurar, mediante una interfase gráfica, sus distintos parámetros.
- El servidor debe poder ejecutarse en distintas plataformas.
- El servidor debe mantener un archivo en el que se registren los accesos de clientes web.
- Es necesario evitar la complejidad y la abundancia del código en el desarrollo del servidor.

### **1.4.3 Definición de entregables**

- Subsistema de atención de conexiones.

Permite procesar nuevas peticiones, es decir, crea una conexión con el cliente cuando éste realiza una petición. Define la creación de la respuesta a las peticiones del cliente.

Es la interfase entre el cliente y el procesamiento del servidor

- Subsistema de procesamiento de protocolo HTTP.

Valida y procesa los distintos tipos de peticiones del protocolo HTTP presentados por el módulo de conexiones y entrega los resultados al módulo de conexiones. Además prepara las páginas que el cliente pide.

- Subsistema de registro de accesos.

Maneja el registro de los usuarios que acceden al servidor, de las peticiones que realizan, y del origen de los mismos. Los registros pueden ser guardados en una base de datos o en un archivo de texto.

- Subsistema de configuración del servidor.

Maneja los parámetros de configuración del servidor, tales como número de puerto, número de conexiones, directorio raíz, página inicial, etc.

- Subsistema de control.

Maneja la interacción entre todos los subsistemas permitiendo que se manejen varias peticiones en paralelo.

- Subsistema de interfase gráfica.

Presenta una aplicación GUI que permite controlar el funcionamiento del servidor.

- CD interactivo multimedia

Es un CD que contiene la materia del taller dividida en capítulos, cada uno de los cuales explica los pasos para desarrollar el servidor. La materia del taller se presenta a través de una aplicación interactiva, la cual guiará al estudiante en la construcción del servidor web.

## Capítulo 2

# MARCO TEORICO

En este capítulo trataremos algunos conceptos que nos ayudarán a entender de mejor manera el proceso de comunicación entre un cliente y un servidor web a través del protocolo HTTP, lo cual nos dará una base de conocimiento sobre la cual podremos construir nuestro servidor.

Es necesario contar con esta base de conocimiento para comprender la estructura del servidor y la función de cada uno de los módulos que lo componen.

En la primera parte de este capítulo analizaremos varios aspectos relacionados con el protocolo HTTP, y en la segunda parte veremos como se fusionan estos conceptos con los módulos que estructuran nuestro servidor.

### 2.1 Protocolo de transferencia de Hipertexto HTTP

HTTP (Hypertext Transfer Protocol) es un protocolo de nivel de aplicación para sistemas de información hipermedia<sup>1</sup>. Este protocolo funciona en base a peticiones o mensajes tipo *request* realizadas por un cliente a un servidor, y en base a las respectivas respuestas o mensajes tipo *response* que el servidor envía al cliente.

---

<sup>1</sup> Sistemas con recursos multimedia tales como páginas web, archivos de texto, de audio, de video, etc.

## 2.1.1 Funcionamiento del Protocolo HTTP

Una comunicación HTTP es iniciada por un cliente o agente de usuario (generalmente éstos clientes son aplicaciones como el Microsoft Internet Explorer o cualquier aplicación o herramienta final de usuario) al solicitar un recurso que se encuentra en un computador en el que se está ejecutando una aplicación servidor. El servidor puede estar corriendo en el mismo computador que inicia la comunicación o en un computador remoto conectado al cliente a través de una red de datos.

El cliente envía una petición sobre un recurso que se encuentra en el servidor en forma de un mensaje tipo request. Este mensaje tiene una estructura definida formada por una línea de identificación del mensaje, un encabezado en el que se pueden incluir ciertas variables y parámetros que manejan el intercambio de información y facilitan la comunicación entre el cliente y el servidor, y un cuerpo del mensaje que contiene la información que se desea transmitir.

Más adelante se explicará con mayor detalle la estructura del mensaje tipo request, sin embargo, cabe mencionar en este momento que la primera línea que forma parte del mensaje tipo request contiene tres partes separadas por un espacio en blanco: el método que se va a aplicar sobre el recurso, una identificación del recurso mediante un URI (Uniform Resource Identifier), y la versión del protocolo HTTP que se está utilizando para determinar el formato del mensaje.

El mensaje de petición emitido por el cliente viaja a través de una conexión TCP/IP establecida entre las aplicaciones cliente y servidor mediante un circuito virtual de la capa de transporte.

Una vez que el mensaje llega hasta el computador que alberga el programa servidor, éste lee la primera línea del mensaje de petición del cliente para verificar si puede ser procesado. En el caso de que el mensaje no se haya corrompido durante su viaje entre origen y destino, éste es procesado y el servidor emite una respuesta a través de un mensaje tipo response que encapsula el recurso o la información requerida por el cliente.

Este mensaje también cuenta con una estructura definida formada por una línea de identificación del mensaje, un encabezado y un cuerpo que, al igual que el otro tipo de mensaje, será analizado con mayor detalle más adelante. Cabe mencionar también que la primera línea que forma parte del mensaje tipo response, llamada línea de estado, contiene también tres partes separadas por un espacio en blanco: la versión del protocolo, el código de estado de la respuesta que le permite determinar al cliente si la petición fue atendida con éxito por el servidor o se produjo algún error, y una frase que describe el significado del código de estado.

El mensaje de respuesta emitido por el servidor viaja a través de la conexión TCP/IP hacia el cliente. Después de que el mensaje de respuesta llega al cliente, el servidor cierra la conexión TCP/IP establecida entre las aplicaciones.

## 2.1.2 Identificadores de recursos

Los URI (Uniform Resource Identifier) son cadenas de caracteres que tienen cierto formato para identificar un recurso a través de un nombre, ubicación o cualquier otra característica particular del mismo. Estos identificadores forman una parte muy importante dentro del protocolo HTTP ya que son utilizados para localizar recursos dentro de una red.

Los URI pueden ser considerados como direcciones únicas que representan distintos tipos de recursos en una red, los cuales pueden ser simples archivos como imágenes, archivos de texto o páginas web, pueden ser directorios, o pueden representar tareas complejas como accesos a una base de datos o búsquedas en Internet.

La estructura básica de un identificador uniforme de recurso es la siguiente:

*protocolo :// dirección : puerto / ruta / archivo ? datos*

- *protocolo*.- El nombre del protocolo que se va a utilizar. Este puede ser:
  - *http*: Permite a un enlace acceder a un recurso remoto.
  - *file*: Permite a un enlace acceder a un archivo en el disco local del computador.
  - *ftp*: Se utiliza para transferir archivos desde un computador remoto hacia el computador local o viceversa.
  - *gopher*: Permite acceder a un servidor gopher.
  - *mailto*: Utiliza el protocolo SMTP y permite a un enlace enviar un mensaje de correo electrónico.
  - *news*: Permite acceder a un artículo o grupo de noticias USENET.
  - *telnet*: Provee los medios necesarios para que un enlace pueda iniciar una sesión telnet en un computador remoto.
  
- *dirección*.- Está compuesta por tres partes separadas por un punto:
  - El nombre del computador (como por ejemplo www)
  - El nombre del dominio (puede ser una dirección IP)
  - El tipo de dominio ( como por ejemplo com, net, org, edu, mil, gov, tv)
  
- *puerto*.- Es el número de puerto o socket TCP a través del cual se va a realizar la comunicación entre el cliente y el servidor. Por defecto el número de puerto es 80.
  
- *ruta*.- El camino, path o directorio en el que se localiza el recurso.
  
- *archivo*.- El nombre del recurso que solicita el cliente. Este nombre esta compuesto por dos partes separadas por un punto:
  - Nombre: Una cadena de caracteres (por ejemplo index, home, etc.)
  - Extensión: El tipo de archivo (por ejemplo html, gif, cgi, jsp, etc.)

- *datos*.- Una cadena de variables con sus respectivos valores que se pasan al recurso para ser procesados. Esta cadena tiene el siguiente formato:

variable\_1=valor\_1&variable\_2=valor\_2&variable\_3=valor\_3

- El nombre de la variable va seguido de un igual y del valor respectivo
- Cada unidad (nombre=valor) esta separada de una siguiente unidad con el signo ampersand (&)
- Si el valor de la variable tiene espacios en blanco, estos son reemplazados por signos de suma (+).

A continuación presentamos algunos ejemplos prácticos de estos identificadores de recursos:

- ftp://ftp.is.co.za/rfc/rfc1808.txt
- gopher://spinaltap.micro.umn.edu/00/Weather/California
- http://www.usfq.edu.ec/index.htm
- mailto:gerbasti@hotmail.com
- news:comp.infosystems.www.servers.unix
- telnet://melvyl.ucop.edu/
- http://abc.com:80/~smith/home.html
- http://ABC.com/%7Esmith/home.html
- http://ABC.com:/%7esmith/home.html

Las tres últimas direcciones apuntan al mismo recurso (home.html). Cabe señalar que en estos identificadores los caracteres especiales pueden ser representados mediante la codificación “% HEX HEX”. En esta codificación la palabra HEX representa un número hexadecimal. Como resultado de esta codificación el carácter “~” es similar a “%7E”.

## 2.1.3 Mensajes HTTP

### 2.1.3.1 Tipos de mensajes

Existen dos tipos de mensajes HTTP: los mensajes de petición de los clientes denominados *request* y los mensajes de respuesta del servidor denominados *response*. Ambos tipos de mensaje utilizan un formato genérico definido por el RFC 822 que consiste de una línea de inicio o identificación, cero o más encabezados, una línea en blanco que indica el final del encabezado y un cuerpo del mensaje en el caso que sea necesario.

*Mensaje genérico = línea de inicio*

*header1= valor*

*header2= valor*

*header3= valor*

*header4= valor*

*Cuerpo del mensaje*

### 2.1.3.2 Encabezado de los mensajes HTTP

Existen varios tipos de encabezados HTTP, entre los cuales podemos citar los siguientes:

- Encabezado general.- Los campos que se incluyen dentro de este encabezado son aplicables tanto a los mensajes request como a los mensajes response y son los siguientes:
  - Cache-Control
  - Connection
  - Date
  - Pragma
  - Trailer



○ Transfer-Encoding

- Encabezado request.- Los campos que se incluyen dentro de este encabezado los veremos en la sección 2.1.4.2.
- Encabezado response.- Los campos que se incluyen dentro de este encabezado los veremos en la sección 2.1.5.2.
- Encabezado de la entidad.- Los campos que se incluyen dentro de este encabezado los veremos en la sección 2.1.3.4.

Cada uno de los campos del encabezado está formado por el nombre del campo, seguido por el caracter dos puntos (:), un espacio en blanco, el valor del campo y finalmente un salto de línea. Los nombres de los campos pueden ser escritos en mayúsculas o minúsculas (case-insensitive).

*Campo de encabezado = nombre: valor*

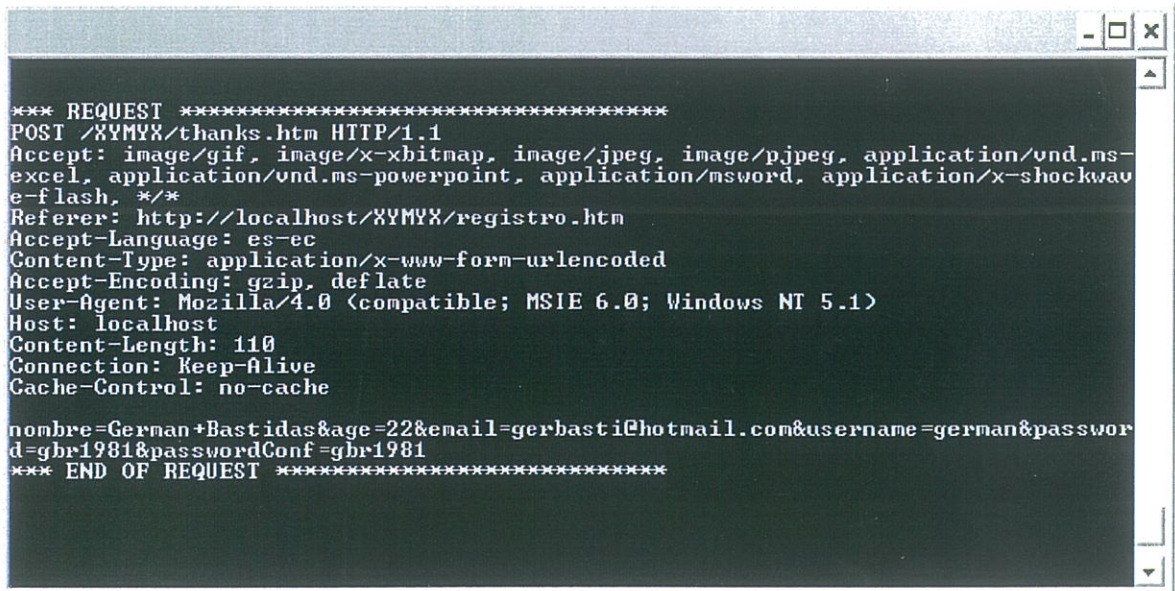
A continuación presentamos algunos ejemplos de campos que van incluidos en un encabezado HTTP:

```
Accept: */*
Server: webserver/1.0
Date: Fri Jan 09 16:17:24 COT 2004
Location: 192.188.53. 6
Content-Type: image/gif
Content-Lenght: 760
```

### 2.1.3.3 Cuerpo del mensaje HTTP

El cuerpo de un mensaje HTTP no es una parte necesaria que debe ser incluida en la estructura del mensaje. Sin embargo tenemos dos casos en los que se incluye información dentro del cuerpo del mensaje:

1. El primer caso es cuando el cliente envía datos al servidor para que estos sean procesados por una aplicación. Para entender esto podemos imaginar una situación de registro en un sitio web en la que un usuario llena sus datos personales en un formulario y los envía a una aplicación que reside en el servidor. Esta aplicación podría ser un programa escrito en PHP o un servlet escrito en Java. Existen muchos tipos de recursos que pueden procesar los datos enviados por un cliente. Los campos del formulario con sus respectivos valores serán incluidos en el cuerpo de un mensaje tipo request siempre y cuando el método HTTP que se utiliza para enviar el formulario sea POST. Existe otro método para enviar información de un formulario que se denomina GET, el cual envía los datos del como parte del URL. A continuación podemos ver dos ejemplos reales en los que se envía datos de un formulario a un servidor.

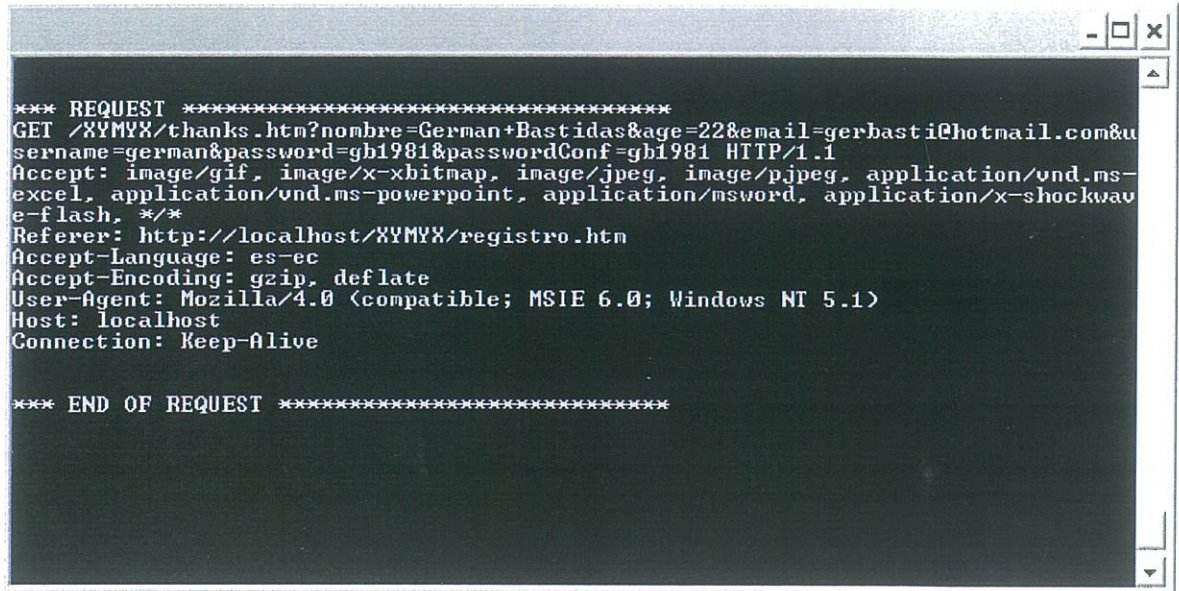


```
*** REQUEST ****
POST /XYMYX/thanks.htm HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-shockwave-flash, */*
Referer: http://localhost/XYMYX/registro.htm
Accept-Language: es-ec
Content-Type: application/x-www-form-urlencoded
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: localhost
Content-Length: 110
Connection: Keep-Alive
Cache-Control: no-cache

nombre=German+Bastidas&age=22&email=gerbasti@hotmail.com&username=german&password=gbr1981&passwordConf=gbr1981
*** END OF REQUEST ****
```

Figura 2. Mensaje request con método POST

En la figura 2 podemos observar un mensaje tipo request en el que se solicita un recurso (thanks.htm) mediante el método POST. Los datos del formulario son enviados como parte del cuerpo del mensaje.



```
*** REQUEST *****
GET /XYMYX/thanks.htm?nombre=German+Bastidas&age=22&email=gerbasti@hotmail.com&
sername=german&password=gb1981&passwordConf=gb1981 HTTP/1.1
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, application/vnd.ms-
excel, application/vnd.ms-powerpoint, application/msword, application/x-shockwav
e-flash, */*
Referer: http://localhost/XYMYX/registro.htm
Accept-Language: es-es
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1)
Host: localhost
Connection: Keep-Alive

*** END OF REQUEST *****
```

Figura 3. Mensaje request con método GET

En la figura 3 podemos observar un mensaje tipo request en el que se solicita un recurso (thanks.htm) mediante el método GET. Los datos del formulario son añadidos al URL.

En los ejemplos anteriores podemos ver claramente la estructura de los mensajes HTTP: la línea de inicio, los headers y el cuerpo del mensaje. Recordemos que el encabezado del mensaje está separado del cuerpo mediante una línea en blanco. En la figura 3 podemos ver que, a pesar de no tener datos en el cuerpo del mensaje, es necesario agregar la línea en blanco que separa el encabezado del cuerpo para que la aplicación que reciba el mensaje, en este caso un servidor web, sepa donde termina el encabezado del mismo.

2. El segundo caso es cuando el servidor envía un mensaje de respuesta que contiene en su cuerpo el recurso solicitado por el cliente. Este es el caso más común en el que se incluye información en el cuerpo de un mensaje HTTP. Imaginemos que un

cliente solicita una página web muy simple que contiene únicamente código HTML. El cliente envía un mensaje tipo request solicitando este recurso que se encuentra en el servidor. El servidor revisará el mensaje de petición del cliente, lo analizará y responderá si corresponde con un mensaje tipo response que contiene la página web solicitada por el cliente. A continuación podemos observar el mensaje tipo response que el servidor envía al cliente como respuesta a su solicitud.

```
*** RESPONSE ****
HTTP/1.1 200
Date: Fri Jan 09 19:44:33 COT 2004
Server: webserv1.0
Connection: Keep-Alive
Location: localhost
Content-Type: text/html
Content-Length: 493
Last-Modified: Fri Jan 09 19:44:33 COT 2004

<html>
<head>
<title>Servidor Web 1.0</title>
</head>
<body>
<table width="100%" height="100%">
  <tr>
    <td align="center" valign="middle">
      <hr>
      <h1>El servidor web esta funcionando</h1>
      
      <hr>
      Si usted puede ver esta página, entonces el Servidor Web esta
      corriendo correctamente
      <hr>
    </td>
  </tr>
</table>
</body>
</html>
*** END OF RESPONSE ****
```

Figura 4. Mensaje response que contiene un archivo HTML

Podemos ver en la figura 4 que en el cuerpo del mensaje se encuentra el contenido del archivo HTML solicitado por el cliente. Como este es un archivo de texto podemos entender claramente el contenido del mismo.

Si observamos el contenido del archivo HTML podemos ver que existe una imagen que forma parte del mismo. Esta imagen es otro recurso que se encuentra en el servidor y por tanto el cliente debe enviar un mensaje tipo request al servidor para obtener esta imagen. La respuesta del servidor correspondiente a esta petición del archivo "logo.gif" se muestra a continuación.

```

*** RESPONSE *****
HTTP/1.1 200
Date: Fri Jan 09 19:52:00 COT 2004
Server: webserver/1.0
Connection: Keep-Alive
Location: localhost
Content-Type: image/gif
Content-Length: 1191
Last-Modified: Fri Jan 09 19:52:00 COT 2004

?%?>? ?ff?_?↓↓?L?>?▼▼?▼▼?""?_?/?%?%?--?%?L? , , ? . ?/? ?00?33?66?
%?; ; ?<?????:? , ?99?>>11?CC?44?JJ?CC?88?99?::? ; ?<<?KK????FF?CC?RR?II?LL?\?\?II
?QQ?\?\?UU?dd?ZZ?cc?ll?dd?kk?ii?oo?rr?uu?tt?xx?pp? `?zz? ! ! ?tt?uv?ΔΔ?zz?ii?~"?>??
? ?ss?yy?jj?????<<????rr?uu????ww?yy????>?????????ww????>????????????
????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????
????????????????????????????????????????????????????????????????????????
MMMMKKKGGDDDBBB??==999666555222111//&&://%▲▲▲---SSS▶▶▶***

▲▲▲???!?◆@? , <??=↑H??!!H?▲???)#J?H?a=u?2j?????>??+I????<S?6????0c??R??8s????
?Hb?+J????H??2?????P?Jm?J?? : ?z?j?<♥^G?A?E?I?hGi?J?E)??8-↑???W?◆NS#0J?◆◆?4" ?Sc
d?@vN◆??&L◆u▲8??@L?6m??/?c◆n▲H??E
@?? 'p??3??o?&p??>@?jm?H?;?R?^??Z??U_U?
d???'u???'↓◆?45?h??<??h?@?? ????4?h??8?H?=<????@◆?d???'c??H&???'L'↓>;
*** END OF RESPONSE *****

```

Figura 5. Mensaje response que contiene un archivo GIF

En este caso el cliente solicitó un recurso de tipo binario (una imagen en formato GIF) que fue enviada por el servidor a través de un mensaje tipo response que podemos observar en la figura 5. Es interesante ver que el contenido del archivo "logo.gif" está definido por el header Content\_type, el cual le indica al cliente que este recurso es una imagen con formato GIF. Como las imágenes son archivos binarios, el contenido del cuerpo del mensaje response es un conjunto de caracteres que no tienen ningún sentido para los seres humanos, pero sí para el computador que lo está recibiendo.

#### 2.1.3.4 Entidades de los mensajes HTTP

Una entidad es la información que se incluye en el cuerpo de los mensajes request y response siempre y cuando corresponda. Los campos que se incluyen en el encabezado de un mensaje que tienen relación con la entidad son los siguientes:

- Allow
- Content-Encoding
- Content-Language
- Content-Length
- Content-Location
- Content-MD5
- Content-Range
- Content-Type
- Expires
- Last-Modified

Algunos de estos campos son opcionales y otros son requeridos. Los campos requeridos se deben incluir en el encabezado del mensaje HTTP siempre y cuando el cuerpo del mensaje incluya la entidad. Estos campos son:

- Content-Length
- Content-Type
- Last-Modified

#### 2.1.4 Mensaje Request

Los mensajes HTTP tipo request son aquellos que un cliente debe construir a través de un navegador web para enviarlos a un servidor y solicitar algún recurso. La estructura básica de este tipo de mensajes es la siguiente:

*Mensaje Request = Línea de inicio de petición*  
*Header o encabezado*  
*Línea en blanco*  
*Cuerpo del mensaje*

#### 2.1.4.1 Línea de inicio del mensaje Request

La línea de inicio de petición esta formada por tres elementos separados por un espacio en blanco:

*método [espacio] recurso [espacio] protocolo*

- *método*.- Es el primer elemento que encontramos en la línea de inicio de un mensaje tipo request y nos indica el método que se va a ejecutar sobre el recurso pedido. Los métodos definidos por el RFC 2616 son los siguientes:
  - “OPTIONS” – Este método representa una petición de información por parte del cliente acerca de las opciones de comunicación disponibles o requerimientos necesarios asociados al recurso solicitado. Además le permite al cliente obtener información acerca de las capacidades del servidor.
  - “GET” – Este método permite obtener cualquier información especificada por el recurso solicitado. Si el recurso es por ejemplo una página web estática, el servidor deberá enviarle al cliente el texto o código que contiene la página; si el recurso es un archivo binario, el servidor debe enviarle los bytes que forman el archivo binario; y si el recurso es un proceso que se debe ejecutar en el servidor, como por ejemplo un script CGI, el servidor debe devolver el resultado del proceso.

- “HEAD” – Este método es similar a GET con la única diferencia de que el servidor debe responder con un mensaje tipo response sin cuerpo. Se utiliza para obtener meta datos del recurso solicitado.
- “POST” – Este método se utiliza para enviar datos de un formulario dentro del cuerpo del mensaje request al servidor para que sean procesados por el recurso solicitado. Generalmente el recurso es un script CGI o una página dinámica. También se puede utilizar el método POST par crear nuevos recursos en el servidor.
- “PUT” – Este método sirve para almacenar un archivo en el servidor con un nombre y en un directorio específico, ambos definidos por el URL de la línea de inicio del mensaje request. El archivo que va a ser almacenado en el servidor es encapsulado en el cuerpo del mensaje.
- “DELETE” – Este método sirve para eliminar un archivo del servidor especificado por el URL de la línea de inicio del mensaje request.
- “TRACE” – Este método sirve para invocar un loop-back del mensaje de petición. Esto se utiliza para que el cliente pueda ver que información esta llegando al servidor con propósitos de testing y diagnóstico.

A parte de estos métodos se pueden definir más métodos con propósitos específicos de las aplicaciones que se estén desarrollando. Si un servidor recibe una petición con un método que no ha sido implementado debe responder con un mensaje que tenga el código de status 501 (Método no implementado)

- *recurso.*- Es el segundo elemento que encontramos en la línea de inicio de un mensaje tipo request que nos indica el recurso sobre el cual se va a ejecutar el método definido anteriormente. Este conjunto de caracteres es un identificador uniforme de recurso URI que definimos en la sección 2.1.2. Hay que recordar que el URI es transmitido con un formato definido por la codificación “% HEX HEX”



que también revisamos anteriormente. En la versión HTTP/1.1 el recurso solicitado es identificado por el URI de la línea de inicio más el valor del encabezado “Host” que determina el nombre del servidor.

- *protocolo*.- El último elemento que compone la línea de inicio del mensaje request define el nombre y la versión del protocolo. Un ejemplo sería “HTTP/1.1” en donde podemos ver que el nombre del protocolo está separado de la versión por el signo slash (/).

#### 2.1.4.2 Headers o encabezados del mensaje Request

Los campos del encabezado de un mensaje request le permiten a la aplicación cliente enviar información adicional al servidor sobre el cliente y sobre el mensaje request. Los campos que se pueden incluir dentro del encabezado son los siguientes:

- Accept
- Accept-Charset
- Accept-Encoding
- Accept-Language
- Authorization
- Expect
- From
- Host
- If-Match
- If-Modified-Since
- If-None-Match
- If-Range
- If-Unmodified-Since
- Max-Forwards
- Proxy-Authorization
- Range
- Referer

- TE
- User-Agent

No es necesario incluir todos estos campos en el encabezado de un mensaje request, sin embargo en la versión HTTP/1.1 el único header que obligatoriamente debe estar presente es "Host". Si el servidor recibe un mensaje que no contenga este campo en el encabezado, éste le devolverá al cliente un mensaje response con el código de status 400 que significa que el mensaje request no ha sido formado adecuadamente.

### 2.1.4.3 Cuerpo del mensaje Request

Como se mencionó antes, el cuerpo de un mensaje request no siempre lleva información para el servidor. Esto dependerá del método especificado en la línea de inicio del mensaje. En la sección 2.1.4.1 revisamos algunos de estos métodos request especificados en el RFC 2616 y vimos que los únicos métodos que incluían información en el cuerpo del mensaje eran POST y PUT, en el primer caso debido a que el cliente envía los datos de un formulario en el cuerpo del mensaje y en el segundo caso debido a que el cliente debe enviar el archivo que va a ser almacenado en el servidor.

### 2.1.4.4 Estructura del mensaje Request

Una vez que hemos revisado las partes fundamentales de un mensaje request vamos a observar en la figura 6 un ejemplo real de la estructura de un mensaje de petición enviado por el navegador Microsoft Internet Explorer 6.0 a nuestro servidor web. En esta figura vamos a encontrar las cuatro partes fundamentales de los mensajes de petición: la línea de identificación o inicio, el encabezado, la línea en blanco que determina el final del encabezado y el inicio del cuerpo, y finalmente el cuerpo del mensaje. Cada una de estas partes está diferenciada claramente con un color específico:

- La línea de inicio es de color verde
- El nombre de los campos del encabezado es de color rojo
- El valor del campo de encabezado es de color negro
- El cuerpo del mensaje es de color azul

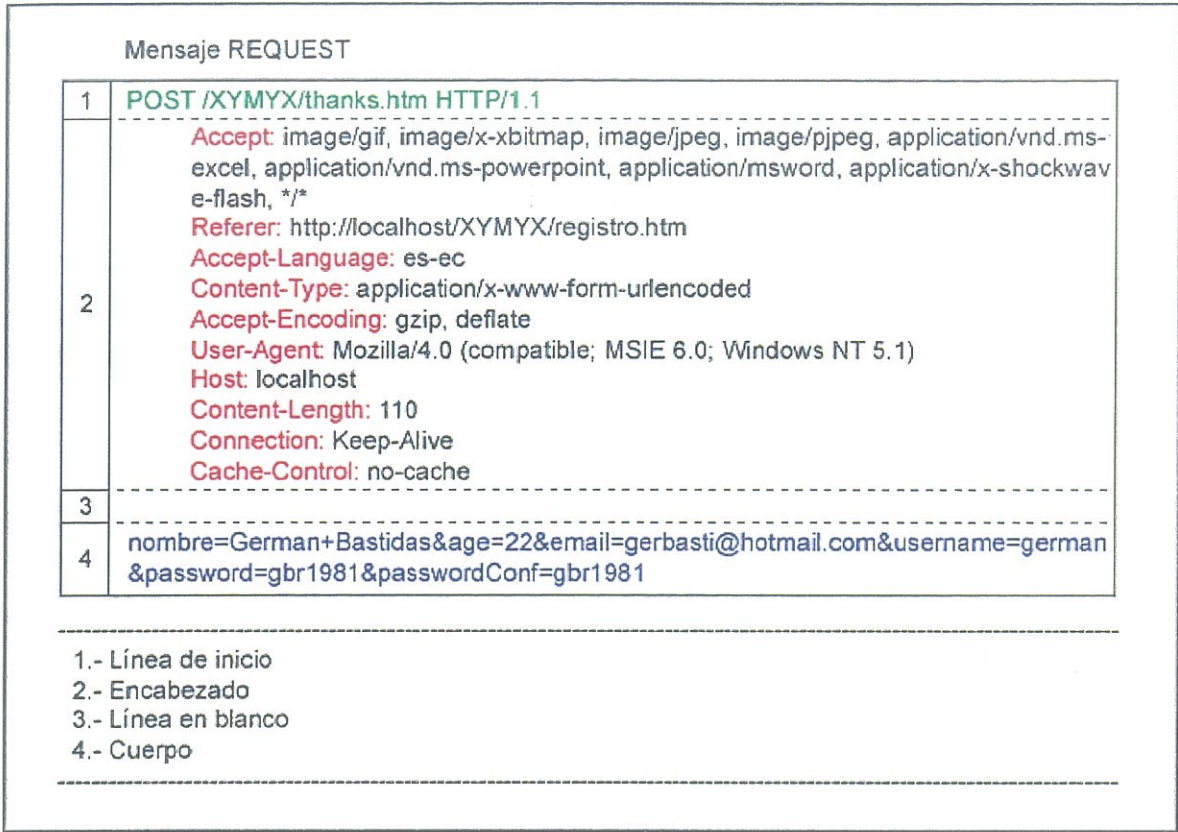


Figura 6. Estructura del mensaje request

### 2.1.5 Mensaje Response

Los mensajes HTTP tipo response son aquellos que un servidor debe construir como respuesta a una petición de un cliente. La estructura básica de este tipo de mensajes es la siguiente:

*Mensaje Response = Línea de estado de respuesta*  
*Header o encabezado*  
*Línea en blanco*  
*Cuerpo del mensaje*

### 2.1.5.1 Línea de estado del mensaje Response

La línea de estado de respuesta esta formada por tres elementos separados por un espacio en blanco:

*protocolo* [espacio] *código de estado* [espacio] *frase de razón*

- *protocolo*.- Al igual que en el mensaje request construido por el cliente, el servidor debe construir un mensaje de respuesta que tenga en la línea de inicio el protocolo HTTP utilizado para armar el mensaje. El primer elemento o grupo de caracteres que encontramos en esta línea de inicio define el nombre y la versión del protocolo. Un ejemplo sería “HTTP/1.1” en donde podemos ver que el nombre del protocolo está separado de la versión por un slash (/).
- *código de estado*.- El segundo elemento que encontramos en la línea de estado del mensaje es el código de estado de la respuesta. Este es un número entero de tres dígitos que representa el resultado del intento del servidor por atender y satisfacer la petición del cliente. El primer dígito del código de estado representa el tipo o clase de respuesta emitida por el servidor. De acuerdo a esta categorización los mensajes de respuesta se agrupan en:
  - 1xx (Mensajes de información) – La petición fue recibida y se puede continuar con el proceso de comunicación.
  - 2xx (Mensajes de éxito) – La acción solicitada por el cliente fue recibida, entendida y aceptada con éxito.
  - 3xx (Mensajes de redirección) – Se deben completar más acciones por parte del cliente para completar una petición.

- 4xx (Mensajes de error del lado del cliente) – El mensaje de petición contiene errores de sintaxis o no pudo ser atendido.
  - 5xx (Mensajes de error del lado del servidor) – Error del servidor al atender una petición aparentemente válida.
- *frase de razón.*- Es una corta descripción textual del código de estado. El código de estado es comprendido por un programa de aplicación mientras que la frase de razón es utilizada para que el código de estado sea comprendido por los seres humanos.

A continuación presentamos los códigos de estado de respuesta definidos por el RFC 2616. Estos códigos van acompañados por la frase de razón respectiva.

1xx ( Mensajes de información )	
100	El cliente debe continuar enviando su petición
101	Cambiar el protocolo de aplicación utilizado en la conexión

2xx ( Mensajes de éxito )	
200	Petición exitosa
201	Un nuevo recurso ha sido creado
202	La petición fue aceptada para ser procesada
203	La información devuelta no es definitiva
204	La respuesta no contiene información en el cuerpo del mensaje
205	El cliente debe resetear el contenido del documento solicitado
206	Respuesta con contenido parcial del recurso

3xx ( Mensajes de redirección )	
300	El recurso solicitado corresponde a varias alternativas
301	El recurso solicitado tiene asignado un nuevo URI
302	Recurso solicitado encontrado bajo un URI temporalmente asignado
303	El recurso puede ser encontrado bajo un URI diferente
304	El recurso no ha sido modificado
305	El recurso solicitado debe ser recuperado desde un servidor proxy
306	Código utilizado en versiones anteriores a HTTP/1.1
307	El recurso solicitado reside temporalmente bajo un URI diferente

4xx ( Mensajes de error del lado del cliente )	
400	La petición no fue entendida por el servidor debido a errores de sintaxis
401	La petición requiere autenticación del usuario
402	Código reservado para uso futuro
403	El servidor entendió la petición pero no le esta permitido responder
404	El recurso solicitado no se encontró en el servidor
405	No es permitido aplicar el método de la petición sobre el recurso
406	El mensaje de respuesta contiene información que no acepta el cliente
407	La petición requiere autenticación con el servidor proxy
408	El tiempo de espera para recibir la petición ha expirado
409	Conflicto con el estado actual del recurso
410	El recurso ha sido removido permanentemente del servidor
411	Es necesario incluir en el mensaje de petición el tamaño del contenido
412	Precondición dada en uno de los headers de la petición es falsa
413	El contenido de la petición es demasiado largo para ser procesado
414	El URI de la petición es demasiado largo para ser procesado
415	Contenido de petición en formato que no puede leer el servidor
416	El rango solicitado del recurso es inválido
417	La expectativa dada por el header Expectation no puede ser satisfecha

5xx ( Mensajes de error del lado del servidor )	
500	Se produjo un error interno en el servidor
501	El método de la petición no está implementado en el servidor
502	El gateway o proxy a recibido una respuesta inválida de otro servidor
503	El servidor no se encuentra disponible por sobrecarga o mantenimiento
504	El tiempo de espera de respuesta de otro servidor ha expirado
505	Versión HTTP no soportada por el servidor

### 2.1.5.2 Headers o encabezados del mensaje Response

Los campos del encabezado de un mensaje response le permiten al servidor enviar información adicional al cliente sobre el acceso al recurso solicitado o sobre el servidor en sí mismo. Los campos que se pueden incluir dentro del encabezado son los siguientes:

- Accept-Ranges
- Age
- ETag
- Location
- Proxy-Authenticate
- Retry-After
- Server
- Vary
- WWW-Authenticate

Ninguno de estos campos es necesario incluir en el encabezado de un mensaje response. Sin embargo algunos de ellos pueden ser utilizados por las aplicaciones clientes. El encabezado “WWW-Authenticate” es utilizado por el protocolo HTTP con fines de autenticación de usuarios. Cuando un recurso requiere de identificación para que sea enviado al cliente, el servidor deberá mandar un mensaje de respuesta que contenga el encabezado “WWW-Authenticate” para indicarle al cliente que debe pedir el recurso mediante un nombre de usuario y una contraseña.

### 2.1.5.3 Cuerpo del mensaje Response

Como se mencionó antes el cuerpo de un mensaje response no siempre lleva información para el cliente. Esto dependerá del método de la petición que se haya realizado. En la sección 2.1.4.1 revisamos algunos de estos métodos request especificados en el RFC 2616.

### 2.1.5.4 Estructura del mensaje Response

Una vez que hemos revisado las partes fundamentales de un mensaje response vamos a observar en la siguiente figura un ejemplo real de este tipo de mensajes enviado por nuestro servidor web como respuesta al mensaje de petición de un cliente.

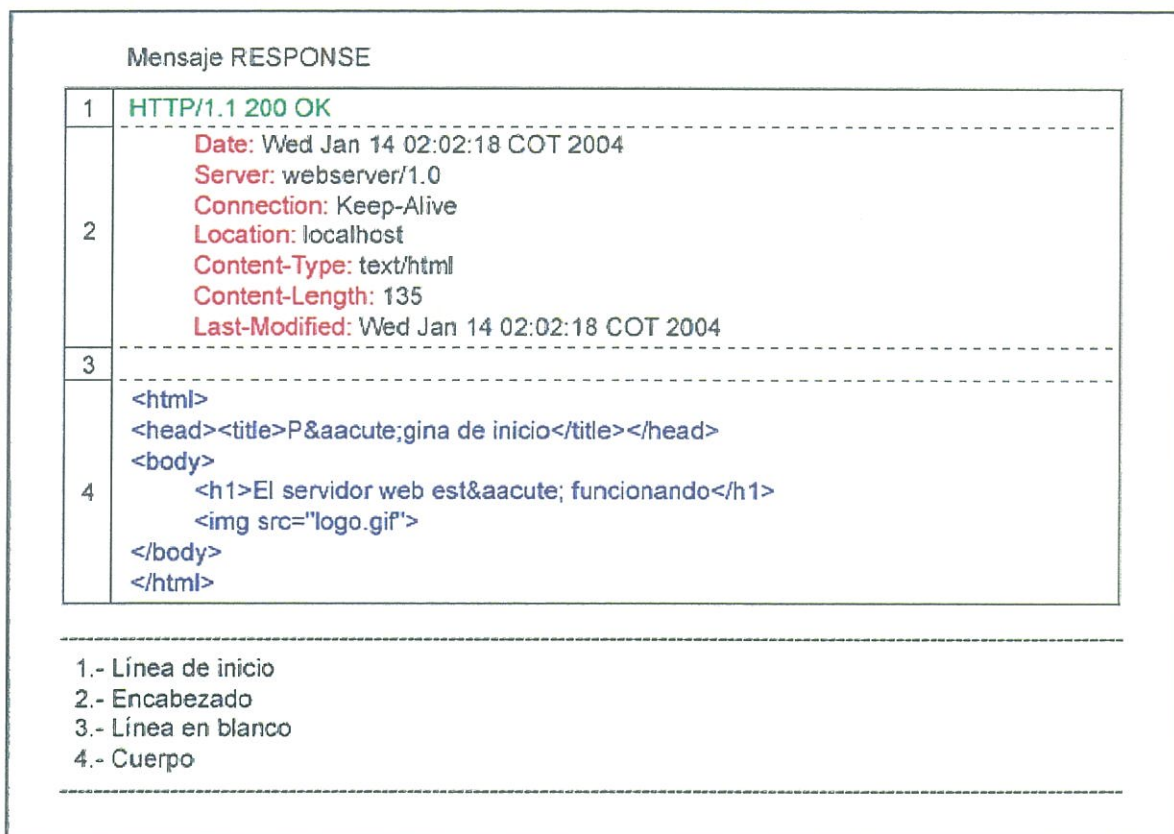


Figura 7. Estructura del mensaje response



## 2.2 Análisis del servidor web

Como vimos en la sección anterior, el proceso de comunicación entre un cliente y un servidor se reduce básicamente al intercambio de mensajes a través de una conexión establecida entre ambas partes. Este proceso de comunicación sirve para que un cliente obtenga un recurso que se encuentra almacenado en un servidor (páginas web, imágenes, sonidos, etc.). Recordemos cuáles eran los pasos para que un cliente pueda ver una página web.

1. El cliente establece una conexión con el servidor web.
2. El cliente construye un mensaje tipo request solicitando una página web. Este mensaje se envía al servidor a través de la conexión. Hay que agregar que la conexión cliente-servidor se realiza a través de un circuito virtual creado a nivel de capa de transporte.
3. El servidor web está pendiente de los mensajes tipo request que deben llegar por un puerto TCP predefinido (generalmente el puerto 80).
4. Al llegar un mensaje tipo request al servidor, éste debe leer el mensaje de petición y crear un mensaje de respuesta apropiado.
5. El servidor envía el mensaje tipo response a través de la conexión.
6. El mensaje de respuesta llega al cliente para ser procesado.
7. El servidor cierra la conexión con el cliente.
8. El cliente lee el mensaje de respuesta y despliega la página web en el navegador.

Como podemos ver existen varias etapas en este proceso de comunicación para que el cliente pueda recuperar un recurso del servidor. Sin embargo, en el proceso de

diseño de nuestro servidor no es necesario que conozcamos todos los detalles de cada uno de estos pasos, sólo debemos tener en cuenta que existen y que se dan en cierto orden. En realidad las etapas que debemos tomar en cuenta para comprender cómo debe funcionar nuestro servidor son la 3, 4 y 5. Estas etapas dan lugar a cuatro estados por los que debe pasar el servidor para cumplir sus propósitos:

- a) Escuchar conexiones
- b) Procesar mensaje request
- c) Crear mensaje response
- d) Enviar respuesta

### 2.2.1 Estados del servidor

En esta sección explicaremos las acciones que debe realizar el servidor web en cada uno de los estados por los que debe pasar para atender peticiones.

1. *Escuchar conexiones.*- El servidor debe crear un socket o puerto TCP. El número de puerto estará definido por el archivo de configuración del servidor. Por defecto el número de puerto para el protocolo HTTP es el 80, pero este número puede ser reemplazado si queremos que el servicio de atención a peticiones de clientes trabaje en otro puerto. Una vez que la aplicación servidor se enlaza con el puerto TCP empieza a escuchar intentos de conexión de los clientes.
2. *Procesar mensaje request.*- El momento en que se detecta que un cliente quiere establecer una conexión con el servidor a través de un puerto TCP, el servidor debe aceptar la conexión y empezar a leer los datos que ingresen por la misma. El servidor empezará a recibir una cadena de bytes que son parte del mensaje de

petición del cliente. A medida que van llegando los bytes del mensaje request, el servidor deberá procesar los mismos para determinar que tipo de respuesta debe emitir. El análisis del mensaje request incluye los siguientes pasos:

- Primero el servidor debe leer la línea de inicio o identificación del mensaje de petición enviado por el cliente.
- Después debe determinar si la línea de inicio está completa o no.
- Si la línea de inicio no está completa, es decir que no cuenta con sus tres elementos básicos (método, URI y protocolo), el servidor debe dejar de leer el mensaje de petición y asignar el valor apropiado al código de estado que se enviará en el mensaje de respuesta.
- Si la línea de inicio está completa entonces el servidor debe determinar el método de la petición, el URI y el protocolo HTTP.
- Luego de tener estos tres valores el servidor debe determinar si el método de la petición es válido y si puede manejar la versión del protocolo HTTP utilizada por el cliente.
- El URI enviado en la línea de inicio del mensaje request debe ser descompuesto en el caso de que los datos de un formulario enviado al servidor a través del método GET fueron agregados al URI.
- Luego el servidor debe leer todos los campos del encabezado del mensaje request.
- Finalmente, si corresponde de acuerdo al método de la petición, el servidor deberá leer el contenido del cuerpo del mensaje de petición del cliente.

3. *Crear mensaje response.*- Una vez leído el contenido del mensaje request, el servidor debe construir un mensaje de respuesta apropiado. Para esto debe determinar que tipo de código de estado se debe incluir en la línea de identificación del mensaje de respuesta. Si el mensaje de petición enviado por el cliente está completo, tiene un método HTTP implementado en el servidor, maneja una versión adecuada del protocolo, contiene en el encabezado los campos necesarios, tiene la estructura adecuada, en fin, si el mensaje de respuesta puede tener un código de estado de petición exitosa (200 OK), el servidor puede empezar a construir el encabezado del mensaje de respuesta con los campos genéricos. Después el servidor debe analizar el recurso que ha solicitado el cliente (a través del URI) para determinar que otros campos se deben incluir en el encabezado. De acuerdo a la naturaleza del mensaje de petición, el servidor deberá agregar información al cuerpo del mensaje de respuesta.

4. *Enviar respuesta.*- Una vez que se ha definido y estructurado el mensaje response, el servidor debe empezar a enviar los bytes de este mensaje de respuesta a través del objeto socket creado en la primer etapa. El servidor debe enviar primero la línea de identificación, seguida del encabezado del mensaje, luego una línea en blanco y al final, si es necesario, debe enviar la información del cuerpo del mensaje. Finalmente el servidor debe cerrar el puerto a través del cual se mantuvo la conexión con el cliente.

## 2.2.2 Arquitectura del servidor

Para analizar la estructura del servidor debemos tomar en cuenta el diseño de su arquitectura. Esto nos permitirá identificar los principales componentes del servidor y la comunicación entre los mismos.

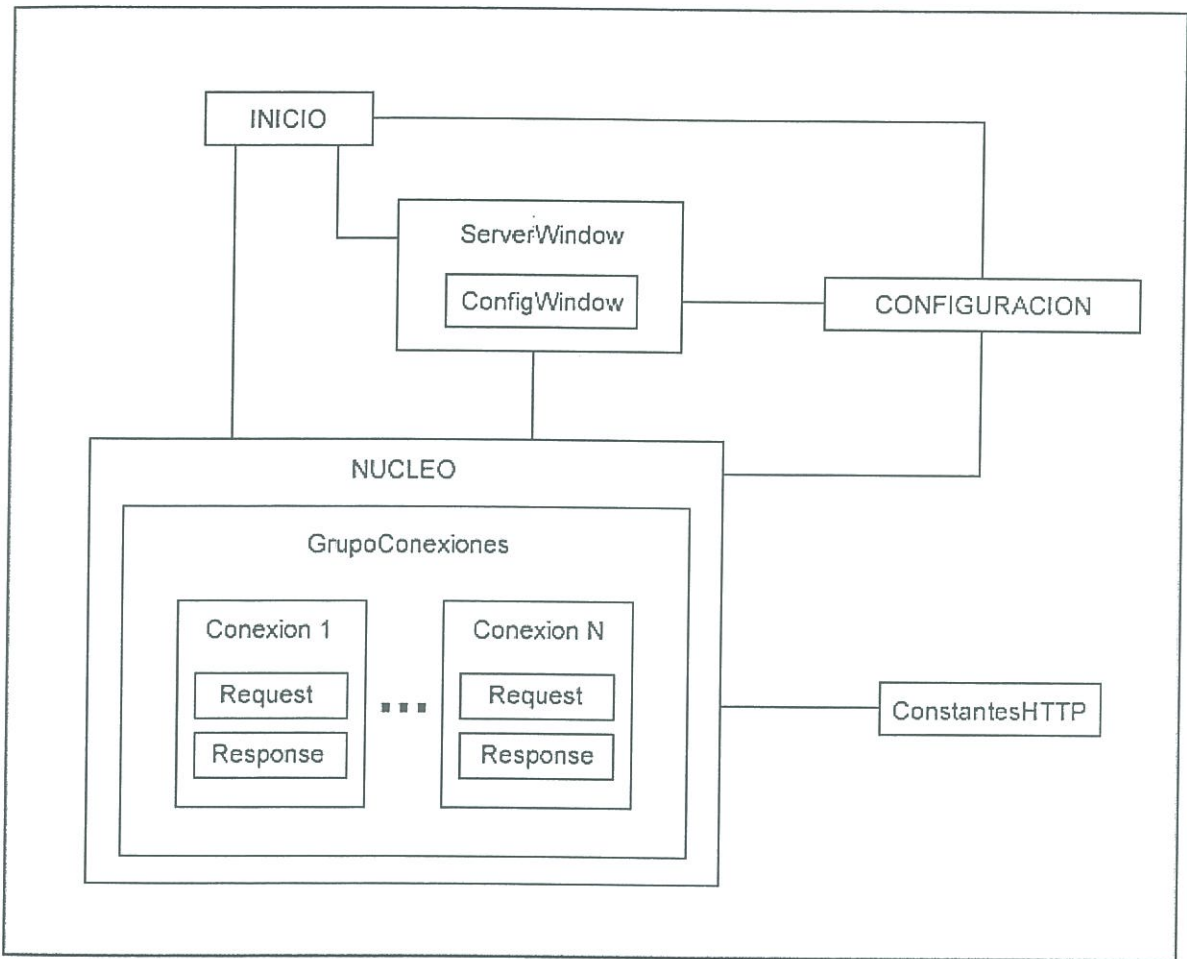


Figura 8. Arquitectura del servidor

En la figura 8 podemos observar la arquitectura del servidor, a un nivel abstracto, a través de un diagrama de bloques que nos permite tener una idea básica de la estructura del servidor, los módulos que la componen y la manera en que estos se relacionan. Este diagrama está compuesto por bloques o recuadros que representan los módulos y por líneas de unión que expresan una relación entre los mismos. Además podemos observar que existen ciertos bloques que se encuentran dentro de otros, lo cual representa

composición de módulos u objetos. Los diez módulos o clases que forman el servidor son los siguientes:

- Módulo Inicio
- Modulo ServerWindow
- Módulo ConfigWindow
- Módulo Configuración
- Módulo Núcleo
- Módulo GrupoConexiones
- Módulo Conexión
- Módulo Request
- Módulo Response
- Módulo ConstantesHTTP

En la sección 2.2.4 explicaremos las funciones que cumple cada uno de los módulos mencionados anteriormente.

### **2.2.3 Procesos**

En esta sección veremos los diferentes procesos que realiza el servidor a través de diagramas de transición de estado. Estos diagramas nos permiten ver las distintas fases o estados por los que tiene que pasar el servidor en respuesta a eventos internos o externos. Tenemos básicamente cinco procesos que se ejecutan como parte del funcionamiento de nuestro servidor web:

- Arranque del servidor sin interfase gráfica
- Arranque del servidor con interfase gráfica
- Atención de petición
- Cambio de la configuración a través de archivo de texto
- Cambio de la configuración a través de aplicación GUI

A continuación se presentan los diagramas de transición de estado correspondientes a los distintos procesos mencionados anteriormente.

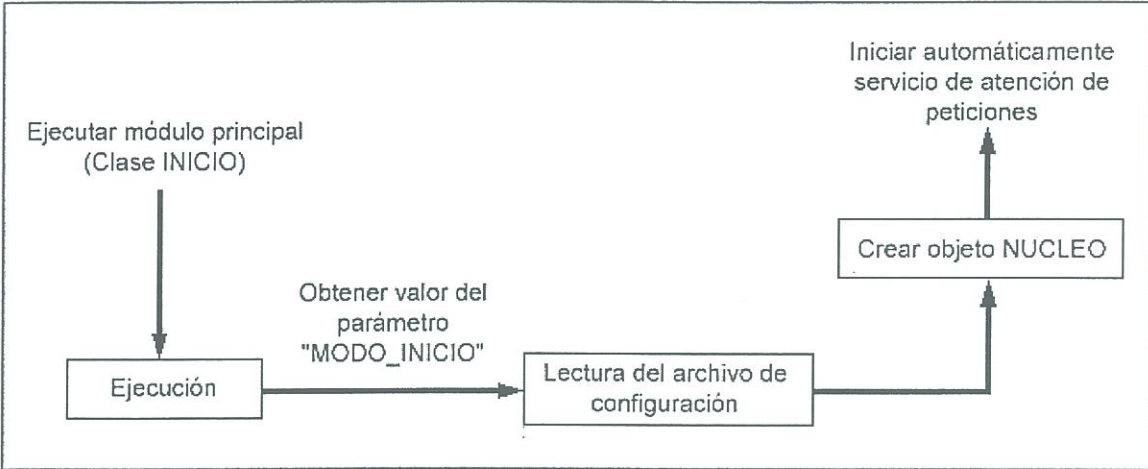


Figura 9. Diagrama de arranque del servidor sin interfase gráfica

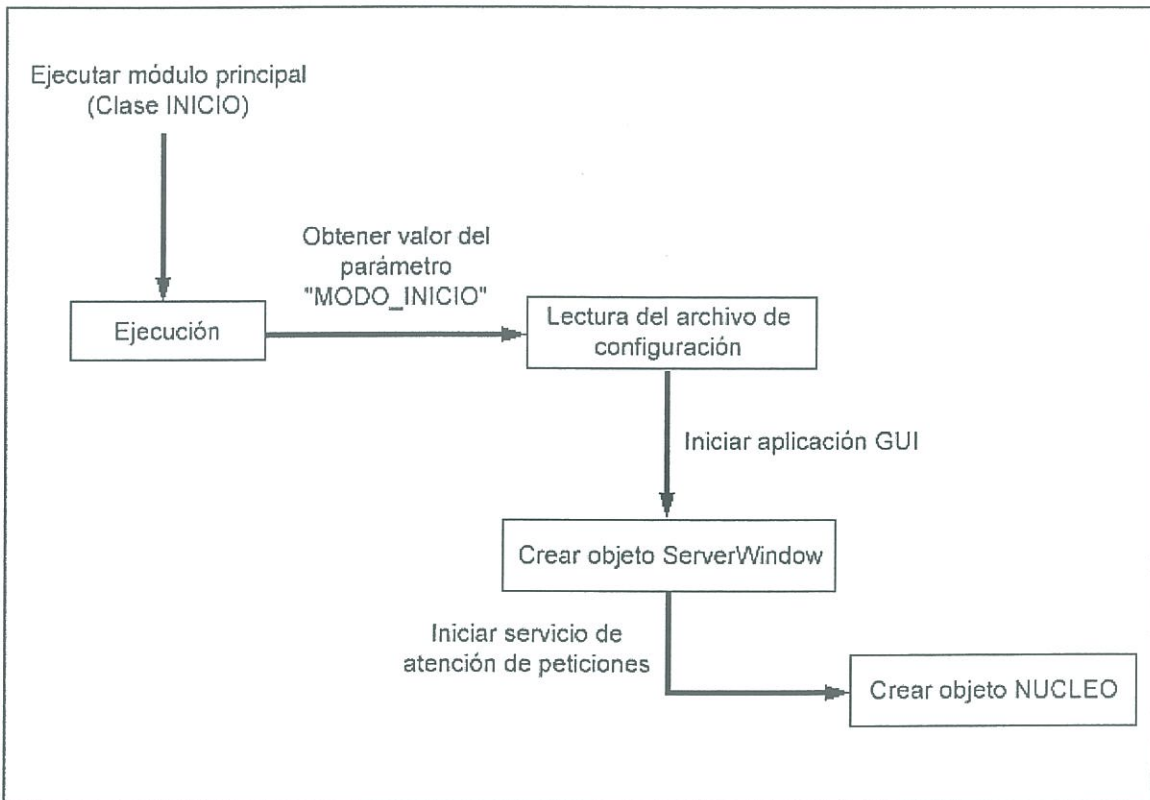


Figura 10. Diagrama de arranque del servidor con interfase gráfica (GUI)

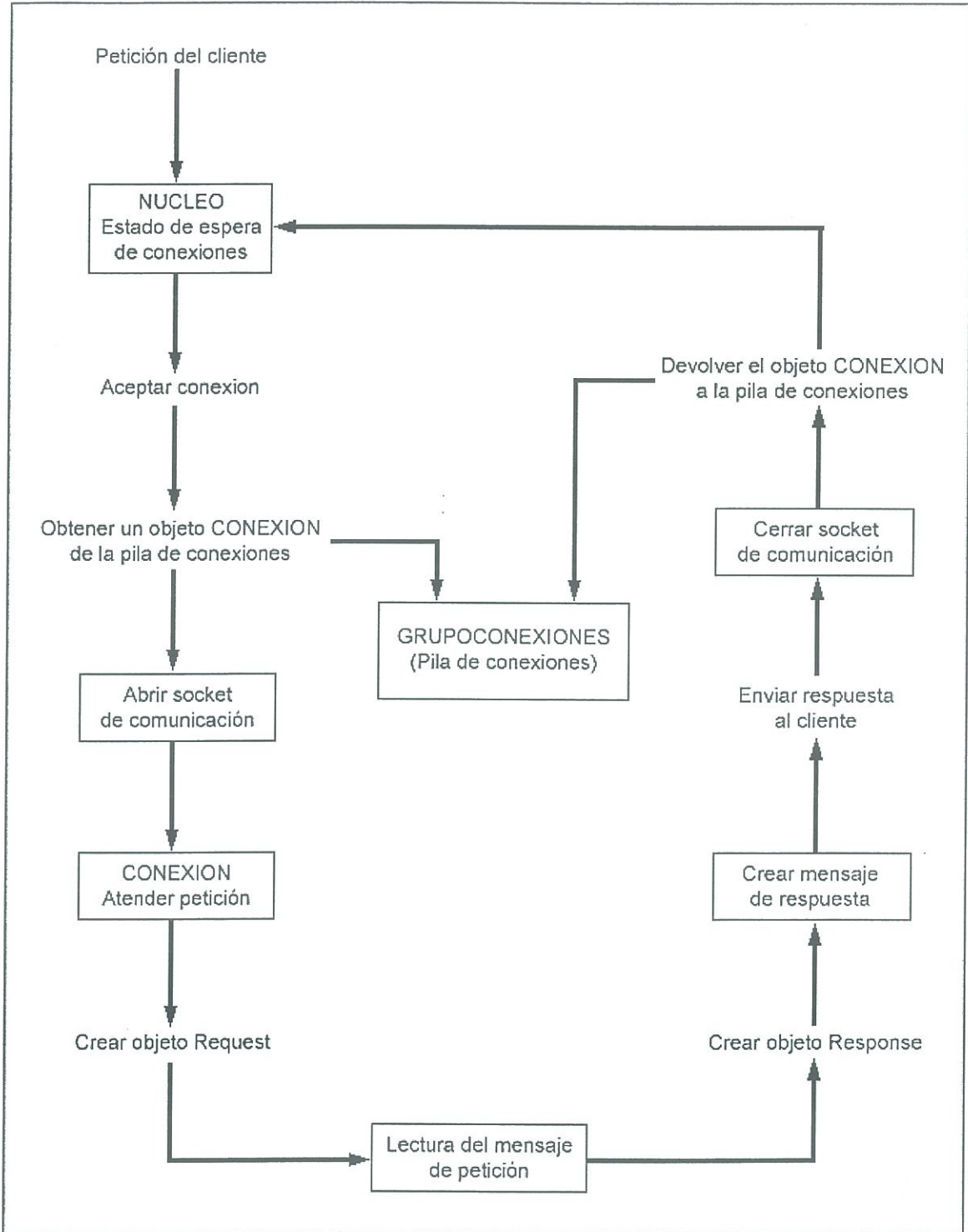


Figura 11. Diagrama de atención de petición



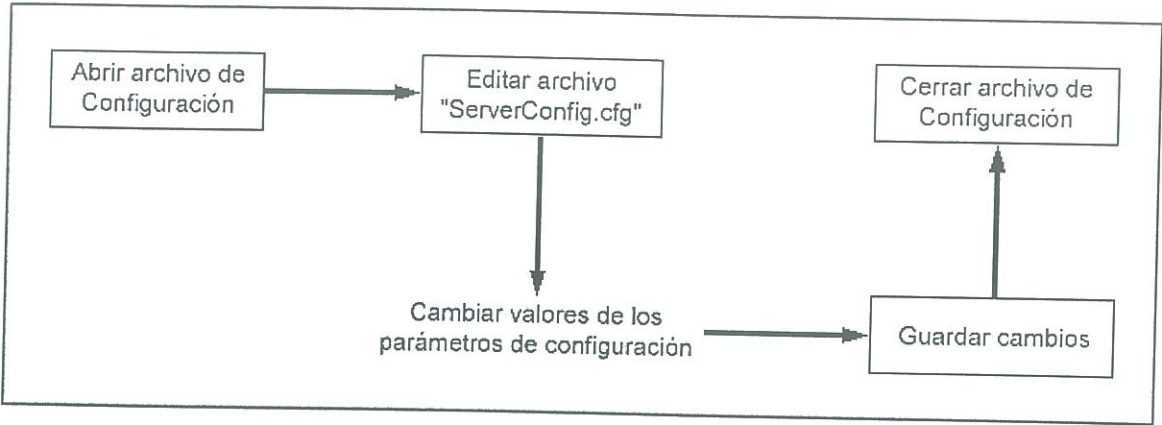


Figura 12. Diagrama de cambio de la configuración a través de un archivo de texto

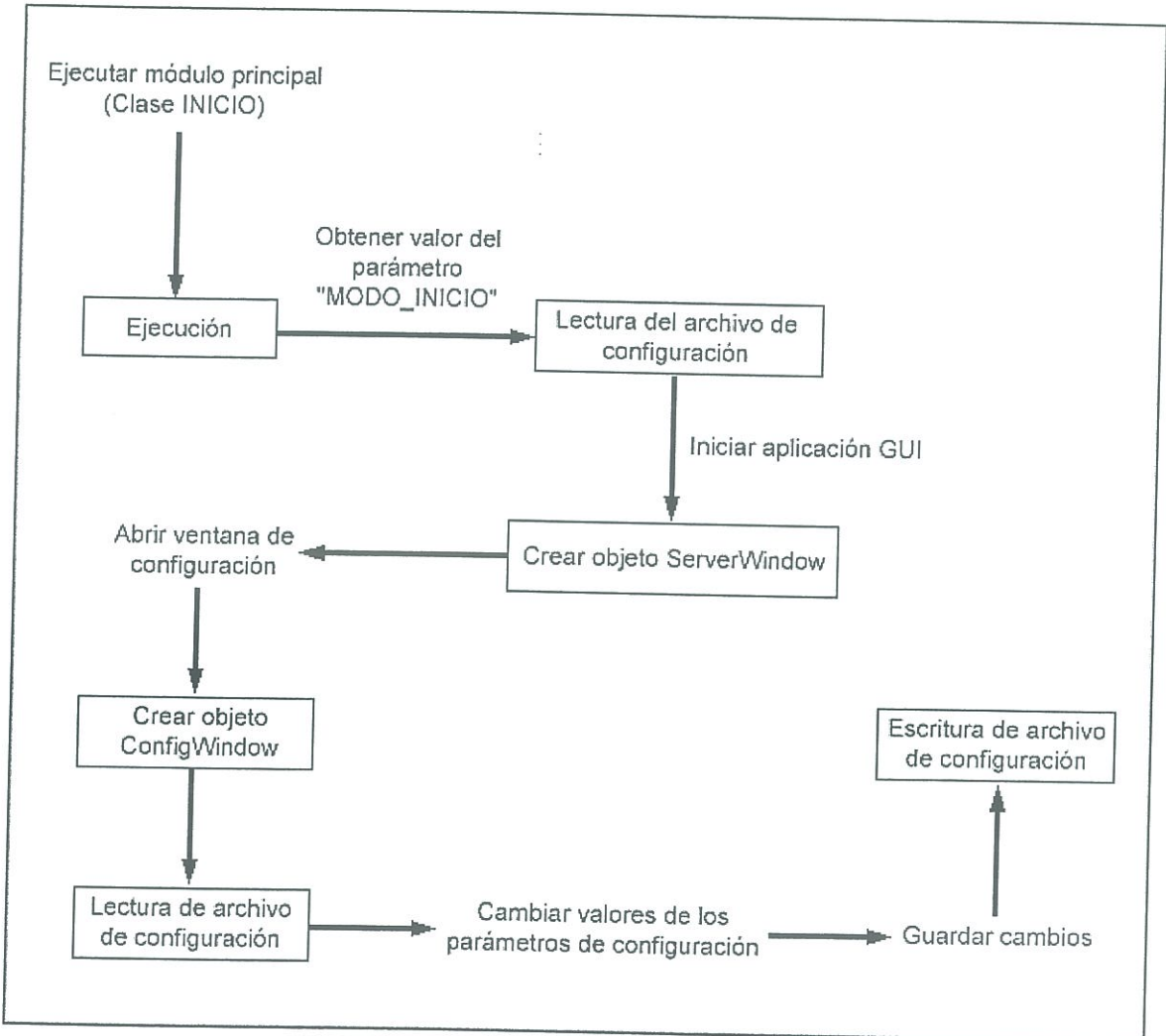


Figura 13. Diagrama de cambio de la configuración a través de aplicación GUI

## 2.2.4 Descripción de los módulos del servidor

Como mencionamos antes el servidor cuenta con diez módulos o clases que lo forman. A continuación explicaremos las funciones que realiza cada uno de estos módulos.

- *Módulo Inicio.*- Este módulo es el que contiene el método principal (main) desde donde se inicia la aplicación. La función de este módulo es crear un objeto de la clase *Configuración* de donde pueda obtener el valor del parámetro *MODO\_INICIO* para determinar de que manera debe arrancar el servidor. Existen dos modos de inicio:
  - Proceso.- Cuando el valor del parámetro *MODO\_INICIO* es 1 el servidor arranca como un proceso sin ninguna interfase gráfica. El servicio de atención de peticiones empieza automáticamente
  - Aplicación GUI.- Cuando el valor del parámetro *MODO\_INICIO* es 2 el servidor arranca como una aplicación con interfase gráfica. El servicio de atención de peticiones no empieza automáticamente.
- *Modulo ServerWindow.*- Este módulo maneja la aplicación GUI del servidor. En esta aplicación el usuario puede realizar varias cosas:
  - Iniciar el servicio de atención de peticiones.
  - Detener el servicio de atención de peticiones.
  - Abrir una ventana para configurar el servidor.
  - Visualizar cómo llegan las peticiones de los clientes en tiempo real.
  - Visualizar el número de conexiones activas.
  - Conocer el número de peticiones totales desde que se inició la aplicación.
  - Conocer el número de Kbytes enviados a los clientes.
- *Módulo ConfigWindow.*- Este módulo permite cambiar los parámetros de configuración del servidor mediante una interfase gráfica. El momento en el que

el usuario acepte los cambios hechos a los parámetros de configuración del servidor, éste los reconocerá y utilizará inmediatamente. Los parámetros de configuración del servidor son los siguientes:

- Puerto TCP.
  - Modo de inicio<sup>2</sup>.
  - Número máximo de conexiones.
  - Directorio raíz.
  - Páginas de inicio.
  - Listar contenido de carpetas.
  - Tiempo de espera máximo para petición de clientes.
- 
- *Módulo Configuración.*- Este módulo se encarga de leer el archivo de configuración “ServerConfig.cfg”. Este es un archivo de texto que puede ser editado por el usuario para cambiar los valores de configuración del servidor. Es importante recalcar que este archivo tiene una estructura específica que no puede ser cambiada por el usuario. El usuario solo podrá cambiar los valores de los parámetros de configuración. En el caso que se cambie la estructura de este archivo o simplemente no exista, el servidor arrancará con valores por defecto.
  
  - *Módulo Núcleo.*- Este es el módulo principal del servidor web. Es un thread que se encarga de escuchar los intentos de conexión de los clientes, aceptar las conexiones, obtener un objeto *Conexión* de la pila *GrupoConexiones* y asignarle a este objeto el trabajo de atender la petición del cliente.
  
  - *Módulo GrupoConexiones.*- Este módulo contiene un stack o pila de objetos de la clase *Conexión*. Provee los métodos necesarios para que el núcleo del servidor pueda extraer objetos tipo *Conexión* para que atiendan las peticiones de los clientes, insertar este mismo tipo de objetos en la pila cuando terminen su trabajo, verificar cuantas conexiones están activas, etc.

---

<sup>2</sup> El valor de este parámetro de configuración solo puede ser cambiado a través de la edición del archivo de texto “ServerConfig.cfg”

- *Módulo Conexión.*- Este módulo es un thread que se encarga de atender una petición de un cliente a través del manejo de los objetos *Request* y *Response*. Además se encarga de registrar en un archivo de texto las peticiones de los clientes con los resultados correspondientes.
- *Módulo Request.*- Este módulo se encarga de leer y descomponer el mensaje de petición del cliente, es decir extrae la información necesaria del mensaje de petición para poder construir un mensaje de respuesta adecuado.
- *Módulo Response.*- Este módulo se encarga de construir y enviar el mensaje de respuesta ante una petición del cliente.
- *Módulo ConstantesHTTP.*- Este módulo contiene variables estáticas que utiliza el servidor web en la comunicación con el cliente.

## Capítulo 3

# DESCRIPCION DEL PROYECTO

En este capítulo veremos los aspectos principales relacionados con el servidor web y el currículo del taller. Detallaremos además los aspectos necesarios para el manejo del servidor y de la herramienta multimedia que nos presenta el material del currículo.

En la primera parte de este capítulo analizaremos el servidor web que hemos desarrollado y en la segunda parte analizaremos la estructura del currículo, la metodología que se va a utilizar, los beneficios del taller, etc.

### 3.1 Descripción del servidor

El objetivo del proyecto es diseñar e implementar un servidor web modular multiplataforma que sirva como base de estudio para elaborar el contenido de un taller de programación de Java avanzado. Por esta razón el servidor que hemos desarrollado no es considerado como el fin último de este proyecto, sino más bien es la base sobre la cual construiremos el taller de Java y por tanto es una aplicación que se ha hecho con fines pedagógicos. Tomando en cuenta esta aclaración podemos limitar el alcance de nuestro servidor web al cumplimiento de la función básica de este tipo de aplicaciones, la cual es atender peticiones de clientes sobre recursos que se encuentran almacenados en el computador que está ejecutando el servidor. En la siguiente sección analizaremos los dos tipos básicos de recursos que pueden estar almacenados en un computador que ejecuta un servidor web, y que son devueltos por éste como respuesta a la petición de un cliente. Tomando en cuenta estos conceptos podremos entender el funcionamiento de nuestra aplicación.

### 3.1.1 Alcance de la aplicación

En el primer capítulo de este documento definimos unos modelos de comunicación entre el cliente y el servidor web que fundamentalmente estaban determinados por dos tipos de recursos: recursos estáticos y recursos dinámicos. A fin de cuentas estos recursos son archivos que, como mencionamos anteriormente, están almacenados en el computador que ejecuta el servidor web.

Existen dos tipos básicos de archivos que pueden ser manejados por un computador:

- Los archivos de texto plano compuestos por caracteres que son legibles para los seres humanos, denominados archivos ASCII<sup>1</sup>. Algunos ejemplos de este tipo de archivos son los siguientes:
  - Archivos de código fuente (Extensión “.c”, “.java”, etc.)
  - Formatos de texto (Extensión “.txt”, “.html”, etc.)
  - Formatos de intercambio (Extensión “.rtf”, “.ps”, etc.)
  
- El resto de archivos que no pertenecen a la categoría anterior son los denominados archivos binarios. Algunos ejemplos de este tipo de archivos son los siguientes:
  - Archivos de imagen (Extensión “.jpg”, “.gif”, “.png”, “.bmp”, etc.)
  - Archivos de sonido (Extensión “.wav”, “.au”, “.mp3”, etc.)
  - Archivos de video (Extensión “.mov”, “.avi”, “.mpg”, etc.)
  - Archivos ejecutables (Extensión “.exe”, “.com”, “.cgi”, etc.)
  - Archivos comprimidos (Extensión “.zip”, “.tar”, etc.)

---

<sup>1</sup> Código estándar americano para el intercambio de información.

Todos los archivos mencionados anteriormente son recursos que pueden estar almacenados en un computador y que pueden ser devueltos por un servidor web como respuesta a la petición de un cliente.

Un navegador o browser es una de las principales aplicaciones que solicita recursos a un servidor web. Los recursos más solicitados por este tipo de aplicaciones son las páginas web que son documentos electrónicos que contienen información sobre un tema en particular. Estos documentos son básicamente archivos de texto que contienen código HTML, el cual es un lenguaje de etiquetas que nos permite construir una página web con texto formateado, hipervínculos, imágenes, sonidos, videos, etc. Estos elementos de una página web, y la página en sí, vendrían a ser recursos estáticos que son entregados al navegador por el servidor web. El proceso de comunicación en este caso es el siguiente:

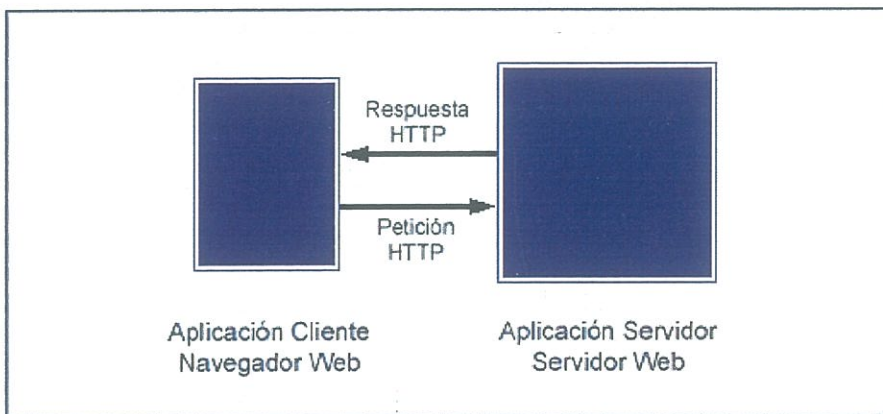


Figura 14. Petición de un recurso estático

Existen además los recursos dinámicos que esencialmente se dividen en dos tipos: unos son archivos de texto que combinan instrucciones escritas en algún lenguaje de programación con código HTML, por ejemplo las tecnologías del lado del servidor como ASP o JSP; y otros son simplemente archivos de código fuente formados por instrucciones escritas en algún lenguaje de programación, como por ejemplo scripts CGI o programas escritos en C, C++, Visual Basic, etc. Estos recursos por ejemplo pueden extraer información de bases de datos para generar dinámicamente código HTML que es interpretado por el navegador. El proceso de comunicación en este caso es el siguiente:

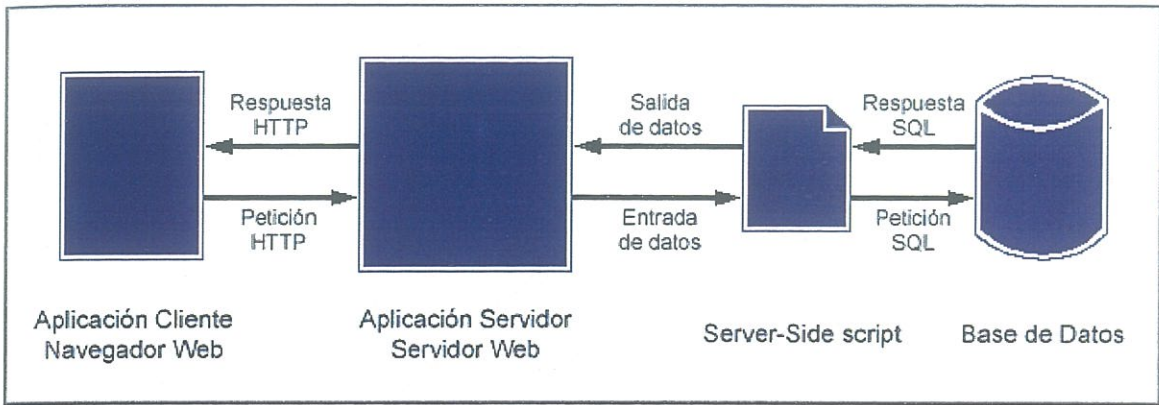


Figura 15. Petición de un recurso dinámico

Podemos observar en la figura 15 que la ventaja de este tipo de recursos es que le permiten a un cliente acceder a información almacenada en el servidor web. Esta información podría estar contenida en bases de datos o en archivos de texto. Debido a la naturaleza dinámica de este tipo de recursos, la información puede ser distinta en cada petición del cliente, a pesar de que el recurso solicitado sea el mismo.

Después del análisis anterior, y comparando las figuras 14 y 15, podemos encontrar una semejanza y una diferencia entre los procesos de petición de recursos estáticos y dinámicos. La semejanza entre ambos procesos es que cuando el cliente solicita un recurso, éste le es devuelto por el servidor web. En cambio la diferencia entre estos dos procesos radica en que la petición de un recurso dinámico requiere de un trabajo extra o de unos pasos adicionales que se tienen que realizar del lado del servidor. Cuando el servidor web recibe una petición de un recurso dinámico debe establecer una comunicación con alguna otra aplicación. Esta aplicación realizará algún proceso que devolverá una respuesta al servidor (generalmente código HTML generado dinámicamente). Finalmente el servidor deberá enviar un mensaje al cliente que contenga el resultado de la transacción realizada por la otra aplicación.

Una vez entendidos estos dos procesos básicos de comunicación entre un servidor web y un cliente, podemos limitar el alcance de nuestro servidor a la atención de peticiones sobre recursos estáticos solamente, porque como dijimos anteriormente, las peticiones sobre recursos dinámicos implican procesos adicionales.



### 3.1.2 Características del servidor web

El servidor web que hemos desarrollado para elaborar el contenido del taller tiene las siguientes características:

- Desarrollado en lenguaje Java.
- Se puede ejecutar en cualquier plataforma que tenga instalado el intérprete de Java (Java Virtual Machine).
- Es ligero y rápido.
- Posee dos modos de inicio: Proceso o Aplicación GUI.
- Soporta el protocolo HTTP/1.1.
- El servidor implementa dos métodos HTTP: GET y POST.
- Atiende cualquier mensaje de petición de un cliente que solicite un recurso estático.
- Puede ser configurado a través de la modificación de un archivo de texto o a través de una interfase gráfica.
- Parametrizar el número de puerto TCP en el que se deberán atender las peticiones.
- Parametrizar el número de conexiones máximas que puede soportar el servidor.
- Parametrizar el directorio raíz que contendrá las páginas web.
- Parametrizar los nombres de archivos que serán reconocidos como páginas de inicio.
- Existe la posibilidad de permitir o denegar el listado de carpetas.
- Se puede determinar el tiempo máximo que debe esperar el servidor para recibir una petición de un cliente una vez que se haya establecido la conexión entre ambos.
- Almacena en un archivo de texto los accesos de los clientes mediante el registro de sus peticiones.

### 3.1.3 Requisitos de hardware y software

Partiendo del concepto de que podemos aprovechar el servidor para aprender conceptos relacionados con la tecnología Java, procesos de comunicación, protocolo HTTP, etc., es necesario considerar algunos requisitos y tomar en cuenta ciertas recomendaciones que asegurarán el funcionamiento adecuado del mismo y facilitarán el proceso de aprendizaje.

#### 3.1.3.1 Requisitos de hardware

- *Espacio en disco.* Para utilizar el servidor es necesario crear una carpeta en el sistema de directorios que vaya a almacenar todos los directorios y archivos que forman parte del servidor. Este directorio podrá llamarse, por ejemplo “Web Server”, y tendrá la siguiente estructura:
  - En la raíz de la nueva carpeta se ubicarán los archivos que contienen el código fuente del servidor, estos son archivos de texto que tienen extensión “.java”.
  - También en la raíz se ubicarán las clases generadas a partir del código fuente, estos archivos tienen extensión “.class”.
  - Debe existir un directorio denominado “CONFIG” que contendrá el archivo de configuración de texto “ServerConfig.cfg”.
  - Debe existir un directorio denominado “DATA” que contendrá el archivo “estilos.css” que utilizará el servidor en el momento de enviar el listado de las carpetas al cliente, además este directorio contendrá algunas imágenes que utiliza el servidor el momento de iniciar la aplicación gráfica.
  - Debe existir un directorio denominado “LOGS” que contendrá el archivo de texto “access.log” que es donde se registran los accesos al servidor.

- Finalmente debe existir el directorio “WWW” que contiene una imagen y un archivo HTML que se podrán visualizar en un navegador web el momento en que el servidor sea encendido y esté atendiendo peticiones.

Toda la estructura mencionada anteriormente ocupará un espacio en disco mínimo de 193 KB distribuido de la siguiente forma:

Clases (bytecodes)	53.50 KB
Código fuente Java	63.90 KB
Directorio CONFIG	0.43 KB
Directorio DATA	68.10 KB
Directorio LOGS <sup>2</sup>	0.00 KB
Directorio WWW	7.02 KB
TOTAL	192.95 KB

- *Memoria RAM.* El servidor debe ser ejecutado en un computador que cuente con 64 MB de memoria RAM mínimo. Esto se debe a que la máquina virtual de Java ocupa aproximadamente 13 MB de memoria para ejecutar una aplicación y el servidor ocupa aproximadamente 7 MB de memoria cuando está atendiendo peticiones. Este último valor dependerá del número de conexiones máximas que acepte el servidor (con 100 conexiones se requiere 7 MB de memoria).
- *Monitor.* Es necesario contar con un monitor que tenga una resolución mínima de 800 x 600 pixels debido a que el tamaño de la ventana de la aplicación GUI del servidor es 800 pixels de ancho por 560 pixels de alto. Lo recomendable sería tener un monitor con una resolución de 1024 x 768 pixels.

---

<sup>2</sup> NOTA: El archivo donde se registran los accesos al servidor tiene un tamaño inicial de 0 KB pero puede crecer indefinidamente.

### 3.1.3.2 Requisitos de software

La ventaja de haber escrito el servidor en Java es que puede ejecutarse en diferentes sistemas debido a un componente de plataforma denominado Máquina Virtual Java (JVM). Este componente es una especie de traductor que convierte los bytecodes<sup>3</sup> de Java en código de máquina durante la ejecución de la aplicación. Por esta razón es necesario instalar el J2SE (Java 2 Standard Edition) versión 1.4.1 en el computador en el que vamos a ejecutar el servidor. El J2SE es una plataforma que incluye dos productos principales:

- Java Runtime Environment (RE). Contiene librerías, la máquina virtual de Java y otros componentes necesarios para ejecutar applets<sup>4</sup> y aplicaciones escritas en Java.
- Java 2 Software Development Kit (SDK). Contiene todos los componentes del Java RE más otras herramientas adicionales como compiladores que son necesarios para desarrollar aplicaciones Java.

También es necesario contar con un editor de textos para examinar el código fuente del servidor y para poder realizar la configuración del mismo a través del archivo de texto “ServerConfig.cfg”.

### 3.1.4 Servicios

El servicio básico que cualquier servidor web debe ofrecer es la atención de peticiones de clientes que solicitan recursos que se encuentran almacenados en el computador que está ejecutando un servidor. Como vimos al inicio de la sección 3.1, existen dos tipos básicos de recursos que pueden ser solicitados por un cliente o navegador web: recursos estáticos y recursos dinámicos. La aplicación que hemos desarrollado, con el propósito de construir el taller de programación en Java, ofrece el

---

<sup>3</sup> Código generado al compilar una aplicación Java

<sup>4</sup> Programas Java que se ejecutan en una página web cuando ésta es accedida desde un navegador.

servicio de atención a peticiones de recursos estáticos que utilicen los métodos HTTP GET y POST.

Por otro lado, tomando en cuenta los fines pedagógicos con que se desarrolló el servidor, pensamos que era necesario implementar los dos modos de inicio siguientes:

- *Proceso.*- En este modo el servidor se ejecuta como un proceso sin ninguna interfase gráfica. La aplicabilidad de este modo de inicio puede relacionarse con la necesidad de contar con un servidor web multiplataforma que atienda peticiones de recursos estáticos, que no consuma muchos recursos del computador y que se ejecute como proceso de fondo (background).
- *Aplicación GUI.*- En este modo el servidor se ejecuta como una aplicación con interfase gráfica. La idea de construir el módulo que implementa esta interfase nace de la necesidad de brindar al estudiante del taller de programación una herramienta que le permita visualizar el trabajo real que realiza el servidor, esto es:
  - Ver en tiempo real cómo arriban las peticiones de los clientes y conocer qué tiempo le toma al servidor atender cada una de ellas.
  - Además el estudiante podrá ver cuantas conexiones activas están procesando los mensajes de petición de los clientes.
  - El estudiante podrá saber cuántas peticiones se han recibido desde que se inicio el servicio de atención del servidor
  - También podrá conocer el número total de bytes que el servidor ha enviado en sus mensajes de respuesta.

A parte de estos beneficios que ofrece la aplicación GUI, cuyo propósito es facilitar el proceso de aprendizaje del estudiante, podemos citar el servicio de configuración del servidor mediante una interfase gráfica que nos evita tener que reiniciar la aplicación para que los nuevos valores de los parámetros de configuración sean reconocidos.

Es preciso aclarar nuevamente que, debido al alcance de la aplicación, no se consideró el implementar la atención de peticiones relacionadas con recursos dinámicos, ya que esto implicaría un proceso extra que debe efectuar el servidor: establecer una comunicación con otra aplicación. Sin embargo, como el código del servidor es abierto, el estudiante tiene la posibilidad de expandir sus conocimientos e implementar un módulo que abarque las peticiones sobre este tipo de recursos. También el estudiante podría ampliar el código del servidor para implementar el resto de métodos de petición HTTP.

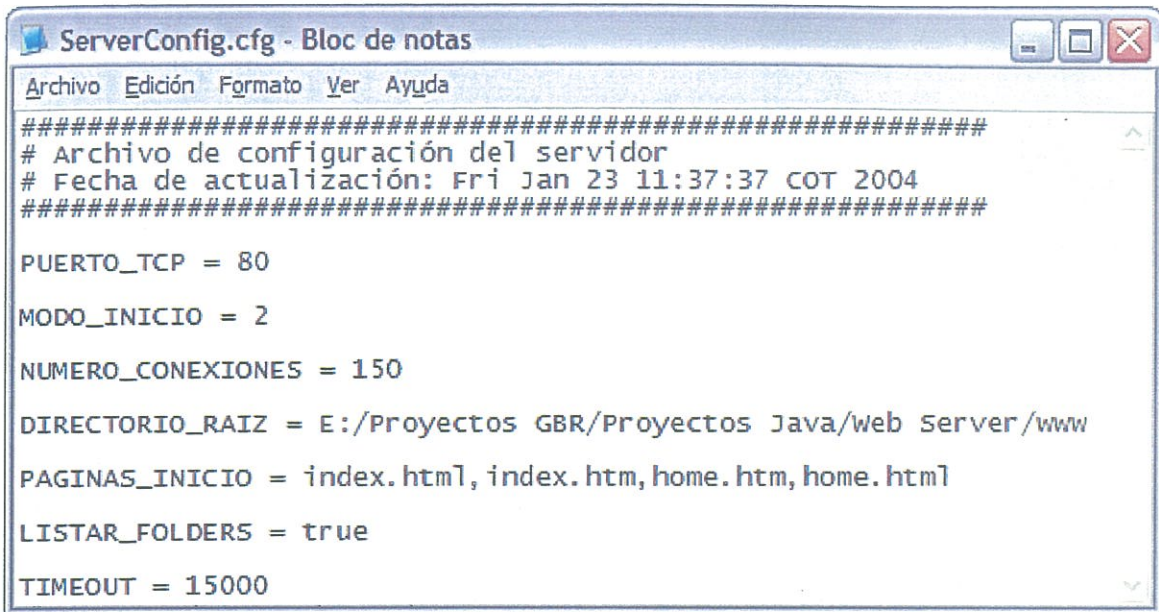
### 3.1.5 Ejecución de la aplicación

Debido a que el servidor es una aplicación Java es necesario utilizar alguna herramienta que forma parte del Java 2 Software Development Kit (SDK) para ejecutar la aplicación. Una de estas herramientas es una pequeña aplicación denominada “java” que se encarga de abrir una ventana de consola, arrancar el Java Runtime Environment (RE), cargar en memoria una clase específica e invocar el método principal (main) de esta clase. Otra herramienta similar denominada “javaw” realiza la misma función que la anterior, con la diferencia de que esta última no abre una ventana de consola. Utilizando una de estas dos herramientas podremos ejecutar el servidor web.

Toda aplicación Java cuenta con una clase de inicio que contiene un método principal denominado “main”, el cual es invocado por la máquina virtual de Java cuando ejecutamos la aplicación. La forma de declarar este método es la siguiente:

```
public static void main ( String args[] )
```

La clase de inicio de nuestro servidor que contiene el método principal (main) se denomina *Inicio*. El momento en que la máquina virtual de Java invoca este método, se crea una instancia de la clase *Configuracion*. Este nuevo objeto lee el contenido del archivo de texto “ServerConfig.cfg” para determinar los valores de los parámetros de configuración del servidor. La estructura y el contenido que hemos diseñado para este archivo es la siguiente:



```
ServerConfig.cfg - Bloc de notas
Archivo Edición Formato Ver Ayuda
#####
# Archivo de configuración del servidor
# Fecha de actualización: Fri Jan 23 11:37:37 COT 2004
#####

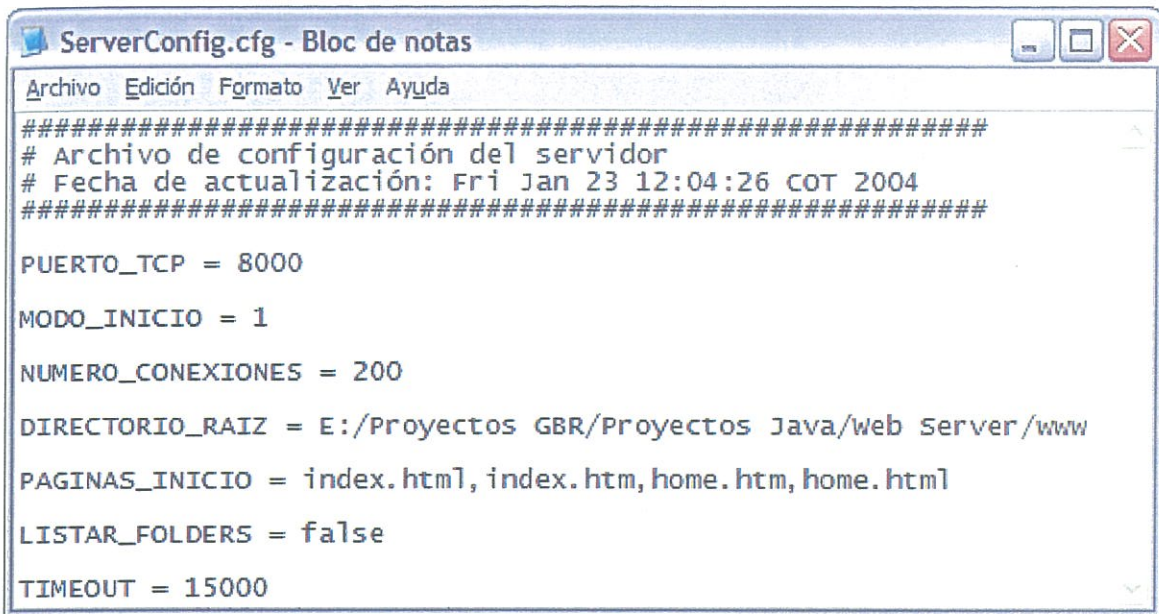
PUERTO_TCP = 80
MODO_INICIO = 2
NUMERO_CONEXIONES = 150
DIRECTORIO_RAIZ = E:/Proyectos GBR/Proyectos Java/web Server/www
PAGINAS_INICIO = index.html, index.htm, home.htm, home.html
LISTAR_FOLDERS = true
TIMEOUT = 15000
```

Figura 16. Estructura del archivo de configuración

Podemos ver en la figura 16 que el archivo “ServerConfig.cfg” contiene un total de 18 líneas. La segunda línea es un titular de aviso que nos advierte que este es el archivo de configuración del servidor. En la tercera línea se indica la fecha de la última actualización de este archivo. A partir de la línea 6 se encuentran los nombres de los parámetros de configuración con sus respectivos valores. Para hacer más legible y notoria la estructura del archivo, los parámetros de configuración se encuentran separados uno de otro mediante una línea en blanco. La estructura de las líneas de configuración es la siguiente:

*NOMBRE = valor*

Para entender como debemos modificar el archivo de configuración pongamos un ejemplo. Supongamos que la configuración inicial es la que se muestra en la figura 15 y necesitamos cambiar el número de puerto TCP de 80 a 8000, queremos que el servidor arranque como proceso y no como aplicación GUI, además necesitamos incrementar el número máximo de conexiones de 150 a 200, y finalmente no queremos que se liste el contenido de las carpetas que se encuentran en el directorio raíz. Para esto primero debemos apagar el servidor (en el caso que estuviere prendido), abrir el archivo “ServerConfig.cfg” con un editor de textos y modificarlo de la siguiente manera:



```
ServerConfig.cfg - Bloc de notas
Archivo Edición Formato Ver Ayuda
#####
# Archivo de configuración del servidor
# Fecha de actualización: Fri Jan 23 12:04:26 COT 2004
#####

PUERTO_TCP = 8000

MODO_INICIO = 1

NUMERO_CONEXIONES = 200

DIRECTORIO_RAIZ = E:/Proyectos GBR/Proyectos Java/web Server/www

PAGINAS_INICIO = index.html, index.htm, home.htm, home.html

LISTAR_FOLDERS = false

TIMEOUT = 15000
```

Figura 17. Estructura del archivo de configuración después de la modificación

Podemos observar que la estructura general del archivo permanece constante, lo único que modificamos fueron los valores de algunos parámetros de configuración. El siguiente paso es guardar los cambios hechos y cerrar el editor de textos. La siguiente vez que se ejecute el servidor, éste arrancará como proceso debido al valor de 1 del parámetro `MODO_INICIO`, además atenderá peticiones en el puerto 8000, el número máximo de conexiones será 200 y no se mostrará el contenido de las carpetas.

En el caso que el archivo de configuración haya sido borrado o haya sufrido alguna modificación en su estructura general, el servidor arrancará como una aplicación con interfase gráfica con valores determinados por defecto para los parámetros de configuración. Iniciará de este modo para que el usuario tenga la posibilidad de configurar gráficamente el servidor a través del módulo *ConfigWindow*, el cual generará automáticamente un nuevo archivo de configuración.

Una vez determinado el modo de inicio del servidor, la clase *Inicio* deberá ejecutar una de las acciones siguientes: si el valor del parámetro `MODO_INICIO` es 1, deberá crear una instancia de la clase *Núcleo*, caso contrario, si el valor del parámetro `MODO_INICIO` es 2, deberá crear una instancia de la clase *ServerWindow*.



### 3.1.6 Primer modo de inicio: Proceso

Si el valor del parámetro MODO\_INICIO es 1, entonces el servidor arranca sin interfase gráfica y empieza a atender inmediatamente las peticiones de los clientes. En el caso que ejecutemos el servidor con la herramienta “javaw”, nos aparecerá un mensaje de información para indicarnos que el servidor ha sido encendido y está funcionando.

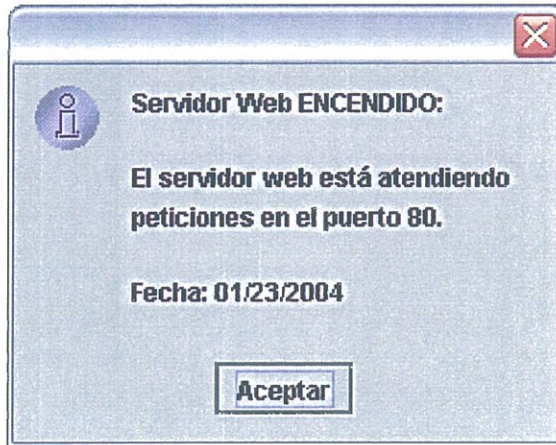


Figura 18. Mensaje de información (Servidor encendido)

Si ejecutamos el servidor con la herramienta “java”, además del mensaje de información de la figura 18, se abrirá la siguiente ventana de consola:

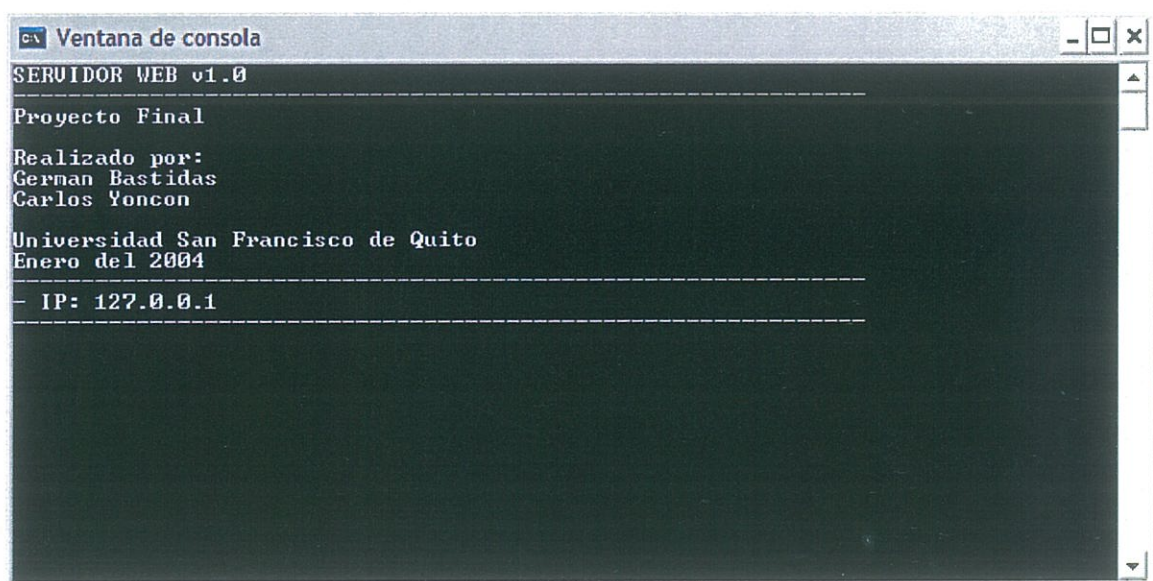


Figura 19. Ventana de consola (Servidor encendido)

Para apagar el servidor, en el primer caso que no tenemos una ventana de consola, debemos terminar manualmente el proceso que fue iniciado por la herramienta “javaw” para ejecutar el servidor web. Por ejemplo en plataformas Windows debemos presionar las teclas “Ctrl+Alt+Supr” para abrir la ventana del Administrador de Tareas de Windows, hacemos clic en la viñeta de procesos y veremos lo siguiente:

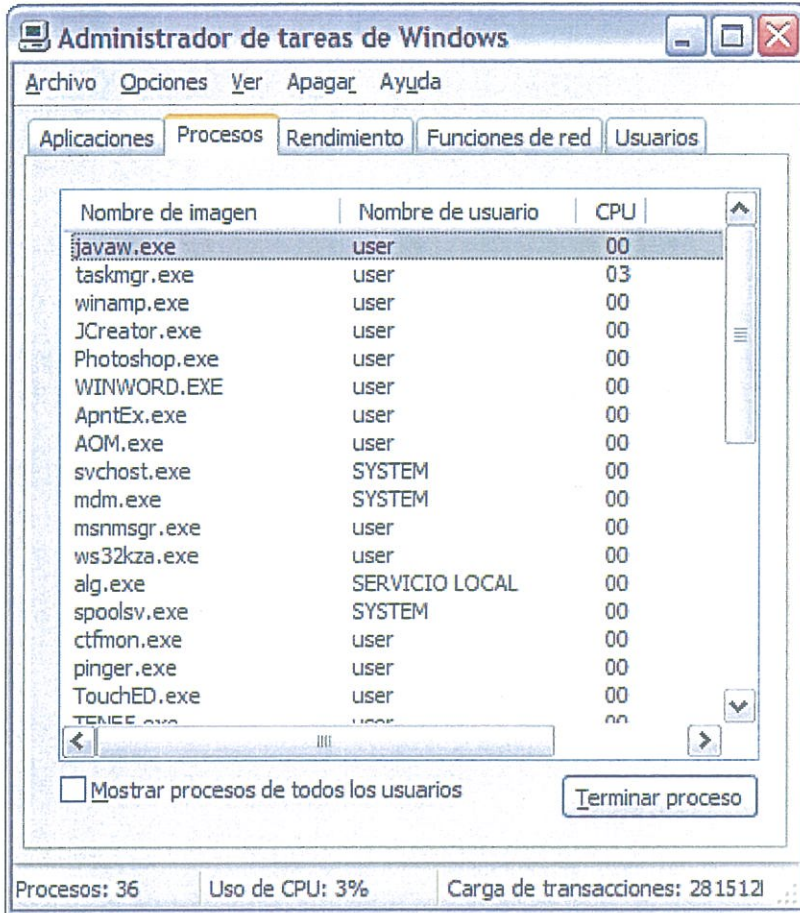


Figura 20. Administrador de Tareas de Windows

En esta ventana seleccionamos el proceso “javaw.exe” y presionamos el botón “Terminar proceso” para finalizar la ejecución del servidor web. En plataformas Linux para ver los procesos que se están ejecutando utilizamos el comando `ps`, una vez que reconozcamos el número de identificación del proceso “javaw”, utilizamos el comando `kill PID`, donde PID es el número de proceso, para terminar la ejecución del servidor web.

Si se utilizó la herramienta “java” para ejecutar el servidor, solamente es necesario cerrar la ventana de consola para terminar la ejecución del servidor.

### 3.1.7 Segundo modo de inicio: Aplicación GUI

Si el valor del parámetro `MODO_INICIO` es 2, entonces el servidor inicia con una interfase gráfica de usuario (GUI). De la misma forma que comentamos en la sección 3.1.6, si ejecutamos el servidor con la herramienta “javaw”, únicamente se abrirá la ventana con la interfase gráfica del servidor, caso contrario, si ejecutamos el servidor con la herramienta “java”, se abrirá además una ventana de consola.

El momento en que se carga en memoria la instancia de la clase `ServerWindow` creada por la clase `Inicio`, aparecerá en pantalla una ventana que contiene la interfase gráfica que nos permite controlar el servidor.

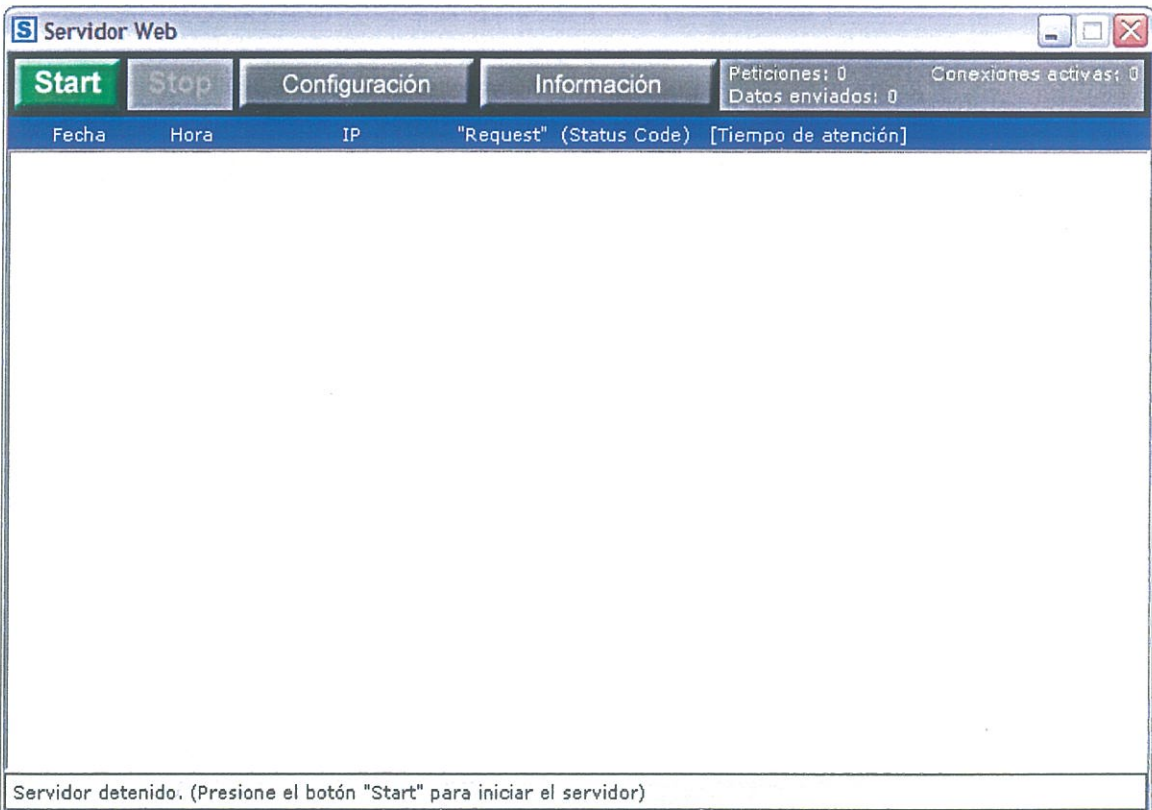


Figura 21. Aplicación GUI (Interfase gráfica del servidor)

Esta ventana está constituida básicamente por tres partes o secciones horizontales que tienen funciones específicas y que detallaremos a continuación.

### 3.1.7.1 Barra de herramientas

La barra de herramientas es la primera sección horizontal de la aplicación GUI del servidor.



Figura 22. Barra de herramientas

Podemos observar que esta barra contiene varios elementos:

- Botones de control. Tenemos dos botones que nos permiten controlar la actividad del servidor:

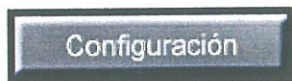


Al presionar el botón Start el servidor empieza a atender peticiones.



Al presionar el botón Stop el servidor detiene la atención peticiones.

- Botón Configuración.



Al presionar este botón se abrirá un diálogo de configuración del servidor a manera de ventana modal. Una ventana modal es aquella que bloquea la interacción del usuario con la ventana que la abrió hasta que la ventana modal sea

cerrada. A continuación mostramos la ventana de configuración que es construida y controlada por la clase *ConfigWindow*.

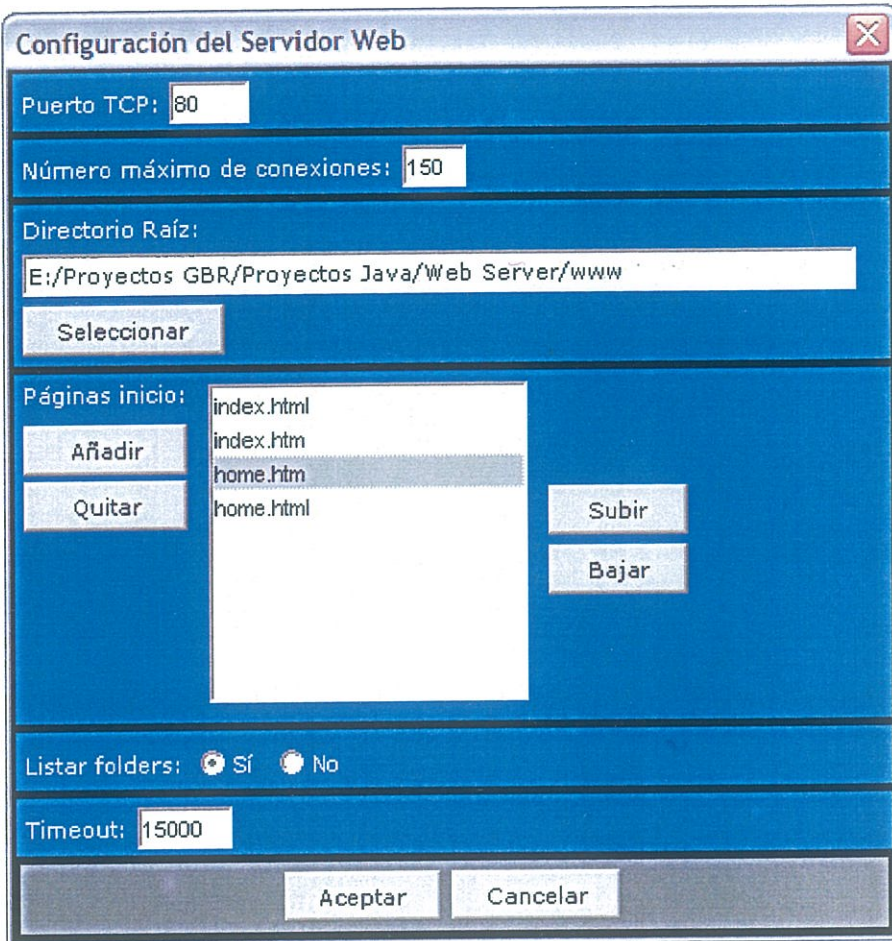


Figura 23. Ventana de configuración

El momento en que aparece esta ventana, se leen los valores de los parámetros de configuración del archivo de texto “ServerConfig.cfg” para que sean desplegados en los campos respectivos. En esta ventana podemos cambiar fácilmente los parámetros de configuración del servidor. No es necesario que el servidor esté detenido para cambiar la configuración ya que el momento en que aceptamos los cambios hechos, automáticamente el núcleo del servidor es notificado y empieza a trabajar en base a los nuevos valores definidos. El momento en que presionemos el botón Aceptar, la aplicación se encargará de modificar el archivo de texto “ServerConfig.cfg” para que incluya los valores de la nueva configuración. Si el archivo no existe, primero crea uno nuevo y luego lo modifica.

- Botón Información.



Al presionar este botón se mostrará un mensaje de información acerca del servidor web. El mensaje de información es el siguiente:



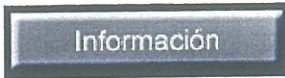
Figura 24. Mensaje acerca del servidor

- Panel de información adicional.



En este panel podemos encontrar información acerca del funcionamiento del servidor, representada a través de tres datos: el número de peticiones totales que se han atendido, la cantidad total de bytes que ha enviado el servidor en sus mensajes de respuesta, y el número de conexiones activas que están atendiendo a los clientes.

- Botón Información.



Al presionar este botón se mostrará un mensaje de información acerca del servidor web. El mensaje de información es el siguiente:



Figura 24. Mensaje acerca del servidor

- Panel de información adicional.



En este panel podemos encontrar información acerca del funcionamiento del servidor, representada a través de tres datos: el número de peticiones totales que se han atendido, la cantidad total de bytes que ha enviado el servidor en sus mensajes de respuesta, y el número de conexiones activas que están atendiendo a los clientes.

### 3.1.7.2 Panel de peticiones

El panel de peticiones es la segunda sección horizontal de la aplicación GUI del servidor. Aquí se despliega información acerca de las peticiones de los clientes, a medida que son atendidas por el servidor.

Fecha	Hora	IP	"Request" (Status Code)	[Tiempo de atención]
01/24/2004	14:26:25	127.0.0.1	"GET /images/menuTop_ON.gif HTTP/1.1"	{200} [0.03 seg]
01/24/2004	14:26:25	127.0.0.1	"GET /images/menuMision_OFF.gif HTTP/1.1"	{200} [0.07 seg]
01/24/2004	14:26:25	127.0.0.1	"GET /images/menuRusher_OFF.gif HTTP/1.1"	{200} [0.111 seg]
01/24/2004	14:26:25	127.0.0.1	"GET /images/menuVision_OFF.gif HTTP/1.1"	{200} [0.0 seg]
01/24/2004	14:26:25	127.0.0.1	"GET /images/menuMision_ON.gif HTTP/1.1"	{200} [0.02 seg]
01/24/2004	14:26:25	127.0.0.1	"GET /images/menuVision_ON.gif HTTP/1.1"	{200} [0.0 seg]
01/24/2004	14:26:26	127.0.0.1	"GET /images/menuContact_ON.gif HTTP/1.1"	{200} [0.05 seg]
01/24/2004	14:26:26	127.0.0.1	"GET /images/menuRusher_ON.gif HTTP/1.1"	{200} [0.02 seg]
01/24/2004	14:26:26	127.0.0.1	"GET /images/menuMid.gif HTTP/1.1"	{200} [0.0 seg]
01/24/2004	14:26:26	127.0.0.1	"GET /images/backTop.gif HTTP/1.1"	{200} [0.0 seg]
01/24/2004	14:26:26	127.0.0.1	"GET /images/menuBottom.gif HTTP/1.1"	{200} [0.02 seg]
01/24/2004	14:26:26	127.0.0.1	"GET /images/backMenu.gif HTTP/1.1"	{200} [0.01 seg]
01/24/2004	14:26:26	127.0.0.1	"GET /images/menuQuienes_OFF.gif HTTP/1.1"	{200} [0.03 seg]
01/24/2004	14:26:26	127.0.0.1	"GET /images/menuQuienes_ON.gif HTTP/1.1"	{200} [0.0 seg]
01/24/2004	14:26:26	127.0.0.1	"GET /images/menuContact_OFF.gif HTTP/1.1"	{200} [0.0 seg]
01/24/2004	14:26:28	127.0.0.1	"GET /top/mision.htm HTTP/1.1"	{200} [0.01 seg]
01/24/2004	14:26:28	127.0.0.1	"GET /mision.htm HTTP/1.1"	{200} [0.02 seg]
01/24/2004	14:26:28	127.0.0.1	"GET /images/tituloMision.gif HTTP/1.1"	{200} [0.01 seg]
01/24/2004	14:26:28	127.0.0.1	"GET /images/splash_girl.gif HTTP/1.1"	{200} [0.0 seg]
01/24/2004	14:26:28	127.0.0.1	"GET /top/vision.htm HTTP/1.1"	{200} [0.0 seg]
01/24/2004	14:26:29	127.0.0.1	"GET /vision.htm HTTP/1.1"	{200} [0.06 seg]
01/24/2004	14:26:29	127.0.0.1	"GET /images/tituloVision.gif HTTP/1.1"	{200} [0.01 seg]
01/24/2004	14:26:29	127.0.0.1	"GET /images/sofia.gif HTTP/1.1"	{200} [0.0 seg]

Figura 25. Panel de peticiones

Este panel está formado por dos partes: una cabecera fija que nos indica el formato en que se presenta la información de las peticiones de los clientes y un área de texto en donde se va agregando las líneas de esta información. A continuación presentamos el formato con que se construye cada línea de petición y una explicación correspondiente.

Fecha	Hora	IP	"Request" (Status Code)	[Tiempo de atención]
-------	------	----	-------------------------	----------------------

Fecha: La fecha en que se atendió la petición (día/mes/año).

Hora: La hora en la que se atendió la petición (HH:MM:SS).

IP: La dirección IP de la máquina cliente que hizo la petición.

Request: La línea de inicio del mensaje request enviado por el cliente.

Status Code: El código de estado de la respuesta devuelta por el servidor.

Tiempo de atención: El tiempo que le tomo al servidor atender la petición.



### 3.1.7.3 Barra de estado

La barra de estado es la tercera sección horizontal de la aplicación GUI del servidor. En esta barra se despliegan mensajes de información acerca del estado del servidor web.



Servidor detenido. (Presione el botón "Start" para iniciar el servidor)

Figura 26. Barra de estado (servidor detenido)

En la figura 26 podemos ver el mensaje inicial que se muestra cuando ejecutamos la aplicación GUI del servidor. Si encendemos el servidor presionando el botón Start, el mensaje de la barra de estado cambiará a lo siguiente:



IP = [ 205.235.14.171 ] Atendiendo peticiones en puerto 80 ( Servidor encendido el 01/25/2004 )

Figura 27. Barra de estado (servidor encendido)

En la figura 27 podemos ver el mensaje que se muestra en la barra de estado una vez que hemos encendido el servidor. Este nos indica la dirección IP de la máquina en que está corriendo el servidor web, el puerto TCP en el que se están atendiendo las peticiones, y la fecha en que se inició el servicio.

## **3.2 Descripción del currículo del taller**

Como mencionamos en la sección 3.1, el objetivo del proyecto es diseñar e implementar un servidor web modular multiplataforma que sirva como base de estudio para elaborar el contenido de un taller de programación de Java avanzado. Por esta razón el producto principal del proyecto vendría a ser el taller de Java.

El taller está dirigido a estudiantes o a profesionales que deseen ampliar o adquirir nuevos conocimientos del lenguaje Java, tomando como base el desarrollo de un servidor web para explicar los conceptos avanzados y las aplicaciones prácticas de éstos, al mismo tiempo.

El objetivo del taller es la formación y capacitación de desarrolladores de Java para que posean un valor agregado que ofrecer ante los nuevos sistemas y tecnologías que se están implantando cada vez con mayor velocidad.

El taller está dividido en partes o módulos de formación que incluyen, a su vez, una serie de capítulos didácticos. La presentación del material del taller es en la forma de CD y con un visualizador apropiado para acceder a dicho contenido.

### **3.2.1 Metodología**

El taller está diseñado para permitir que el estudiante pueda seguir un esquema de autoaprendizaje. El taller utiliza un esquema facilitador de aprendizaje que consiste en la presentación del contenido del currículo con una herramienta multimedia (Macromedia Flash). Utilizando esta herramienta, el alumno puede observar los ejemplos de código conjuntamente con el contenido del capítulo relacionado y de esa forma ajustar su propio ritmo de aprendizaje.

El taller también puede ser impartido de manera presencial con la ayuda de un instructor. Bajo esta modalidad se utiliza la misma herramienta multimedia para impartir

el taller, y adicionalmente el instructor puede ayudar y guiar a los estudiantes a alcanzar los objetivos de aprendizaje.

El curso de programación Java se caracteriza por su método activo, que implica la participación constante del alumno. Así mismo, para la modalidad o metodología presencial cada clase contará con un número de alumnos reducido de forma que el contacto con el profesor sea continuo, así como una mecánica fundamentalmente práctica.

### **3.2.2 Beneficios**

¿Quiénes se benefician del taller?

- La Institución Educativa: Agrega valor a sus programas de estudio al incluir educación para una posible certificación en tecnología Sun.
- El estudiante: Puede obtener una certificación de uno de los líderes de la industria de IT, lo que eleva enormemente el valor de su curriculum.
- La Industria: Cuenta con profesionales capacitados en tecnología de punta, que ofrecen solución inmediata a sus demandas.

### **3.2.3 Prerrequisitos del estudiante**

Para el pleno aprovechamiento de los conceptos avanzados de Java presentados en el currículo, el perfil de conocimientos del estudiante se debe ajustar a los siguientes prerrequisitos:

### 1. Conceptos básicos de programación lineal y estructurada.

Es necesario conocer las bases de la programación lineal, ya sea bajo Java o que conozca algún otro lenguaje de programación. Estos conceptos son los siguientes:

- Funciones
- Expresiones
- Arreglos
- Estructuras de control

### 2. Entendimiento de los conceptos básicos de la Programación Orientada a Objetos y sintaxis de Java.

Las explicaciones del currículo asumen que el estudiante puede entender y ha utilizado previamente los conceptos básicos de la programación orientada a objetos. Los conceptos que se utilizan en el currículo y que el estudiante debe de conocer son:

- Clase
- Objeto
- Métodos
- Sobrecarga de Métodos
- Tipos de Datos
- Manejo de Eventos

### 3. Conocimiento y manejo de las clases básicas de Swing o GUI de Java.

Esto incluye la utilización de métodos de las clases para manejo de interfaces gráficas de Java. Se requiere que el estudiante haya tenido experiencia o conozca las clases básicas de Swing (Java SDK 1.2.8) para generar una interfase gráfica básica.

- JFrame
- JPanel
- JButton
- JText

### 3.2.4 Proyección del perfil del instructor

Para la modalidad presencial del taller o curso se requiere que un instructor dirija y asesore a los estudiantes para el cumplimiento de los objetivos del curso. Para tal designio es necesario que el perfil del instructor cumpla con los siguientes requerimientos:

1. Nivel de estudios: Formación de tercer nivel en áreas relacionadas con Programación Java y desarrollo de aplicaciones Java.
2. Experiencia docente dando cursos de programación.
3. Conocimiento del contenido del taller y manejo fluido de los conceptos presentados en el currículo, así como también habilidad para resolver problemas relacionados con el JDK de Java.
4. Experiencia mínima: 1 año.
5. Edad: Indiferente.

### **3.2.5 Estructura general del currículo**

El currículo del taller está dividido en partes conceptuales que estructuralmente ayudan a entender el desarrollo de un servidor web. Cada parte se subdivide en capítulos con objetivos de aprendizaje específicos. Cada uno de estos capítulos presenta la siguiente estructura:

- Introducción al capítulo y exposición de objetivos a alcanzar con su estudio.
- Esquema de los contenidos.
- Desarrollo del capítulo.
- Glosario de términos nuevos.

En la siguiente sección presentamos una descripción detallada del contenido y de los objetivos de cada capítulo.

### **3.2.6 Descripción detallada del material del currículo**

#### **INTRODUCCIÓN**

Esta parte está orientada a ofrecer una explicación general del contenido del currículo, los prerrequisitos para los estudiantes y del manejo de la interfase de la herramienta multimedia para acceder al contenido del currículo.

En esta primera parte del currículo se examinarán los siguientes puntos:

- Introducción al curso
- Requerimientos del curso
- Descripción del manejo de la interfase del currículo

## **PARTE I – Definición de requerimientos y diseño**

Se da una descripción del funcionamiento de los servidores web y de los requerimientos de diseño para desarrollar uno con Java. Esta parte está dirigida a proveer al estudiante una guía para de desarrollo de software del servidor web a lo largo de todo el taller.

### **Capítulo 1. Servidores Web**

Se tratan las definiciones de un servidor web genérico y su funcionamiento. Al final de este capítulo el estudiante estará en capacidad de describir de forma general cómo funciona un servidor web, paso a paso, y además tendrá una guía de lo que se espera desarrollar a lo largo de todo el taller. Los puntos a tratar en este capítulo son:

- ¿Qué es un servidor web?
- ¿Cómo funciona un servidor web?

### **Capítulo 2. Crear un servidor web**

En este capítulo se presenta una introducción a los requerimientos del servidor web a desarrollarse, también se establece una arquitectura basada en los conocimientos del capítulo anterior. El modelo servirá como referencia a los estudiantes para los avances en código de subsecuentes capítulos. Los temas de éste capítulo son:

- Definición de requerimientos del servidor
- Diseño y Arquitectura
- Modo de inicialización

## **PARTE II – Protocolo HTTP**

Esta parte describe el funcionamiento del principal protocolo de comunicación utilizado entre servidores web y clientes (navegadores). El objetivo de esta parte es obtener una comprensión de cómo se transmite la información hacia y desde el servidor web, de esta forma el estudiante podrá conocer lo que se debe implementar en Java para lograr un servidor funcional.

### **Capítulo 3. Conceptos de HTTP**

Se presentan los conceptos generales para entender el protocolo de comunicación entre el servidor y el cliente (navegador). El objetivo de este capítulo es que el estudiante conozca cómo funciona el protocolo HTTP para poder aplicar esos conocimientos en implementaciones de clases futuras.

Los tópicos de este capítulo son:

- Funcionamiento del protocolo HTTP
- Identificadores de recursos

### **Capítulo 4. Mensajes HTTP**

Este capítulo contiene material complementario al capítulo anterior. Se describen los principales tipos de mensajes del protocolo HTTP. EL objetivo de este capítulo es brindar al estudiante el conocimiento del formato de los datos que se utilizan en la comunicación con un servidor web, y de esa forma capacitarle para que pueda implementar las clases que realizan este trabajo.

Los temas que trata este capítulo son:

- Mensaje tipo Request
- Mensaje tipo Response



## **PARTE III – Parámetros de configuración del servidor**

En esta parte se empieza el desarrollo del servidor en sí con la generación de código por parte del estudiante. Se desarrollan las primeras clases relacionadas con los parámetros de configuración del servidor utilizando los conceptos estudiados en las partes anteriores. El objetivo de esta parte es entender los parámetros de configuración del servidor y manejar la permanencia de la configuración mediante archivos.

### **Capítulo 5. Conceptos de Java**

Este capítulo ofrece una explicación de los conceptos de programación Java que se utilizarán en esta parte del currículo. El objetivo de este capítulo en particular es que el estudiante pueda describir y manejar las clases Properties y clases relacionadas con la Entrada/Salida (I/O) de datos desde y hacia archivos.

Los temas de este capítulo son:

- Propiedades (java)
- Manejo de archivos

### **Capítulo 6. Parámetros de configuración del servidor**

Este capítulo presenta una explicación de los distintos parámetros de configuración de un servidor web tradicional. También se empieza la generación del código relacionado con el manejo de dichos parámetros en la implementación del servidor web.

El objetivo al finalizar este capítulo es que el estudiante pueda describir para que sirve cada parámetro y esté en capacidad de desarrollar las distintas clases relacionadas a la configuración del servidor.

- Parámetros y sus usos
- Archivo de texto con parámetros
- Código de clase configuración

## PARTE IV – Núcleo del servidor

Aquí se introducen los conceptos avanzados de hilos y sockets en Java y se desarrollan las clases que definen al servidor web como aplicación. En esta parte también se definen las clases que sirven de comunicadores entre un cliente y el servidor web.

### Capítulo 7. Conceptos de Java

Aquí se trata los conceptos de Java relacionados con la parte del desarrollo y funcionamiento de las clases que darán funcionalidad al servidor para atender conexiones. Específicamente se tratan los conceptos de hilos en el servidor para atender a varios clientes simultáneamente y sockets para establecer un canal de comunicación con cada cliente. Al final de este capítulo el estudiante deberá manejar fluidamente los conceptos de este capítulo y tener una idea de cómo aplicarlos al desarrollo del servidor. Los temas de este capítulo son:

- Threads
- Sockets

### Capítulo 8. Inicialización del servidor

Este capítulo describe la implementación de las clases base sobre las cuales se construirá la funcionalidad del mismo. Se define además la forma de iniciar el servidor basándose en los requerimientos de la parte I. Al final de este capítulo el estudiante deberá conocer y ser capaz de definir las clases para iniciar el servidor y dejarlo corriendo como una aplicación. Los temas de este capítulo son:

- Clase Inicio
- Clase Núcleo

## Capítulo 9. Comunicación con Clientes

En este capítulo se introduce las clases que establecen una comunicación entre el servidor y el cliente. Al final de este capítulo el estudiante deberá comprender el funcionamiento de las clases desarrolladas para este fin y deberá implementar las clases para establecer las conexiones. Los temas de este capítulo son:

- Clase Conexión
- Clase GrupoConexiones

## PARTE V – Petición y respuesta

Esta parte trata los conceptos y el desarrollo de las partes fundamentales de manejo de peticiones y envío de las respuestas del servidor web. El resultado de esta parte es obtener las clases que implementan el servicio de petición y respuesta del servidor web.

## Capítulo 10. Conceptos de Java

El capítulo trata los conceptos de Java relacionados con esta parte. Se hace un breve repaso del protocolo HTTP para que el estudiante recuerde los tipos de mensaje request y response, con el propósito de que pueda desarrollar la clase que se encarga de procesar los mensajes de petición y la clase que se encarga de construir el mensaje de respuesta.

Al final de este capítulo el estudiante debe ser capaz de manejar los métodos y las clases para manejos de cadenas de caracteres o strings. Los temas de este capítulo son:

- Repaso del protocolo HTTP
- Conceptos de java – Strings

## **Capítulo 11. Peticiones de clientes**

Este capítulo ofrece la explicación y desarrollo de la clase para atender las peticiones HTTP de los clientes. Dependiendo del tipo de mensaje HTTP se analizará y se atenderá la petición de distinta forma.

El objetivo de este capítulo es que el estudiante pueda describir e implementar la clase que recoge las peticiones los clientes. Los temas de este capítulo son:

- Mensaje tipo Request
- Clase Request

## **Capítulo 12. Respuestas del servidor**

Este capítulo ofrece la explicación y desarrollo de la clase para responder a los clientes. Dependiendo del tipo de mensaje de petición HTTP recibido se responderá al cliente de distinta forma.

El objetivo de este capítulo es que el estudiante pueda describir e implementar la clase que envía las respuestas a los clientes. Los temas de este capítulo son:

- Mensaje tipo Response
- Clase Response

## **PARTE VI – Interfase gráfica del servidor**

En esta parte se presenta conceptos adicionales para implementar una interfase gráfica del servidor, la cual servirá para la administración de la configuración y para el monitoreo y registro de las conexiones al servidor. El objetivo de esta parte es que el estudiante desarrolle una aplicación gráfica para controlar el servidor y para guardar registros.

### **Capítulo 13.** Interfase gráfica del servidor

En este capítulo se trata tanto el desarrollo de la interfase gráfica para el segundo modo de inicio del servidor como el módulo gráfico para manejar los parámetros de configuración del servidor de forma directa.

El objetivo de este capítulo es que el estudiante, luego de generar un servidor web funcional, pueda agregar funcionalidades gráficas para complementar el servidor. Los temas de este capítulo son:

- Descripción de aplicación
- Descripción del módulo para manejo de la configuración
- Código de aplicación

### **Capítulo 14.** Registro de accesos o logs

Este capítulo describe una funcionalidad adicional del servidor. Se detalla un código sencillo para guardar registros de peticiones de los clientes con los respectivos resultados de las respuestas.

El objetivo de este capítulo es que el estudiante implemente la funcionalidad de guardar registros del servidor en un archivo de texto y pueda opcionalmente hacer posteriores análisis de tráfico y peticiones en base al contenido de este archivo. Los temas de este capítulo son:

- Puntos claves para guardar registros
- Código para guardar registros de peticiones y clientes

# CONCLUSIONES Y RECOMENDACIONES

## Conclusiones

- Java es un lenguaje de programación que permite y facilita el desarrollo de todo tipo de aplicaciones. Podemos enumerar entre éstas a las aplicaciones empresariales de gran escala, aplicaciones distribuidas, páginas web con contenido dinámico e interactivo, aplicaciones para dispositivos electrónicos de consumo como teléfonos celulares y PDA, etc.
- La simplicidad, la programación orientada a objetos, la robustez, la seguridad, la portabilidad, etc., son características que hacen del lenguaje de programación Java uno de los preferidos a la hora de desarrollar una aplicación, por eso es que el desarrollador puede enfocarse en las particularidades de la implementación de un sistema más que en los aspectos relacionados con la tecnología que se está utilizando.
- Esencialmente el funcionamiento de los protocolos de comunicación se basa en el intercambio de mensajes estructurados de cierta forma, para permitir así el intercambio de información.
- Podría retomarse este proyecto como base para la construcción de un servidor web que soporte un mayor número de parámetros de configuración, implemente más métodos de petición y atienda tanto recursos estáticos como dinámicos.
- El desarrollo del currículo a través de herramientas multimedia permite a los estudiantes tomar las riendas de su capacitación en lenguaje Java a través de un proceso de autoaprendizaje. Estas herramientas nos permitieron transmitir de manera más efectiva los conceptos claves que nos llevaron a alcanzar los objetivos del proyecto.

- El proceso de E-Learning, implementado a través de las herramientas multimedia, ofrece a los alumnos una capacitación didáctica e interactiva y al mismo tiempo representa para las instituciones educativas una reducción en los costos y la posibilidad de expandir la cobertura a más individuos.

## Recomendaciones

- El presente proyecto podría ser considerado por los directivos de la facultad de Ingeniería de Sistemas de la Universidad San Francisco de Quito como material de un taller o seminario que complemente los estudios en programación avanzada de los alumnos del Politécnico. Se recomienda esto, porque no solo se aprenden conceptos relacionados con el lenguaje Java, sino que además se profundiza en los fundamentos de los procesos de comunicación entre aplicaciones. Esto les permite a los estudiantes comprender de mejor manera el funcionamiento de una de las arquitecturas más utilizadas actualmente en los sistemas de información: la arquitectura cliente-servidor.
- Como seguimiento de este proyecto se podrían añadir modificaciones en el código del servidor que nos permitan desplegar en pantalla la entrada y salida de mensajes, con la finalidad de que el estudiante pueda visualizar contenidos y estructuras reales de los mensajes de petición y respuesta.
- Los profesionales en el área de desarrollo de aplicaciones distribuidas deben tener muy en cuenta las especificaciones de protocolos y estándares con el fin de asegurar la interoperabilidad entre distintos sistemas.
- Es importante incentivar a los estudiantes de Ingeniería de Sistemas a que tomen cursos de Multimedia, ya que, a parte de contar con herramientas de apoyo para el desarrollo de aplicaciones, estos cursos les permitirán desarrollar soluciones que faciliten los procesos de comunicación entre el usuario y la aplicación.

## **ANEXOS**

**( Código fuente del Servidor Web )**



```

/*
*****
*
* Inicio.java
*
* Esta clase lee la variable MODO_INICIO del archivo de
* configuracion para determinar como arranca el servidor:
*
* - MODO_INICIO = 1
*   El servidor se ejecuta sin interface grafica
*
* - MODO_INICIO = 2
*   El servidor se ejecuta como una aplicacion (GUI)
*
*****
*/

public class Inicio{

    ////////////////////////////////////////////////////////////////////

    public static void main( String args[] ) {

        Configuracion config = new Configuracion();

        // Empieza sin interface grafica
        if ( config.getModoInicio() == 1 ){

            Nucleo servidor = new Nucleo( config );

        }

        // Empieza con aplicacion (GUI)
        if ( config.getModoInicio() == 2 ){

            new ServerWindow();

        }

    } // fin del metodo main

    ////////////////////////////////////////////////////////////////////

} // fin de la clase Inicio

```

```

/*
*****
*
* Configuracion.java
*
* Esta clase lee el archivo de configuracion ServerConfig.cfg
* que se encuentra en el directorio /config/
*
*****
*/

import java.io.*;
import java.util.*;

public class Configuracion {

    // Archivo de configuracion
    private String archivo_Config = "config" + File.separator + "ServerConfig.cfg";

    // Lista de propiedades de configuracion
    private Properties propiedades = new Properties();

    // Variables de configuracion por defecto
    private int var_puerto_TCP = 80;
    private int var_modulo_inicio = 2;
    private int var_numero_conexiones = 150;
    private String var_directorio_raiz = System.getProperty("user.dir") +
        System.getProperty("file.separator") +
        "www" +
        System.getProperty("file.separator");

    private Vector var_paginas_inicio = new Vector();
    private String var_listar_folders = "true";
    private int var_timeout = 15000;

    // Tabla de mapeo de MIME types
    private Hashtable htMimeTypes = new Hashtable();

    ///////////////////////////////////////////////////////////////////

    public Configuracion() {

        try {

            cargarPropiedades();
            cargarMimeTypes();

        }
        catch ( Exception e ) {

            System.out.println("Error en la lectura del archivo de configuracion");

        }

    } // fin del metodo constructor

    ///////////////////////////////////////////////////////////////////

```

```

public void cargarPropiedades() {

String valor;
File file_Cfg = new File( archivo_Config );

if ( file_Cfg.exists() ) {

////////////////////////////////////
// cargar variables de configuracion en propiedades
////////////////////////////////////

try{
    InputStream inputStr;
    inputStr = new BufferedInputStream( new FileInputStream(file_Cfg) );
    propiedades.load( inputStr );
    inputStr.close();
}
catch( Exception e ){}

////////////////////////////////////
// leer propiedades
////////////////////////////////////

valor = propiedades.getProperty( "PUERTO_TCP", "80" );
var_puerto_TCP = Integer.parseInt( valor );

////////////////////////////////////

valor = propiedades.getProperty( "MODO_INICIO", "2" );
var_modos_inicio = Integer.parseInt(valor);

////////////////////////////////////

valor = propiedades.getProperty( "NUMERO_CONEXIONES", "150" );
var_numero_conexiones = Integer.parseInt(valor);

////////////////////////////////////

valor = propiedades.getProperty( "DIRECTORIO_RAIZ", var_directorio_raiz );

if ( valor != null ) {
    if ( new File(valor).isDirectory() ) var_directorio_raiz = valor;
}

////////////////////////////////////

valor = propiedades.getProperty( "PAGINAS_INICIO", "index.html" );

StringTokenizer items = new StringTokenizer( valor, "," );

while ( items.hasMoreTokens() ) var_paginas_inicio.add(items.nextToken());

////////////////////////////////////

valor = propiedades.getProperty( "LISTAR_FOLDERS", "true" );
var_listar_folders = valor;

```

```

////////////////////////////////////
valor = propiedades.getProperty( "TIMEOUT", "15000" );
var _timeout = Integer.parseInt(valor);

////////////////////////////////////

}

} // fin del metodo cargarPropiedades

////////////////////////////////////

private void cargarMimeTypes() {

    putMimeType( "", "content/unknown" );
    putMimeType( "htm", "text/html" );
    putMimeType( "shtml", "text/html" );
    putMimeType( "html", "text/html" );
    putMimeType( "shtm", "text/html" );
    putMimeType( "asp", "text/html" );

    putMimeType( "txt", "text/plain" );
    putMimeType( "text", "text/plain" );
    putMimeType( "ini", "text/plain" );
    putMimeType( "c", "text/plain" );
    putMimeType( "cc", "text/plain" );
    putMimeType( "c++", "text/plain" );
    putMimeType( "h", "text/plain" );
    putMimeType( "pl", "text/plain" );
    putMimeType( "java", "text/plain" );
    putMimeType( "inf", "text/plain" );

    putMimeType( "jpeg", "image/jpeg" );
    putMimeType( "jpg", "image/jpeg" );
    putMimeType( "jpe", "image/jpeg" );
    putMimeType( "gif", "image/gif" );
    putMimeType( "png", "image/png" );
    putMimeType( "bmp", "image/x-bitmap" );

    putMimeType( "mov", "video/quicktime" );
    putMimeType( "qt", "video/quicktime" );

    putMimeType( "js", "application/x-javascript" );

    putMimeType( "midi", "audio/midi" );
    putMimeType( "mid", "audio/midi" );

    putMimeType( "mp2", "audio/mpeg" );
    putMimeType( "mp3", "audio/mpeg" );
    putMimeType( "mpga", "audio/mpeg" );

    putMimeType( "mpe", "video/mpeg" );
    putMimeType( "mpeg", "video/mpeg" );
    putMimeType( "mpg", "video/mpeg" );
    putMimeType( "avi", "video/x-msvideo" );
}

```

```

putMimeType("php", "application/x-httpd-php");
putMimeType("php3", "application/x-httpd-php3");
putMimeType("phtml-msql2", "application/x-httpd-php-msql2");
putMimeType("phtml", "application/x-httpd-php-msql2");
putMimeType("phtml", "application/x-httpd-php-msql2");

putMimeType("ps", "application/postscript");

putMimeType("ra", "audio/x-realaudio");
putMimeType("ram", "audio/x-pn-realaudio");

putMimeType("pdf", "application/pdf");
putMimeType("swf", "application/x-shockwave-flash");

putMimeType("uu", "application/octet-stream");
putMimeType("exe", "application/octet-stream");
putMimeType("bin", "application/octet-stream");
putMimeType("class", "application/octet-stream");
putMimeType("dll", "application/octet-stream");

putMimeType("zip", "application/zip");
putMimeType("tar", "application/x-tar");

putMimeType("au", "audio/basic");
putMimeType("snd", "audio/basic");
putMimeType("wav", "audio/x-wav");

putMimeType("xml", "text/xml");
putMimeType("dtd", "text/xml");

putMimeType("ai", "application/postscript");
putMimeType("eps", "application/postscript");

putMimeType("aif", "audio/x-aiff");
putMimeType("aiff", "audio/x-aiff");

putMimeType("css", "text/css");

putMimeType("dcr", "application/x-director");
putMimeType("dir", "application/x-director");
putMimeType("dxr", "application/x-director");

putMimeType("doc", "application/msword");
putMimeType("ppt", "application/powerpoint");

putMimeType("gtar", "application/x-gtar");
putMimeType("gz", "application/x-gzip");

putMimeType("rtf", "application/rtf");

putMimeType("tif", "image/tiff");
putMimeType("tiff", "image/tiff");

putMimeType("vrml", "model/vrml");
putMimeType("wrl", "model/vrml");
} // fin del metodo cargarMimeTypes

```

```
////////////////////////////////////
public void putMimeType( String extension, String tipo ) {
    htMimeTypes.put( extension, tipo );
} // fin del metodo putMimeType
////////////////////////////////////
public String getMimeType( String extension ) {
    String mimeType = (String) htMimeTypes.get(extension);
    if ( mimeType != null )
        return mimeType;
    else
        return "unknown/unknown";
} // fin del metodo getMimeType
////////////////////////////////////
public int getPuertoTCP() {
    return var_puerto_TCP;
}
////////////////////////////////////
public int getModoInicio() {
    return var_modos_inicio;
}
////////////////////////////////////
public int getNumeroConexiones() {
    return var_numero_conexiones;
}
////////////////////////////////////
public String getDirectorioRaiz() {
    return var_directorio_raiz;
}
////////////////////////////////////
public Vector getPaginasInicio() {
    return var_paginas_inicio;
}
////////////////////////////////////
```

```
public boolean listarFolders() {  
    if ( var_listar_folders.toLowerCase().equals("true" )  
        return true;  
    else  
        return false;  
}  
  
////////////////////////////////////  
public int getTimeout() {  
    return var_timeout;  
}  
  
////////////////////////////////////  
public String getProperty( String p ) {  
    return propiedades.getProperty( p );  
}  
  
////////////////////////////////////  
} // fin de la clase Configuracion
```

```

/*
*****
*
* Nucleo.java
*
* Esta clase es la base del servidor web. Controla la creacion
* de nuevas conexiones para atender peticiones de los clientes
*
*****
*/

import java.io.*;
import java.net.*;
import java.util.*;
import java.text.*;
import javax.swing.*;

public class Nucleo extends Thread {

    private String direccionIP;

    private GrupoConexiones grupoCon;

    private int modoInicio;

    private ServerSocket serverSocket;
    private Socket userSocket;

    public Configuracion config;

    SimpleDateFormat dateFormat = new SimpleDateFormat( "MM/dd/yyyy" );

    ///////////////////////////////////////////////////////////////////

    public Nucleo( Configuracion config ) {

        this.config = config;

        try{

            direccionIP = InetAddress.getLocalHost().getHostAddress();

        }
        catch (Exception e) {}

        modoInicio = config.getModoInicio();

        startServer();

    } // fin del metodo constructor

    ///////////////////////////////////////////////////////////////////

```



```

public boolean startServer() {

    grupoCon = new GrupoConexiones( config );

    try{
        serverSocket = new ServerSocket( config.getPuertoTCP() );
        userSocket = new Socket();

        this.start();
        if( modoInicio == 1 ){
            System.out.println("SERVIDOR WEB v1.0");
            System.out.println("-----");
            System.out.println("Proyecto Final\n\n" +
                "Realizado por:\n" +
                "German Bastidas\n" +
                "Carlos Yoncon\n\n" +
                "Universidad San Francisco de Quito\n" +
                "Enero del 2004");
            System.out.println("-----");
            System.out.println("- IP: " + direccionIP );
            System.out.println("-----");

            JOptionPane.showMessageDialog( null,
                "Servidor Web ENCENDIDO:\n\n"+
                "El servidor web esta atendiendo\n"+
                "peticiones en el puerto " + config.getPuertoTCP() + ".\n\n" +
                "Fecha: " + dateFormat.format( new Date() ) + "\n\n",
                "",
                JOptionPane.INFORMATION_MESSAGE);
        }
        if( modoInicio == 2 ){
            ServerWindow.setStatus( "IP = [ " + direccionIP + " ]" +
                " Atendiendo peticiones en puerto " + config.getPuertoTCP() +
                " ( Servidor encendido el " + dateFormat.format(new Date()) + " )");
        }
        return true;
    }
    catch ( BindException be ) {
        JOptionPane.showMessageDialog( null,
            "ERROR:\n\nUna aplicacion " +
            "esta utilizando el puerto " + config.getPuertoTCP() + ".\n\n" +
            "Cambie el numero de puerto TCP.\n\n",
            "Error al iniciar el servidor",
            JOptionPane.ERROR_MESSAGE);
        if( modoInicio == 1)
            System.exit(1);
        if( modoInicio == 2)
            ServerWindow.setStatus("Error: Una aplicacion esta utilizando el" +
                "puerto" + config.getPuertoTCP() +
                ".\nCambie el numero de puerto TCP.");
    }
    catch( Exception e ){

    }

    return false;
} // fin del metodo startServer

```

```
////////////////////////////////////////////////////////////////
```

```
public void stopServer() {  
    try{  
        grupoCon = null;  
        userSocket.close();  
        serverSocket.close();  
    }  
    catch( IOException e ){}  
  
    if( modoInicio == 2 )  
        ServerWindow.setStatus("Servidor detenido. (Presione el boton \"Start\" +  
                                \"para iniciar el servidor)\");  
  
} // fin del metodo stopServer
```

```
////////////////////////////////////////////////////////////////
```

```
public void run() {  
    try{  
        while ( true ) {  
            userSocket = serverSocket.accept();  
  
            synchronized( grupoCon ){  
                if ( !grupoCon.isEmpty() ) {  
                    Conexion con = grupoCon.extraer();  
  
                    if( modoInicio == 2)  
                        ServerWindow.setNumConexiones(grupoCon.conexionesActuales());  
  
                    con.setSocket( userSocket );  
  
                }  
                else {  
                    userSocket.close();  
                }  
            }  
        }  
  
    }  
    catch( IOException e ){}  
  
} // fin del metodo run()
```

```
////////////////////////////////////////////////////////////////
```

```
} // fin de la clase Nucleo
```

```

/*
*****
*
* GrupoConexiones.java
*
* Esta clase es una pila que contiene los objetos Conexion
* creados para atender las peticiones de los clientes. Provee
* los metodos necesarios para poder manejar la utilizacion de
* conexiones.
*
*****
*/

import java.util.*;

public class GrupoConexiones {

    private Configuracion config;
    private Vector conexiones;
    private int maxConexiones;
    private int modoInicio;

    ////////////////////////////////////////////////////////////////////

    public GrupoConexiones( Configuracion config ) {

        this.config = config;

        maxConexiones = config.getNumeroConexiones();

        conexiones = new Vector();

        for ( int i = 1; i <= maxConexiones; i++ ){

            Conexion newCon = new Conexion( config, this );
            newCon.setName( "Conexion " + i );
            newCon.start();

            conexiones.addElement( newCon );

        }

    } // fin del metodo constructor

    ////////////////////////////////////////////////////////////////////

    public int conexionesActuales(){
        return ( maxConexiones - conexiones.size() );
    }

    ////////////////////////////////////////////////////////////////////

    public boolean isEmpty() {
        return conexiones.isEmpty();
    }

    ////////////////////////////////////////////////////////////////////

```

```
public Conexion extraer() {
    return (Conexion)conexiones.remove( 0 );
}

/////////////////////////////////////////////////////////////////

public void insertar(Conexion c) {
    conexiones.addElement(c);
}

/////////////////////////////////////////////////////////////////

} // fin de la clase GrupoConexiones
```

```

/*
*****
*
* Conexion.java
*
* Esta clase atiende peticiones de los clientes que se
* conectan al servidor web
*
*****
*/

import java.io.*;
import java.net.*;
import java.util.*;
import java.text.*;

public class Conexion extends Thread {

    private Socket userSocket;
    private Configuracion config;

    private Request request;
    private Response response;

    private String resultado;

    private GrupoConexiones grupoCon;

    private final int BUF_SIZE = (10*1024);
    private byte[] buf;

    SimpleDateFormat dateFormat = new SimpleDateFormat( "MM/dd/yyyy HH:mm:ss" );

    ///////////////////////////////////////////////////////////////////

    public Conexion( Configuracion cfg, GrupoConexiones gc ) {

        config = cfg;
        grupoCon = gc;
        resultado = "";
        buf = new byte[ BUF_SIZE ];
        userSocket = null;

    } // fin del metodo constructor

    ///////////////////////////////////////////////////////////////////

    public synchronized void setSocket( Socket so ) throws SocketException {

        userSocket = so;
        notify();

    } // fin del metodo setSocket

    ///////////////////////////////////////////////////////////////////

```

```

public synchronized void run() {
    while( true ){
        try {
            wait(); // Espera hasta que el nucleo le asigne un socket y pueda
                   // procesar la peticion
        }
        catch (InterruptedException e) {}

        procesarPeticion();

        if ( config.getModoInicio() == 2 && resultado.length() > 0 ){
            ServerWindow.display( resultado );
            ServerWindow.sumarPeticion();
        }

        userSocket = null;

        // Devolver este objeto a la pila de conexiones

        synchronized( grupoCon ){

            grupoCon.insertar( this );

            if( config.getModoInicio() == 2 )
                ServerWindow.setNumConexiones( grupoCon.conexionesActuales() );

        }

    }

} // fin del metodo run

////////////////////////////////////

private void procesarPeticion() {

    try {

        long tInicio = System.currentTimeMillis();

        userSocket.setSoTimeout( config.getTimeout() );
        request = new Request( userSocket );

        long tFinal = System.currentTimeMillis();

        userSocket.setSoTimeout(0);
        response = new Response( userSocket );

        if( request.getStatusCode() == 200 )
            procesarURL( request.getRequestURL() );
        else
            response.construirHeader( request.getStatusCode(), "", 0, "" );

    }

}

```

```

resultado = "";
resultado += dateFormat.format( new Date() ) + " ";
resultado += "      " + request.getRemoteAddr() + "      ";
resultado += "\"\" + request.getRequestHeader() + "\"";
resultado += " ( " + response.getResponse() + " )";

if ( resultado.length() > 0 ) serverLog();

resultado += " [ " + ( tFinal - tInicio ) / 1000.0 + " seg]";

request.close();
response.close();

}
catch(Exception e){

} // fin del metodo procesarPeticion

////////////////////////////////////

private void procesarURL( String url ) {

try{

File target;
long size = 0;
String location = "";
String contentType = "";

String absPath = config.getDirectorioRaiz();
String relPath = URLDecoder.decode( url, "UTF-8" );

if ( relPath.indexOf("/data:") != -1 ){

String str = relPath.substring( relPath.indexOf(":") + 1 );
absPath = System.getProperty("user.dir") +
System.getProperty("file.separator");
if ( str.equals("estilos" ) )
relPath = "estilos.css";
else if ( str.equals("folder" ) )
relPath = "imageFolder.gif";
else if ( str.equals("file" ) )
relPath = "imageFile.gif";
}

target = new File( absPath , relPath );
////////////////////////////////////
// Si el recurso no existe
////////////////////////////////////

if( !target.exists() ) {
response.construirHeader( ConstantesHTTP.HTTP_NOT_FOUND,
contentType,
size, location );

return;
}
}
}

```

```

////////////////////////////////////
// Si el recurso es un directorio
////////////////////////////////////

if( target.isDirectory() ) {

    if( !relPath.endsWith("/") ) {
        location = url + "/";
        response.construirHeader( ConstantesHTTP.HTTP_SEE_OTHER,
                                contentType,
                                size, location );

        return;
    }

    // Verificar si existe en el directorio una pagina de inicio

    Iterator paginasInicio = config.getPaginasInicio().iterator();
    boolean existeIndex = false;

    while( paginasInicio.hasNext() ) {
        String nombre = (String)paginasInicio.next();
        File index = new File( target, nombre );
        if( index.exists() ) {
            target = index;
            existeIndex = true;
        }
    }

    if ( !existeIndex ){
        if ( config.listarFolders() ){
            location = url;
            contentType = "text/html";
            response.construirHeader( ConstantesHTTP.HTTP_OK,
                                    contentType,
                                    size, location );

            mostrarDirectorio( target );
        }
        else{
            userSocket.setKeepAlive( true );
            response.construirHeader( ConstantesHTTP.HTTP_FORBIDDEN,
                                    "", 0, "" );
        }
    }
}

////////////////////////////////////
// Si el recurso es un archivo
////////////////////////////////////

if( target.isFile() ) {

    String nombre = target.getName();

    String extension;

    int indexPoint = nombre.lastIndexOf('.');

```



```

    if ( indexPoint > 0 ) {

        extension = nombre.substring( indexPoint + 1 );
        contentType = config.getMimeType( extension );
        size = target.length();
        location = request.getHeader("Host");

    }

    response.construirHeader( ConstantesHTTP.HTTP_OK,
        contentType, size, location );
    sendFile( target );

}

}
catch( Exception e ){
    System.out.println("Error al procesar el URL");
}
} // fin del metodo procesarURL

////////////////////////////////////

private void sendFile( File f ) throws IOException {

    InputStream is = new FileInputStream( f.getAbsolutePath() );

    try {

        int n;
        while ( ( n = is.read(buf) ) > 0 ) response.write( buf, 0, n );

    }
    finally {
        is.close();
    }

    if ( config.getModoInicio() == 2 )
        ServerWindow.sumarBytes( f.length() );

} // fin del metodo sendFile

////////////////////////////////////

private void mostrarDirectorio( File dir ) throws IOException {

    StringWriter writer = new StringWriter();

    String tamano = "";
    String html = "";

    String raiz = config.getDirectorioRaiz();
    String path = dir.getPath().substring( raiz.length() );

    path = path.replace( '\\', '/' );
    if ( path.equals("/") ) path = "";

```

```

html += "<html><head>\n";
html += "<link href=\"/data:estilos\" rel=\"stylesheet\" +
html += "type=\"text/css\">\n";
html += "</head>\n";
html += "<body bgcolor=\"#FFFFFF\" color=\"#000000\">\n";
html += "<font face=\"verdana\"><h3>Listado del directorio \" + path;
html += "\</h3></font>";

html += "<table>\n";
html += "<tr class=\"titulo\">\n";
html += "<td width=\"20\" align=\"left\"></td>\n";
html += "<td width=\"200\" align=\"left\">Nombre</td>\n";
html += "<td width=\"20\" align=\"left\"></td>\n";
html += "<td width=\"100\" align=\"right\">Tamano</td>\n";
html += "<td width=\"20\" align=\"left\"></td>\n";
html += "<td width=\"100\" align=\"left\">Tipo</td>\n";
html += "<td width=\"20\" align=\"left\"></td>\n";
html += "<td width=\"200\" align=\"left\">Ultima modificacion</td>\n";
html += "</tr>\n";

if( !dir.getName().equals("") ) {

    html += "<tr class=\"item\">\n";
    html += "<td><img src=\"/data:folder\"></td>\n";
    html += "<td colspan=\"7\"> +
        "<a class=\"directorio\" href=\"..\">../ (Parent)</a></td>\n";
    html += "</tr>\n";

}

String[] elementos = dir.list();

// Listado de directorios

for ( int i = 0; elementos != null && i < elementos.length; i++ ) {

    File f = new File( dir, elementos[i] );

    if ( f.isDirectory() ){

        html += "<tr class=\"item\">\n";
        html += "<td><img src=\"/data:folder\"></td>\n";
        html += "<td><a class=\"directorio\" href=\"\" + elementos[i] + "\>\" +
            elementos[i] + "</a></td>\n";
        html += "<td></td>\n";
        html += "<td></td>\n";
        html += "<td></td>\n";
        html += "<td>Carpeta de archivos</td>\n";
        html += "<td></td>\n";
        html += "<td align=\"left\"><span class=\"fecha\"> +
            new Date( f.lastModified() ) + "</span></td>\n";
        html += "</tr>\n";

    }

}

}

```

```

// Listado de archivos
for ( int i = 0; elementos != null && i < elementos.length; i++ ) {

    File f = new File( dir, elementos[i] );

    long t = f.length() / 1024;
    if ( t < 1 )
        tamano = "1 KB";
    else
        tamano = t + " KB";

    if ( f.isFile() ){

        html += "<tr class=\"item\">\n";
        html += "<td></td>\n";
        html += "<td align=\"left\"><a class=\"archivo\" href=\"\" +
        elementos[i] + \"\>\" + elementos[i] + </a></td>\n";
        html += "<td></td>\n";
        html += "<td align=\"right\">\" + tamano + "</td>\n";
        html += "<td></td>\n";
        html += "<td>Archivo " +
            f.getName().substring(f.getName().indexOf(".")+1).toUpperCase() +
            "</td>\n";
        html += "<td></td>\n";
        html += "<td align=\"left\"><span class=\"fecha\">\" +
            ( new Date(f.lastModified()) ) + "</span></td>\n";
        html += "</tr>\n";

    }

}

html += "</table>\n";
html += "<br><hr><br><i>Servidor Web coriendo en el puerto " +
    config.getPuertoTCP() + "</i>";
html += "</body></html>";

writer.write( html );
response.escribirMensaje( writer.toString() );

} // fin del metodo mostrarDirectorio

////////////////////////////////////

```

```

private void serverLog(){

    FileWriter file_Log;
    File f = new File ( "logs" + File.separator + "access.log" );

    try{

        if ( f.exists() ){
            file_Log = new FileWriter( f, true );
        }
        else{
            file_Log = new FileWriter( "logs/access.log", true );
        }

        PrintWriter output = new PrintWriter( file_Log );

        output.println( resultado );
        output.flush();
        output.close();

    }
    catch(Exception e){
        e.printStackTrace();
    }

} // fin del metodo serverLog

////////////////////////////////////

} // fin de la clase Conexion

```

```

/*
*****
*
* Request.java
*
* Esta clase extrae la informacion que contienen los mensajes
* de peticion de los clientes hacia el servidor web
*
*****
*/

import java.io.*;
import java.net.*;
import java.util.*;

public class Request {

    private Socket userSocket;
    private BufferedReader input;

    private static int BUFFER_SIZE = 1000;

    private String metodo = "";
    private String protocolo = "";
    private String requestURL = "";
    private String queryString = "";
    private String remoteAddress = "";
    private String remoteHost = "";
    private String contenido = "";
    private String requestHeader = "";

    private int statusCode = 0;

    private Hashtable headers;
    private byte[] postData;

    ////////////////////////////////////////////////////////////////////

    public Request( Socket s ) throws IOException{

        try{

            userSocket = s;

            boolean metodoValido = false;
            String linea;
            ArrayList lineas = new ArrayList();

            remoteAddress = userSocket.getInetAddress().getHostAddress();
            remoteHost = userSocket.getInetAddress().getHostName();

            input = new BufferedReader(
                new InputStreamReader( userSocket.getInputStream() ) );

            ////////////////////////////////////////////////////////////////////

```

```

// Lectura de la linea de identificacion del mensaje
////////////////////////////////////

requestHeader = input.readLine();
contenido = requestHeader + "\n";

// Determinar si la línea de identificación esta completa

if ( requestHeader.length() == 0 ) return;

StringTokenizer lineaItems = new StringTokenizer(requestHeader, " ");

if ( lineaItems.countTokens() != 3 ) {

    statusCode = ConstantesHTTP.HTTP_BAD_REQUEST;
    return;

}

// Determinar si el metodo es valido

metodo = lineaItems.nextToken();

for( int i = 0; i < ConstantesHTTP.metodos.length; i++ ) {

    if( metodo.equalsIgnoreCase(ConstantesHTTP.metodos[i]) ) {
        metodoValido = true;
        break;
    }

}

if( !metodoValido ) {
    statusCode = ConstantesHTTP.HTTP_UNSUPPORTED_TYPE;
    return;
}

// Determinar el identificador de recurso uniforme URI

String uri = lineaItems.nextToken();

int indexSeparador = uri.indexOf("?");

if ( indexSeparador == -1 ) {
    requestURL = uri;
    queryString = null;
}
else {
    requestURL = uri.substring( 0, indexSeparador );
    queryString = uri.substring( indexSeparador + 1 );
}

// Determinar el protocolo

protocolo = lineaItems.nextToken();
////////////////////////////////////
// Lectura de los headers del mensaje

```

```

////////////////////////////////////
headers = new Hashtable();

linea = input.readLine();

while( !linea.equals("") ) {

    contenido += linea + "\n";
    int colonPosition = linea.indexOf(":");

    String nombre = linea.substring( 0, colonPosition );
    String valor = linea.substring( colonPosition + 1 ).trim();

    headers.put( nombre.toUpperCase() ,valor);
    linea = input.readLine();

}

contenido += "\n";

////////////////////////////////////
// Lectura del cuerpo del mensaje (Datos POST)
////////////////////////////////////

if ( metodo.equalsIgnoreCase("POST") ) {

    String tipo = (String)headers.get("CONTENT-TYPE");

    if( tipo.equals("application/x-www-form-urlencoded") ) {
        int leidos = 0;
        int size = Integer.parseInt((String)headers.get("CONTENT-LENGTH"));
        postData = new byte[size];

        while( leidos < size ) {
            int result = input.read();
            postData[leidos] = (byte)result;
            contenido += (char)result;
            leidos++;
        }
    }

}

statusCode = ConstantesHTTP.HTTP_OK;

}
catch( SocketTimeoutException ste ){
    statusCode = ConstantesHTTP.HTTP_CLIENT_TIMEOUT;
}

} // fin del metodo constructor

////////////////////////////////////
public String getRequestURL() {
    return requestURL;
}

```

```
////////////////////////////////////  
public String getRemoteAddr() {  
    return remoteAddress;  
}  
////////////////////////////////////  
public int getStatuscode() {  
    return statusCode;  
}  
////////////////////////////////////  
public String getRequestHeader() {  
    return requestHeader;  
}  
////////////////////////////////////  
public String getHeader( String nombre ) {  
    return headers.get( nombre.toUpperCase() ).toString();  
}  
////////////////////////////////////  
public void close() throws IOException {  
    input.close();  
}  
////////////////////////////////////  
} // fin de la clase Request
```



```

/*
*****
*
* Response.java
*
* Esta clase maneja el formato de los mensajes de respuesta
* que el servidor web envia a los clientes
*
*****
*/

import java.io.*;
import java.net.*;
import java.util.*;

public class Response {

    private Socket userSocket;
    private final String HTTP_PROTOCOL = "HTTP/1.1";
    private PrintStream output;
    private String header;
    private String response;

    ////////////////////////////////////////////////////////////////////

    public Response( Socket s ) throws IOException {

        userSocket = s;
        header = "";
        response = "";
        output = new PrintStream( userSocket.getOutputStream() );

    } // fin del metodo constructor

    ////////////////////////////////////////////////////////////////////

    public void construirHeader(int c,String t,long s,String l) throws IOException {

        header += HTTP_PROTOCOL + " " + c;
        header += ConstantesHTTP.HTTP_EOL;

        header += "Date: " + new Date();
        header += ConstantesHTTP.HTTP_EOL;

        header += "Server: webserver/1.0";
        header += ConstantesHTTP.HTTP_EOL;

        header += "Connection: Keep-Alive";
        header += ConstantesHTTP.HTTP_EOL;

        if ( l.length() > 0 ){

            header += "Location: " + l;
            header += ConstantesHTTP.HTTP_EOL;

        }

    }

```

```

if ( t.length() > 0 ){
    header += "Content-Type: " + t;
    header += ConstantesHTTP.HTTP_EOL;
}
if ( s > 0 ){
    header += "Content-Length: " + s;
    header += ConstantesHTTP.HTTP_EOL;

    header += "Last-Modified: " + new Date();
    header += ConstantesHTTP.HTTP_EOL;
}

header += ConstantesHTTP.HTTP_EOL;

output.print( header );
response = "" + c;
} // fin del metodo construirHeader

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
public void escribirMensaje( String data ) throws IOException {
    output.print( data );
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
public void write( byte[] buf, int off, int len ) {
    output.write( buf, off, len );
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
public void close() {
    output.flush();
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
public String getResponse() {
    return response;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
} // fin de la clase Response

```

```

/*
*****
*
* ConstantesHTTP.java
*
* Esta clase contiene variables estaticas necesarias para
* la comunicacion http
*
*****
*/

```

```

public class ConstantesHTTP {
    public static String[] metodos = {"GET", "POST", "HEAD", "PUT"};
    public static final String HTTP_EOL = "\r\n";
    // 1XX: informacion
    public static final int HTTP_CONTINUE = 100;
    public static final int HTTP_SWITCHING_PROTOCOLS = 101;
    // 2XX: success
    public static final int HTTP_OK = 200;
    public static final int HTTP_CREATED = 201;
    public static final int HTTP_ACCEPTED = 202;
    public static final int HTTP_NOT_AUTHORITATIVE = 203;
    public static final int HTTP_NO_CONTENT = 204;
    public static final int HTTP_RESET = 205;
    public static final int HTTP_PARTIAL = 206;
    // 3XX: redirecion
    public static final int HTTP_MULT_CHOICE = 300;
    public static final int HTTP_MOVED_PERM = 301;
    public static final int HTTP_MOVED_TEMP = 302;
    public static final int HTTP_SEE_OTHER = 303;
    public static final int HTTP_NOT_MODIFIED = 304;
    public static final int HTTP_USE_PROXY = 305;
    // 4XX: error cliente
    public static final int HTTP_BAD_REQUEST = 400;
    public static final int HTTP_UNAUTHORIZED = 401;
    public static final int HTTP_PAYMENT_REQUIRED = 402;
    public static final int HTTP_FORBIDDEN = 403;
    public static final int HTTP_NOT_FOUND = 404;
    public static final int HTTP_BAD_METHOD = 405;
    public static final int HTTP_NOT_ACCEPTABLE = 406;
    public static final int HTTP_PROXY_AUTH = 407;
    public static final int HTTP_CLIENT_TIMEOUT = 408;
    public static final int HTTP_CONFLICT = 409;
    public static final int HTTP_GONE = 410;
    public static final int HTTP_LENGTH_REQUIRED = 411;
    public static final int HTTP_PRECON_FAILED = 412;
    public static final int HTTP_ENTITY_TOO_LARGE = 413;
    public static final int HTTP_REQ_TOO_LONG = 414;
    public static final int HTTP_UNSUPPORTED_TYPE = 415;
    // 5XX: error servidor
    public static final int HTTP_SERVER_ERROR = 500;
    public static final int HTTP_INTERNAL_ERROR = 501;
    public static final int HTTP_BAD_GATEWAY = 502;
    public static final int HTTP_UNAVAILABLE = 503;
    public static final int HTTP_GATEWAY_TIMEOUT = 504;
    public static final int HTTP_VERSION = 505;
} // fin de la clase ConstantesHTTP

```

```

/*
*****
*
* ServerWindow.java
*
* Esta clase presenta una version grafica del servidor en la
* que se puede iniciar o detener el servidor, cambiar la
* configuracion y ver las peticiones de los clientes
*
*****
*/

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;
import java.util.*;
import java.text.*;

public class ServerWindow extends JFrame {

    private static Nucleo server;

    private Container c;
    private Font fuente = new Font( "Verdana", Font.PLAIN, 11 );

    private static JButton btnStart, btnStop, btnConfig, btnInfo;
    private static JLabel lblStatus, lblPeticiones, lblBytes, lblConexiones;
    private static JTextArea textArea;

    private static String estado = "apagado";
    private static long contPeticiones = 0;
    private static long contBytes = 0;
    private static int contConexiones = 0;

    private int width = 800;
    private int height = 560;

    ///////////////////////////////////////////////////////////////////

    public ServerWindow() {

        super( "Servidor Web" );

        c = this.getContentPane();
        c.setLayout( null );

        UIManager.LookAndFeelInfo look[] = UIManager.getInstalledLookAndFeels();
        try{
            UIManager.setLookAndFeel(look[2].getClassName());
            SwingUtilities.updateComponentTreeUI(this);
        }
        catch( Exception e ){
            e.printStackTrace();
        }
    }
}

```

////////////////////////////////////

```
JPanel panelNorte = new JPanel();
```

```
panelNorte.setLayout( null );  
panelNorte.setBackground( new Color(128,128,128) );  
panelNorte.setBorder( new LineBorder( new Color(0,0,0), 4) );  
panelNorte.setBounds( 0, 0, width - 6 , 40);
```

```
btnStart = new JButton( new ImageIcon("data/imageStart.jpg") );  
btnStart.setRolloverIcon( new ImageIcon("data/imageStartOVER.jpg") );  
btnStart.setPressedIcon( new ImageIcon("data/imageStartDOWN.jpg") );  
btnStart.setCursor( new Cursor( Cursor.HAND_CURSOR ) );  
btnStart.setFocusPainted( false );  
btnStart.setBounds( 2, 2, 70, 36 );  
btnStart.setBorder( new LineBorder( new Color(0,0,0), 2 ) );  
btnStart.addActionListener(  
    new ActionListener(){  
        public void actionPerformed(ActionEvent e){  
            prenderServidor();  
        }  
    }  
);
```

```
btnStop = new JButton( new ImageIcon("data/imageStop.jpg") );  
btnStop.setRolloverIcon( new ImageIcon("data/imageStopOVER.jpg") );  
btnStop.setPressedIcon( new ImageIcon("data/imageStopDOWN.jpg") );  
btnStop.setCursor( new Cursor( Cursor.HAND_CURSOR ) );  
btnStop.setFocusPainted( false );  
btnStop.setBounds( 72, 2, 70, 36 );  
btnStop.setBorder( new LineBorder( new Color(0,0,0), 2 ) );  
btnStop.setEnabled( false );  
btnStop.addActionListener(  
    new ActionListener(){  
        public void actionPerformed(ActionEvent e){  
            apagarServidor();  
        }  
    }  
);
```

```
btnConfig = new JButton( new ImageIcon("data/imageConfig.jpg") );  
btnConfig.setRolloverIcon( new ImageIcon("data/imageConfigOVER.jpg") );  
btnConfig.setPressedIcon( new ImageIcon("data/imageConfigDOWN.jpg") );  
btnConfig.setCursor( new Cursor( Cursor.HAND_CURSOR ) );  
btnConfig.setFocusPainted( false );  
btnConfig.setBorder( new LineBorder( new Color(0,0,0), 2 ) );  
btnConfig.setBounds( 142, 2, 150, 36 );  
btnConfig.addActionListener(  
    new ActionListener(){  
        public void actionPerformed(ActionEvent e){  
            ConfigWindow cfgWin = new ConfigWindow();  
        }  
    }  
);
```

```

btnInfo = new JButton( new ImageIcon("data/imageInfo.jpg") );
btnInfo.setRolloverIcon( new ImageIcon("data/imageInfoOVER.jpg") );
btnInfo.setPressedIcon( new ImageIcon("data/imageInfoDOWN.jpg") );
btnInfo.setCursor( new Cursor( Cursor.HAND_CURSOR ) );
btnInfo.setFocusPainted( false );
btnInfo.setToolTipText( "Informacion del Servidor" );
btnInfo.setBorder( new LineBorder( new Color(0,0,0), 2 ) );
btnInfo.setBounds( 292, 2, 150, 36 );
btnInfo.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent e){
            JOptionPane.showMessageDialog( null,
                "SERVIDOR WEB version 1.0\n\n" +
                "Proyecto Final de Ingenieria de Sistemas\n\n" +
                "Realizado por:\n" +
                "German Bastidas\n" +
                "Carlos Yoncon\n\n" +
                "Universidad San Francisco de Quito\n" +
                "Enero del 2004\n\n",
                "Acerca del Servidor Web",
                JOptionPane.INFORMATION_MESSAGE);
        }
    }
);

```

```

lblPeticiones = new JLabel( "Peticiones: 0" );
lblPeticiones.setFont( new Font( "Verdana", Font.PLAIN, 11) );
lblPeticiones.setForeground( new Color(255,255,255) );
lblPeticiones.setBounds( 448, 6, 150, 12);

```

```

lblBytes = new JLabel( "Datos enviados: 0" );
lblBytes.setFont( new Font( "Verdana", Font.PLAIN, 11) );
lblBytes.setForeground( new Color(255,255,255) );
lblBytes.setBounds( 448, 20, 300, 12);

```

```

lblConexiones = new JLabel( "Conexiones activas: 0" );
lblConexiones.setFont( new Font( "Verdana", Font.PLAIN, 11) );
lblConexiones.setForeground( new Color(255,255,255) );
lblConexiones.setBounds( 620, 6, 180, 12);

```

```

panelNorte.add(btnStart);
panelNorte.add(btnStop);
panelNorte.add(btnConfig);
panelNorte.add(btnInfo);
panelNorte.add(lblPeticiones);
panelNorte.add(lblBytes);
panelNorte.add(lblConexiones);

```

```

////////////////////////////////////

```

```

JPanel panelCentro = new JPanel();

panelCentro.setLayout( null );
panelCentro.setBackground( new Color(0,0,255) );
panelCentro.setBounds( 0, 40, width, height - 96 );

JLabel lblTitulo = new JLabel();
lblTitulo.setFont( new Font( "Verdana", Font.PLAIN, 11) );
lblTitulo.setForeground( new Color(255,255,255) );
lblTitulo.setBounds( 0, 0, width, 20);
lblTitulo.setText( "      Fecha      Hora" +
"      IP" +
"      \"Request\" (Status Code) [Tiempo de atencion]");

textArea = new JTextArea();
textArea.setEditable( false );
textArea.setFont( new Font( "Courier", Font.PLAIN, 11) );

JScrollPane scroll = new JScrollPane( textArea );
scroll.setBounds( 0, 20, width - 6, height - 116 );

panelCentro.add(lblTitulo);
panelCentro.add(scroll);

////////////////////////////////////

JPanel panelSur = new JPanel();
panelSur.setLayout( null );
panelSur.setBounds( 0, height - 55, width - 6, 23);
panelSur.setBorder( new LineBorder( new Color(0,0,0), 1) );

lblStatus = new JLabel("Servidor detenido. (Presione el boton \"Start\" +
" para iniciar el servidor)");
lblStatus.setFont( new Font( "Verdana", Font.PLAIN, 11) );
lblStatus.setForeground( new Color(0,0,0) );
lblStatus.setBounds( 5, 1, width - 6, 20);

panelSur.add(lblStatus);

////////////////////////////////////

c.add(panelNorte);
c.add(panelCentro);
c.add(panelSur);

this.addWindowListener(new WindowAdapter(){
    public void windowClosing(WindowEvent e){
        System.exit(0);
    }
});
this.setSize( width , height );
this.setIconImage( (new ImageIcon("data/imageLogo.jpg")).getImage() );
this.setResizable( false );
this.show();

} // fin del metodo constructor

```

```
////////////////////////////////////////////////////////////////
```

```
private static void prenderServidor(){  
    server = new Nucleo( new Configuracion() );  
    btnStart.setEnabled( false );  
    btnStop.setEnabled( true );  
    estado = "encendido";  
}  
// fin del metodo prenderServidor
```

```
////////////////////////////////////////////////////////////////
```

```
private static void apagarServidor(){  
    server.stopServer();  
    server = null;  
    btnStart.setEnabled( true );  
    btnStop.setEnabled( false );  
    estado = "apagado";  
}  
// fin del metodo apagarServidor
```

```
////////////////////////////////////////////////////////////////
```

```
public static void display( String text ){  
    textArea.append( " " + text + " \n" );  
    textArea.setCaretPosition( textArea.getText().length() );  
}  
// fin del metodo display
```

```
////////////////////////////////////////////////////////////////
```

```
public static void setStatus( String text ){  
    lblStatus.setText( text );  
}  
// fin del metodo setStatus
```

```
////////////////////////////////////////////////////////////////
```

```
public static void sumarPeticion(){  
    contPeticiones++;  
    lblPeticiones.setText( "Peticiones: " +  
        NumberFormat.getInstance().format(contPeticiones) );  
}  
// fin del metodo sumarPeticion
```

```
////////////////////////////////////////////////////////////////
```



```

public static void sumarBytes( long kb ){

    contBytes = contBytes + kb;
    lblBytes.setText( "Datos enviados: " +
        NumberFormat.getInstance().format(contBytes/1024) + " KB");

} // fin del metodo sumarBytes

////////////////////////////////////

public static void setNumConexiones( int num ){

    lblConexiones.setText( "Conexiones activas: " +
        NumberFormat.getInstance().format(num) );

} // fin del metodo setNumConexiones

////////////////////////////////////

public static void readConfig(){

    if ( estado.equals("encendido") ) {

        apagarServidor();

        prenderServidor();

    }

} // fin del metodo readConfig

////////////////////////////////////

} // fin de la clase ServerWindow

```

```

/*
*****
*
* ConfigWindow.java
*
* Esta clase permite configurar las variables utilizadas por el
* servidor web sin necesidad de reiniciarlo
*
*****
*/

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.util.*;

public class ConfigWindow extends JDialog {

    private Configuracion config;

    private Container c;
    private Font fuente = new Font( "Verdana", Font.PLAIN, 11);

    private JPanel panels[];
    private JButton btnFile, btnAnadir, btnQuitar;
    private JButton btnUp, btnDown, btnAceptar, btnCancelar;
    private JTextField txtPuerto, txtConexiones, txtRaiz, txtTimeout;
    private JRadioButton rbtYes, rbtNo;
    private ButtonGroup radioGroup;
    private JList listaIndex;

    private int ancho = 450;
    private int alto = 470;

    ///////////////////////////////////////////////////////////////////

    public ConfigWindow( ) {

        config = new Configuracion();

        c = this.getContentPane();
        c.setLayout( new BorderLayout() );

        ///////////////////////////////////////////////////////////////////

        JPanel panelCentro = new JPanel();

        panelCentro.setLayout( null );
        panelCentro.setBorder( new LineBorder( new Color(0,0,0), 2) );

        panels = new JPanel[6];
        FlowLayout flowLay = new FlowLayout();
        flowLay.setAlignment( FlowLayout.LEFT );

```

```

panels[0] = new JPanel();
panels[0].setLayout( flowLay );
panels[0].setBackground( new Color(94,117,237) );
panels[0].setBorder( new LineBorder( new Color(0,0,0), 2) );
panels[0].setBounds( 0, 0, ancho - 6, 33 );

JLabel lblPuerto = new JLabel( "Puerto TCP:" );
lblPuerto.setFont( fuente );
lblPuerto.setForeground( new Color(255,255,255) );
panels[0].add( lblPuerto );

txtPuerto = new JTextField( 5 );
txtPuerto.setText( "" + config.getPuertoTCP() );
panels[0].add( txtPuerto );

panelCentro.add( panels[0] );

panels[1] = new JPanel();
panels[1].setLayout( flowLay );
panels[1].setBackground( new Color(94,117,237) );
panels[1].setBorder( new LineBorder( new Color(0,0,0), 2) );
panels[1].setBounds( 0, 33, ancho - 6, 33 );

JLabel lblConexiones = new JLabel( "Numero maximo de conexiones:" );
lblConexiones.setFont( fuente );
lblConexiones.setForeground( new Color(255,255,255) );
panels[1].add( lblConexiones );

txtConexiones = new JTextField( 4 );
txtConexiones.setScrollOffset( 2 );
txtConexiones.setText( "" + config.getNumeroConexiones() );
panels[1].add( txtConexiones );

panelCentro.add( panels[1] );

panels[2] = new JPanel();
panels[2].setLayout( flowLay );
panels[2].setBackground( new Color(94,117,237) );
panels[2].setBorder( new LineBorder( new Color(0,0,0), 2) );
panels[2].setBounds( 0, 66, ancho - 6, 85);

JLabel lblRaiz = new JLabel( "Directorio Raiz:" );
lblRaiz.setFont( fuente );
lblRaiz.setForeground( new Color(255,255,255) );
panels[2].add( lblRaiz );

txtRaiz = new JTextField( 38 );
txtRaiz.setFont( fuente );
txtRaiz.setText( "" + config.getDirectorioRaiz() );
panels[2].add( txtRaiz );

btnFile = new JButton( "Seleccionar" );
btnFile.setFont(fuente);

```

```

btnFile.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent e){
            nuevoDirectorio();
        }
    }
);
panels[2].add( btnFile );

panelCentro.add( panels[2] );

panels[3] = new JPanel();
panels[3].setLayout( null );
panels[3].setBackground( new Color(94,117,237) );
panels[3].setBorder( new LineBorder( new Color(0,0,0), 2) );
panels[3].setBounds( 0, 151, ancho - 6, 180 );
panels[3].setAlignmentY( JPanel.TOP_ALIGNMENT );

JLabel lblIndex = new JLabel( "Paginas inicio:" );
lblIndex.setFont( fuente );
lblIndex.setForeground( new Color(255,255,255) );
lblIndex.setBounds( 7, 2, 100, 20);
panels[3].add( lblIndex );

listaIndex = new JList( config.getPaginasInicio() );
listaIndex.setVisibleRowCount( 4 );
listaIndex.addListSelectionListener(
    new ListSelectionListener(){
        public void valueChanged(ListSelectionEvent e){
            itemSelected( listaIndex.getSelectedIndex() );
        }
    }
);

JScrollPane scList = new JScrollPane(listaIndex);
scList.setBounds( 100, 8, 160, 160 );
panels[3].add( scList );

btnAnadir = new JButton("Anadir");
btnAnadir.setFont( fuente );
btnAnadir.setBounds( 7, 28, 82, 25);
btnAnadir.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent e){
            anadirIndex();
        }
    }
);
panels[3].add( btnAnadir );

btnQuitar = new JButton( "Quitar" );
btnQuitar.setEnabled( false );
btnQuitar.setFont( fuente );
btnQuitar.setBounds( 7, 56, 82, 25 );

```

```

btnQuitar.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent e){
            removerIndex();
        }
    }
);
panels[3].add( btnQuitar );

btnUp = new JButton( "Subir" );
btnUp.setEnabled( false );
btnUp.setFont( fuente );
btnUp.setBounds( 270, 60, 70, 25 );
btnUp.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent e){
            moverIndex( listaIndex.getSelectedIndex(), -1 );
        }
    }
);
panels[3].add( btnUp );

btnDown = new JButton( "Bajar" );
btnDown.setEnabled( false );
btnDown.setFont( fuente );
btnDown.setBounds( 270, 90, 70, 25 );
btnDown.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent e){
            moverIndex( listaIndex.getSelectedIndex(), 1 );
        }
    }
);
panels[3].add( btnDown );

panelCentro.add( panels[3] );

panels[4] = new JPanel();
panels[4].setLayout( flowLay );
panels[4].setBackground( new Color(94,117,237) );
panels[4].setBorder( new LineBorder( new Color(0,0,0), 2) );
panels[4].setBounds( 0, 331, ancho - 6, 33 );

JLabel lblListar = new JLabel( "Listar folders:" );
lblListar.setFont( fuente );
lblListar.setForeground( new Color(255,255,255) );
panels[4].add( lblListar );

rbtYes = new JRadioButton( "Si" );
rbtYes.setForeground( new Color(255,255,255) );
rbtYes.setBackground( new Color(94,117,237) );
panels[4].add( rbtYes );

```

```

rbtNo = new JRadioButton( "No" );
rbtNo.setForeground( new Color(255,255,255) );
rbtNo.setBackground( new Color(94,117,237) );
panels[4].add( rbtNo );

if ( config.listarFolders() )
    rbtYes.doClick();
else
    rbtNo.doClick();

radioGroup = new ButtonGroup();
radioGroup.add( rbtYes );
radioGroup.add( rbtNo );

panelCentro.add( panels[4] );

panels[5] = new JPanel();
panels[5].setLayout( flowLay );
panels[5].setBackground( new Color(94,117,237) );
panels[5].setBorder( new LineBorder( new Color(0,0,0), 2) );
panels[5].setBounds( 0, 364, ancho - 6, 33 );

JLabel lblTimeout = new JLabel( "Timeout:" );
lblTimeout.setFont( fuente );
lblTimeout.setForeground( new Color(255,255,255) );
panels[5].add( lblTimeout );

txtTimeout = new JTextField( 6 );
txtTimeout.setText( "" + config.getTimeout() );
panels[5].add( txtTimeout );

panelCentro.add( panels[5] );

////////////////////////////////////

JPanel panelSur = new JPanel();

panelSur.setLayout( new FlowLayout() );
panelSur.setBorder( new LineBorder( new Color(0,0,0), 4) );
panelSur.setBackground( new Color(128,128,128) );

btnAceptar = new JButton( "Aceptar" );
btnAceptar.setFont( fuente );
btnAceptar.addActionListener(
    new ActionListener(){
        public void actionPerformed(ActionEvent e){
            aceptar();
        }
    }
);

btnCancelar = new JButton( "Cancelar" );
btnCancelar.setFont(fuente);

```

```

    btnCancelar.addActionListener(
        new ActionListener(){
            public void actionPerformed(ActionEvent e){
                cancelar();
            }
        }
    );

panelSur.add(btnAceptar);
panelSur.add(btnCancelar);

////////////////////////////////////

c.add(panelCentro, BorderLayout.CENTER);
c.add( panelSur, BorderLayout.SOUTH );

this.setModal( true );
this.setTitle( "Configuracion del Servidor Web" );
this.setSize( ancho, alto );
this.setResizable( false );
this.show();

} // fin del metodo constructor

////////////////////////////////////

private void aceptar(){

    String directorioRaiz = txtRaiz.getText();

    Vector items = new Vector( listaIndex.getModel().getSize() );

    for( int i = 0; i < listaIndex.getModel().getSize(); i++ ){

        items.add( listaIndex.getModel().getElementAt(i) );

    }

    String paginasInicio = items.toString();

    paginasInicio = paginasInicio.substring(1, paginasInicio.length() - 1 );
    paginasInicio = paginasInicio.replaceAll(" ", "");

    String listarFolders;

    if ( rbtYes.isSelected() )

        listarFolders = "true";

    else

        listarFolders = "false";

```

```

// Escritura de variables de configuracion en el archivo ServerConfig.cfg
File file_Cfg = new File ( "config" + File.separator + "ServerConfig.cfg" );

try{

    if ( !file_Cfg.exists() ) file_Cfg.createNewFile();

    PrintStream output = new PrintStream( new FileOutputStream(file_Cfg) );

    output.println("#####");
    output.println("# Archivo de configuracion del servidor");
    output.println("# Fecha de actualizacion: " + new Date() );
    output.println("#####");
    output.println("");
    output.println("PUERTO_TCP = " + txtPuerto.getText() );
    output.println("");
    output.println("MODO_INICIO = " + config.getModoInicio() );
    output.println("");
    output.println("NUMERO_CONEXIONES = " + txtConexiones.getText() );
    output.println("");
    output.println("DIRECTORIO_RAIZ = " + directorioRaiz );
    output.println("");
    output.println("PAGINAS_INICIO = " + paginasInicio );
    output.println("");
    output.println("LISTAR_FOLDERS = " + listarFolders );
    output.println("");
    output.println("TIMEOUT = " + txtTimeout.getText() );

    output.flush();

    output.close();

}
catch(Exception e){

    e.printStackTrace();

}

ServerWindow.readConfig();

this.dispose();
} // fin del metodo aceptar

////////////////////////////////////

private void cancelar(){

    this.dispose();

} // fin del metodo cancelar

////////////////////////////////////

```



```

private void anadirIndex(){

String newIndex = JOptionPane.showInputDialog( this,
    "Ingrese el nombre de la pagina de inicio", "Nueva pagina de inicio",
    JOptionPane.QUESTION_MESSAGE );

Vector items = new Vector();
for( int i = 0; i < listaIndex.getModel().getSize(); i++ ){
    items.add( listaIndex.getModel().getElementAt(i) );
}

items.add( newIndex );

listaIndex.removeAll();
listaIndex.setListData( items );

btnQuitar.setEnabled( false );
btnUp.setEnabled( false );
btnDown.setEnabled( false );

} // fin del metodo anadirIndex

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

private void removerIndex(){

Vector items = new Vector();
for( int i = 0; i < listaIndex.getModel().getSize(); i++ ){
    items.add( listaIndex.getModel().getElementAt(i) );
}

items.removeElementAt( listaIndex.getSelectedIndex() );

listaIndex.removeAll();
listaIndex.setListData( items );

btnQuitar.setEnabled( false );
btnUp.setEnabled( false );
btnDown.setEnabled( false );

} // fin del metodo removerIndex

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

private void itemSelected( int index ){

btnQuitar.setEnabled( true );
btnUp.setEnabled( true );
btnDown.setEnabled( true );

if ( index == 0 ) btnUp.setEnabled(false);

if ( index == listaIndex.getModel().getSize()-1 ) btnDown.setEnabled(false);

} // fin del metodo itemSelected

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

```

```

private void moverIndex( int index, int direccion ){

    Vector items = new Vector();

    for( int i = 0; i < listaIndex.getModel().getSize(); i++ ){
        items.add( listaIndex.getModel().getElementAt(i) );
    }

    Object temp = items.remove( index );
    items.insertElementAt( temp, index + direccion );

    listaIndex.removeAll();
    listaIndex.setListData( items );
    listaIndex.setSelectedIndex( index + direccion );

} // fin del metodo moverIndex

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

private void nuevoDirectorio(){

    JFileChooser fileChooser = new JFileChooser();

    fileChooser.setDialogTitle( "Seleccione el directorio raiz" );
    fileChooser.setSelectionMode( JFileChooser.DIRECTORIES_ONLY );
    fileChooser.setCurrentDirectory( new File( txtRaiz.getText() ) );

    int result = fileChooser.showOpenDialog( null );

    if (result == JFileChooser.CANCEL_OPTION)
        return;

    File fileName = fileChooser.getSelectedFile();

    String newPath = fileName.getAbsolutePath();

    newPath = newPath.replace('\\', '/');

    if ( fileName.isDirectory() ) txtRaiz.setText( newPath );

} // fin del metodo nuevoDirectorio

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

} // fin de la clase ConfigWindow

```

# GLOSARIO

**ASCII.** American Standard Code for Information Interchange. Es un estándar que asigna un valor numérico a cada carácter.

**Agente de usuario.** Cliente que inicia una petición tipo request. Puede ser un navegador web, un spider o cualquier otra herramienta final de usuario.

**C.** Lenguaje de programación de alto nivel desarrollado en la década de los 70 por Dennis Ritchie en los laboratorios de Bell. Su implementación está basada en los lenguajes B y BCLP más ciertas características adicionales como la definición de tipos de variables.

**C++.** Lenguaje de programación de alto nivel desarrollado en la década de los 80 por Bjarne Stroustrup en los laboratorios de Bell. C++ añade la funcionalidad de la programación orientada a objetos a su predecesor C.

**CGI.** Common Gateway Interface.

**Cliente.** Programa que establece una conexión con un programa servidor con el propósito de mandar peticiones para solicitar recursos del servidor.

**DNS.** Domain Name Service. Servicio de traducción de nombres de dominio. Mapea un nombre de dominio a una dirección IP.

**Dirección IP.** Es un conjunto de 32 dígitos binarios expresados en cuatro grupos decimales separados por un punto. Describe la dirección lógica de una computadora en una red. Por ejemplo 192.188.53.6

**Encriptar.** Proceso en el que se transforma la información para que no pueda ser entendida por ninguna persona.

**FTP.** Protocolo de transferencia de archivos.

**Gateway.** Servidor o equipo de salida.

**HTTP.** Protocolo de transferencia de hipertexto.

**Hipertexto.** Texto que recibe un diseño o formato mediante etiquetas.

**Hipervínculos.** Texto o imágenes que permiten conectar un documento web con otro recurso en la red como páginas web, imágenes, programas, etc.

**HTML.** Hypertext Markup Language. Lenguaje de etiquetas utilizado para dar formato a los elementos de una página web.

**PDA.** Agenda electrónica personal.

**Protocolo.** Conjunto de reglas y especificaciones que permiten la comunicación entre dos unidades.

**Proxy.** Software que sirve de intermediario entre un cliente y un servidor. En el caso de que el recurso solicitado por un cliente este almacenado temporalmente en el servidor proxy, éste lo envía al cliente, caso contrario el servidor proxy pasa a convertirse en el cliente y solicita el recurso al servidor.

**Puerto.** Ver Socket.

**Recurso.** Cualquier cosa que tenga identidad. Por ejemplo documentos electrónicos, imágenes, videos, servicios, etc.

**RFC.** Request For Comments. Son documentos que especifican estándares utilizados en el Internet.

**Servidor.** Programa de aplicación que acepta conexiones con el objetivo de escuchar peticiones enviadas por los programas clientes. El servidor atiende estas peticiones a través de mensajes de respuesta.

**SMTP.** Simple Mail Transport Protocol. Protocolo utilizado para el transporte de correo electrónico.

**Socket.** Objeto de software que conecta una aplicación con un protocolo de red. Mediante el uso de estos objetos se puede establecer una comunicación entre dos aplicaciones remotas.

**Spider.** Programa que atraviesa la web en busca de algún recurso. Se utilizan mucho para obtener resultado de una búsqueda en Internet.

**TCP/IP.** Transmisión Control Protocol / Internet Protocol. Suite de protocolos de comunicación usados para comunicar un computador con Internet.

**Testing.** Proceso en el cual se ejecuta un programa para encontrar errores.

**Thread.** Procesos conocidos como hilos de ejecución que forman parte de un programa y que pueden ejecutarse independientemente y de manera concurrente con otros procesos.

**URI.** Uniform Resource Identifier. Cadena de caracteres que permite identificar un recurso.

# BIBLIOGRAFIA

- STALLINGS William, Comunicaciones y Redes de Computadores, Prentice Hall, 2000
- SILBERSCHATZ Abraham, GALVIN Meter, Sistemas Operativos, Prentice Hall, 5ta edición, 1999.
- DEITEL H.M, P. J. Deitel, y T. R. Nieto, Internet & World Wide Web How to program, Prentice Hall, 2da edición, 2000.
- DEITEL H.M, P. J. Deitel, y T. R. Nieto, Java How to program, Prentice Hall, 3ra edición, 1999.
- DEITEL H.M, P. J. Deitel, y T. R. Nieto, C++ How to program, Prentice Hall, 3ra edición, 2001.

## Internet:

- <http://www.lib.berkeley.edu/TeachingLib/Guides/Internet/WhatIs.html>
- <http://www.monografias.com/trabajos11/infintern/infintern.shtml>
- <http://www.isoc.org/internet/history/brief.shtml>
- <http://www.compute.net/html/hisint.html>

## Documentos electrónicos:

- RFC 2616 (Especificación sobre el protocolo HTTP/1.1)
- RFC 2396 (Especificación sobre la sintaxis general de los identificadores uniformes de recursos)