U N I V E R S I D A D D E G U A Y A Q U I L

F a c u l t a d d e C i e n c i a s M a t e m á t i c a s y F í s i c a s

C a r r e r a d e I n g e n i e r í a e n S i s t e m a s C o m p u t a c i o n a l e s

"Implementación de Linux como Router"

TESIS DE GRADO

Previo a la Obtención del Título de:

INGENIERO EN SISTEMAS COMPUTACIONALES

Autores:

Boris Alex López Macías

Jorge Giovanny Sánchez De La Torre

José Daniel Sotomayor Navarro

GUAYAQUIL-ECUADOR

Año: 2008

A G R A D E C IM IE N T O

A Dios ante todo, ya que él con su infinito amor nos doto de sabiduría para poder tomar las mejores decisiones e ir por el sendero del bien.

A nuestros Padres, que con paciencia y dedicación, nos dieron su apoyo, consejos, y parte de sus vidas para que nosotros sigamos adelante y alcancemos nuestras metas.

A nuestros profesores, que nos transmitieron sus vastos conocimientos y experiencias adquiridas a lo largo de su carrera profesional.

A nuestros amigos y seres queridos que siempre cuando nos sentíamos vencidos, nos daban aliento para seguir adelante. Gracias por su apoyo incondicional.

D E D IC A T O R IA

Nuestra tesis la dedicamos con amor y cariño, a Dios que nos diste la oportunidad de vivir y de regalarnos unas familias maravillosas, a nuestros Padres quienes han estado en cada momento a nuestro lado, creyeron en nosotros dándonos preparación hasta lograr vernos convertidos en auténticos profesionales. A nuestros seres queridos que supieron confiar en nosotros animándonos a que siguiéramos nuestros sueños.

TRIBUNAL DE GRADUACIÓN

Presidente	1er Vocal
2do Vocal	Vocal Secundario

DECLARACIÓN EXPRESA

"La autoría de la tesis de grado corresponde exclusivamente al suscrito(s), perteneciendo a la Universidad de Guayaquil los derechos que generen la aplicación de la misma"

(Reglamento de Graduación de la Carrera de Ingeniería en Sistemas Computacionales, Art. 26)

Boris Alex López Macías

borisjudoka@hotmail.com

Jorge Giovanny Sánchez De La Torre

a_androide17@hotmail.com

José Daniel Sotomayor Navarro

jdsn13@hotmail.com

RESUMEN

Actualmente las Organizaciones han optimizado y mejorado la forma de administrar sus redes internas mediante la implementación de la tecnología.

Sin embargo, debido a la necesidad de mejorar la utilización de los recursos de red que establecen la comunicación con el mundo (Internet) y también el aprovechamiento de las herramientas Open Source, se realizó una "Implementación de una PC Linux como Router" el cual es administrado y controlado por medio de una aplicación Web con la finalidad de optimizar la comunicación de 2 o más redes y ofrecerle a la Organización una disminución en el presupuesto de adquisición y mantenimiento de dispositivos de ruteo.

Esta aplicación fue diseñada usando la arquitectura de Aplicaciones Cliente / Servidor, utilizando como lenguaje de programación la herramienta MyEclipse de la plataforma Java.

El acceso a la aplicación será evaluado mediante los roles o privilegios que tenga configurado el usuario del mismo, proporcionándole así un entorno de administración seguro.

INDICE GENERAL

AGRADECIMIENTO	I
DEDICATORIA	II
TRIBUNAL DE GRADUACIÓN	III
DECLARACIÓN EXPRESA	IV
RESUMEN	V
INDICE GENERAL	VI

TABLA DE CONTENIDO

CAPÍTULO 1 1

1 INTRODUCCIÓN 1

1.1. Antecedentes 1

1.2. Definición de Router 2

1.3. Opciones de Acceso a la Aplicación 4

1.4. Visión 5

1.5. Misión 5

1.6. Objetivos Generales 6

1.7. Objetivos Específicos 6

1.8. Alcances del Proyecto 8

1.8.1. Descripción del Módulo para la Administración del Router 8

1.8.2. Gestión y Control 10

1.9. FODA de la Aplicación Web del PC Linux como Router 12

1.9.1. Fortalezas 12

1.9.2. Oportunidades 13

1.9.3. Debilidades 14

1.9.4. Amenazas 14

1.10. Metodología 15

1.11. Arquitectura 18

1.12. Tecnología 19

1.13. Recursos 20

1.13.1. Recurso de Hardware 21

1.13.2. Recurso de Software 24

1.13.3. Recurso Humano: 25

CAPÍTULO 2 26

2. ANALISIS 26

2.1. Requerimientos 26

2.2. Mecánica de Control 27

2.3. Diagrama de Clases a utilizar 30
2.3.1. Análisis del Diagrama de Clases 30

2.4. Casos de Uso 33

2.5. Diagrama de Flujo de Datos 43
2.5.1. Ingreso al navegador 45
2.5.2. Digitar dirección del servidor de aplicación en el browser del navegador 45
2.5.3. Carga de página inicial 45
2.5.4. Solicitud de usuario y contraseña 46
2.5.5. Confirmación de rol 46
2.5.6. Carga de página principal del usuario administrador 47
2.5.7. Gestión de usuarios 47
2.5.8. Carga de página principal del usuario operador 50

CAPITULO 3 52

3. DISEÑO 52

3.1. Diseño de la Solución 52
3.1.1. Repositorio de información 53
3.1.2. Modelado del Sistema 56

3.2. Diseño de Clases 56
3.2.1. Creacion_user_admin 57
3.2.2. Logon_v2 57
3.2.3. Confirma_rol 59
3.2.4. Crear_usuarios 60
3.2.5. Cambio_clave_v2 61
3.2.6. Eliminar_usuarios 62
3.2.7. Crear_bitacora 62
3.2.8. Responder_solicitud 63
3.2.9. Verificar_archivos_bitacora_control 65
3.2.10. Eliminar_logoneados_v2 65
3.2.11. Encriptador 66
3.2.12. Verificar_archivos_basicos 67
3.2.13. Configura_basica 68
3.2.14. Configura_host_pass_ripd 68
3.2.15. Procesa_Arch_tabla 69
3.2.16. Mascara 70
3.2.17. Lista_iface 71
3.2.18. Lista_eth 72
3.2.19. Informacion_interface 73
3.2.20. Ruta_Tabla 73
3.2.21. Config_rip 74

3.3. Diseño de Pantallas 75
3.3.1. Pantalla Inicial 76
3.3.2. Pantalla de Inicio de Sesión 77

3.3.3. Pantalla Principal del Usuario Administrador 78
3.3.4. Opción Usuarios 79
3.3.5. Opción Bitácora de Control 79
3.3.6. Opción Configuración Básica 80
3.3.7. Opción Router Estático 81
3.3.8. Opción Router Dinámico – Rip 82
3.3.9. Opción Router Dinámico – Ospf 83
3.3.10. Pantalla Principal del Usuario Operador 84
3.3.11. Opción Usuarios 85
3.3.12. Opción Consultar Tablas de Ruteo 86

CAPITULO 4 88

4. DESARROLLO DEL SOFTWARE 88

4.1. Adaptación 88
4.1.1. Contenido de zebra.conf 91
4.1.2. Contenido de ripd.conf 92
4.1.3. Contenido de ospf.conf 93

4.2. Integración de Recursos 94

4.3. Instalación del Software 95

4.4. Modelo Funcional 97

CAPÍTULO 5 99

5. CONCLUSIONES Y RECOMENDACIONES 99

5.1. Conclusiones 99

5.2. Recomendaciones 100

GLOSARIO DE TERMINOS 102

BIBLIOGRAFÍA 104

INDICE DE FIGURAS

ILUSTRACIÓN 1-1 MODELO LINEAL SECUENCIAL 17

ILUSTRACIÓN 1-2 MODELO EN ESPIRAL 18

ILUSTRACIÓN 1-3 ARQUITECTURA 19

ILUSTRACIÓN 2-1 DIAGRAMA DE CLASES 32

ILUSTRACIÓN 2-2 ACTORES DE LOS CASOS DE USOS 33

ILUSTRACIÓN 2-3 CASO CARGA DE PAGINA INICIAL 34

ILUSTRACIÓN 2-4 CASO CREACIÓN DE USUARIOS 35

ILUSTRACIÓN 2-5 CASO ASIGNACIÓN DE PRIVILEGIOS 36

ILUSTRACIÓN 2-6 CASO INGRESO DE USUARIO 37

ILUSTRACIÓN 2-7 CASO CONFIGURACIÓN DE ROUTER 39

ILUSTRACIÓN 2-8 CASO MONITOREO DE ACTIVIDADES 40

ILUSTRACIÓN 2-9 CASO SOLICITUD DE REQUERIMIENTO 41

ILUSTRACIÓN 2-10 CASO RESPUESTA DE REQUERIMIENTO 42

ILUSTRACIÓN 2-11 DIAGRAMA DE FLUJO DE DATOS 44

ILUSTRACIÓN 3-1 PANTALLA INICIAL 76

ILUSTRACIÓN 3-2 PANTALLA DE INICIO DE SESIÓN 77

ILUSTRACIÓN 3-3 PANTALLA PRINCIPAL DEL ADMINISTRADOR 78

ILUSTRACIÓN 3-4 PANTALLA DE OPCIÓN USUARIOS 79

ILUSTRACIÓN 3-5 PANTALLA DE OPCIÓN BITÁCORA DE CONTROL 80

ILUSTRACIÓN 3-6 PANTALLA DE OPCIÓN CONFIGURACIÓN BÁSICA 81

ILUSTRACIÓN 3-7 PANTALLA DE OPCIÓN ROUTER ESTÁTICO 82

ILUSTRACIÓN 3-8 PANTALLA DE OPCIÓN ROUTER DINÁMICO - RIP 83

ILUSTRACIÓN 3-9 PANTALLA DE OPCIÓN ROUTER DINÁMICO - OSPF 84

ILUSTRACIÓN 3-9 PANTALLA PRINCIPAL DEL OPERADOR 85

ILUSTRACIÓN 3-11 PANTALLA DE OPCIÓN USUARIOS OPERADOR 86

ILUSTRACIÓN 3-11 PANTALLA DE OPCIÓN CONSULTAR TABLAS DE RUTEO 87

ILUSTRACIÓN 4-1 ESQUEMA DEMOSTRATIVO DEL FUNCIONAMIENTO 97

INDICE DE TABLAS

TABLA 1-1- DESCRIPCIÓN PC LINUX 22

TABLA 1-2- DESCRIPCIÓN PC HOSTS 23

TABLA 1-3-DESCRIPCIÓN DE OTROS DISPOSITIVOS 23

TABLA 1-4- RECURSOS DE SOFTWARE 24

TABLA 2-1– CASO CARGA DE PÁGINA INICIAL 35

TABLA 2-2– CASO CREACIÓN DE USUARIOS 36

TABLA 2-3– CASO ASIGNACIÓN DE PRIVILEGIOS 37

TABLA 2-4– CASO INGRESO DE USUARIO 38

TABLA 2-5– CASO CONFIGURACIÓN DEL ROUTER 39

TABLA 2-6– CASO MONITOREO DE ACTIVIDADES 40

TABLA 2-7 CASO SOLICITUD DE REQUERIMIENTO 41

TABLA 2-8– CASO RESPUESTA DE REQUERIMIENTO 42

TABLA 3-1– ESTRUCTURA DEL ARCHIVO USER 53

TABLA 3-2– ESTRUCTURA DEL ARCHIVO LOGONEADOS 54

TABLA 3-3– ESTRUCTURA DEL ARCHIVO HIST_LOGON 54

TABLA 3-4– ESTRUCTURA DEL ARCHIVO BITACORA 55

TABLA 3-5– ESTRUCTURA DEL ARCHIVO USER_MIRROR 56

TABLA 3-6– ESTRUCTURA DE CLASE CREACIÓN_USER_ADMIN 57

TABLA 3-7– ESTRUCTURA DE CLASE LOGON_V2 58

TABLA 3-8– ESTRUCTURA DE CLASE CONFIRMA_ROL 59

TABLA 3-9– ESTRUCTURA DE CLASE CREAR_USUARIOS 60

TABLA 3-10– ESTRUCTURA DE CLASE CAMBIO_CLAVE_V2 61

TABLA 3-11– ESTRUCTURA DE CLASE ELIMINAR_USUARIOS 62

TABLA 3-12– ESTRUCTURA DE CLASE CREAR_BITACORA 63

TABLA 3-13– ESTRUCTURA DE CLASE RESPONDER_SOLICITUD 64

TABLA 3-14– ESTRUCTURA DE CLASE VERIFICAR_ARCHIVOS_BITACORA_CONTROL 65

TABLA 3-15– ESTRUCTURA DE CLASE ELIMINAR_LOGONEADOS_V2 66

TABLA 3-16– ESTRUCTURA DE CLASE ENCRIPTADOR 67

TABLA 3-17– ESTRUCTURA DE CLASE VERIFICAR_ARCHIVOS_BASICOS 67

TABLA 3-18 – ESTRUCTURA DE CLASE CONFIGURA_BASICA 68
TABLA 3-19 – ESTRUCTURA DE CLASE CONFIGURA_HOST_PASS_RIPD 69
TABLA 3-20 – ESTRUCTURA DE CLASE PROCESA_ARCH_TABLA 70
TABLA 3-21 – ESTRUCTURA DE CLASE MASCARA 71
TABLA 3-22 – ESTRUCTURA DE CLASE MASCARA 71
TABLA 3-23 – ESTRUCTURA DE CLASE LISTA_ETH 72
TABLA 3-24 – ESTRUCTURA DE CLASE INFORMACIÓN_INTERFACE 73
TABLA 3-25 – ESTRUCTURA DE CLASE RUTA_TABLA 74
TABLA 3-26 – ESTRUCTURA DE CLASE CONFIG_RIP 75

C A P Í T U L O 1

1 IN T R O D U C C IÓ N

1.1. A n t e c e d e n t e s

En vista de que en la actualidad la comunicación se ha convertido en un factor fundamental para el desarrollo y crecimiento de las grandes Organizaciones y a esto se le agrega el hecho de que las redes de computadoras es un tema completamente globalizado, siempre se está buscando la forma de innovar en el mercado con nuevas herramientas que ofrezcan mayores facilidades y ventajas a la hora de comunicar una red con otra.

Por este motivo, la demanda de este tipo de herramientas se ha vuelto cada día mas grande por aquellas Empresas que ante la competencia buscan crecer, no solo en nombre sino también en calidad,

prestaciones y servicios buscando siempre dar mayor seguridad a sus clientes (usuarios).

Para seguir estos preceptos se escogió el tema de **"Implementación de una Máquina Linux como Router"** que permitirá ayudar y llevar a cabo un mayor control, recepción, seguridad y evaluación de las necesidades que puedan existir en las diversas redes de la Empresa, las cuales serán evaluadas y monitoreadas por el área de Sistemas.

1.2. Definición de Router

El Router, que según la traducción sería "enrutador". Dispositivo hardware o software para interconexión de redes de computadoras que opera en la capa tres (nivel de red) del modelo OSI. El Router interconecta segmentos de red o redes enteras. Hace pasar paquetes de datos entre redes tomando como base la información de la capa de red.

El Router toma decisiones (basado en diversos parámetros) con respecto a la mejor ruta para el envío de datos a través de una red interconectada y luego redirige los paquetes hacia el segmento y el puerto de salida adecuados.

Es el dispositivo conectado a la computadora que permite que los mensajes a través de la red se envíen de un punto (emisor) a otro (destinatario), de manera tal que entre el alto volumen de tráfico que hay en Internet, nos va a asegurar que el mensaje llegue a su destinatario y no a otro lado.

Para realizar esta transmisión a través de las redes de comunicación, el Router se encarga de chequear cada uno de los paquetes (o pequeñas unidades de 1.500 bytes aproximadamente) en los que se divide la información que se envía a través de Internet de un lugar a otro, para asegurarse de que llegue al destino correcto.

Entre sus características, se destaca que siempre buscará la ruta más corta o la que tenga menos tráfico para lograr su objetivo y, por otra parte, que si no funciona una ruta, tiene la capacidad de buscar una alternativa.

En cuanto a los diversos Routers que hay en el mercado, se pueden diferenciar a partir de distintas particularidades, como tipos de conectores que utilizan, protocolos que conocen y velocidad de

transporte. De esta manera, nos encontramos con Routers internos y externos, centrales y periféricos, locales y remotos.

1.3. Opciones de Acceso a la Aplicación

Los usuarios finales de la aplicación al momento de utilizar la misma necesitan ingresar la siguiente información:

- ✓ Nombre de usuario
- ✓ Clave de usuario

Tanto el Gerente y Líder de Proyecto (Administrador) podrán acceder a las opciones que dispondrá el módulo de acuerdo a los roles que se les haya configurado.

El usuario por defecto será **admin** y su contraseña por consiguiente (default) es la misma; pudiendo ser cambiada para efectos de mayor seguridad.

Para el desarrollo de este aplicativo nos basaremos en el uso de las herramientas eclipse.

1.4. Visión

Que al finalizar el proyecto, la aplicación se convierta en una poderosa herramienta de trabajo en lo que respecta a este tipo de configuraciones y gracias a ello sea tomada en cuenta y utilizada por alguna Organización o Empresa para proveer algún determinado servicio a otras Organizaciones, llegando a convertirse en la herramienta de configuración de Router estándar a nivel nacional.

1.5. Misión

Configurar como Router una PC que tenga instalada una distribución del Sistema Operativo Linux especificando las tablas de ruteo dinámicas y estáticas de manera gráfica por medio de una Interfase Web (Aplicación), considerando todas las medidas necesarias para que la seguridad de las redes participantes no sea violada bajo ningún concepto y que su utilización sea amigable y sencilla para los que la utilicen (Usuarios). También se garantizará que la información que será objeto de acceso sea confiable y veraz, para esto la aplicación utilizarán protocolos de red seguros.

1.6. Objetivos Generales

- ✓ Poder realizar la administración y configuración del ruteo de paquetes de PCs locales hacia diferentes redes mediante una interfaz gráfica amigable para usuarios con poca experiencia en Linux.

- ✓ La interfaz manejará las tablas de los protocolos de enrutamiento que utilizan los Routers existentes.

- ✓ La máquina Linux configurada como Router se comportará como tal, para comunicar redes entre sí y compartir información, además que permitirá tomar la decisión de cuales la ruta más adecuada en cada momento para enviar un paquete.

- ✓ Administración del Router (Linux) vía Web.

1.7. Objetivos Específicos

Se tendrán presente los siguientes objetivos:

- ✓ Configurar una máquina Linux como ruteador, especificar tablas de ruteo dinámicas y estáticas de manera gráfica por medio de una interfase Web.

- ✓ Definir un entorno seguro de usuarios a nuestra red.

- ✓ Realizar conexiones seguras de al menos 2 redes físicamente diferentes.

- ✓ Permitir al administrador de la red realizar acceso remoto a equipos denominados cliente.

- ✓ Definir una forma segura en que los datos que se van a transmitir a través de la red, para que no puedan ser leídos por clientes o usuarios no autorizados de la red.

- ✓ Definir un entorno seguro en donde se manejará la información que concierne a los usuarios.

- ✓ Lograr el ingreso y control de los requerimientos del área solicitante por medio de una interfaz.

- ✓ Utilizar protocolos de red seguros, tales como el TCP/IP e ICMP.

- ✓ Utilizar protocolos de ruteo seguros, tales como RIP u OSPF.

1.8. Alcances del Proyecto

Administración del Router vía Web, para los usuarios. Para esto existirá un usuario administrador que se encargará de la creación de los mismos y los permisos que ellos deberán tener (Roles). Estos usuarios quedarán registrados en una bitácora de control que indicará entre otras cosas quiénes ingresaron a la aplicación Web, tiempo que estuvieron dentro de esta y qué fue lo que solicitaron al administrador.

Desarrollo e implementación de la aplicación Web que permitirá administrar de forma adecuada el manejo de los sucesos del ruteador.

1.8.1. Descripción del Módulo para la Administración del Router

- ✓ Para acceder a la aplicación se colocará la dirección Web en el browser del navegador, en ese momento se cargará la página principal que dará la bienvenida al usuario administrador.

- ✓ Esta página principal contendrá links o vínculos tales como **Bienvenida, Gestión de Usuarios, Bitácora de Control, Configuración Básica** y **Administración del Router**.

- ✓ El usuario Administrador tendrá acceso a links (vínculos) que los usuarios normales no van a tener.

- ✓ Estas páginas van a interactuar directamente con los archivos de configuración de Linux, en el momento que se realice algún ingreso a través de las páginas JSP, y se presione el botón de grabar o ejecución de acción se actualizarán dichos archivos con los nuevos datos ingresados o configurados.

- ✓ Con respecto a la información que corresponde a los usuarios cabe indicar que se la manejará por medio de archivos que estarán debidamente encriptados para protegerlos ante la difícil pero posible extracción de los mismos por algún intruso.

1.8.2. Gestión y Control

Se considerarán los siguientes puntos:

- ✓ En lo que respecta a Gestión de Usuarios, se manejará todo lo concerniente con la administración de los usuarios del sistema, es decir, creación, eliminación y cambio de contraseña del mismo. Cabe indicar que solo el Administrador tendrá acceso total a las 3 funciones descritas anteriormente, lo que no ocurre con un usuario corriente (Operador) que solo podrá cambiar su contraseña.

- ✓ En lo que respecta a Bitácora de Control, se tendrá el control total de la información que corresponde al usuario, como por ejemplo: Consulta de usuarios registrados, Consulta de usuarios que están utilizando la aplicación en ese momento, Consulta de un historial de sesiones de todos los usuarios y Consulta de todos los requerimientos del usuario Operador hacia el Administrador. Dentro de esta visualización de requerimientos existirá una opción para responder los mismos. Cabe indicar que solo el Administrador tendrá acceso a este link. El usuario Operador solo tendrá una

opción en la cual se le permitirá el ingreso de su petición o sugerencia.

- ✓ En lo que respecta a Configuración Básica, trata la forma en que el usuario podrá manejar las direcciones IP, configuración de tarjetas, etc. de una forma más amigable, fácil y rápida.

- ✓ En lo que respecta a Administración del Router, es precisamente aquí en donde se configuran las tablas de ruteo de las diferentes redes ya sea estática o dinámicamente.

- ✓ Cabe recalcar que solo podrá hacer esto el usuario que tenga los permisos específicos en otras palabras el rol Administrador. Los usuarios con rol Operador estarán restringidos realizando solamente consultas.

1.9. FODA de la Aplicación Web del PC Linux como Router

1.9.1. Fortalezas

- ✓ Administración del Router vía Web.
- ✓ La aplicación Web permitirá la comunicación de varias redes LAN.
- ✓ El acceso hacia los objetos de una determinada red estará restringido para los usuarios que no tengan los permisos correspondientes.
- ✓ Interfaz amigable y de fácil manipulación para los usuarios.
- ✓ Llevará un control de todos los usuarios que accesaron a la aplicación.
- ✓ Garantizará la seguridad de todas las redes involucradas o participantes debido a los permisos de accesibilidad para los usuarios.

- ✓ Llevará un mejor control de las actividades de los usuarios de las diferentes redes participantes.

- ✓ Optimización de los recursos suministros, tiempo y humano.

1.9.2. Oportunidades

- ✓ Ampliar los conocimientos por parte del grupo de trabajo.

- ✓ Manejo de plataformas que no son convencionales, es decir, podremos especializarnos en un entorno casi desconocido como lo son las distribuciones del Sistema Operativo Linux.

- ✓ La opción de escoger cuál es la distribución Linux que más convendría para la realización de la aplicación Web.

- ✓ Posibilidad de que una vez finalizado el proyecto sea utilizado por Organizaciones para proveer algún tipo de servicio.

- ✓ Aportar con una herramienta que además de comunicar redes locales lo hará con normas de seguridad aceptables.

- ✓ Ahorro de recursos utilizados por la Empresa.

1.9.3. Debilidades

- ✓ Fallas repentinas de los componentes del computador que hace la simulación del Router, lo cual interferiría en la realización del proyecto de manera continua ya que esto involucraría paralizar las actividades hasta que se den soluciones al inconveniente presentado.

- ✓ Mal funcionamiento de los paquetes de configuración de la distribución Linux que hace posible que el PC se comporte como Router.

1.9.4. Amenazas

- ✓ Saturación del PC Router ocasionado por cortes de energía inesperados.

- ✓ Instalación de software o aplicaciones innecesarias que afectarían el rendimiento de la PC Router.

- ✓ Filtración u obtención de información del proyecto por medio de algún programa o accesos de hackers a la red.

- ✓ Mal uso de la aplicación por usuarios nuevos o inexpertos que podrían afectar el rendimiento de la misma.

- ✓ Fallas en el servidor encargado de cargar la página Web de la aplicación.

- ✓ Orientación por parte del Cliente hacia otras aplicaciones expuestas al mercado por la competencia.

1.10. Metodología

Un modo alternativo de presentar modelos de actividad que toma en cuenta la retroalimentación entre etapas y la repetición de tareas, es el llamado **Modelo en Espiral**.

En este diagrama, el uso de la espiral implica que las diferentes actividades de la ingeniería de requerimientos son repetidas hasta que se toma la decisión final, que es la aceptación del documento de especificación de requerimientos.

Es decir, si en el diseño preliminar se encuentran problemas, entonces recorremos el ciclo nuevamente (extracción-análisis-especificación-validación) hasta que todos sean resueltos, que es lo mismo que decir que este ciclo continuará hasta que se pueda elaborar un documento aceptable.

Pero también existen factores externos que pueden determinar la finalización del ciclo, como por ejemplo la presión por cumplir con un determinado cronograma. Luego del análisis respectivo de varios modelos, podemos concluir que dado el escenario de trabajo es más válida la aplicación del modelo en espiral para desarrollar la aplicación Web. Y es que el modelo en espiral representa de manera más real cómo se irán desarrollando las actividades del proceso, esto es, debido al desconocimiento del tema, se genera un grado demasiado alto de incertidumbre que sólo puede disminuirse al repetir el ciclo de trabajo una y otra vez, permitiendo así ajustar todos los parámetros, cada vez en mayor detalle, hasta lograr un resultado satisfactorio.

Este modelo combina dos modelos importantes como son: el ***Modelo Lineal Secuencial*** y el ***Modelo de Construcción de Prototipos***.

En el **Modelo Lineal Secuencial (Ciclo de vida clásico)** se identifican un conjunto de requisitos de la aplicación a desarrollar, tales como, entradas, procesamientos y salidas. Con esto se elabora un prototipo inicial como punto de partida que sirve para detallar objetivos más concretos del producto final y que satisfaga al usuario.

Es un enfoque sistemático y secuencial del desarrollo del software que comienza en un nivel de sistemas y progresa de la siguiente manera:

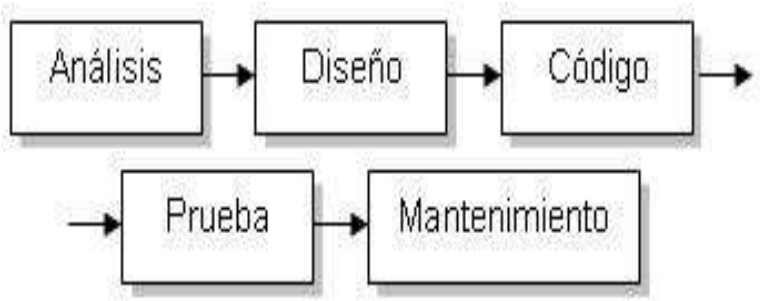

Ilustración 1-1 Modelo Lineal Secuencial

El **Modelo en Espiral** proporciona el potencial para el refinamiento continuo del modelo inicial y desarrollo de versiones incrementales del software así como también el análisis de riesgos del mismo.

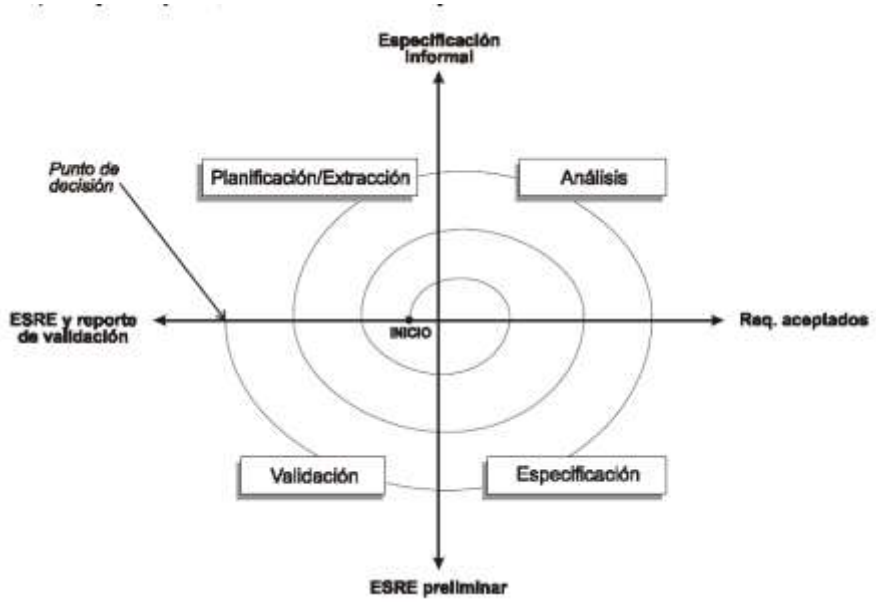

Ilustración 1-2 Modelo en Espiral

La finalidad como tal, es optimizar la reutilización de código, así como construir y almacenar objetos complejos.

1.11. Arquitectura

Para el desarrollo de nuestro sistema se utilizará la arquitectura 3 capas. En la aplicación Cliente – Servidor se nos presenta la ventaja que se tendrá dividida a la aplicación en capas, tales como: Composición Visual, Reglas y Objetos del negocio. Esto permitirá mantener un estándar de programación, manejar el concepto de los roles de trabajo sobre la aplicación, privilegios de acceso, entre otras funciones.

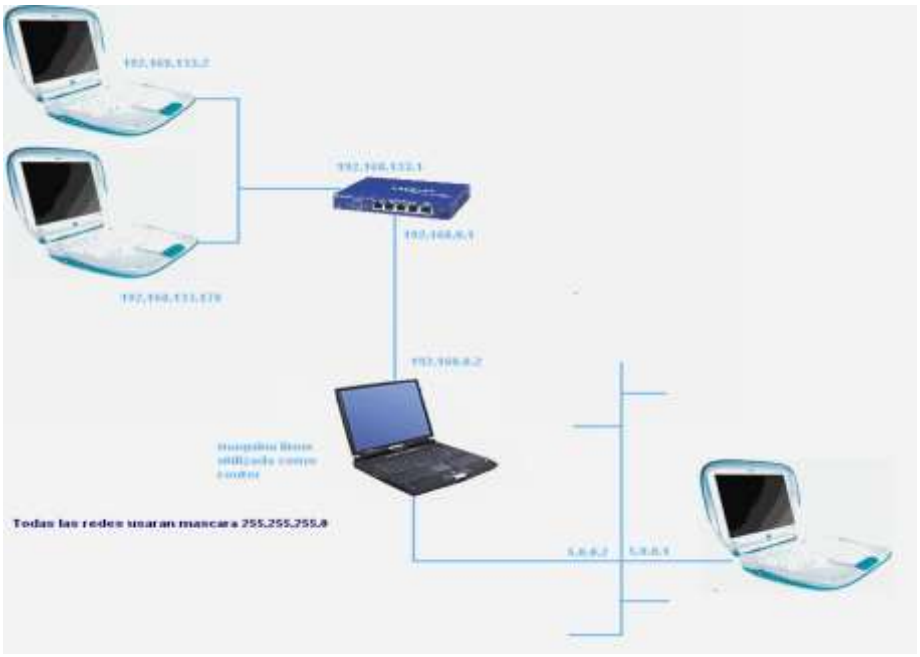

Ilustración 1-3 Arquitectura

1.12. Tecnología

Debido a que la mayor parte del desarrollo será realizado bajo la plataforma de Java, la herramienta que usaremos para crear sus objetos será My Eclipse.

My Eclipse:

Eclipse posee un editor visual, que ofrece, compilación incremental de código, un potente depurador, un navegador de clases, un gestor de archivos y proyectos, pero no se limita sólo a esto; la versión estándar

de My Eclipse proporciona también una biblioteca de refactorización de código y una lista de tareas, además incluye una herramienta para completar el código.

El asistente de contenido, encargado de mostrar los métodos y atributos de las clases con las que se está trabajando, ya sea que formen parte de las APIs de JAVA o de cualquier otra clase en el build path, aunque estén en ficheros JAR.

Cabe notar que la herramienta My Eclipse es una aplicación amigable para el usuario; la desventaja que se presenta en esta aplicación al momento de su ejecución es que se necesita de un mayor performance y capacidad de los equipos.

1.13. Recursos

Para la creación de los nodos de red, necesitaremos algunos componentes, o recursos indispensables para cumplir con los objetivos del mismo.

Estos recursos se subdividen en 3 tipos, los cuales se detallan a continuación:

- ✓ RECURSO DE HARDWARE
- ✓ RECURSO DE SOFTWARE
- ✓ RECURSO HUMANO

1.13.1. Recurso de Hardware

Para nuestra implementación se va requerir del siguiente hardware detallado a continuación:

- ✓ HARDWARE SERVIDOR
- ✓ HARDWARE CLIENTES
- ✓ HARDWARE OTROS

1.13.1.1. Hardware Servidor

Los equipos que harán la función de CORE de este proyecto serán computadores personales con las siguientes características:

Router Linux

Cantidad	Descripción	Características
1	MOTHERBOARDS	Socket478 BIOSTAR DDR2 Audio / Video / Lan
1	Procesador	Pentium 4 3.0 Ghz Velocidad de Bus: 800Mhz
2	Memoria Ram	512 Mb PC 400 o superior
1	Disco duro	120 Gb IDE 7200 r.p.m.
1	Lectores Opticos	DVD-RW / CD-ROM
1	Monitor	14 pulgadas
1	Teclado	Estándar
1	Mouse óptico	Estándar
2	Ethernet	• Tarjeta de Red PCI 10/100 • Tarjeta de red incorporada en placa base

Tabla 1-1- Descripción PC Linux

1.13.1.2. Hardware Clientes

En nuestro proyecto vamos a contar con 2 computadores portátiles a las cuales se les va instalar máquina virtual con lo que obtendríamos 4 PC; 2 PC Físicas y 2 PC Virtuales en nuestra red. Las características de las PC se detallan a continuación:

PCs de la Red (Hosts)

Cantidad	Descripción	Características
1	MOTHERBOARDS	Pentium 3 o Superior
1	Procesador	Pentium 3 1.6 Ghz o superior
2	Memoria Ram	512 Mb o superior
1	Disco duro	40 Gb
1	Lectores Opticos	DVD-RW / CD-ROM
1	Monitor	14 pulgadas
1	Teclado	Estándar
1	Mouse óptico	Estándar
1	Ethernet	Tarjeta de red incorporada en placa base

Tabla 1-2- Descripción PC Hosts

1.13.1.3. Hardware Otros

Se necesitarán también de otros componentes de hardware, que nos ayudarán a la interconexión de nuestra Red, los cuales se detallan a continuación:

Cantidad	Descripción	Características
1	Switch	Switch de 8 puertos 10/100 TX
8	Conectores	Conectores RJ45 (de ocho hilos)
5 A	Cables de red UTP	Categoría 5 o 6

Tabla 1-3-Descripción de Otros Dispositivos

1.13.2. Recurso de Software

En lo que respecta al software, a continuación se detalla:

Software	Descripción
 **Linux**	Linux Distribución Centos 5 Paquetes Adicionales: Dns, squid, iptables.
	Microsoft Windows xp. Para los clientes de nuestra red.
	Vmware Workstation 5 Es un programa que sirve para crear máquinas virtuales, lo que permite tener varios sistemas operativos corriendo bajo otros. Con cual obtendríamos como mínimo 2 host en la red por maquina física.
	Editor Java MyEclipse Vamos usar el MyEclipse para desarrollar nuestra aplicación.
	**Java Development Kit V.1.5**
	The **Apache** HTTP Server

Tabla 1-4- Recursos de Software

1.13.3. Recurso Humano:

Para el desarrollo del proyecto será necesario contar con 3 recursos, los cuales se citan a continuación:

- ✓ Boris Alex López Macías.
- ✓ Jorge Giovanny Sánchez De La Torre.
- ✓ José Daniel Sotomayor Navarro.

CAPÍTULO 2

2. ANALISIS

2.1. Requerimientos

Inicialmente se pensó en interconectar 2 o 3 redes mediante un solo Router, siendo esto posible solamente configurando el ruteo estático. Pero, para efectos de demostración de que la aplicación también será capaz de manejar el ruteo dinámico se llegó a la conclusión de implementar 2 o 3 Routers, es decir, se utilizarán 3 PC con distribuciones de Linux para que hagan las veces de Router.

Básicamente el proyecto consiste en manejar las tablas de ruteo estáticas y dinámicas de la máquina Linux que hará las veces de Router mediante una aplicación Web que se construirá, pero además de eso se crearán diferentes entornos para los usuarios que

manejarán esta aplicación, es decir, se manejará el esquema de Cliente – Servidor en donde el usuario Operador de una determinada estación de trabajo de la red solicitará un requerimiento de configuración al usuario Administrador de la red en donde este gestionará dicho pedido enviando una respuesta. En conclusión, existirán 2 tipos de usuarios, **Usuario Administrador** y **Usuario Operador** en donde solamente el usuario Administrador tendrá el control total de la aplicación. La aplicación que se encargará de manejar todas las actividades descritas anteriormente la hemos denominado **Router Easy.**

En lo que respecta a la información que se manejará, inicialmente se consideró hacerlo por medio de una base de datos, pero debido a que el volumen de información que se tratará no es tan extenso se tomó la decisión de manejarla por medio de archivos que estarán debidamente encriptados dándoles así un alto grado de seguridad.

2.2. Mecánica de Control

Mediante la implementación de la interfaz de **Router Easy** se proveerá al Administrador de la aplicación todas las opciones necesarias para la configuración, mantenimiento y el control de todas las actividades que

se presenten en un determinado momento o tiempo de utilización dentro de una red.

Su funcionalidad básica inicia con el ingreso de la dirección IP del Router o Nombre de Dominio de la red del mismo (opcional) con lo cual aparecerá la página inicial de nuestra aplicación Router Easy.

El Router Easy brindará los siguientes servicios:

- ✓ **Gestión de Usuarios.-** Consiste en actividades de mantenimiento de usuarios, como las siguientes:
    - Creación de usuario.
    - Cambio de clave de usuario.
    - Eliminación de usuario.

- ✓ **Bitácora de Control.-** Consiste en consultar toda la información que corresponde a los usuarios, desde datos personales hasta monitoreo de actividades de los mismos; esto será posible por medio de las sesiones que estos inicien. Las opciones que se manejarán serán:
    - Consulta de usuarios registrados.
    - Consulta de usuarios logoneados en ese momento.

- Consulta de un historial de sesiones de los usuarios.
- Consulta de peticiones de los usuarios, pudiendo éstas ser contestadas al instante.

✓ **Configuración Básica .-** Consiste en la configuración de:

- Las Interfaces de Red.
- Las máscaras de subred de esas interfaces.

✓ **Administración del Router.-** Consiste en la configuración de las Tablas de Ruteo, las opciones que manejará serán:

- Ruteo Estático.
- Ruteo Dinámico.

Para la utilización de estos servicios u opciones se presentará una pantalla en la cual se solicitará el ingreso del usuario y el password asignados que limitará el uso del módulo de acuerdo a los privilegios otorgados. Una vez verificado el usuario y su contraseña, este procederá a escoger cualquiera de las opciones que ofrece el menú de la página principal (descritos arriba); siempre y cuando el usuario que ingrese tenga todos los permisos o el rol de Administrador. En caso de que el usuario que acceda a la página principal tenga el rol de Operador, las opciones que estarán a su disposición serán:

- ✓ **Gestión de Usuarios.-** Donde el usuario tendrá acceso a las siguientes opciones:
  - Cambio de clave de usuario.
  - Peticiones.

- ✓ **Consulta de Tablas de Ruteo.-** Donde el usuario podrá saber cómo se han configurado las tablas de ruteo, las opciones que se desplegarán serán:
  - Ruteo Estático.
  - Ruteo Dinámico.

2.3. Diagrama de Clases a utilizar

2.3.1. Análisis del Diagrama de Clases

El diagrama de clases correspondiente al Módulo a implementar contendrá las siguientes clases o estructuras indicadas a continuación:

- ✓ Creacion_user_admin
- ✓ Logon_v2
- ✓ Confirma_rol
- ✓ Crear_usuarios

- ✓ Cambio_clave_v2
- ✓ Eliminar_usuarios
- ✓ Crear_bitacora
- ✓ Responder_solicitud
- ✓ Verificar_archivos_bitacora_control
- ✓ Eliminar_logoneados_v2
- ✓ Encriptador
- ✓ Verificar_archivos_basicos
- ✓ Configura_basica
- ✓ Configura_host_pass_ripd
- ✓ Procesa_Arch_tabla
- ✓ Mascara
- ✓ Lista_iface
- ✓ Lista_eth
- ✓ Información_interface
- ✓ Ruta_Tabla
- ✓ Config_rip

La formación de este modelo de procesos esta basado en el análisis detallado de todos los requerimientos que esta aplicación debe satisfacer. A continuación se presenta el diagrama de clases.

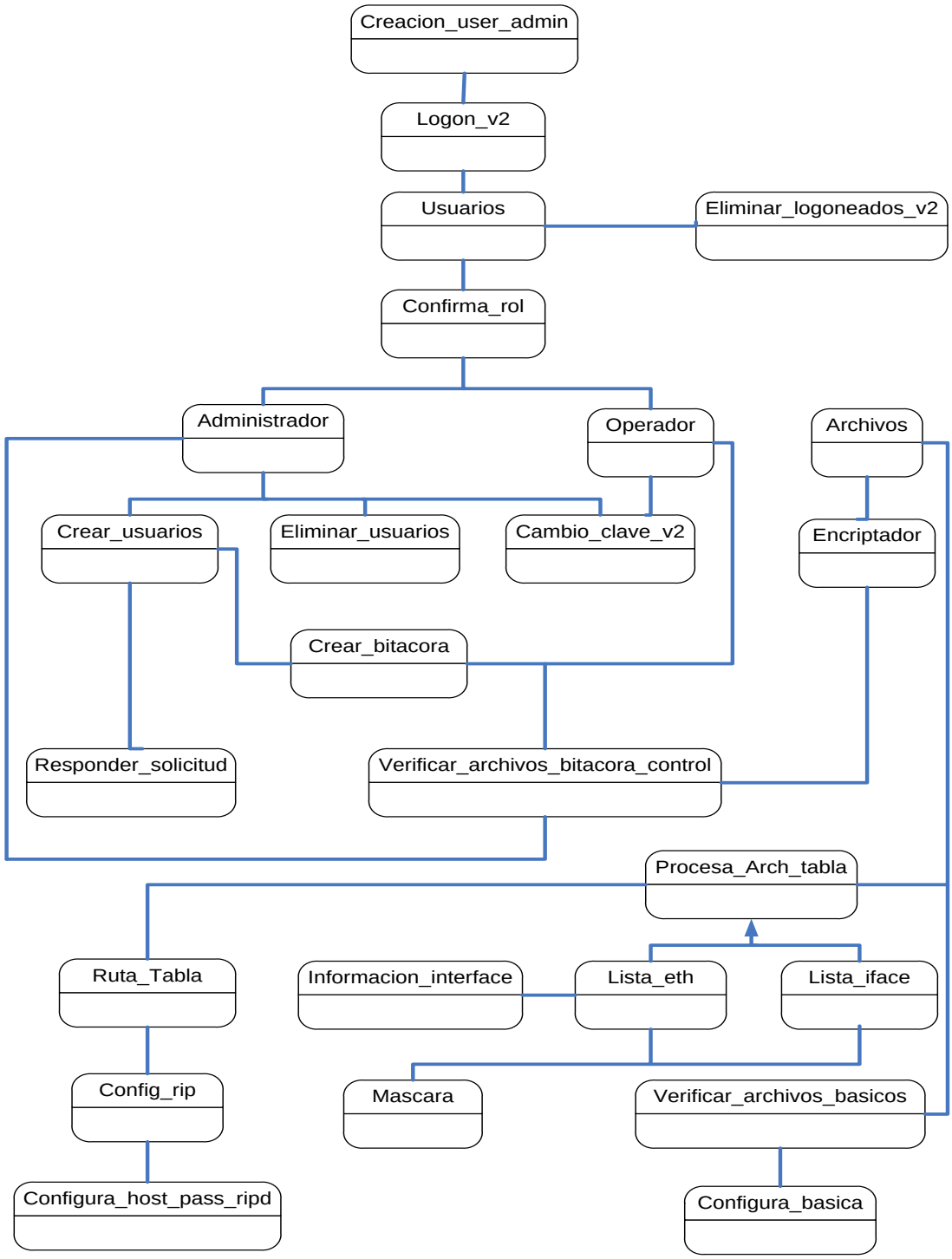

Ilustración 2-1 Diagrama de Clases

2.4. Casos de Uso

La utilización de los Casos de Uso nos permitirá mostrar las diversas secuencias de acciones que la aplicación realizará para obtener un resultado útil para el usuario.

Estas acciones estarán comandadas o manejadas por las clases de usuarios existentes a los cuales dentro de la nomenclatura de los casos de usos se los conoce como actores.

Actores que intervienen:

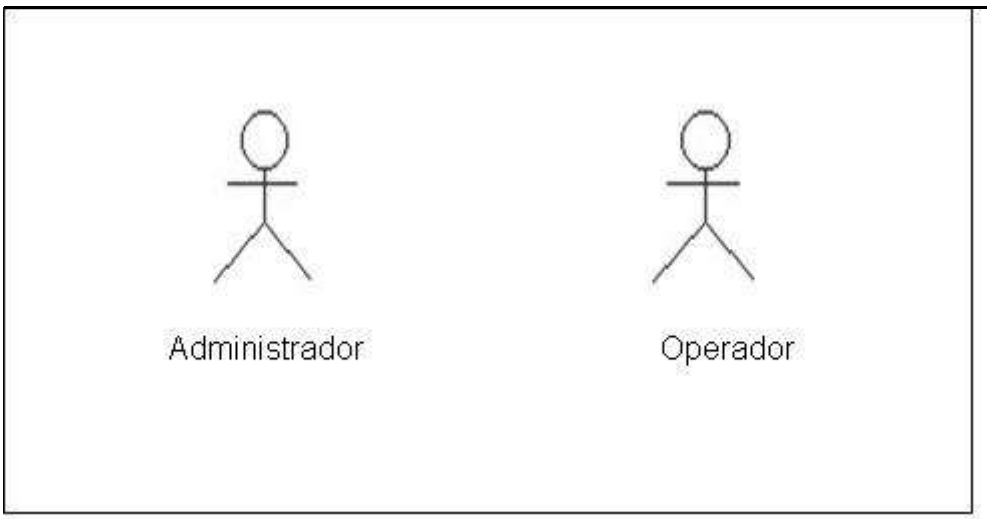

Ilustración 2-2 Actores de los Casos de Usos

Caso de Uso 1

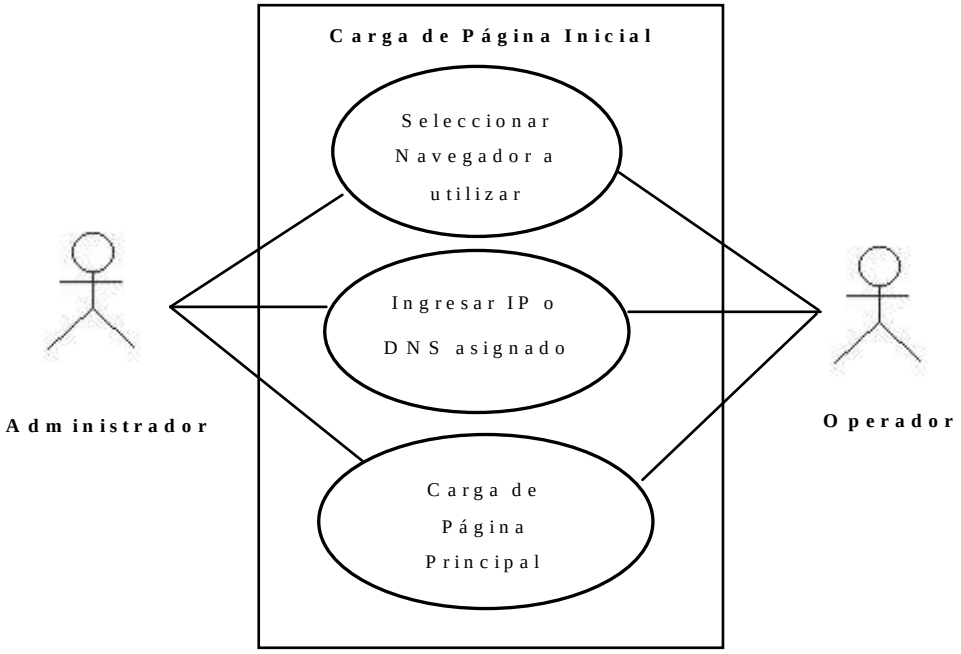

Ilustración 2-3 Caso Carga de Pagina Inicial

Nombre:	Carga de la Página Inicial de la Aplicación
Descripción: Permitirá acceder a la página o interfaz inicial de la aplicación Web digitando el URL designado, en el browser de un determinado navegador.	
Actores: 1. Administrador (Principal) 2. Operador (Secundarios).	
Precondiciones: Tener levantado el servidor Web que permitirá que la página inicial y la aplicación en general funcione.	
Flujo Normal: 1. Tener los servicios que nos van a permitir levantar la aplicación. 2. Asignar a esa dirección IP inicial, un DNS con el cual se identificará de una mejor manera el URL de la aplicación (opcional). 3. Los actores hacen uso de esa dirección para acceder a la aplicación.	

Flujo Alternativo: Crear un icono del navegador en el escritorio para cargar la aplicación de una forma más directa.
Poscondiciones: Si todo esta correcto cargará la página inicial de Router Easy.

Tabla 2-1– Caso Carga de Página Inicial

Caso de Uso 2

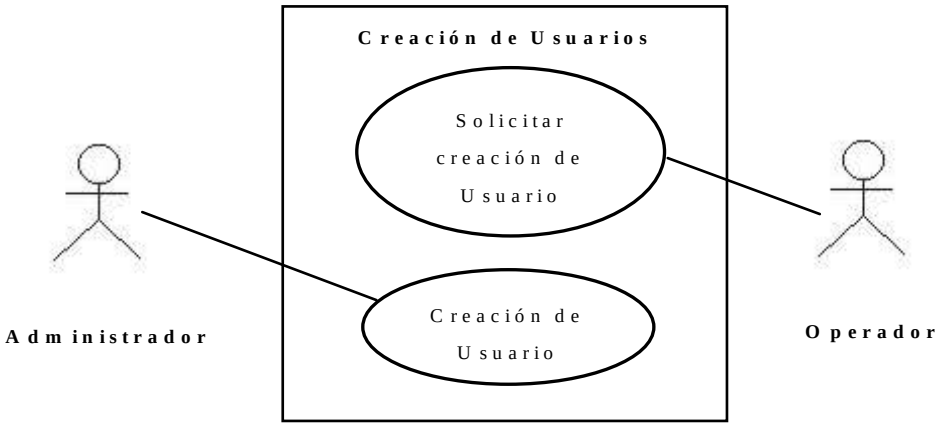

Ilustración 2-4 Caso Creación de Usuarios

Nombre:	Creación de Usuarios
Descripción: Una vez que el usuario haya ingresado a la página inicial de la aplicación se presentará un botón que permitirá acceder a la siguiente página, en donde se presentarán las opciones de ingreso de un usuario y su respectiva contraseña en caso de que éste (usuario) ya exista, caso contrario se debe solicitar al Administrador que realice la creación de el nuevo usuario.	
Actores: 1. Administrador. 2. Operador.	
Precondiciones: Haber cargado correctamente la página inicial o de bienvenida.	
Flujo Normal: 1. Registro del usuario. 2. Asignación de user y password.	
Flujo Alternativo: No aplica.	
Poscondiciones: El actor Operador estará listo para logonearse y acceder a la página principal de la aplicación.	

Tabla 2-2– Caso Creación de Usuarios

Caso de Uso 3

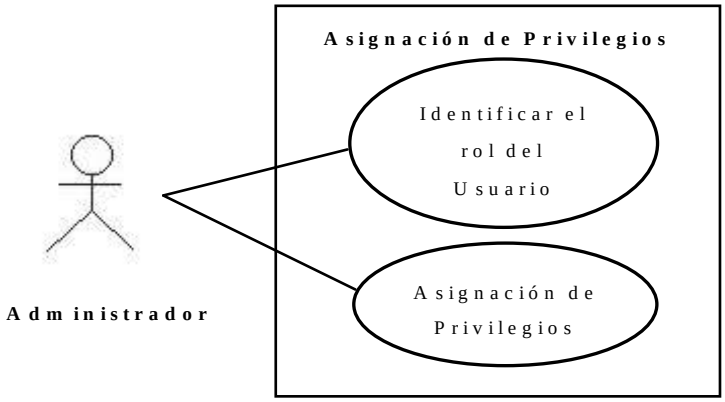

Ilustración 2-5 Caso Asignación de Privilegios

Nombre:	Asignación de Privilegios
Descripción: Se otorgarán todos los permisos para la libre manipulación de la aplicación. Este privilegio solo lo tendrá el usuario Administrador, ya que el resto de usuarios serán creados con el rol de Operador.	
Actor: 1. Administrador	
Precondiciones: Previamente debe haberse solicitado la creación del usuario.	
Flujo Normal: Creación del usuario. Identificación del rol del usuario (Siempre será Operador). Asignación del rol.	
Flujo Alternativo: No aplica.	
Poscondiciones: Los actores interactuarán con la aplicación sin restricciones.	

Tabla 2-3– Caso Asignación de Privilegios

Caso de Uso 4

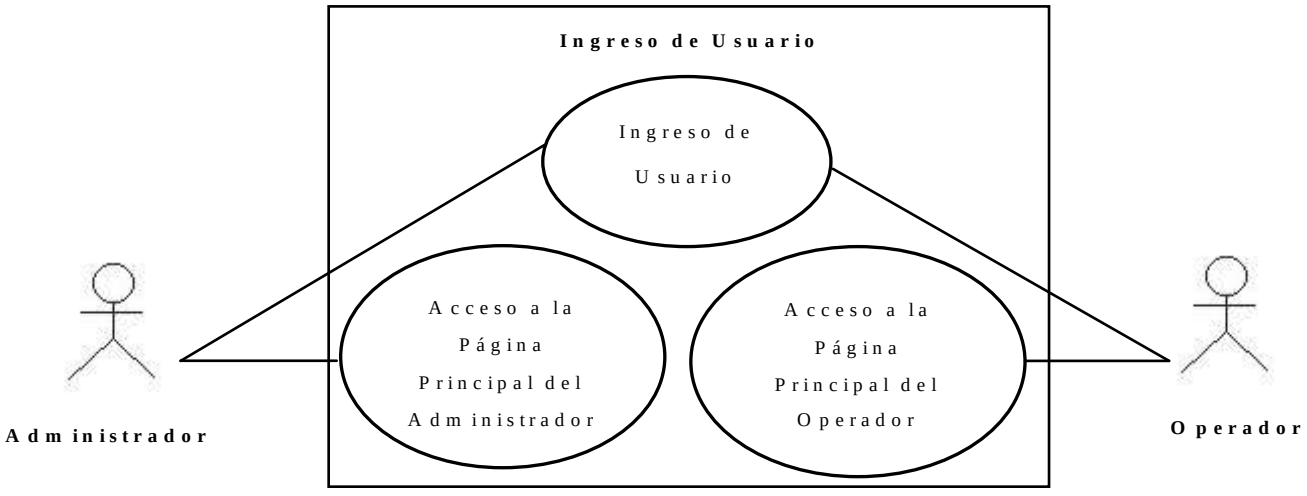

Ilustración 2-6 Caso Ingreso de Usuario

Nombre:	Ingreso de Usuarios a la Página Principal
Descripción: Permitirá ingresar a los usuarios existentes a su correspondiente página principal, en donde podrán tener acceso (dependiendo de los roles) a varios links o enlaces. En el caso del usuario Operador tendrá acceso a toda la información concerniente a las tablas de ruteo, pero no tendrá el poder de configurar nada. Caso que no ocurre con el usuario Administrador que tiene el control total del Router.	
Actores: 1. Administrador. 2. Operador.	
Precondiciones: El usuario que se logonee o ingrese debe existir en la base de datos del sistema (Archivo contenedor de Usuarios), caso contrario no accederá a la página principal.	
Flujo Normal: 1. El usuario debe existir, es decir, debe constar en la base de datos del sistema. 2. El usuario accederá a la pantalla o página principal de la aplicación. 3. Si es un usuario sin privilegios (Operador), interactuará con 2 enlaces que son Gestión de Usuario y Consulta de Tablas de Ruteo. 4. Si es un usuario con privilegios (Administrador), tendrá el acceso total de la aplicación.	
Flujo Alternativo: No aplica.	
Poscondiciones: Acceso a todos los enlaces de la aplicación dependiendo del rol del usuario	

Tabla 2-4– Caso Ingreso de Usuario

Caso de Uso 5

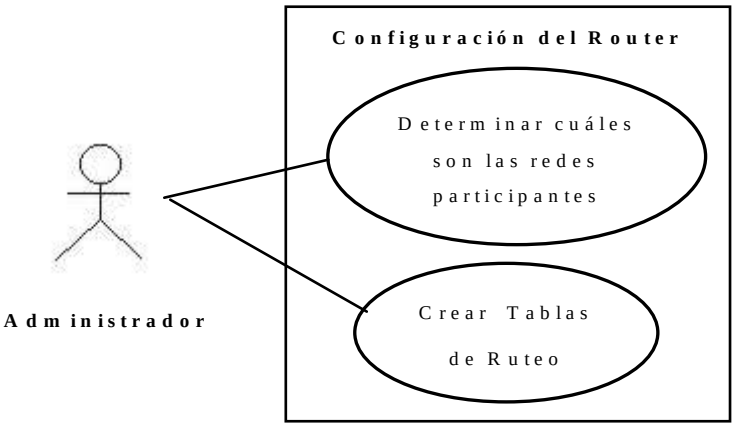

Ilustración 2-7 Caso Configuración de Router

Nombre:	Configuración del Router
Descripción: Permitirá al usuario Administrador configurar la forma en que el Router comunicará a las redes participantes.	
Actores: 1. Administrador.	
Precondiciones: La máquina Linux que hará las veces de un Router deberá estar configurada correctamente, es decir, estará lista para cumplir con los requerimientos solicitados.	
Flujo Normal: 1. Determinar cuáles y cuántas son las redes participantes. 2. Hacer o configurar la respectiva tabla de ruteo estática. 3. Hacer o configurar la respectiva tabla de ruteo dinámica.	
Flujo Alternativo: No aplica.	
Poscondiciones: Comunicación exitosa de 2 o más redes.	

Tabla 2-5– Caso Configuración del Router

Caso de Uso 6

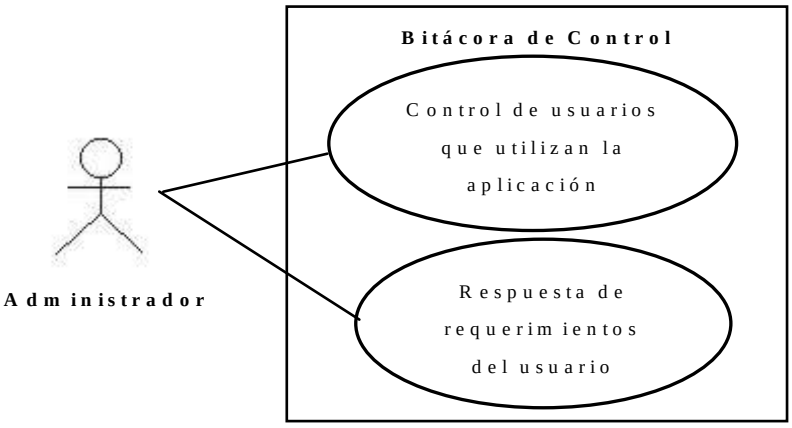

Ilustración 2-8 Caso Monitoreo de Actividades

Nombre:	Monitoreo de Actividades o Bitácora de Control
Descripción: Permitirá tener acceso a todos los datos o información que se tenga de los usuarios como el número de usuarios logoneados, la hora en que este inició y cerró sesión y lo que solicitó cuando estuvo utilizando la aplicación.	
Actores: 1. Administrador.	
Precondiciones: Necesariamente el actor que tenga acceso a este enlace debe ser el usuario Administrador.	
Flujo Normal: 1. Petición de user y password para el usuario. 2. Si el Usuario ingresado posee el rol de Administrador tendrá acceso al enlace de Bitácora de Control. 3. Libre manipulación de la información.	
Flujo Alternativo: No aplica.	
Poscondiciones: Posibles cambios de la información con la que trabaja la aplicación.	

Tabla 2-6– Caso Monitoreo de Actividades

Caso de Uso 7

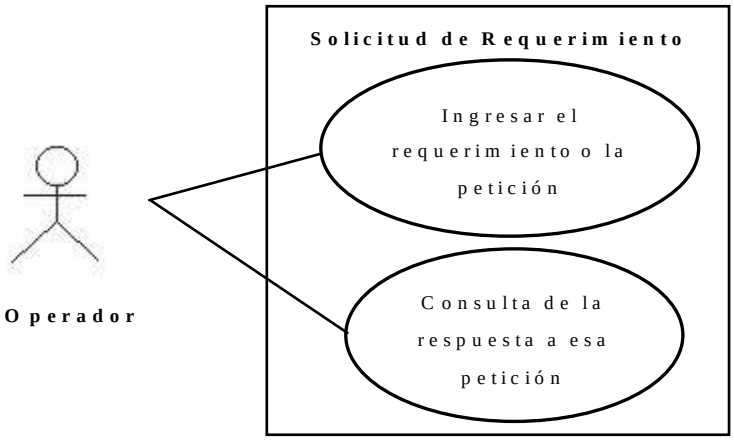

Ilustración 2-9 Caso Solicitud de Requerimiento

Nombre:	Solicitud de Requerimiento
Descripción: En vista de que el usuario Operador solo podrá realizar consultas de todo lo que esta configurado en el Router, este dispondrá de una opción en la cual podrá solicitarle o sugerirle al usuario Administrador la configuración de una determinada ruta.	
Actores: 1. Operador.	
Precondiciones: El actor que tenga acceso a este enlace debe ser el usuario Operador.	
Flujo Normal: 1. Petición de user y password para el usuario. 2. Si el Usuario ingresado posee el rol de Operador tendrá acceso al enlace de Gestión de Usuario – Petición.	
Flujo Alternativo: No aplica.	
Poscondiciones: No podrá realizar más de una petición a la vez. Podrá hacerlo siempre y cuando su primera petición haya sido contestada.	

Tabla 2-7 Caso Solicitud de Requerimiento

Caso de Uso 8

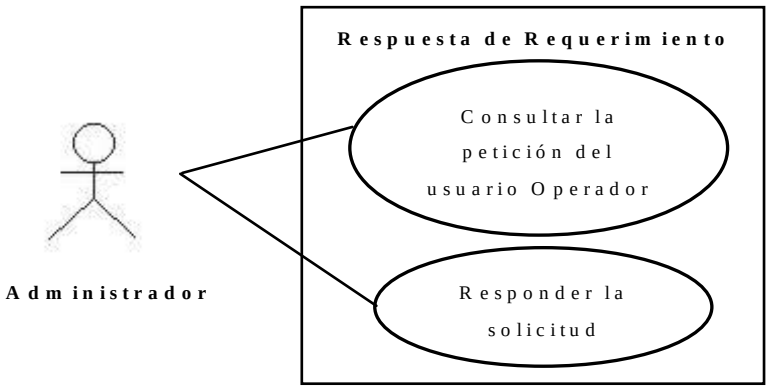

Ilustración 2-10 Caso Respuesta de Requerimiento

Nombre:	Respuesta de Requerimiento
Descripción: El usuario Administrador podrá resolver las inquietudes del usuario Operador contestando todas las peticiones que este formule.	
Actores: 1. Operador.	
Precondiciones: El actor que tenga acceso a este enlace debe ser el usuario Administrador.	
Flujo Normal: 1. Petición de user y password para el usuario. 2. Si el Usuario ingresado posee el rol de Administrador tendrá acceso al enlace de Bitácora de Control – Requerimientos.	
Flujo Alternativo: No aplica.	
Poscondiciones: No podrá contestar 2 veces una misma petición.	

Tabla 2-8– Caso Respuesta de Requerimiento

2.5. Diagrama de Flujo de Datos

Durante la etapa de análisis se obtuvo el DFD (Diagrama de Flujo de Datos), el cual se encargará de definir paso a paso todas las actividades que se llevarán a cabo a lo largo de la utilización de la aplicación de acuerdo a los roles que posean los usuarios que se logoneen.

Se muestra de manera general el diagrama de flujo de datos.

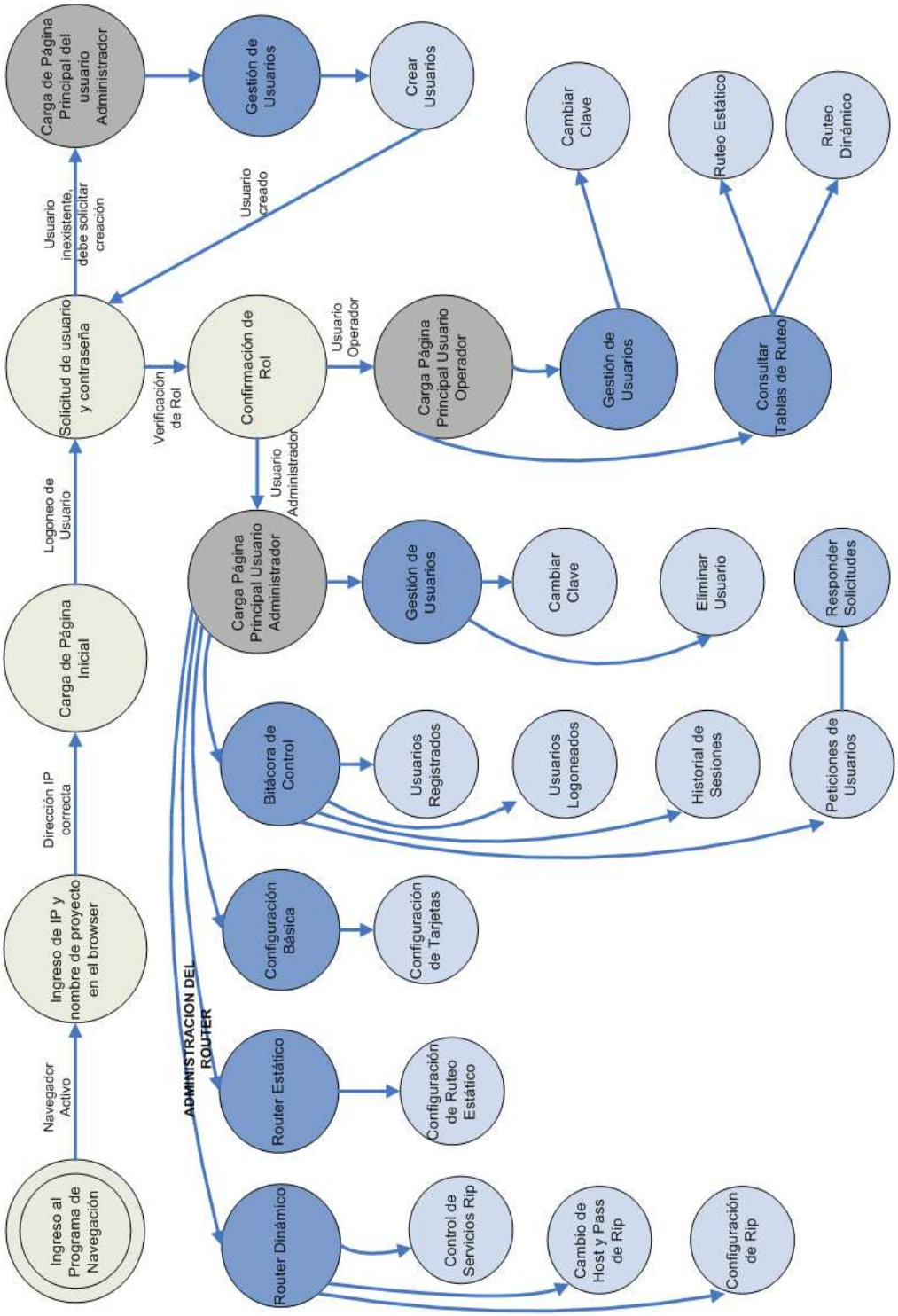

Figura 2.11 – Diagrama de Flujo de Datos

A continuación se explica, de manera general, los procesos que se involucran en el Diagrama de Flujo de Datos.

2.5.1. Ingreso al navegador

Se escoge el aplicativo navegador de sitios Web de preferencia, el cual ser utilizado para levantar la aplicación Router Easy.

2.5.2. Digitar dirección del servidor de aplicación en el browser del navegador

Luego de haber ingresado al navegador, se procederá a digitar la dirección IP correspondiente al servidor de la aplicación junto con el nombre del proyecto que es lo que se encargará de levantar la aplicación Web.

2.5.3. Carga de página inicial

Mostrará la página inicial en la cual se visualizará el nombre de la aplicación que se va a utilizar (Router Easy) y adicionalmente se presentará un enlace o botón encargado de acceder a la siguiente página.

2.5.4. Solicitud de usuario y contraseña

A todos los visitantes que accedieron a esta página se les solicitará el ingreso de su respectivo user y password para que estos puedan acceder a la página principal de la aplicación. En caso de que el usuario que esta intentando ingresar no exista, deberá solicitar al usuario Administrador la creación del mismo, caso contrario nunca podrá utilizar la aplicación.

2.5.5. Confirmación de rol

Una vez que el visitante ingrese correctamente su usuario y contraseña se procederá a verificar el rol que éste tiene ya que de acuerdo a ello se cargarán las siguientes páginas:

- ✓ Página Principal Administrador
- ✓ Página Principal Operador

La funcionalidad de las páginas descritas se presenta a continuación:

2.5.6. Carga de página principal del usuario administrador

Si el rol identificado corresponde al del usuario Administrador se visualizará la página principal del mismo, con la cual se tendrá el control total de la aplicación Web, ya que las opciones que vienen consigo lo permitirán. Las opciones o enlaces que esta página ofrece son:

- ✓ Gestión de Usuarios.
- ✓ Bitácora de Control.
- ✓ Configuración Básica.
- ✓ Router Estático.
- ✓ Router Dinámico.

2.5.7. Gestión de usuarios

Dentro de esta opción se manejará todo lo que concierne al mantenimiento de los usuarios. Las opciones de mantenimiento de usuarios que se desplegarán son:

- ✓ **Crear Usuario.-** Se podrán crear todos los usuarios Operadores que se requieran.
- ✓ **Cambio de Clave.-** El usuario Administrador podrá cambiar su clave de ingreso las veces que desee.

- ✓ **Eliminar Usuario.-** Se podrán eliminar todos los usuarios Operadores que se requieran.

2.5.7.1. Bitácora de control

Con esta opción se podrá consultar toda la información que corresponde a los usuarios, desde datos personales hasta monitoreo de actividades de los mismos; esto será posible por medio de las sesiones que estos inicien. Las opciones que se manejarán son:

- ✓ **Usuarios Registrados.-** Se mostrará toda la información de los usuarios que están autorizados para utilizar la aplicación, es decir, esta opción realiza un mapeo del archivo de usuarios.
- ✓ **Usuarios Logoneados.-** Se mostrarán todos los usuarios que están utilizando la aplicación en ese momento.
- ✓ **Historial de Sesiones.-** Se mostrará de manera específica todas las veces en que los usuarios iniciaron y finalizaron sesión.
- ✓ **Requerimientos.-** Dentro de esta opción se podrán visualizar todas las peticiones o requerimientos realizadas

por los usuarios Operadores, las mismas que podrán ser atendidas al instante.

2.5.7.2. Configuración básica

Dentro de esta opción se podrán configurar los parámetros de red básicos, esto se realizará mediante la opción que se desplegará:

- ✓ **Configuración de Tarjetas.-** Este enlace permitirá configurar todas las tarjetas de red que posea la máquina o PC Linux que se comportará como un Router, se podrá definir cuál será su dirección IP con su respectiva máscara.

2.5.7.3. Router estático

Esta opción permitirá eliminar o agregar manualmente las rutas con las cuales una red se podrá comunicar con otras redes, especificando el Gateway por el cual se realizará dicha comunicación.

2.5.7.4. Router dinámico

Esta opción configurará de manera automática las rutas por las cuales la comunicación de las redes será más eficiente.

2.5.8. Carga de página principal del usuario operador

Si el rol identificado corresponde al del usuario Operador se visualizará la página principal del mismo, con la cual entre otras cosas solo se podrán realizar consultas de cómo están configuradas las tablas de ruteo. Las opciones o enlaces que esta página ofrece son:

- ✓ Gestión de Usuarios.
- ✓ Consultar Tablas de Ruteo.

2.5.8.1. Gestión de usuarios

Dentro de esta opción solo se contará con los siguientes servicios:

- ✓ **Cambio de Clave.-** El usuario Operador podrá cambiar su clave de ingreso las veces que desee.

- ✓ **Peticiones.-** En vista de que el usuario Operador solo puede realizar consultas de las configuraciones que se han realizado, se creó este enlace con el fin de que éste pueda solicitar al usuario Administrador algún tipo de configuración específica.

2.5.8.2. Consultar tablas de ruteo

Se desplegarán las siguientes opciones:

- ✓ **Ruteo Estático.-** Permitirá visualizar la tabla de ruteo estática, es decir, aquella que ha sido configurada manualmente por el usuario Administrador.
- ✓ **Ruteo Dinámico.-** Permitirá visualizar la tabla de ruteo dinámica, es decir, aquella que el *"router inteligente"* define en el momento que hace el mapeo de todas las redes participantes. Logrando con ello determinar cuál es la ruta óptima para la comunicación de las mismas.

CAPITULO 3

3. DISEÑO

3.1. Diseño de la Solución

El diseño de esta aplicación tiene un alto nivel estratégico y de decisión para resolver los problemas que se presentan con mayor frecuencia al momento de determinar cuales serán las rutas que se utilizarán en la comunicación de diferentes redes. Los grandes problemas se deben resolver partiendo desde la contemplación de todas las soluciones posibles del mismo, para esto, es indispensable siempre partir desde el análisis del problema, para luego construir el diseño de la solución. Este diseño está basado fundamentalmente en los siguientes factores:

- ✓ Repositorio de la información que se maneja.

- ✓ Modelado del Sistema.

3.1.1. Repositorio de información

El diseño del repositorio de la información de la aplicación está basado en el manejo de archivos de texto planos, los cuales simulan el comportamiento de una base de datos. A continuación se presentan los archivos que se utilizan junto a su correspondiente descripción:

- ✓ **User.-** Este archivo contiene la información correspondiente a todos los usuarios registrados. Su estructura es la siguiente:

User name del usuario
Password del usuario
Rol o Privilegio del usuario
Nombres y Apellidos del usuario
Fecha de creación del usuario

Tabla 3-1– Estructura del archivo user

- ✓ **Logoneados.-** Este archivo contiene la información de todos los usuarios que están utilizando la aplicación en un determinado momento. Su estructura es la siguiente:

User name del usuario
Password del usuario
Rol o Privilegio del usuario
Nombres y Apellidos del usuario
Fecha de creación del usuario

Tabla 3-2– Estructura del archivo logoneados

- ✓ **Hist_logon.-** Este archivo contiene la información histórica de todos los usuarios que iniciaron y cerraron su respectiva sesión. Su estructura es la siguiente:

User name del usuario
Fecha de inicio de sesión
Fecha de fin de sesión

Tabla 3-3– Estructura del archivo hist_logon

- ✓ **Bitácora.**- Este archivo contiene la información que corresponde a todas las peticiones o sugerencias que el usuario Operador solicita al usuario Administrador. Su estructura es la siguiente:

User name del usuario
Fecha de solicitud de petición
Petición
Petición contestada

Tabla 3-4– Estructura del archivo bitacora

- ✓ **User_mirror.**- Este archivo contiene la información de todas las peticiones que los usuarios Operadores solicitan junto a su correspondiente respuesta, este archivo será múltiple, es decir, todo usuario que solicite un requerimiento tendrá su respectivo archivo User_mirror. Su estructura es la siguiente.

User name del usuario
Fecha de solicitud de petición
Petición
Respuesta de la petición

Tabla 3-5– Estructura del archivo user_mirror

3.1.2. Modelado del Sistema

Los procesos de las clases que forman parte del **Core** de la aplicación se encuentran especificados en el diagrama de clases que se detalla en el Capítulo 2 de este Tomo.

3.2. Diseño de Clases

A continuación se presenta una descripción detallada de todas las clases utilizadas para la implantación del sistema **Router Easy**.

3.2.1. Creacion_user_admin

Esta clase es la que se encarga de crear al usuario Administrador cuando se inicia la ejecución de la aplicación. Además, es la encargada de crear los archivos en donde se manejará la información para la Gestión de Usuarios. A continuación se muestra su estructura.

CREACION_USER_ADMIN
Atributos: private String usuario private String clave private String privilegio private String nombre private String fecha StringTokenizer str_token
Métodos: public String getCurrentJavaSqlDate1( ) public void carga_campos(StringTokenizer str_token) public void lee_archivo( ) public void crea_usuario( )

Tabla 3-6– Estructura de clase Creación_user_admin

3.2.2. Logon_v2

Esta clase se encarga de validar que todos los usuarios que inicien sesión se encuentren previamente registrados en el archivo de usuarios, luego de eso los almacena según su orden

de ingreso a la aplicación en los archivos de logoneo de usuarios. A continuación se muestra su estructura.

LOGON_V2
Atributos: private String usuario private String clave private String privilegio private String nombre private String fecha StringTokenizer str_token private int cont private int result private result_usu private result_pass
Métodos: public String getCurrentJavaSqlDate1( ) public void carga_campos(StringTokenizer str_token) public String retorna_user(StringTokenizer str_token) public String retorna_clave(StringTokenizer str_token) public String retorna_privilegio(StringTokenizer str_token) public String retorna_nombre(StringTokenizer str_token) public int lee_archivo(String FileSource, String Path, String Usuario, String Password) public int logoneo(String Usuario, String Password) public int valida_usuario(String Usuario, String Password) public int valida_pass(String Usuario, String Password) public int lee_Logon_v2(String user, String path) public String recupera_file(String path) public int valida_archivo_nulo(String Path) public int valida_usuario(String user) public String devuelve_user(String Path, String user) public String retorno_userhist(String Path, String user) public void crea_hist_logon(String Usuario, String Path) public void crea_Logoneados(String Usuario, String Path)

Tabla 3-7– Estructura de clase Logon_v2

3.2.3. Confirma_rol

Esta clase se encarga de identificar cual es el privilegio o el rol del usuario que esta iniciando sesión, de acuerdo a ello el usuario tendrá acceso al entorno de trabajo que le corresponde. A continuación se muestra su estructura.

CONFIRMA_ROL
Atributos: private String usuario private String clave private String privilegio private String nombre private String fecha StringTokenizer str_token private int cont private int result
Métodos: public void carga_campos(StringTokenizer str_token) public String retorna_user(StringTokenizer str_token) public String retorna_clave(StringTokenizer str_token) public String retorna_privilegio(StringTokenizer str_token) public int valida_usuario(String user) public int confirma_privilegio(String user)

Tabla 3-8– Estructura de clase Confirma_rol

3.2.4. Crear_usuarios

Esta clase se encarga de crear a todos los usuarios que utilizarán la aplicación y almacenará la información de los mismos en el archivo de usuarios. A continuación se muestra su estructura.

CREAR_USUARIOS
Atributos: private String usuario private String clave private String privilegio private String nombre private String fecha StringTokenizer str_token private int cont private int result
Métodos: public String getCurrentJavaSqlDate1( ) public void carga_campos(StringTokenizer str_token) public int lee_archivo_valida(String Path, String user) public int lee_archivo(String FileSource, String Path, String user, String pass, String confirma_pass, String privilegio, String nombre) public String recupera_file(String FileSource, String Path, String user, String pass, String confirma_pass, String privilegio, String nombre) public int valida_usuario(String user) public int valida_archivo_nulo(String Path) public void crea_usuario(String user, String pass, String confirma_pass, String privilegio, String FileSource, String Path, String nombre)

Tabla 3-9– Estructura de clase Crear_usuarios

3.2.5. Cambio_clave_v2

Esta clase se encarga de cambiar la clave de los usuarios. A continuación se muestra su estructura.

CAMBIO_CLAVE_V2
Atributos: private String usuario private String clave private String privilegio private String nombre private String fecha StringTokenizer str_token private int cont private int result
Métodos: public String getCurrentJavaSqlDate1( ) public void carga_campos(StringTokenizer str_token) public void carga_campos_e(StringTokenizer str_token) public String retorna_user(StringTokenizer str_token) public String retorna_clave(StringTokenizer str_token) public String retorna_privilegio(StringTokenizer str_token) public String retorna_nombre(StringTokenizer str_token) public int valida_usuario(String user) public String lee_archivo_logon(String user, String pass, String new_pass, String confirma_new_pass) public String lee_archivo(String FileSource, String Path, String user, String pass) public String actualiza_archivo_logon(String FileSource, String Path, String user, String pass) public String recupera_file(String FileSource, String Path, String user, String pass, String new_pass, String confirma_new_pass) public String visualiza_logoneados(String user, String pass) public String visualiza_logoneados_h(String user, String pass) public String graba_new_archivo(String FileSource, String Path, String user, String pass, String new_pass, String confirma_new_pass)

Tabla 3-10– Estructura de clase Cambio_clave_v2

3.2.6. Eliminar_usuarios

Esta clase se encarga de eliminar a los usuarios, toda la información relacionada con el usuario a eliminar será eliminada del archivo. A continuación se muestra su estructura.

ELIMINAR_USUARIOS
Atributos: private String usuario private String clave private String privilegio private String nombre private String fecha StringTokenizer str_token private int cont private int result
Métodos: public void carga_campos(StringTokenizer str_token) public int lee_archivo(String Path, String user) public int valida_usuario(String user) public void elimina_registro(String user, String Path)

Tabla 3-11– Estructura de clase Eliminar_usuarios

3.2.7. Crear_bitacora

Esta clase se encarga de crear el archivo en donde se almacenarán los requerimientos de todos los usuarios. A continuación se muestra su estructura.

CREAR_BITACORA
Atributos: private String usuario private String fecha StringTokenizer str_token private int cont private int result
Métodos: public String getCurrentJavaSqlDate1( ) public int lee_archivo_valida(String Path, String user) public int lee_archivo(String Path, String user) public String recupera_file(String Path, String user) public int valida_usuario(String user) public int valida_archivo_nulo(String Path) public int valida_archivo_nulo_m(String mirror) public void valida_usuario(String user, String Path, String path, String mirror, String desc) public void crea_usuario(String user, String Path, String desc) public void crea_solicitudes(String user, String Path, String path, String mirror, String desc) public int cuenta_lineas(String path)

Tabla 3-12– Estructura de clase Crear_bitacora

3.2.8. Responder_solicitud

Esta clase se encarga de llenar los archivos de peticiones de los usuarios con las respuestas a sus respectivas solicitudes. A continuación se muestra su estructura.

RESPONDER_SOLICITUD
Atributos: private String usuario private String fecha private String trama1 StringTokenizer str_token private int cont private int result
Métodos: public String getCurrentJavaSqlDate1( ) public String retorna_user(StringTokenizer str_token) public String retorna_trama(StringTokenizer str_token) public String recupera_file(String path, String user) public String recupera_file(String mirror, String user) public String lee_archivo(String Path, String user) public int valida_usuario(String user) public int valida_archivo_nulo(String path) public int valida_archivo_nulo_m(String mirror) public void Respuesta_afirmativa(String user, String Path, String path, String mirror, String respuesta) public void Respuesta_negativa(String user, String Path, String path, String mirror, String respuesta) public void contesta_pedido_P(String user, String Path, String path, String mirror, String respuesta) public void contesta_pedido_N(String user, String Path, String path, String mirror, String respuesta) public int cuenta_lineas(String path) public void actualiza_bitacora_ac(String Path, String user) public void actualiza_bitacora_re(String Path, String user)

Tabla 3-13– Estructura de clase Responder_solicitud

3.2.9. Verificar_archivos_bitacora_control

Esta clase se encarga de mostrar el contenido de todos los archivos de gestión de usuarios, es decir le permitirá al usuario Administrador monitorear las actividades de los usuarios. A continuación se muestra su estructura.

VERIFICAR_ARCHIVOS_BITACORA_CONTROL
Atributos: StringTokenizer str_token private int cont
Métodos: public void lee_archivo_user( ) public void lee_archivo_logoneados( ) public void lee_archivo_hist_logon( ) public void lee_archivo_bitacora( ) public void lee_archivo_userX(String user, String path) public void lee_archivo_userX_mirror(String user, String mirror)

Tabla 3-14– Estructura de clase Verificar_archivos_bitacora_control

3.2.10. Eliminar_logoneados_v2

Esta clase se encarga de eliminar de los archivos de logoneo a todos los usuarios que cierran su sesión al cerrar la aplicación. A continuación se muestra su estructura.

ELIMINAR_LOGONEADOS_V2
Atributos: private String usuario private String clave private String privilegio private String nombre private String fecha StringTokenizer str_token private int cont private int result
Métodos: public String getCurrentJavaSqlDate1( ) public void carga_campos(StringTokenizer str_token) public void carga_campos_e(StringTokenizer str_token) public String retorna_user(StringTokenizer str_token) public String recupera_file(String path, String user) public String ver_hist_logon(String path, String user) public int lee_archivo(String user) public int valida_usuario(String user) public void elimina_registro(String user)

Tabla 3-15– Estructura de clase Eliminar_logoneados_v2

3.2.11. Encriptador

Esta clase se encarga de transformar toda la información que se encuentra en los archivos que se están utilizando, de esta forma se está protegiendo la misma en caso de que dichos archivos caigan en malas manos. A continuación se muestra su estructura.

ENCRIPTADOR
Atributos:
Métodos: public static String encriptar(String ps_palabra) public static String decriptar(String ps_palabra)

Tabla 3-16– Estructura de clase Encriptador

3.2.12. Verificar_archivos_basicos

Esta clase se encarga de mostrar cómo están configurados los archivos ***hosts, resolv.conf y sysctl.conf***. A continuación se muestra su estructura.

VERIFICAR_ARCHIVOS_BASICOS
Atributos: StringTokenizer str_token private int cont
Métodos: public void lee_archivo_area(String FileSource, String Path) public void control_archivos(String name_arch)

Tabla 3-17– Estructura de clase Verificar_archivos_basicos

3.2.13. Configura_basica

Esta clase se encarga de configurar de acuerdo al criterio del usuario los archivos ***hosts, resolv.conf y sysctl.conf***. A continuación se muestra su estructura.

CONFIGURA_BASICA
Atributos:
Métodos: public void escritura_archivo(String ip1, String ip2, String ip3, String ip4, String localhost, String localdomain) public void escritura_name_server(String ip_name_server1, String ip_name_server2, String ip_name_server3, String ip_name_server4) public void escritura_sysctl( )

Tabla 3-18– Estructura de clase Configura_basica

3.2.14. Configura_host_pass_ripd

Esta clase se encarga de cambiar la configuración del archivo ***ripd.conf*** que será el que determinará el ruteo dinámico. A continuación se muestra su estructura.

CONFIGURA_HOST_PASS_RIPD
Atributos: StringTokenizer str_token private int cont
Métodos: public void escritura_archivo_ripd(String linea) public void lee_archivo(String Path, String hostname_router, String pass, String ena_pass)

Tabla 3-19– Estructura de clase Configura_host_pass_ripd

3.2.15. Procesa_Arch_tabla

Esta clase se encarga de la creación y ejecución de archivos shell que tendrán consigo diferentes comandos utilizados para definir las tablas de ruteo. A continuación se muestra su estructura.

PROCESA_ARCH_TABLA
Atributos: private StringTokenizer str_token private Ruta_Tabla Tabla
Métodos: public void crea_archivo_comando(String nombre_archi, String pv_linea) public int borra_ruta(String Destination, String Genmask, String Gateway, String Interfaz) public int ejecuta_comando(String comando) public void actualiza_archi_tabla( ) public Vector getTablas( ) public int agrega_ruta(String Destination, String Genmask, String Gateway, String Interfaz)

Tabla 3-20– Estructura de clase Procesa_arch_tabla

3.2.16. Mascara

Esta clase se encarga de realizar conversiones de las máscaras que tengan las interfaces de red que se estén utilizando. Por ejemplo si la máscara de la dirección IP 192.168.1.2 está con el formato 192.168.1.2 / 24, la máscara resultante será 255.255.255.0.

La clase también podrá realizar la conversión del formato 255.255.255.0 a / 24. A continuación se muestra su estructura.

MASCARA
Atributos:
Métodos: public String convierte_mask(String ps_mask) public Vector getmaskvalidas( )

Tabla 3-21– Estructura de clase Mascara

3.2.17. Lista_iface

Esta clase se encarga de la creación de archivos con la información de todas las interfaces de red, además de poder visualizar las mismas. A continuación se muestra su estructura.

LISTA_IFACE
Atributos: private StringTokenizer str_token private StringTokenizer str_token2 private String id_interface private String interfaz private lista_iface lv_iface
Métodos: public lista_iface(StringTokenizer str_token) public lista_iface( ) public String getid_interface( ) public String getinterfaz( ) public void act_archi_lista_interfaces( ) public Vector getlista_iface( )

Tabla 3-22– Estructura de clase Mascara

3.2.18. Lista_eth

Esta clase se encarga de configurar las tarjetas red que el router tenga en ese momento, además de crear un archivo con la información de esas tarjetas de red activas. A continuación se muestra su estructura.

LISTA_ETH
Atributos: private StringTokenizer str_token private StringTokenizer str_token2 private String id_interface private String interfaz private lista_eth lv_iface
Métodos: public lista_eth(StringTokenizer str_token) public lista_eth( ) public String getinterfaz( ) public void act_archi_lista_interfaces( ) public int cambia_ruta(String pv_interfaz, String pv_direccion, String pv_mask) public Vector getlista_iface( ) public void crea_archivos_network(String pv_interfaz, String pv_direccion, String pv_mask) public String getnetwork_if(String pv_iface) public void crea_archi_masivos_if_up( ) public Vector getlista_iface_up( ) public void act_archi_lista_interfaces_up( ) public void crea_archivos_interfaz(String pv_interfaz)

Tabla 3-23– Estructura de clase Lista_eth

3.2.19. Informacion_interface

Esta clase procesa y visualiza los archivos que fueron creados en la clase Lista_eth. A continuación se muestra su estructura.

INFORMACION_INTERFACE
Atributos: StringTokenizer str_token StringTokenizer str_token1 private String result private String trash private String ip private String bcast private String mask
Métodos: public String getCurrentJavaSqlDate1( ) public String retorna_trash(StringTokenizer str_token) public String retorna_ip(StringTokenizer str_token) public String retorna_bcast(StringTokenizer str_token) public String retorna_mask(StringTokenizer str_token) public String result(StringTokenizer str_token1) public String Ip(String Path) public String Bcast(String Path) public String Mask(String Path)

Tabla 3-24– Estructura de clase Información_interface

3.2.20. Ruta_Tabla

Esta clase se encarga de formar una estructura de datos en la cual se procederá a almacenar la información correspondiente a la tabla de rutas. A continuación se muestra su estructura.

RUTA_TABLA
Atributos: private String Destination private String Gateway private String Genmask private String Flags private String Metric private String Ref private String Use private String face
Métodos: public Ruta_Tabla(StringTokenizer str_token) public Ruta_Tabla() public String getDestination() public String getGateway() public String getGenmask() public String getFlags() public String getMetric() public String getRef() public String getUse() public String getface()

Tabla 3-25– Estructura de clase Ruta_tabla

3.2.21. Config_rip

Esta clase se encarga de configurar los archivos que hacen posible el ruteo de las redes, es decir, el zebra.conf y el ripd.conf. Además levanta los servicios del zebra (puerto 2601) y del ripd (puerto 2602). A continuación se muestra su estructura.

CONFIG_RIP
Atributos: private String eth private String ip_direccion private String ip_mask private String network private StringTokenizer str_token
Métodos: public config_rip(String pv_interfaz, String pv_direccion, String pv_mask, String pv_network) public config_rip(StringTokenizer s) public String geteth( ) public String getip_direccion( ) public String getip_mask( ) public String getip_network( ) public Vector get_ifconfig_all_up( ) public void crea_all_datos_if( ) public Vector visualiza_datos_interfaces( ) public int arma_zebra_rip(String pv_hostname, String pv_password, String pv_ena_password, String pv_neigboar) public int arma_zebra_ospf(String pv_hostname, String pv_password, String pv_ena_password, String pv_neigboar) public String recupera_hostname(String Path) public String recupera_password(String Path) public String recupera_enable_pass(String Path) public void crea_archi_tabla_zebra( )

Tabla 3-26– Estructura de clase Config_rip

3.3. Diseño de Pantallas

A continuación se presenta un resumen de las principales pantallas con las que los usuarios podrán interactuar con el Router.

3.3.1. Pantalla Inicial

Esta pantalla se cargará una vez que se ingrese en el browser del navegador la dirección IP, el puerto de acceso y el nombre del proyecto correspondientes al servidor de la aplicación.

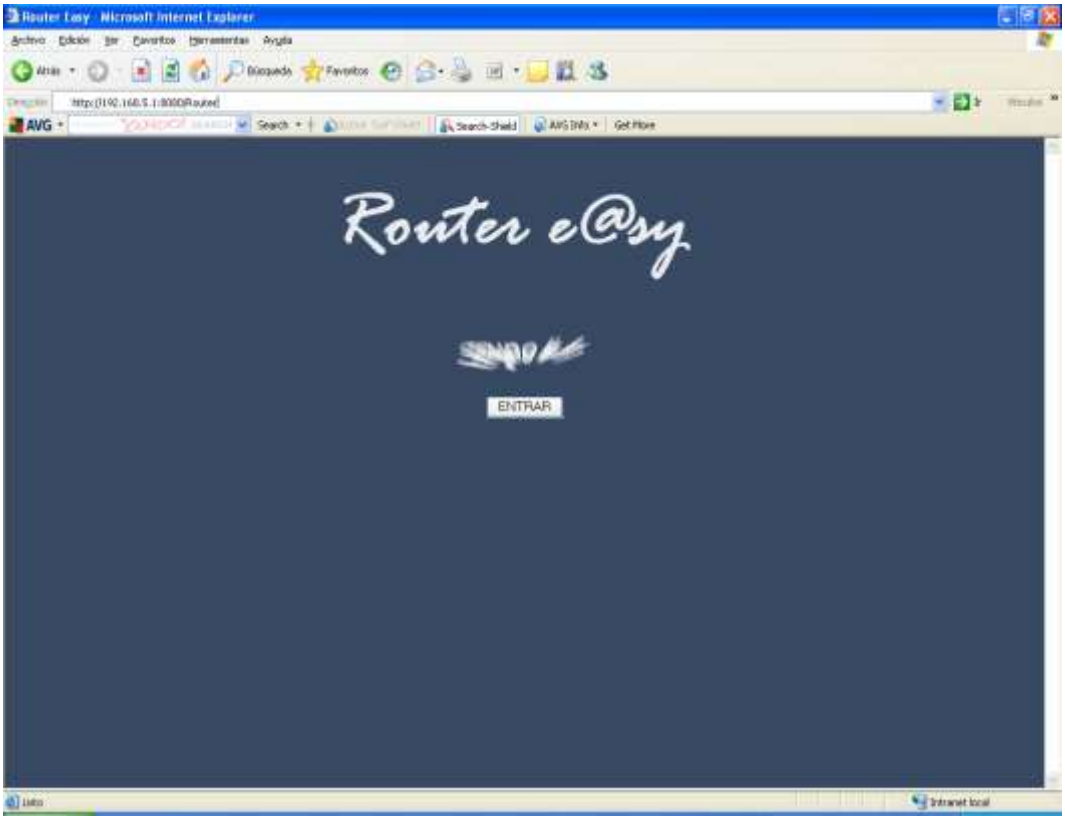

Ilustración 3-1 Pantalla Inicial

3.3.2. Pantalla de Inicio de Sesión

Esta pantalla se cargará cuando se presione el botón **ENTRAR** que se encuentra en la pantalla inicial.

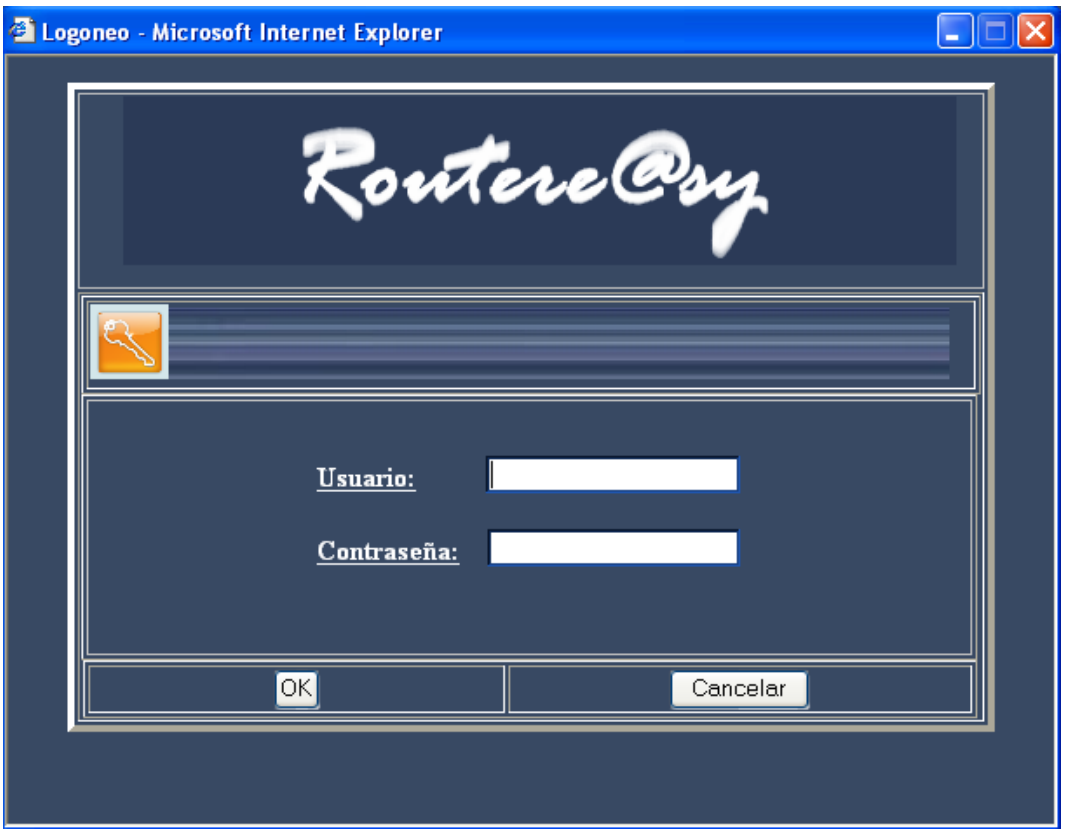

Ilustración 3-2 Pantalla de Inicio de Sesión

3.3.3. Pantalla Principal del Usuario Administrador

Esta pantalla se cargará cuando el usuario Administrador se haya logoneado. Esta traerá consigo varias opciones con las cuales el Administrador tendrá el control total de la aplicación Router Easy.

Ilustración 3-3 Pantalla Principal del Administrador

3.3.4. Opción Usuarios

Esta opción es utilizada para realizar el respectivo mantenimiento de los usuarios que utilizan la aplicación.

Ilustración 3-4 Pantalla de Opción Usuarios

3.3.5. Opción Bitácora de Control

Esta opción es utilizada para monitorear las actividades de los usuarios que utilizan la aplicación.

Ilustración 3-5 Pantalla de Opción Bitácora de Control

3.3.6. Opción Configuración Básica

Esta opción es utilizada para realizar configuraciones básicas tales como definir la dirección IP de la tarjeta de red con su respectiva máscara de subred.

Ilustración 3-6 Pantalla de Opción Configuración Básica

3.3.7. Opción Router Estático

Esta opción permite agregar o quitar manualmente rutas de comunicación entre redes de la tabla de ruteo.

Ilustración 3-7 Pantalla de Opción Router Estático

3.3.8. Opción Router Dinámico – Rip

Esta opción permite encontrar mediante el protocolo de ruteo **RIP** cuál es la ruta óptima para la transmisión de paquetes de datos desde una red X hacia a una red Y.

Ilustración 3-8 Pantalla de Opción Router Dinámico - RIP

3.3.9. Opción Router Dinámico – Ospf

Esta opción permite encontrar mediante el protocolo de ruteo **OSPF** cuáles la ruta óptima para la transmisión de paquetes de datos desde una red A hacia a una red B.

Ilustración 3-9 Pantalla de Opción Router Dinámico - OSPF

3.3.10. Pantalla Principal del Usuario Operador

Esta pantalla se cargará cuando el usuario Operador se haya logoneado. Esta traerá consigo solamente opciones de consulta.

Ilustración 3-10 Pantalla Principal del Operador

3.3.11. Opción Usuarios

Esta opción permite que el usuario Operador pueda cambiar su contraseña de ingreso, además de poder solicitar algún requerimiento al usuario Administrador.

Ilustración 3-11 Pantalla de Opción Usuarios Operador

3.3.12. Opción Consultar Tablas de Ruteo

Esta opción permite que el usuario Operador pueda consultar cuáles son las tablas de ruteo estáticas y dinámicas que el usuario Administrador ha configurado.

Ilustración 3-12 Pantalla de Opción Consultar Tablas de Ruteo

CAPITULO 4

4. DESARROLLO DEL SOFTWARE

4.1. Adaptación

Para hacer posible que una computadora que trabaja con cualquier distribución del sistema operativo Linux se comporte como un Router fue necesario la instalación en dicho sistema de 3 paquetes con los cuales se generaron los archivos de configuración de los ruteos estáticos y dinámicos.

Estos paquetes cuyo nombre común es **Quagga** vienen en todas las distribuciones de Linux y es necesaria la instalación de los 3 para el funcionamiento de lo deseado. A continuación se muestra específicamente el nombre de los paquetes:

- ✓ **quagga-0.98.3-2.i386.rpm**
- ✓ **quagga-contrib-0.98.3-2.i386.rpm**
- ✓ **quagga-devel-0.98.3-2.i386.rpm**

Para el desarrollo del proyecto se escogió trabajar con la distribución **Centos 5** debido a que está destinada para el uso de equipos servidores, sin embargo, cabe indicar que los paquetes mencionados anteriormente se obtuvieron descargándolos del sitio Web oficial de Quagga, ya que los paquetes Quagga que vienen en la distribución Centos 5 traen consigo ciertos errores funcionales que imposibilitó la tarea de efectuar la simulación de un Router.

A continuación se explicará brevemente qué es y cómo funciona Quagga.

Quagga es un paquete de software de encaminamiento que proporciona comunicación basada en servicios de TCP/IP con protocolos de encaminamiento que soportan RIPv1, RIPv2, RIPng, OSPF, BGP-4 y BGP-4+, de los cuales solo se utilizarán RIPv2 y OSPF para efectuar el ruteo dinámico.

Un sistema operativo con Quagga instalado actúa como un Router dedicado. Con Quagga una máquina intercambia información de ruteo con otros Routers utilizando protocolos de ruteo. Quagga utiliza esa información para actualizar el núcleo de las tablas de ruteo de forma que la información correcta esté en el lugar correcto. Además Quagga permite la configuración dinámica y es posible ver la información de la tabla de ruteo desde el interfaz de terminal de Quagga.

Una vez instalados los paquetes, se realizaron las configuraciones pertinentes a los archivos que permitirían realizar ambos tipos de ruteo (Estático y Dinámico).

A continuación se presentan los archivos de configuración que permiten que una PC se comporte como un Router:

- ✓ **zebra.conf**
- ✓ **ripd.conf**
- ✓ **ospf.conf**

4.1.1. Contenido de zebra.conf

```
!
! Zebra configuration saved from vty
!   2008/09/01 17:16:25
!
hostname zebrita
password zebra
enable password zebra
log file /var/log/quagga/zebra.log
!
interface lo
!
interface eth0
 ip address 192.168.1.1/24
 ipv6 nd suppress-ra
!
interface eth1
 ip address 192.168.4.1/24
 ipv6 nd suppress-ra
!
interface eth2
 ip address 192.168.3.2/24
 ipv6 nd suppress-ra
!
interface sit0
 ipv6 nd suppress-ra
!
line vty
!
```

En la parte de las interfaces **(interface eth0, interface eth1, interface eth2)** se coloca a continuación de **ip address** la dirección correspondiente de esa interface de red de la computadora.

4.1.2. Contenido de ripd.conf

```
!
! Zebra configuration saved from vty
!   2008/09/01 17:16:25
!
hostname zebrita
password zebra
enable password zebra
log file /var/log/quagga/ripd.log
!
interface lo
!
interface eth0
!
interface eth1
!
interface eth2
!
router rip
 network 192.168.1.0/24
 network 192.168.4.0/24
 network 192.168.3.0/24
!
line vty
!
```

En la parte que indica **network** y de acuerdo a la interfaces especificadas en el archivo anterior se coloca las direcciones de las redes con su respectiva máscara, con las cuales se entablará la comunicación.

4.1.3. Contenido de ospf.conf

```
!
! Zebra configuration saved from vty
!   2008/09/29 20:57:31
!
hostname zebra1
password zebra
enable password zebra
log file /var/log/quagga/ospfd.log
log stdout
!
!
!
interface eth0
!
interface eth1
!
interface eth2
!
interface lo
!
interface sit0
!
router ospf
 network 0.0.0.0/0 area 0
!
line vty
!
```

Se coloca la nomenclatura que corresponde a las interfaces de red del computador **(eth0, eth1, eth2).**

La ubicación de estos archivos en la distribución de Linux que se utilizó es: ***/ etc / quagga.***

4.2. Integración de Recursos

Una vez configurados correctamente los archivos mencionados, se procedió a construir el esquema de trabajo con cual se demostraría el funcionamiento de este proyecto.

Para llevar esto a cabo fue necesaria la integración de los equipos que se mencionan en el Capítulo 1 de este Tomo, pero que, para efectos de especificarlos de mejor manera se muestran a continuación.

- ✓ 7 Computadores Personales, de los cuales 3 serán los encargados de hacer las veces de Ruteadores.
- ✓ 2 Computadores Portátiles.
- ✓ 1 Switch.
- ✓ 12 Cables de red.

Se tomó la decisión de utilizar 3 PC como Router para poder demostrar que el esquema que será presentado será capaz de encontrar cuáles la ruta óptima de comunicación, es decir, será capaz de rutear o comunicar dinámicamente las redes que se encuentren interconectadas a estos Ruteadores. Cada uno de estos PC tendrá instalado 3 interfaces de red.

4.3. Instalación del Software

Para poder controlar los Ruteadores con la aplicación Web que se construyó fue necesario la instalación de la misma en las 3 PC.

Para esto, se siguieron los siguientes pasos:

- ✓ Se creó la carpeta java dentro de la carpeta usr.

- ✓ Luego en dicha carpeta se copió el archivo jdk-1_5_0_16-linux-i586-rpm.bin, al cual se le dieron permisos de ejecución. A continuación se muestra el comando: ***chmod +x jdk-1_5_0_16-linux-i586-rpm.bin.***

- ✓ Después se ejecutó el mismo y se generó un archivo rpm. Se muestra la ejecución: ***./jdk-1_5_0_16-linux-i586-rpm.bin.***

- ✓ Luego se instaló en el sistema el paquete rpm que se generó. Se muestra el comando: ***rpm Uvh ./jdk-1_5_0_16-linux-i586.rpm.***

- ✓ Este programa se grabó dentro del directorio ***/usr/java/jdk1.5.0_16.***

- ✓ Luego se copió el archivo apache-tomcat-5.5.27.tar.gz dentro del directorio /usr/java, y se procedió a descomprimirlo con el siguiente comando: ***tar xzf apache-tomcat-5.5.27.tar.gz.***

- ✓ Luego, dentro de la carpeta java se creó la carpeta tomcat5 y dentro de ésta se agregó el archivo recientemente descomprimido. Se muestra el comando: ***mv apache-tomcat-5.5.27 tomcat5***.

- ✓ Después editamos el archivo Profile que se encuentra dentro de la carpeta /etc. A este archivo se le agregaron las siguientes líneas al final del mismo, a continuación de la palabra **done**:
    - JAVA_HOME=/usr/java/jdk1.5.0_16
    - PATH=$PATH:$JAVA_HOME/bin
    - CATALINA_HOME=/usr/java/tomcat5
    - export JAVA_HOME CATALINA_HOME PATH

- ✓ Luego de esto, se ejecutó el archivo Profile por medio del comando: ***source profile.***

- ✓ Se copió el contenido de la carpeta WebRoot de nuestro proyecto y se lo pegó en el directorio /usr/java/tomcat5/webapps/Router.

- ✓ Finalmente, se arrancó el servidor de aplicaciones Web (tomcat5) entrando en el directorio /usr/java/tomcat5/bin y ejecutando el comando: ***sh startup.sh.***

4.4. Modelo Funcional

Para demostrar de una manera clara y concisa que el Router es capaz de manejar las tablas de ruteo estática y dinámicamente, se presenta el siguiente Esquema Funcional.

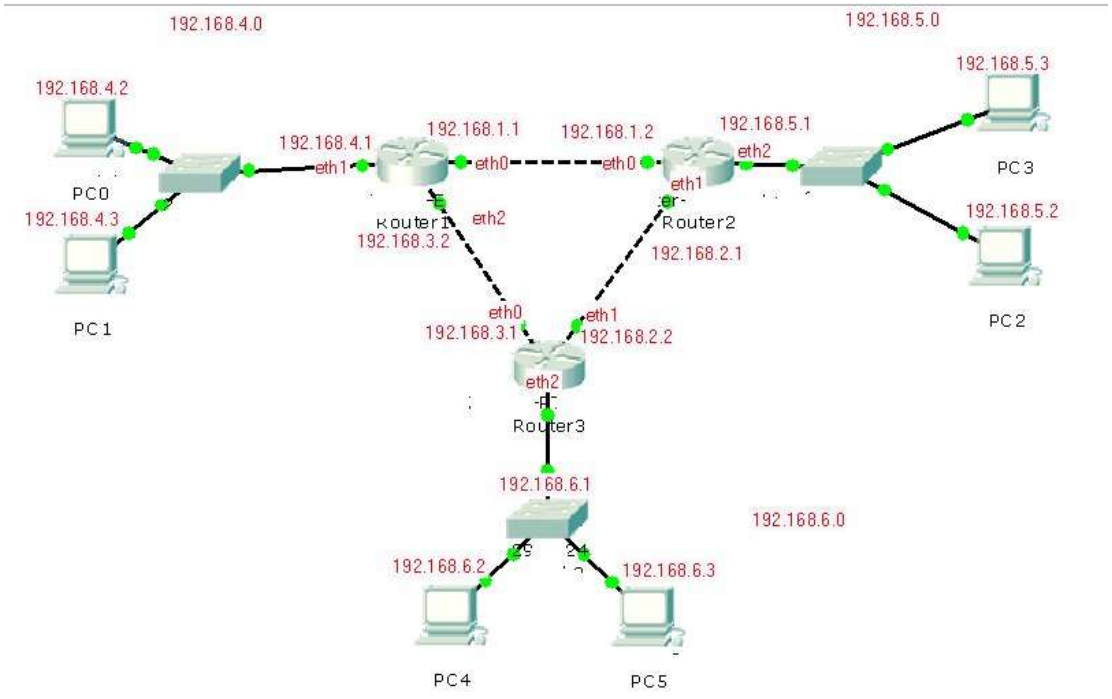

Ilustración 4-1 Esquema Demostrativo del Funcionamiento

De esta forma las 3 redes que intervienen se encuentran interconectadas.

RED A → 192.168.4.0 (PC 0 – PC 1)

RED B → 192.168.5.0 (PC 2 – PC 3)

RED C → 192.168.6.0 (PC 4 – PC 5)

CAPÍTULO 5

5. CONCLUSIONES Y RECOMENDACIONES

5.1. Conclusiones

El entorno de los usuarios de las diferentes redes interconectadas es confiable y seguro.

La comunicación entre las redes participantes es total y permanente.

La aplicación tiene la capacidad de configurar el ruteo de paquetes solamente hacia las redes que se deseen.

En caso de que existan conexiones con nuevas redes, el Router tendrá la capacidad de actualizar las tablas de ruteo, agregando en ellas las nuevas rutas correspondientes a esa nueva red.

Este proyecto constituye una valiosa alternativa para incursionar en el mundo de las herramientas Open Source y así ahorrar el gasto que genera la adquisición de dispositivos afines.

Se consiguió con éxito el cumplimiento de lo que se requería en este proyecto, es decir, se logró la configuración de las tablas de ruteo estáticas y dinámicas del Router Linux (Administración del Router) por medio de la aplicación Web Router Easy.

5.2. Recomendaciones

Cuando se quiera instalar los paquetes que hacen posible el funcionamiento del Router (quagga), visitar la página Web oficial de quagga http://www.quagga.net/download.php y descargarlos luego de seleccionar la distribución Linux con la que se esté trabajando.

Realizar la capacitación correspondiente a los usuarios Administradores de las redes que gestionarán las actividades del Router por medio de la aplicación (Consultar el Manual de Usuario).

Mantener la computadora Router (Servidor) siempre en actividad para salvaguardar la información de las tablas de ruteo. Para esto el Servidor deberá estar debidamente protegido por medio de reguladores de voltaje y por lo menos 2 UPS.

Para que la instalación del proyecto sea apropiada, revisar el Capítulo 4 (Desarrollo del Software) en el cual se detalla como se implantaron los objetos en el sistema operativo Centos 5.

GLOSARIO DE TERMINOS

Apache Tomcat 5:

Software utilizado como servidor de aplicaciones Web

Centos 5:

Distribución del Sistema Operativo Linux.

Hackers:

Persona desagradable cuya principal actividad es causar daños y perjuicios a las redes empresariales hurtando información confidencial o ingresando en las mismas algún malware (software dañino – virus informático).

Java:

Lenguaje de programación de objetos.

JSP:

Servidor de páginas Web del lenguaje de programación Java.

MyEclipse:

Herramienta de creación y edición de código Java y JSP.

OSPF:

Protocolo de enrutamiento cuya principal función es la de calcular la ruta mas corta que exista para el envío de paquetes de datos.

Quagga:

Software de encaminamiento que proporciona comunicación basada en servicios de TCP/IP.

RIP:

Protocolo de encaminamiento de información.

Router:

Dispositivo que se encarga de interconectar varias redes.

BIBLIOGRAFÍA

Angie Nash and Jason Nash, LPIC 1 Certification Bible, Hungry Minds, .

Indianapolis, IN ✦ Cleveland, OH ✦ New York, NY, 2001.

Bert Hubert, Enrutamiento avanzado y control de tráfico en Linux, Thomas Graf (Section Author) 2003/12/25.

Kunihiro Ishiguro, A routing software package for TCP/IP networks, Quagga, July 2006.

Richard Petersen, The Complete Reference Sixth Edition, New York Chicago San Francisco, 2008

Schroder Carla, Linux Networking Cookbook™, Printed in the United States of America, O'Reilly Media 2008.

Terry Collings & Kurt Wall, Red Hat® Linux® Networking and System Administration Hungry Minds, New York, NY 10022.

http://bulma.net/

http://www.ibm.com/developerworks/opensource/

http://www.investigacion.frc.utn.edu.ar/labsis/index.html

http://www.linuxdoc.org/

http://www.linuxparatodos.com

http://www.quagga.net/docs.php

http://www.quagga.net/download.php

U N I V E R S I D A D D E G U A Y A Q U I L

F a c u l t a d d e C i e n c i a s M a t e m á t i c a s y F í s i c a s

C a r r e r a d e I n g e n i e r í a e n S i s t e m a s C o m p u t a c i o n a l e s

“Implementación de Linux como Router”

TESIS DE GRADO

Previo a la Obtención del Título de:

INGENIERO EN SISTEMAS COMPUTACIONALES

Autores:

Boris Alex López Macías

Jorge Giovanny Sánchez De La Torre

José Daniel Sotomayor Navarro

GUAYAQUIL-ECUADOR

Año: 2008

A G R A D E C IM IE N T O

A D ios ante todo, ya que él con su infinito amor nos doto de sabiduría para poder tomar las mejores decisiones e ir por el sendero del bien.

A nuestros Padres, que con paciencia y dedicación, nos dieron su apoyo, consejos, y parte de sus vidas para que nosotros sigamos adelante y alcancemos nuestras metas.

A nuestros profesores, que nos transmitieron sus vastos conocimientos y experiencias adquiridas a lo largo de su carrera profesional.

A nuestros amigos y seres queridos que siempre cuando nos sentíamos vencidos, nos daban aliento para seguir adelante. Gracias por su apoyo incondicional.

D E D IC A T O R IA

Nuestra tesis la dedicamos con amor y cariño, a Dios que nos diste la oportunidad de vivir y de regalarnos unas familias maravillosas, a nuestros Padres quienes han estado en cada momento a nuestro lado, creyeron en nosotros dándonos preparación hasta lograr vernos convertidos en auténticos profesionales. A nuestros seres queridos que supieron confiar en nosotros animándonos a que siguiéramos nuestros sueños.

TRIBUNAL DE GRADUACIÓN

Presidente	1er Vocal
2do Vocal	Vocal Secundario

D E C L A R A C I Ó N E X P R E S A

"La autoría de la tesis de grado corresponde exclusivamente al suscrito(s), perteneciendo a la Universidad de Guayaquil los derechos que generen la aplicación de la misma"

(Reglamento de Graduación de la Carrera de Ingeniería en Sistemas Computacionales, Art. 26)

Boris Alex López Macías

blopez@corlasosa.com

Jorge Giovanny Sánchez De La Torre

jsanchez@corlasosa.com

José Daniel Sotomayor Navarro

jose.sotomayor@sasf.net

INDICE GENERAL

AGRADECIMIENTO	I
DEDICATORIA	II
TRIBUNAL DE GRADUACIÓN	III
DECLARACIÓN EXPRESA	IV
INDICE GENERAL	V

TABLA DE CONTENIDOS

CAPÍTULO 1 1

1 MANUAL TECNICO 1

1.1 Objetivo 1

1.2 Descripción de Procesos del Sistema 1

1.2.1 Métodos de la Clase Creación_user_admin 2

1.2.2 Métodos de la Clase Logon_v2 5

1.2.3 Métodos de la Clase Confirma_rol 8

1.2.4 Métodos de la Clase Crear_usuarios 9

1.2.5 Métodos de la Clase Cambio_clave_v2 12

1.2.6 Métodos de la Clase Eliminar_usuarios 13

1.2.7 Métodos de la Clase Crear_bitacora 15

1.2.8 Métodos de la Clase Responder_solicitud 18

1.2.9 Métodos de la Clase Verificar_archivos_bitacora_control 19

1.2.10 Métodos de la Clase Eliminar_logoneados_v2 24

1.2.11 Métodos de la Clase Encriptador 26

1.2.12 Métodos de la Clase Configura_host_pass_ripd 27

1.2.13 Métodos de la Clase Procesa_Arch_tabla 28

1.2.14 Métodos de la Clase Mascara 36

1.2.15 Métodos de la Clase list_eth 39

CAPÍTULO 2 48

2 MANUAL DE USUARIO 48

2.1 INICIO DE SESION DE ADMINISTRADOR 54

2.1.1 Bienvenido 56

2.1.2 Usuarios 57

2.1.3 Bitácora de Control 70

2.1.4 Configuración Básica 73

2.1.5 Router Estático 76

2.1.6 Router Dinámico – Rip 84

2.1.7 Router Dinámico – Ospf 91

2.1.8 Cerrar Aplicación 100

2.2 INICIO DE SESION DE OPERADOR 101

2.2.1 Usuarios 102

2.2.2 Consulta de Tablas de Ruteo 109

INDICE DE FIGURAS

Ilustración 2-1 Pantalla de Inicio de la aplicación 48

Ilustración 2-2 Pantalla de Inicio de la aplicación 49

Ilustración 2-3 Alerta de ingreso de Usuario 50

Ilustración 2-4 Alerta de ingreso de Password 51

Ilustración 2-5 Alerta de ingreso correcto de Usuario y Password 52

Ilustración 2-6 Alerta de haber sobrepasado el número de intentos 53

Ilustración 2-7 Alerta de cerrado de aplicación 54

Ilustración 2-8 Pantalla de Bienvenida 55

Ilustración 2-9 Pantalla de Breve Introducción 56

Ilustración 2-10 Menú de Usuarios 57

Ilustración 2-11 Crear Usuarios 58

Ilustración 2-12 Alerta de ingreso de toda la información 59

Ilustración 2-13 Ingreso de Usuario 60

Ilustración 2-14 Alerta de Clave y confirmación no coinciden 61

Ilustración 2-15 Alerta de usuario creado exitosamente 62

Ilustración 2-16 Cambiar clave 63

Ilustración 2-17 Cambiar clave 64

Ilustración 2-18 Confirmación de clave cambiada 65

Ilustración 2-19 Pantalla Principal de eliminar usuario 66

Ilustración 2-20 Alerta ingrese usuario a eliminar 67

Ilustración 2-21 Alerta no puede eliminar usuario inexistente 68

Ilustración 2-22 Eliminar Usuarios 69

Ilustración 2-23 Alerta de Usuario eliminado exitosamente 69

Ilustración 2-24 Consulta de Usuarios 70

Ilustración 2-25 Usuarios Logoneados 71

Ilustración 2-26 Historial de Sesiones 72

Ilustración 2-27 Peticiones de Usuarios 73

Ilustración 2-28 Configuración de Tarjeta de Red 74

Ilustración 2-29 Alerta de ingreso completo de Red 75

Ilustración 2-30 Información de Tarjeta de red correcta 75

Ilustración 2-31 Alerta de configuración de red exitosa 76
Ilustración 2-32 Tabla de Ruteo 77
Ilustración 2-33 Adicionar Nueva ruta 78
Ilustración 2-34 Confirmación de Ruta 79
Ilustración 2-35 Alerta de Configuración exitosa 80
Ilustración 2-36 Tabla de Ruteo con ruta nueva 81
Ilustración 2-37 Borrando Ruta 82
Ilustración 2-38 Alerta de Ruta borrada exitosamente 83
Ilustración 2-39 Ruta borrada 83
Ilustración 2-40 Subir servicios Ripd y Zebra 84
Ilustración 2-41 Bajar servicios Ripd y Zebra 85
Ilustración 2-42 Alerta de ingreso de toda la configuración 86
Ilustración 2-43 Alerta tienen que coincidir password y su confirmación 86
Ilustración 2-44 Alerta no coinciden enable password y su confirmación 87
Ilustración 2-45 Configuración Completa: host, pass y enable cambiados 87
Ilustración 2-46 Pantalla ingreso Host, pass, enable, interfaces y neighbor 88
Ilustración 2-47 Actualización de los archivos Rip y Zebra 89
Ilustración 2-48 Consulta archivo ripd 90
Ilustración 2-49 Consulta archivo zebra 90
Ilustración 2-50 Consulta Tabla de Ruteo 91
Ilustración 2-51 Subir servicios Ospfd y Zebra 92
Ilustración 2-52 Subir servicios Ospfd y Zebra 93
Ilustración 2-53 Alerta de ingrese toda la información 94
Ilustración 2-54 Alerta password y su confirmación deben de coincidir 95
Ilustración 2-55 Ingreso de la configuración 95
Ilustración 2-56 Alerta de configuración completa exitosa 96
Ilustración 2-57 Configuración de los archivos ospfd y zebra 97
Ilustración 2-58 Alerta de actualización correcta en los archivos 98
Ilustración 2-59 Visualización del archivo ospfd 99
Ilustración 2-60 Visualización del archivo zebra 99
Ilustración 2-61 Tabla de Ruteo Dinámico – Ospf 100
Ilustración 2-62 Pantalla de Inicio de Operador 101
Ilustración 2-63 Cambio de clave Operador 102
Ilustración 2-64 Alerta de ingreso de información 103
Ilustración 2-65 Alerta de clave anterior no corresponde al usuario 104
Ilustración 2-66 Alerta nueva clave y su confirmación no coinciden 105
Ilustración 2-67 Alerta de cambio de clave exitosa 106

Ilustración 2-68 Ingreso de Requerimiento 107
Ilustración 2-69 Alerta de ingreso de requerimiento 108
Ilustración 2-70 Petición enviada exitosamente 108
Ilustración 2-71 Respuesta de Peticiones 109
Ilustración 2-72 Tabla de Ruteo Estático 110
Ilustración 2-73 Tabla de Ruteo Dinámico 111

C A P Í T U L O 1

1. M A N U A L T E C N I C O

1.1 O b j e t i v o

Este capítulo tiene como finalidad orientar o dar a conocer al lector la forma en que se desarrollaron los principales procesos de esta aplicación, además de especificar cuáles son las funciones que realizan o en base a qué criterios fueron creados, es decir, se mostrará una descripción detallada del por qué de su creación.

1.2 D e s c r i p c i ó n d e P r o c e s o s d e l S i s t e m a

A continuación se presenta la descripción de los principales métodos o procesos de las clases utilizadas para el desarrollo de la aplicación.

1.2.1 Métodos de la Clase Creación_user_admin

getCurrentJavaSqlDate1( ).- Este método fue creado para obtener la fecha y hora del sistema.

```
public String    getCurrentJavaSqlDate1() {
        // TODO Auto-generated method stub
        java.util.Date dt = new java.util.Date();
        java.text.SimpleDateFormat sdf = new
java.text.SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
        String fecha=sdf.format(dt);
        System.out.println(dt);
        System.out.println(sdf.format(dt));
        return fecha;
}
```

carga_campos(StringTokenizer str_token).- Este método es utilizado para definir puntos clave en un determinado archivo de texto y de acuerdo a ello poder extraer algún carácter o palabra del mismo.

```
public void carga_campos (StringTokenizer str_token)
        {
                                usuario     = str_token.nextToken();
                                System.out.println(usuario);
                                clave     = str_token.nextToken();
                                System.out.println(clave);
                                privilegio = str_token.nextToken();
                                System.out.println(privilegio);
                                nombre = str_token.nextToken();
                                System.out.println(nombre);
                                fecha       = str_token.nextToken();
                                System.out.println(fecha);


        }
```

lee_archivo( ).- Este método es utilizado para leer línea por línea el archivo donde se encuentran registrados todos los usuarios autorizados para manejar la aplicación.

```
public void lee_archivo () throws Exception, SQLException
                                        {
            String FileSource = "user" ;
            String Path = "//etc//usuarios//"+FileSource;
            String linea=null;
            File f = new File(Path);

            if (!f.exists())
            {
                    System.out.print("Archivo no existe");
                    crea_usuario();
                    //return;
             }else{
                    //f.canRead();
                    FileReader fr=new FileReader(Path);
                    BufferedReader entrada=new BufferedReader(fr);
                    linea=entrada.readLine();
                    System.out.println(linea);

        if(linea!=null){
             linea= e.decriptar(linea);
             str_token       = new  StringTokenizer(linea,"|");
             carga_campos(str_token);
             if (this.usuario.equals("ADMIN")){
                    System.out.println("usuario es admin");
             }
        }
        entrada.close();

             }

    }
```

crea_usuario( ).- Este método se encarga de crear los archivos repositorios de la información de usuarios, es decir, crea los archivos *user*, *logoneados*, *hist_logon* y *bitácora*.

```
public void crea_usuario() throws SQLException, Exception
                {
                String variable1="";
                String mens;
                        String user="ADMIN";
                        String pass="ADMIN";
                        String time=null;
                        time=getCurrentJavaSqlDate1();
                java.util.Date fecha2=null;
                fecha2=getCurrentJavaSqlDate();
                System.out.println("\nString "+ fecha2);
                String privilegio="ADMINISTRADOR";
                String nombre="GRUPO4 - SEMINARIO LINUX";
                        String
variable=user+"|"+pass+"|"+privilegio+"|"+nombre+"|"+time+".";
                        FileWriter fstream = new
FileWriter("//etc//usuarios//user");
                        BufferedWriter out = new
BufferedWriter(fstream);

                        FileWriter fstream3 = new
FileWriter("//etc//usuarios//bitacora");
                        BufferedWriter out3 = new
BufferedWriter(fstream3);
                        FileWriter fstream4 = new
FileWriter("//etc//usuarios//hist_logon");
                        BufferedWriter out4 = new
BufferedWriter(fstream4);
                        variable1= e.encriptar(variable);
                        mens=" USUARIOS|LOGONEADOS|EN|ESTE|MOMENTO ";
                        mens=e.encriptar(mens);
                        out.write(variable1);
                        out.close();
                        FileWriter fstream2 = new
FileWriter("//etc//usuarios//logoneados");
                        BufferedWriter out2 = new
BufferedWriter(fstream2);
                        out2.write(mens);
                        out2.close();
                        System.out.println(variable);
                }
```

1.2.2 Métodos de la Clase Logon_v2

getCurrentJavaSqlDate1( ).- Este método fue creado para obtener la fecha y hora del sistema.

```
public String    getCurrentJavaSqlDate1() {
        // TODO Auto-generated method stub
        java.util.Date dt = new java.util.Date();
        java.text.SimpleDateFormat sdf = new
java.text.SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
        String fecha=sdf.format(dt);
        System.out.println(dt);
        System.out.println(sdf.format(dt));
        return fecha;
}
```

carga_campos(StringTokenizer str_token).- Este método es utilizado para definir puntos clave en un determinado archivo de texto y de acuerdo a ello poder extraer algún carácter o palabra del mismo.

```
public void carga_campos (StringTokenizer str_token)
        {
                        usuario     = str_token.nextToken();
                        System.out.println(usuario);
                        clave     = str_token.nextToken();
                        System.out.println(clave);
                        privilegio = str_token.nextToken();
                        System.out.println(privilegio);
                        nombre = str_token.nextToken();
                        System.out.println(nombre);
                        fecha       = str_token.nextToken();
                        System.out.println(fecha);

        }
```

lee_archivo(FileSource, Path, Usuario, Password).- Este método se encarga de validar que el usuario y la contraseña del visitante se encuentren registrados en el archivo de usuarios. Este proceso es utilizado en el momento de inicio de sesión de la aplicación.

```
public int lee_archivo (String FileSource,String Path, String
Usuario, String Password) throws Exception, SQLException
        {
                FileReader fr=new FileReader(Path);
                BufferedReader entrada=new BufferedReader(fr);
                String linea=null;
                linea=entrada.readLine();
                while(linea!=null)
                {
                        System.out.println(linea);
                        linea = e.decriptar(linea);
                        str_token       = new
StringTokenizer(linea,"|");
                        carga_campos(str_token);
                        result_usu=this.valida_usuario(Usuario,
Password);
                if(result_usu==1){
                        System.out.println("Bienvenido " + Usuario
+"...");
                      //break;
                        result_pass=this.valida_pass(Usuario,
Password);
                      if(result_pass==2){
                           if(result_usu==1 && result_pass==2){
                                System.out.println("Bienvenido " +
Usuario +"... Su password esta correcto");
                                 result_usu=3;
                                break;
                           }else{
                                System.out.println("Clave Incorrecta");
                           }
                      }
                }


                linea=entrada.readLine();
                        cont++;
                entrada.close();
           return result_usu;
        }
```

crea_Logoneados(Usuario, Path).- Este método se encarga de guardar a todos los usuarios que han iniciado sesión en los archivos *logoneados* y *hist_logon*.

```
public void crea_Logoneados(String Usuario,
                                String Path
                                ) throws SQLException, Exception
        {
                String archivo;
                String file;
                String existe;
                String path="//etc//usuarios//logoneados";
                int Logon_v2=0;
                Logon_v2 v2 = new Logon_v2();
                System.out.println("valida user 1");
                        existe=devuelve_user(Path, Usuario);
                        archivo = recupera_file(path);
                        existe=e.encriptar(existe);
                        System.out.println("var existe: "+existe);
                        System.out.println("Archivo Encriptado");
                        if (archivo=="nada"){
                                file=existe;
                                FileWriter fstream = new
FileWriter("//etc//usuarios//logoneados");
                                BufferedWriter out = new
BufferedWriter(fstream);
                                out.write(file);
                                out.close();
                                v2.crea_hist_logon(Usuario, Path);
                                System.out.println(file);
                        }else{
                                Logon_v2=lee_Logon_v2(Usuario, path);
                                if (Logon_v2==0){
                                        file = archivo+"\n"+existe;
                                        FileWriter fstream = new
FileWriter("//etc//usuarios//logoneados");
                                        BufferedWriter out = new
BufferedWriter(fstream);
                                        out.write(file);
                                        out.close();
                                        v2.crea_hist_logon(Usuario,
Path);
                                        System.out.println(file);
                                }
                        }

        }
```

1.2.3 Métodos de la Clase Confirma_rol

confirma_privilegio(user).- Este método es utilizado para determinar cual es el Rol del usuario que esté intentando ingresar a la aplicación.

```
public int confirma_privilegio (String user
                                    )throws Exception, SQLException
     {
               String linea=null;
               String usuario=null;
               String clave=null;
               String privilegio=null;
               int resultado=2;
               FileReader fr=new
FileReader("//etc//usuarios//user");
                     BufferedReader entrada=new
BufferedReader(fr);
                     linea=entrada.readLine();
                     while(linea!=null)
                     {
                             linea=e.decriptar(linea);
                             System.out.println(linea);
                             str_token       = new
StringTokenizer(linea,"|");
                          usuario=retorna_user(str_token);
                          clave=retorna_clave(str_token);
                          privilegio=retorna_privilegio(str_token);
                          result=valida_usuario(user);
                          if (result==1){
                                    if
(privilegio.equals("ADMINISTRADOR") ){
                                             resultado=1;
                                             linea=null;
                                    }else{
                                             resultado=0;
                                             linea=null;
                                    }
                          }
                          if (resultado==2){
                             linea=entrada.readLine();
                          }
                     }
                     entrada.close();
                     return resultado;
     }
```

1.2.4 Métodos de la Clase Crear_usuarios

lee_archivo(File,Path,user,pass,conf,privilegio,nombre).

- Este método es utilizado para realizar la validación de existencia del usuario antes de ser insertado en el archivo de usuarios.

```
public int lee_archivo (String FileSource,
                                        String Path,
                                        String user,
                                        String pass,
                                        String confirma_pass,
                                        String privilegio,
                                        String nombre) throws
Exception, SQLException
                {
                      String linea=null;
                      int resultado=0;
                      FileReader fr=new FileReader(Path);
                      BufferedReader entrada=new BufferedReader(fr);
                        linea=entrada.readLine();
                        while(linea!=null)
                        {
                                linea=e.decriptar(linea);

                                str_token       = new
StringTokenizer(linea,"|");
                                carga_campos(str_token);
                                result=valida_usuario(user);
                        if (result==1){
                                        resultado=1;

                                }
                      linea=entrada.readLine();
                      System.out.println(linea);
                                cont++;
                        }
                        entrada.close();
                        return resultado;
        }
```

recupera_file(File,Path,user,pass,conf,privilegio,nombre).- Este método se encarga de extraer o recuperar toda la información del archivo de usuarios.

```
public String recupera_file(String FileSource,String Path,
                                String user,
                                String pass,
                                String confirma_pass,
                                String privilegio,
                                String nombre) throws Exception,
SQLException                                                    {
                    String linea=null;
                    String File;
                    String File_i;
                    String File_r;
                    String linea_act=null;
                    String FileL;
                    String lineap=null;
                    FileReader f = new FileReader(Path);
                       BufferedReader entrada=new BufferedReader(f);
                       linea=entrada.readLine();
                       File=linea;
                       File_i=null;
                       File_r=null;
                       lineap=e.decriptar(linea);
                       FileL="";
                       while(linea!=null){
                           File_r = linea;
                           linea=e.decriptar(linea);
                           str_token       = new
StringTokenizer(linea,"|");
                           carga_campos(str_token);
                           if (lineap == linea){
                                  FileL=lineap;
                           }else{
                                  FileL=FileL+"\n"+linea;
                           }
                    linea=entrada.readLine();
                    if (linea==null){
                         File = File_i;
                         cont++;
                    }else{
                         System.out.println(linea);
                         linea_act = linea;
                         System.out.println(File);
                         File = File+"\n"+linea_act;
                         File_i = File;
                         cont++;}
                     } entrada.close();
                       if (cont==1){return File_r;
                       }else{    return File;    }
     }
```

crea_usuario(File,Path,user,pass,conf,privilegio,nombre).- Este método es el que se encarga de actualizar el archivo de usuarios insertando los nuevos registros.

```
public void crea_usuario(String user,String pass,
String confirma_pass,String privilegio,String FileSource,
String Path,String nombre) throws SQLException, Exception {
                    String archivo;
                    String file;
                    int confirma=0;
                    String time=null;
                    time=getCurrentJavaSqlDate1();
                    Crear_usuarios r= new Crear_usuarios();
               java.util.Date fecha2=null;
               confirma=valida_archivo_nulo(Path);
               if (confirma==1){
                    user="ADMIN";pass="ADMIN";
                    privilegio="ADMINISTRADOR";
                    nombre="GRUPO4 - SEMINARIO LINUX";
                    String
variable=user+"|"+pass+"|"+privilegio+"|"+nombre+"|"+time+".";
                         FileWriter fstream = new FileWriter(Path);
                         BufferedWriter out = new
BufferedWriter(fstream);
                       out.write(variable);    out.close();
                         FileWriter fstream1 = new
FileWriter("//etc//usuarios//solicitudes//"+user);
                         BufferedWriter out1 = new
BufferedWriter(fstream1);
                         FileWriter fstream2 = new
FileWriter("//etc//usuarios//solicitudes//"+user+"_mirror");
                         BufferedWriter out2 = new
BufferedWriter(fstream2);
               }else{
                    String
variable=user+"|"+pass+"|"+privilegio+"|"+nombre+"|"+time+".";
                    archivo = r.recupera_file(FileSource, Path,
user, pass, confirma_pass, privilegio, nombre);
                    variable=e.encriptar(variable);
                    file = archivo+"\n"+variable;
                    FileWriter fstream = new FileWriter(Path);
                    BufferedWriter out = new
BufferedWriter(fstream);
                    out.write(file);  out.close();
                    FileWriter fstream1 = new
FileWriter("//etc//usuarios//solicitudes//"+user);
                         BufferedWriter out1 = new
BufferedWriter(fstream1);
                         FileWriter fstream2 = new
FileWriter("//etc//usuarios//solicitudes//"+user+"_mirror");
                         BufferedWriter out2 = new
BufferedWriter(fstream2);   }     }
```

1.2.5 Métodos de la Clase Cambio_clave_v2

lee_archivo_logon(user,pass,new_pass,conf_new_pass).- Este método es utilizado para encontrar el registro del usuario que quiere cambiar su clave, una vez encontrada procede a cambiarla.

```
public String lee_archivo_logon (String user,String pass,
String new_pass,String confirma_new_pass) throws Exception,
SQLException {
      String linea=null;          String resultado=null;
      String usuario=null;        String clave=null;
      String privilegio=null;     String nombre=null;
      String new_clave=null;      String time=null;
             time=getCurrentJavaSqlDate1();
             FileReader fr=new
FileReader("//etc//usuarios//logoneados");
                    BufferedReader entrada=new
BufferedReader(fr);
                    linea=entrada.readLine();
                    while(linea!=null)
                    {    linea=e.decriptar(linea);
                       str_token       = new
StringTokenizer(linea,"|");
                         usuario=retorna_user(str_token);
                         clave=retorna_clave(str_token);
                         privilegio=retorna_privilegio(str_token);
                         nombre=retorna_nombre(str_token);
                         result=valida_usuario(user);
                         if (result==1){
                                  if
((pass.equals(clave))&&(new_pass.equals(confirma_new_pass))){
                                       new_clave=new_pass;

resultado=usuario+"|"+new_clave+"|"+privilegio+"|"+nombre+"|"+t
ime+".";
                                       linea=null;
                                  }else{

                                       resultado="clave
incorrecta";
                                       linea=null;
                                  }
                         }else{
                           resultado="user incorrecto";
                           linea=entrada.readLine();
                         }
                    }
                    entrada.close();
                    return resultado; }
```

graba_new_archivo(File,Path,user,pass,newp,conf_npass).- Este método se encarga de actualizar los archivos de usuarios registrados y usuarios logoneados en base al archivo o parámetro de retorno del método anterior.

```
public void graba_new_archivo(String FileSource,
                              String Path,
                              String user,
                              String pass,
                              String new_pass,
                              String confirma_new_pass
                             )throws SQLException, Exception
    {

    String file;
    Cambio_clave_v2 n = new Cambio_clave_v2();
    file = recupera_file(FileSource,Path,user,pass,new_pass,
confirma_new_pass);
    FileWriter fstream = new FileWriter(Path);
    BufferedWriter out = new BufferedWriter(fstream);
    out.write(file);
    out.close();
    n.actualiza_archivo_logon(FileSource, Path, user, pass);
    }
```

1.2.6 Métodos de la Clase Eliminar_usuarios

valida_usuario(user).- Este método se encarga de verificar la existencia de un usuario.

```
public int valida_usuario(String user){
            if(this.usuario.equals(user)){
                    System.out.println("usuario correcto");
                    return 1;
            }else{
                    System.out.println("usuario incorrecto");
                    return 0;
            }
      }
```

elimina_registro(user, Path).- Este método es utilizado para eliminar el registro de un usuario determinado.

```
public void elimina_registro (String user,
                              String Path) throws Exception,
SQLException
     {
               String linea=null;
               String LineaVale;
               String Linea_ant;
               String new_file;
               String linea_des=null;
               FileReader fr=new FileReader(Path);
               BufferedReader entrada=new BufferedReader(fr);
               linea=entrada.readLine();
               Linea_ant=linea;
               new_file=linea;
               while(linea!=null)
               {
                       linea_des=e.decriptar(linea);
                       str_token       = new
StringTokenizer(linea_des,"|");
                       carga_campos(str_token);
                       result=valida_usuario(user);

                       if (result==1){
                               LineaVale=linea;
                               if (LineaVale==Linea_ant){
                                       linea=entrada.readLine();
                                       new_file=linea;
                               }
                       }else{
                               LineaVale=linea;
                               if (LineaVale==Linea_ant){
                                       new_file=LineaVale;
                               }else{
                                    new_file=new_file + "\n" +
LineaVale;
                               }
                       }
                               linea=entrada.readLine();
                               cont++;
               }
               entrada.close();
               FileWriter fstream = new FileWriter(Path);
               BufferedWriter out = new BufferedWriter(fstream);
               out.write(new_file);
               out.close();
     }
```

1.2.7 Métodos de la Clase Crear_bitacora

recupera_file(Path, user).- Este método es utilizado para extraer toda la información del archivo *bitacora*.

```
public String recupera_file(String Path,
                                                String user
                                                ) throws Exception,
SQLException
        {
                    String linea=null;
                    String File;
                    String File_i;
                    String File_r;
                    String linea_act=null;
                    String FileL;
                    String lineap=null;
                    FileReader f = new FileReader(Path);
                       BufferedReader entrada=new BufferedReader(f);
                       linea=entrada.readLine();
                       File=linea;
                       File_i=null;
                       File_r=null;
                       FileL="";
                       while(linea!=null)
                       {
                           File_r = linea;
                           if (lineap == linea){
                                   FileL=lineap;
                           }else{
                                   FileL=FileL+"\n"+linea;
                           }
                    linea=entrada.readLine();
                    if (linea==null){
                         File = File_i;
                         cont++;
                    }else{
                         linea_act = linea;
                         File = File+"\n"+linea_act;
                         File_i = File;
                         cont++;
                    }
                     }
                       entrada.close();
                       if (cont==1){
                              return File_r;
                       }else{
                            return File;
                       }
     }
```

crea_usuario(user, Path, desc).- Este método es utilizado para almacenar en el archivo *bitácora* todas las peticiones que los usuarios Operadores soliciten.

```
public void crea_usuario(String user,String Path,String desc)
throws SQLException, Exception
{       String archivo;
        String file;
        int confirma=0;
        Crear_bitacora r= new Crear_bitacora();
        String time=null;
        time=getCurrentJavaSqlDate1();
        confirma=valida_archivo_nulo(Path);
        if (confirma==1){
         String variable="- R E Q U E R I M I E N T O S -"+"\n"+
                         "USUARIO: |"+user+"|\n"+
                         "FECHA: "+time+"\n"+
                         "PETICION.- "+"\n"+
                         "    "+desc+"\n"+
                         "--------------------------------------";
          file = variable;
          FileWriter fstream = new FileWriter(Path);
          BufferedWriter out = new BufferedWriter(fstream);
          out.write(file);
          out.close();
        }else{
         String variable="- R E Q U E R I M I E N T O S -"+"\n"+
                         "USUARIO: |"+user+"|\n"+
                         "FECHA: "+time+"\n"+
                         "PETICION.- "+"\n"+
                         "    "+desc+"\n"+
                         "--------------------------------------";
          archivo = r.recupera_file(Path, user);
          file = archivo+"\n"+variable;
          FileWriter fstream = new FileWriter(Path);
          BufferedWriter out = new BufferedWriter(fstream);
          out.write(file);
          out.close();
         }
     }
```

crea_solicitudes(user, Path, desc).- Este método es utilizado para almacenar en el archivo *user_mirror* todas las peticiones que el Operador realice. En el momento de que el Administrador conteste la petición, ésta se guardará en este archivo a continuación de la petición.

```
public void crea_solicitudes(String user,String Path,
String path, String mirror,String desc) throws SQLException,
Exception
     {
       String archivo;
       String file;
       String filem;
       int confirma=0;
       int nulo=0;
       Crear_bitacora r= new Crear_bitacora();
       String time=null;
       time=getCurrentJavaSqlDate1();
       confirma=cuenta_lineas(path);
       if (confirma==11 || confirma==0){
         r.crea_usuario(user, Path, desc);
         String variable="- R E Q U E R I M I E N T O  -"+"\n"+
                         "USUARIO: "+user+"\n"+
                         "FECHA: "+time+"\n"+
                         "PETICION.- "+"\n"+
                         "    "+desc+"\n"+
                         "-------------------------------------";
         file = variable;
         FileWriter fstream = new FileWriter(path);
         BufferedWriter out = new BufferedWriter(fstream);
         out.write(file);
         out.close();
         nulo = r.valida_archivo_nulo_m(mirror);
         if(nulo==1){
              filem = variable;
              FileWriter fstream1 = new FileWriter(mirror);
              BufferedWriter out1 = new BufferedWriter(fstream1);
              out1.write(filem); out1.close();
         }else{
              archivo = r.recupera_file(mirror, user);
              filem = archivo+"\n"+variable;
              FileWriter fstream1 = new FileWriter(mirror);
              BufferedWriter out1 = new BufferedWriter(fstream1);
              out1.write(filem); out1.close();
           }
       }else{
           System.out.println("No puede ingresar 2 requerimientos
a las vez");                    }
     }
```

1.2.8 Métodos de la Clase Responder_solicitud

Respuesta_afirmativa(user, Path, path, mirror, respuesta).- Este método es utilizado para responder de forma afirmativa el requerimiento del usuario Operador que se almacena en el archivo *bitacora*. La respuesta a esa petición el usuario Operador la podrá consultar en su respectivo archivo *user_mirror*.

```
public void Respuesta_afirmativa(String user,String Path,
                    String path, String mirror, String respuesta
                     ) throws SQLException, Exception{
   int decide=0;
   decide=lee_archivo(Path, user);
   if (decide == 1){
        System.out.println("usuario existe");
   }else{
        System.out.println("usuario no existe retorno 0");
        Responder_solicitud w = new Responder_solicitud();
        w.contesta_pedido_P(user, Path, path, mirror, respuesta);
               }
        }
```

Respuesta_negativa(user, Path, path, mirror, respuesta).- Este método tiene la misma función del método anterior, solo que éste responde la petición de forma negativa.

```
public void Respuesta_negativa(String user,String Path,
                    String path, String mirror, String respuesta
                    ) throws SQLException, Exception{
   int decide=0;
   decide=lee_archivo(Path, user);
   if (decide == 1){
        System.out.println("usuario existe");
   }else{
        System.out.println("usuario no existe retorno 0");
        Responder_solicitud w = new Responder_solicitud();
        w.contesta_pedido_N(user, Path, path, mirror, respuesta);
               }
        }
```

1.2.9 Métodos de la Clase Verificar_archivos_bitacora_control

lee_archivo_user( ).- Este método es utilizado para consultar la información que existe en el archivo de usuarios *user*.

```
public void lee_archivo_user () throws Exception, SQLException
        {
                String Path = "//etc//usuarios//user";
                FileReader fr=new FileReader(Path);
                BufferedReader entrada=new BufferedReader(fr);
                String linea=null;
                out_puts.println("<textarea cols=75 rows=20 > ");
                linea=entrada.readLine();
                linea=e.decriptar(linea);
                while(linea!=null)
                {
                 System.out.println(linea);
                 out_puts.println(linea);
                 str_token       = new   StringTokenizer(linea,"\n");

                        linea=entrada.readLine();
                        if (linea!=null){
                                linea=e.decriptar(linea);
                        }
                }
                entrada.close();
                out_puts.println("</textarea>");
        }
```

lee_archivo_logoneados( ).- Este método es utilizado para consultar la información que existe en el archivo *logoneados*.

```
public void lee_archivo_logoneados () throws Exception,
SQLException
        {
                String Path = "//etc//usuarios//logoneados";
                FileReader fr=new FileReader(Path);
                BufferedReader entrada=new BufferedReader(fr);
                String linea=null;
                out_puts.println("<textarea cols=78 rows=20 > ");
                linea=entrada.readLine();
                linea=e.decriptar(linea);
                while(linea!=null)
                {
                 System.out.println(linea);
                 out_puts.println(linea);
                 str_token       = new   StringTokenizer(linea,"\n");


                        linea=entrada.readLine();
                        if  (linea!=null){
                                linea=e.decriptar(linea);
                        }
                }
                entrada.close();
                out_puts.println("</textarea>");
        }
```

lee_archivo_hist_logon( ).- Este método es utilizado para consultar la información que existe en el archivo *hist_logon*.

```
public void lee_archivo_hist_logon () throws Exception,
SQLException
        {
                String Path = "//etc//usuarios//hist_logon";
                FileReader fr=new FileReader(Path);
                BufferedReader entrada=new BufferedReader(fr);
                String linea=null;
                out_puts.println("<textarea cols=78 rows=20 > ");
                linea=entrada.readLine();
                while(linea!=null)
                {
                        System.out.println(linea);
                        out_puts.println(linea);
                        str_token       = new
StringTokenizer(linea,"\n");

                        linea=entrada.readLine();
                }
                entrada.close();
                out_puts.println("</textarea>");
        }
```

lee_archivo_bitacora( ).- Este método es utilizado para consultar la información que existe en el archivo *bitacora*.

```
public void lee_archivo_bitacora () throws Exception,
SQLException
        {
                String Path = "//etc//usuarios//bitacora";
                FileReader fr=new FileReader(Path);
                BufferedReader entrada=new BufferedReader(fr);
                String linea=null;
                out_puts.println("<textarea cols=72 rows=18 > ");
                linea=entrada.readLine();
                while(linea!=null)
                {
                 System.out.println(linea);
                 out_puts.println(linea);
                 str_token       = new   StringTokenizer(linea,"\n");


                        if(linea==null){
                            break;
                        }else{
                                cont++;
                        }
                        linea=entrada.readLine();

                }
                entrada.close();
                out_puts.println("</textarea>");
        }
```

lee_archivo_userX_mirror(user, mirror).- Este método es utilizado para consultar la información que existe en el archivo *user_mirror*.

```
public void lee_archivo_userX_mirror (String user,
String mirror) throws Exception, SQLException
     {

               FileReader fr=new FileReader(mirror);
               BufferedReader entrada=new BufferedReader(fr);
               String linea=null;
               out_puts.println("<textarea cols=71 rows=18 > ");
               linea=entrada.readLine();
               while(linea!=null)
               {
                System.out.println(linea);
                out_puts.println(linea);
                str_token       = new   StringTokenizer(linea,"\n");


                       if(linea==null){
                              break;
                       }else{
                              cont++;
                       }
                       linea=entrada.readLine();

               }
               entrada.close();
               out_puts.println("</textarea>");

        }
```

1.2.10 Métodos de la Clase Eliminar_logoneados_v2

recupera_file(path, user).- Este método es utilizado para actualizar el archivo histórico de logoneo indicando la fecha en la cual el usuario terminó su sesión.

```
public String recupera_file(String path,String user
                                )throws Exception, SQLException
    {
            String linea=null;
            String File;
            String linea_act=null;
            String time=null;
            String trama=null;
            String act;
            String ant;
            time=getCurrentJavaSqlDate1();
            int resultado=0;
            FileReader f = new FileReader(path);
            BufferedReader entrada=new BufferedReader(f);
            linea=entrada.readLine();
            ant=linea;
            act=linea;
            File=linea;
            while(linea!=null)
            {
            str_token       = new  StringTokenizer(linea,"|");
            carga_campos_e(str_token);
            resultado=valida_usuario(user);
            if (resultado==1){
                    trama=" ---> "+time+".";
                    linea_act = "Fin de Sesión: "+linea+trama;
            }else{
                    linea_act = linea;
            }
            if(ant==act){
                    File = linea_act;
            }else{
                    File = File+"\n"+linea_act;
            }
            linea=entrada.readLine();
            act=linea;

            }
            entrada.close();
            return File;
      }
```

elimina_registro(user).- Este método se encarga de eliminar el registro de un usuario del archivo *logoneados* cuando éste cierra su sesión.

```
public void elimina_registro (String user
                                ) throws Exception, SQLException
    {
      String Path="//etc//usuarios//logoneados";
      String path="//etc//usuarios//hist_logon";
      String linea=null;
      String LineaVale;
      String Linea_ant;
      String new_file;
      String archivo=null;
      String linea_des=null;
      FileReader fr=new FileReader(Path);
      BufferedReader entrada=new BufferedReader(fr);
      linea=entrada.readLine();
      Linea_ant=linea;
      new_file=linea;
      System.out.println(linea);
      while(linea!=null)
      {
      linea_des=e.decriptar(linea);
      str_token       = new  StringTokenizer(linea_des,"|");
      carga_campos(str_token);
      result=valida_usuario(user);
      if (result==1){
              LineaVale=linea;
              if (LineaVale==Linea_ant){
                      linea=entrada.readLine();
                      new_file=linea;
              }
      }else{
              LineaVale=linea;
              if (LineaVale==Linea_ant){
                      new_file=LineaVale;
              }else{
                   new_file=new_file + "\n" + LineaVale;
              }
      }
      linea=entrada.readLine();
      }
      entrada.close();
      archivo=recupera_file(path, user);
      FileWriter fstream1 = new FileWriter(path);
      BufferedWriter out1 = new BufferedWriter(fstream1);
      out1.write(archivo);
      out1.close();
      FileWriter fstream = new FileWriter(Path);
      BufferedWriter out = new BufferedWriter(fstream);
      out.write(new_file);
      out.close();    }
```

1.2.11 Métodos de la Clase Encriptador

encriptar(ps_palabra).- Este método es utilizado para encriptar el contenido de los archivos.

```
public static String encriptar(String ps_palabra)
   {
      return Twofish.encriptar(ps_palabra);
   }
```

decriptar(ps_palabra).- Este método es utilizado para desencriptar el contenido de los archivos.

```
public static String decriptar(String ps_palabra)
   {
      return Twofish.decriptar(ps_palabra);
   }
```

1.2.12 Métodos de la Clase Configura_host_pass_ripd

lee_archivo(Path, hostname_router, pass, ena_pass).- Este método es utilizado para leer el archivo de configuración del ***ripd.conf*** y cambiar sus parámetros de configuración.

```
public void lee_archivo (String Path, String hostname_router,
                         String pass, String ena_pass) throws
Exception, SQLException
        {
        FileReader fr=new FileReader(Path);
        BufferedReader entrada=new BufferedReader(fr);
        String linea=null;
        String valida_host="";
        String ant="";
        String act="";
        String File="";
        linea=entrada.readLine();
        File=linea;
        ant=linea;
        while(linea!=null)
        {
           act=linea;
           if (linea.length()> 8){
                valida_host=linea.substring(0,8);
                if (valida_host.equals("hostname")){
                              linea="hostname "+hostname_router;
                }else
                    if (valida_host.equals("password")){
                        linea="password "+pass;
                    }else
                    if (valida_host.equals("enable p")){
                        linea="enable password "+ena_pass;
                    }
                }
                if (act==ant){
             File = ant;
                }else{
                    File = File+"\n"+linea;
                }
                linea=entrada.readLine();
           }
                entrada.close();
                escritura_archivo_ripd(Path,File);
        }
```

escritura_archivo_ripd(Path, linea).- Este método es utilizado para guardar el archivo de configuración ***ripd.conf*** con los cambios que se generaron en el método anterior.

```
public void escritura_archivo_ripd (String Path,String linea)
throws IOException {
        String variable=linea;
        FileWriter fstream = new FileWriter(Path);
        BufferedWriter out = new BufferedWriter(fstream);
        out.write(variable);
        out.close();
}
```

1.2.13 Métodos de la Clase Procesa_Arch_tabla

crea_archivo_comando(String nombre_archi, String pv_linea).- El método sirve para crear shell con comandos necesarios para, los diferentes configuraciones, visualización de tablas de ruteo, activar servicios, ect...

Este método cuenta con dos parámetros de entradas los cuales se detallan a continuación: nombre_archi aquí se le envia la ruta del archivo y nombre ejemplo: /ect/quagga/comando_ip.sh.

El siguiente parámetro que recibe este método es pv_linea, va contener la(s) línea(s) de comando del archivo (Shell), ejemplo:

route > /etc/tablas_rutas.txt

Este comando utilizamos para guardar la tabla de rutas en el archivo para su posterior lectura.

```
public void crea_archivo_comando (String nombre_archi,String
pv_linea )throws IOException
 {

        String permiso_ejecuta="chmod 777 "+nombre_archi;
        try{
                FileWriter fw= new FileWriter(nombre_archi);
                BufferedWriter entrada=new BufferedWriter(fw);
                String linea=pv_linea;
                entrada.write(linea);
                entrada.close();
                ejecuta_comando(permiso_ejecuta);

              }
        catch(Exception e){
              System.out.println("Error al crear Archivo:
"+nombre_archi+" - "+e);

          }

 }
```

ejecuta_comando(String comando) .- Este método sirve para ejecutar comandos o ejecutar archivo(shell) que fueron creados con el método anterior. Si el comando se

ejecuto correctamente el comando va a retornar un numero entero si es 0, esto significa que el comando se ejecuto correctamente.

```
public  int ejecuta_comando(String comando){
         try
         {

             final Process process = Runtime.getRuntime().exec(comando);
             new Thread(){
                 public void run(){
                    try{
                       InputStream is = process.getInputStream();
                       byte[] buffer = new byte[1024];
                       for(int count = 0; (count = is.read(buffer)) >= 0;){
                             System.out.write(buffer, 0, count);
                       }
                    }
                    catch(Exception e){
                     e.printStackTrace();
                    }
                }
                }.start();
                new Thread(){
                    public void run(){
                    try{
                         InputStream is = process.getErrorStream();
                         byte[] buffer = new byte[1024];
                         for(int count = 0; (count = is.read(buffer)) >= 0;){
                               System.err.write(buffer, 0, count);
                         }
                     }
                     catch(Exception e){
                          e.printStackTrace();
                    }
                    }.start();

                              int returnCode = process.waitFor();
                              System.out.println("Return code = " + returnCode);
                              return returnCode;

                    }
                    catch (Exception e){
                              e.printStackTrace();
                              int returnCode=13;
                              return returnCode;

                    }

 }
```

ejecuta_comando2(String comando) .- Este método tiene la misma función que el método anterior, la diferencia es que este almacena el mensaje de error en un variable privada de la clase. Para obtener el valor de variable que

esta contenida en la clase, usaremos el método getmsn() que nos retorna un String con el error .

```
public int ejecuta_comando2(String comando){
          try
          {
               final Process process = Runtime.getRuntime().exec(comando);
               int retorno=0;
               String lee_buff="";
               try{

                    InputStream is = process.getInputStream();
                    byte[] buffer = new byte[1024];
                    for(int count = 0; (count = is.read(buffer)) >= 0;){
                        lee_buff+=new String(buffer,0,count);
                    }

                }
               catch(Exception e){
                    e.printStackTrace();
                    this.msn=e.getMessage();
                    System.out.println("Entro en 1ra exception");
                  }
               this.code_error = process.waitFor();
               this.msn=lee_buff;

               if (this.code_error > 0){
                  try{
                    InputStream is = process.getErrorStream();
                    byte[] buffer = new byte[1024];
                    for(int count = 0; (count = is.read(buffer)) >= 0;){
                          lee_buff+=new String(buffer,0,count);
                    }
                  }
                  catch(Exception e){
                    e.printStackTrace();
                    System.out.println("Entro en 2da exception");
                    this.msn=e.getMessage();
                   }
                }

                    retorno = process.waitFor();
                    return retorno;

                    }
                    catch (Exception e){
                              e.printStackTrace();
                              int retorno=0;
                              System.out.println("Entro en 3ra exception");
                              this.msn=e.getMessage();
                              this.code_error=13;
                              retorno=13;
                              return retorno;
                    }

 }
```

Actualiza_archi_tabla().- Este método actualiza el archivo que contiene la tabla con las rutas que en ese momento tiene configurada la maquina .

```
public void  actuliza_archi_tabla()throws Exception{
    String ruta_archivo="//etc//";
    String nombre_archivo="actualizaroute2.sh";
    String tabla_archivo="rutas_estaticas.txt";
    String comando_archi="route > "+ruta_archivo+tabla_archivo;
    String ejecuta_archivo="sh "+ruta_archivo+nombre_archivo ;
        try{
                crea_archivo_comando(ruta_archivo+nombre_archivo,comando_archi);
                ejecuta_comando(ejecuta_archivo);
                }
        catch(Exception e){
                System.out.println("Error al actualizar archivo de rutas"+e);
        }


}
```

getTablas() .- Este método retorna un vector con objetos de tipo Ruta_Tabla, el vector que retorna contiene las rutas que tiene configurada nuestro Linux hasta este momento.

```
public Vector getTablas ()throws Exception {

    Vector Tabla_rutas = new Vector();

    FileReader fr=new FileReader("//etc//rutas_estaticas.txt");
    BufferedReader entrada=new BufferedReader(fr);
            String linea=null;
            int cont=0;
                while((linea=entrada.readLine())!=null)
                    {
                        cont++;
                        if(cont>2){
                                str_token       = new  StringTokenizer(linea," ");
                                Tabla = new Ruta_Tabla (str_token);
                                Tabla_rutas.add (Tabla);
                        }
                    }// fin del while
                    entrada.close();
                    fr.close();
    return Tabla_rutas;
}
```

agrega_ruta(String Destination, String Genmask, String Gateway, String Interfaz) .- Este método recibe los parámetros que le envía la interfaz web, la función de este método es agregar las rutas estáticas. Este metodo recibe

estos parámetros y ejecuta el comando requerido para agregar la ruta que requiere el usuario

```
public int agrega_ruta(String Destination,String Genmask, String Gateway, String Interfaz
                    ) throws SQLException, Exception
          {
                    String lv_comando="";
                    String nombre_archivo="//etc//agrega_ruta.sh";
                    System.out.println("destination"+Destination);
                    String comando_ejecuta_sh="";
                    String mask_convertida="";
                    int retorno=0;
          try{
                    if(Destination.equals("default")){
                         if (Gateway.equals("*")){
                              lv_comando="route add "+Destination+" "+Interfaz;
                              crea_archivo_comando(nombre_archivo,lv_comando);
                              comando_ejecuta_sh="sh "+nombre_archivo;
                              retorno=ejecuta_comando2(comando_ejecuta_sh);
                        }
                        else {
                                  if (Interfaz.equals("*")){
                                     lv_comando="route add "+Destination+" gw "+Gateway;
                              crea_archivo_comando(nombre_archivo,lv_comando);
                              comando_ejecuta_sh="sh "+nombre_archivo;
                              retorno=ejecuta_comando2(comando_ejecuta_sh);
                                 }
                                else{
                                  lv_comando="route add "+Destination+" gw "+Gateway+" "+Interfaz;
                                  crea_archivo_comando(nombre_archivo,lv_comando);
                                  comando_ejecuta_sh="sh "+nombre_archivo;
                                  retorno=ejecuta_comando2(comando_ejecuta_sh);
                               }
                        }

                    }
                    else{


CONTINUARA… … … … … … … .
```

```
                    else{
                        if (Gateway.equals("*")){
                            Mascara mk =new Mascara();
                            mask_convertida= mk.convierte_mask(Genmask);
                            lv_comando="route add -net "+Destination+"/"+mask_convertida+"
"+Interfaz;
                                crea_archivo_comando(nombre_archivo,lv_comando);
                                comando_ejecuta_sh="sh "+nombre_archivo;
                                retorno=ejecuta_comando2(comando_ejecuta_sh);
                            }
                            else{
                                    if (Interfaz.equals("*")){
                                            Mascara mk =new Mascara();
                                            mask_convertida= mk.convierte_mask(Genmask);
                                            lv_comando="route add -net
"+Destination+"/"+mask_convertida+" gw "+Gateway;
                                            crea_archivo_comando(nombre_archivo,lv_comando);
                                            comando_ejecuta_sh="sh "+nombre_archivo;
                                            retorno=ejecuta_comando2(comando_ejecuta_sh);
                                    }
                                    else{
                                            Mascara mk =new Mascara();
                                            mask_convertida= mk.convierte_mask(Genmask);
                                            lv_comando="route add -net
"+Destination+"/"+mask_convertida+" gw "+Gateway+" "+Interfaz;
                                            crea_archivo_comando(nombre_archivo,lv_comando);
                                            comando_ejecuta_sh="sh "+nombre_archivo;
                                            retorno=ejecuta_comando2(comando_ejecuta_sh);
                                    }
                            }

                    }
                    return retorno;
            }
            catch(Exception e){
                    retorno=69;
                    return retorno;
            }
}
```

borra_ruta (String Destination,String Genmask, String Gateway, String Interfaz) .- Este método recibe los parámetros que le envía la interfaz web, la función de este método es borrar las rutas estáticas. Este método recibe estos parámetros y ejecuta el comando requerido para borrar la ruta que requiere el usuario.

```
public int borra_ruta(String Destination, String Genmask,String Gateway,String Interfaz
                                        ) throws SQLException, Exception
            {
          String lv_comando="";
          String nombre_archivo="//etc//borra_ruta.sh";
          System.out.println("destination"+Destination);
          String comando_ejecuta_sh="";
          String mask_convertida="";
          int retorno=0;
         try{
          if(Destination.equals("default")){

                if ((Gateway.equals("*"))){
               lv_comando="route del "+Destination+" "+Interfaz;
                        crea_archivo_comando(nombre_archivo,lv_comando);
                        comando_ejecuta_sh="sh "+nombre_archivo;
                        retorno=ejecuta_comando2(comando_ejecuta_sh);
                }
                else {
                         lv_comando="route del "+Destination+" gw "+Gateway+" "+Interfaz;
                        crea_archivo_comando(nombre_archivo,lv_comando);
                        comando_ejecuta_sh="sh "+nombre_archivo;
                        retorno=ejecuta_comando2(comando_ejecuta_sh);
                 }
          }
                 else{
                     if ((Gateway.equals("*"))){
                        Mascara mk =new Mascara();
                        mask_convertida= mk.convierte_mask(Genmask);
                        lv_comando="route del -net "+Destination+"/"+mask_convertida;
                        crea_archivo_comando(nombre_archivo,lv_comando);
                        comando_ejecuta_sh="sh "+nombre_archivo;
                        retorno=ejecuta_comando2(comando_ejecuta_sh);
                      }
                      else{
                                Mascara mk =new Mascara();
                                mask_convertida= mk.convierte_mask(Genmask);
                                lv_comando="route del -net
"+Destination+"/"+mask_convertida+" gw "+Gateway+" "+Interfaz;
                                crea_archivo_comando(nombre_archivo,lv_comando);
                                comando_ejecuta_sh="sh "+nombre_archivo;
                                retorno=ejecuta_comando2(comando_ejecuta_sh);
                      }
                 }
              return retorno;
             }
        catch(Exception e){
              retorno=69;
              return retorno;
        }
  }
```

1.2.14 Métodos de la Clase Mascara

convierte_mask (String ps_mask) .- Este método recibe una cadena de caracteres (String), esta cadena de caracteres contiene una mascara con el siguiente formato "255.255.255.0".

Lo que hace la función con esta cadena es convertir esta mascara en un formato de numero de bits ejemplo :

255.255.255.0 → 24

255.255.0.0 → 16

```
public String convierte_mask(String ps_mask)
        {
                String retorno="";

                if (ps_mask.compareTo("0.0.0.0")== 0) {
                        retorno="0";
                        return retorno;

                }
                        else if (ps_mask.compareTo("128.0.0.0")== 0) {
                                retorno="1";
                                return retorno;
                        }
                        else if (ps_mask.compareTo("192.0.0.0")== 0) {
                                retorno="2";
                                return retorno;
                        }

                        else if (ps_mask.compareTo("224.0.0.0")== 0) {
                                retorno="3";
                                return retorno;
                        }
                                                .
                                                .
                                 Valido todas las mascaras validas
                                                .
                                                .
                        else if (ps_mask.compareTo("255.255.255.254")== 0) {
                                retorno="31";
                                return retorno;
                        }
                        else if (ps_mask.compareTo("255.255.255.255")== 0) {
                                retorno="32";
                                return retorno;
                        }

                else{
                        retorno="ERROR";
                        return retorno;
                }

        }
```

getmaskvalida().- Este método retorta un vector de String con todas las Mascaras validas del protocolo ip versión.

```
public Vector getmaskvalidas()
{
            String lv_mask="";
      Vector lista_mask= new Vector();

            lv_mask="255.255.255.254";
            lista_mask.add(lv_mask);

      lv_mask="255.255.255.252";
            lista_mask.add(lv_mask);

      lv_mask="255.255.255.248";
            lista_mask.add(lv_mask);

                  ::
                  ::
          Se resume el metodo
                  ::
                  ::
      lv_mask="240.0.0.0";
            lista_mask.add(lv_mask);

            lv_mask="224.0.0.0";
            lista_mask.add(lv_mask);

      lv_mask="192.0.0.0";
            lista_mask.add(lv_mask);

      lv_mask="128.0.0.0";
            lista_mask.add(lv_mask);

            return lista_mask;


}
```

1.2.15 Métodos de la Clase list_eth

act_archi_interface () .- Este método guarda en un archivo la lista de las interfaces de red que tiene servidor Linux.

```
public void  act_archi_lista_interfaces()throws Exception{
          String ruta_archivo="//etc//";
          String nombre_archivo_comando="actualiza_interfaces_eth.sh";
          String lista_interfaces="lista_interfaz_eth.txt";
          String comando_archi="ip link list > "+ruta_archivo+lista_interfaces;

          String ejecuta_archivo="sh "+ruta_archivo+nombre_archivo_comando ;
          try{

crea_archivo_comando(ruta_archivo+nombre_archivo_comando,comando_archi);
                    ejecuta_comando(ejecuta_archivo);
                    }
          catch(Exception e){
                    System.out.println("Error al actualizar archivo de rutas"+e);
          }

}
```

cambia_ruta (String pv_interfaz, String pv_direccion, String pv_mask) .- Este método configura las interfaces de red del servidor de red.

```
public int cambia_ruta(String pv_interfaz,String pv_direccion,String pv_mask)
{
          String lv_comando="";
          String nombre_archivo="//etc//cambia_ip.sh";
          String comando_ejecuta_sh="";
          String mask_convertida="";
          int retorno=0;
          try{
                Mascara mk =new Mascara();
                mask_convertida= mk.convierte_mask(pv_mask);
                lv_comando="ifconfig "+pv_interfaz+" "+pv_direccion+"/"+ mask_convertida+"
"+"up" ;
               crea_archivo_comando(nombre_archivo,lv_comando);
                comando_ejecuta_sh="sh "+nombre_archivo;
                retorno=ejecuta_comando(comando_ejecuta_sh);
                return retorno;
          }
          catch(Exception e){
            retorno=69;
            return retorno;
          }
}
```

getlista_iface().- Este metodo almacena en un vector las interfaces que tiene el servidor linux. Por ejemplo: eth0, eth1,eth2, ect...

```
public Vector getlista_iface ()throws Exception {

          Vector vector_iface= new Vector();
           act_archi_lista_interfaces();
           FileReader fr=new FileReader("//etc//lista_interfaz_eth.txt");//linux
           BufferedReader entrada=new BufferedReader(fr);
           String linea=null;
          String lo="lo";
          String sit="sit0";
           int cont=0;
           while((linea=entrada.readLine())!=null)
          {
              cont++;
              if((cont%2)!=0){
                     str_token       = new  StringTokenizer(linea," ");
                     lv_iface=new lista_eth(str_token);
                if (!lv_iface.getinterfaz().equals(lo)&& !lv_iface.getinterfaz().equals(sit)){
                     vector_iface.add(lv_iface);
                }
           }
          }// fin del while
          entrada.close();
          fr.close();
          return vector_iface;
}
```

crea_archivos_network(String pv_interfaz,String pv_direcion, String pv_mask).- Este metodo calcula el nombre de red a la cual pertenece un Ip con formato, 192.168.1.20.

El resultado de esta operación es guarda en un archivo con el nombre datos_network_eth0.txt

```
public void  crea_archivos_network(String pv_interfaz,String pv_direcion,String
pv_mask)throws Exception{
      String ruta_archivo="//etc//";
      String nombre_archivo_comando="comando_network.sh";
      String nobre_archi_datos="datos_network_"+pv_interfaz+".txt";
      String comando_archi="ipcalc -n "+pv_direcion+" "+pv_mask+" > "
+ruta_archivo+nobre_archi_datos;
      String ejecuta_archivo="sh "+ruta_archivo+nombre_archivo_comando ;
      try{
          crea_archivo_comando(ruta_archivo+nombre_archivo_comando,comando_archi);
          ejecuta_comando(ejecuta_archivo);
          }
          catch(Exception e){
                    System.out.println("Error al crear archivo: "+e);
          }

}
```

getnetwork_if (String pv_iface).- Este método tiene la funcionalidad de leer la información que se creo con el método anterior, procesar esta información y presentar el nombre de red.

```
public String getnetwork_if (String pv_iface )throws Exception {
         String path="//etc//";//linux
        String archi_network="datos_network_"+pv_iface+".txt";
         FileReader fr=new FileReader(path+archi_network);
         BufferedReader entrada=new BufferedReader(fr);
         String result_network=null;
         String linea=null;
          while((linea=entrada.readLine())!=null)
         {
               str_token       = new   StringTokenizer(linea,"=");
               str_token.nextToken();
               result_network=str_token.nextToken();

         }// fin del while
         entrada.close();

         fr.close();
      return result_network;
}
```

crea_archi_masivos_if_up ().- Este método tiene la funcionalidad crear varios archivos con la información de cada interfaz de red que esta activa en ese momento.

```
public void crea_archi_masivos_if_up(){
 try{
          lista_eth fila2 = new lista_eth();
          Vector colefila2 =  fila2.getlista_iface_up();
          Enumeration un2=colefila2.elements();
          while (un2.hasMoreElements()){
                    fila2 = (lista_eth) un2.nextElement();
                     System.out.println(fila2.getinterfaz());
                     fila2.crea_archivos_interfaz(fila2.getinterfaz());

     }
 }
 catch(Exception e){

 }
}
```

arma_zebra_rip ().- Este método modifica los archivos zebra.conf y quagga.conf, recibe como parámetros hostname de router, password , enable password y neigbor, la demas información que se necesita para configurar estos archivos la recuperamos y guardamos en estos 2 archivos.

```
public int arma_zebra_rip (String pv_hostname, String pv_password, String pv_ena_password,
String pv_neigboar)throws IOException
          {


        String lv_eth="";
        String lv_eth2="";
        String lv_direccion_ip="";
        String lv_mask="";
        String lv_network="";
             String texto_zebra="!\n"+
                                                                    "! Zebra configuration saved from
vty\n"+
                                                                    "!   2008/09/01 17:16:25\n"+
                                                                    "!\n"+
                                                                    "hostname "+pv_hostname+"\n"+
                                                                    "password "+pv_password+"\n"+
                                                                    "enable password
"+pv_ena_password+"\n";
             String texto_rip=texto_zebra;
             texto_rip=texto_rip+"log file /var/log/quagga/ripd.log\n"+
                        "!\n";
             texto_zebra=texto_zebra+"log file /var/log/quagga/zebra.log\n"+
                                                                    "!\n";
                    Mascara m=new Mascara();
                    FileReader fr=new FileReader("//etc//all_datos_interfaces.txt");//linux
                    //FileReader fr=new FileReader("c:\\archivo\\all_datos_interfaces.txt");
                    BufferedReader entrada=new BufferedReader(fr);
                    //rip
                    FileReader fr2=new FileReader("//etc//all_datos_interfaces.txt");//linux

                    //FileReader fr2=new FileReader("c:\\archivo\\all_datos_interfaces.txt");
                    BufferedReader entrada2=new BufferedReader(fr2);
                    String linea="";
                    String linea2="";
                    int resultado=0;

                    try{


                              texto_rip=texto_rip+
                              "interface lo\n"+
                              "!\n";

                              texto_zebra=texto_zebra+
                              "interface lo\n"+
                              "!\n";

                              while((linea2=entrada2.readLine())!=null)
                              {
                                        str_token = new  StringTokenizer(linea2,"|");
                                        config_rip lv_iface2=new config_rip(str_token);
                                        lv_eth2=lv_iface2.geteth();
                                   texto_rip=texto_rip+"interface "+lv_eth2+"\n"+"!\n";

                              }

                              texto_rip=texto_rip+"router rip\n";
                    fr2.close();
                    entrada2.close();
                              while((linea=entrada.readLine())!=null)
                              {
                                        str_token = new  StringTokenizer(linea,"|");
                                        config_rip lv_iface=new config_rip(str_token);
                                   lv_eth=lv_iface.geteth();
                                   lv_direccion_ip=lv_iface.getip_direcion();
                                   lv_mask=lv_iface.getip_mask();
                                   lv_network=lv_iface.getip_network();
                                   texto_zebra=texto_zebra+"interface "+ lv_eth+"\n"+
                                                                                 " ip address
"+lv_direccion_ip+"/"+m.convierte_mask(lv_mask)+"\n"+
                                                                              " ipv6 nd suppress-
ra\n"+
                                                                              "!\n";
                                        texto_rip=texto_rip+" network
"+lv_network+"/"+m.convierte_mask(lv_mask)+"\n";
```

arma_zebra_ospfd ().- Este método modifica los archivos zebra.conf y ospfd.conf, recibe como parámetros hostname de router, password , enable password y neigbor, la demas información que se necesita para configurar estos archivos la recuperamos y guardamos en estos 2 archivos.

```
                        public int arma_zebra_ospf (String pv_hostname, String pv_password,
String pv_ena_password, String pv_neigboar)throws IOException
          {


        String lv_eth="";
        String lv_eth2="";
        String lv_direccion_ip="";
        String lv_mask="";
        String lv_network="";
             String texto_zebra="!\n"+
                                                        "! Zebra configuration saved from
vty\n"+
                                                        "!   2008/09/01 17:16:25\n"+
                                                        "!\n"+
                                                        "hostname "+pv_hostname+"\n"+
                                                        "password "+pv_password+"\n"+
                                                        "enable password
"+pv_ena_password+"\n";
             String texto_ospf=texto_zebra;
             texto_ospf=texto_ospf+"log file /var/log/quagga/ospfd.log\n"+
                        "!\n";
             texto_zebra=texto_zebra+"log file /var/log/quagga/zebra.log\n"+
                                                        "!\n";
                Mascara m=new Mascara();
                FileReader fr=new FileReader("//etc//all_datos_interfaces.txt");//linux
                //FileReader fr=new FileReader("c:\\archivo\\all_datos_interfaces.txt");
                BufferedReader entrada=new BufferedReader(fr);
                //rip
                FileReader fr2=new FileReader("//etc//all_datos_interfaces.txt");//linux
                //FileReader fr2=new FileReader("c:\\archivo\\all_datos_interfaces.txt");
                BufferedReader entrada2=new BufferedReader(fr2);
                String linea="";
                String linea2="";
                int resultado=0;

                try{


                        texto_ospf=texto_ospf+
                        "interface lo\n"+
                        "!\n";

                        texto_zebra=texto_zebra+
                        "interface lo\n"+
                        "!\n";

                        while((linea2=entrada2.readLine())!=null)
                        {
                                str_token = new  StringTokenizer(linea2,"|");
                                config_rip lv_iface2=new config_rip(str_token);
                                lv_eth2=lv_iface2.geteth();
                                texto_ospf=texto_ospf+"interface "+lv_eth2+"\n"+"!\n";

                        }

                        texto_ospf=texto_ospf+"router ospf\n"+" network 0.0.0.0/0 area
0\n";
                fr2.close();
                entrada2.close();
                        while((linea=entrada.readLine())!=null)
                        {
                                str_token = new  StringTokenizer(linea,"|");
                                config_rip lv_iface=new config_rip(str_token);
                        lv_eth=lv_iface.geteth();
                        lv_direccion_ip=lv_iface.getip_direcion();
                        lv_mask=lv_iface.getip_mask();
                        lv_network=lv_iface.getip_network();
```

CONTINUARA...

```
                                    texto_zebra=texto_zebra+"interface "+ lv_eth+"\n"+
                                                                                " ip address
"+lv_direccion_ip+"/"+m.convierte_mask(lv_mask)+"\n"+
                                                                                " ipv6 nd suppress-
ra\n"+
                                                                                "!\n";
                                        //texto_rip=texto_rip+" network
"+lv_network+"/"+m.convierte_mask(lv_mask)+"\n";

                                }
                                if (pv_neigboar != ""){
                                        texto_ospf=texto_ospf+" neighbor "+pv_neigboar+"\n";
                                }
                                texto_ospf=texto_ospf+
                                "!\n"+
                                "line vty\n"+
                                "!\n";
                                texto_zebra=texto_zebra+
                                "interface sit0\n"+
                                " ipv6 nd suppress-ra\n"+
                                "!\n"+
                                "line vty\n"+
                                "!\n";

                                fr.close();
                                entrada.close();

                                ejecuta_comando("service zebra stop");//linux
                                ejecuta_comando("service ospfd stop");//linux
                            FileWriter fstream = new
FileWriter("//etc//quagga//zebra.conf");//linux
                                //FileWriter fstream = new FileWriter("c:\\archivo\\zebra.conf");
                                BufferedWriter out = new BufferedWriter(fstream);
                                out.write(texto_zebra);
                                out.close();
                                fstream.close();
                                ejecuta_comando("service zebra start");//linux

                            FileWriter fstream2 = new FileWriter("//etc//quagga//ospfd.conf");
//linux
                                //FileWriter fstream2 = new FileWriter("c:\\archivo\\ospfd.conf");
                                BufferedWriter out2 = new BufferedWriter(fstream2);
                                out2.write(texto_ospf);
                        out2.close();
                                fstream2.close();
                                ejecuta_comando("service ospfd start");

                        System.out.println("Archivo generado con exito");
                                System.out.println(texto_zebra);
                                return resultado;

                }
                catch(Exception e){

                    System.out.println("Error al leer archivo "+e.getMessage());
                    resultado=1;
                    return resultado;
                }


            }
```

CAPÍTULO 2

2. MANUAL DE USUARIO

Pantalla de Inicio de la Aplicación

Ilustración 2-1 Pantalla de Inicio de la aplicación

Pantalla de logoneo, cuando se inicia por primera vez la aplicación existe solo el usuario Administrador, para poder ingresar a la misma el Usuario y Contraseña a utilizar son admin.

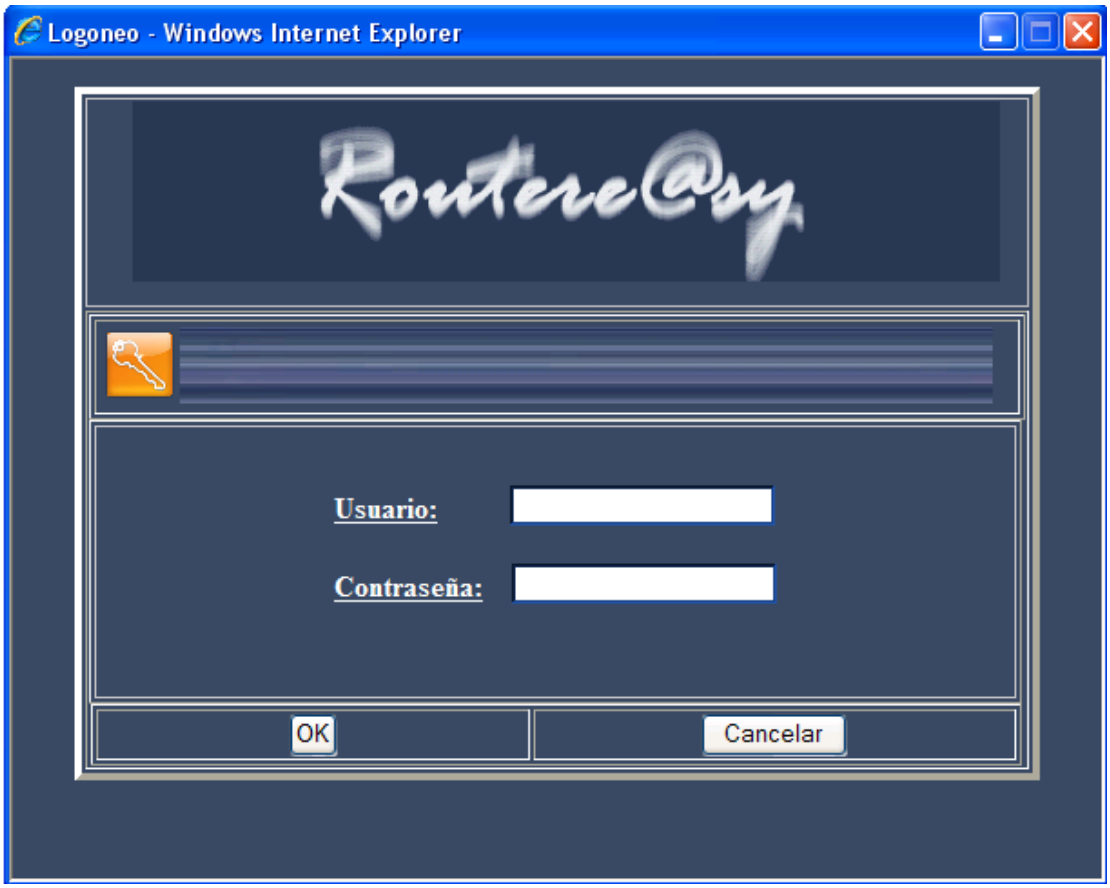

Ilustración 2-2 Pantalla de Inicio de la aplicación

En caso de no haber ingresado nada en las cajas de texto de la aplicación se presentará una alerta descriptiva del evento que sucedió.

En este claro ejemplo, se presiono el botón "OK", no se ha ingresado usuario

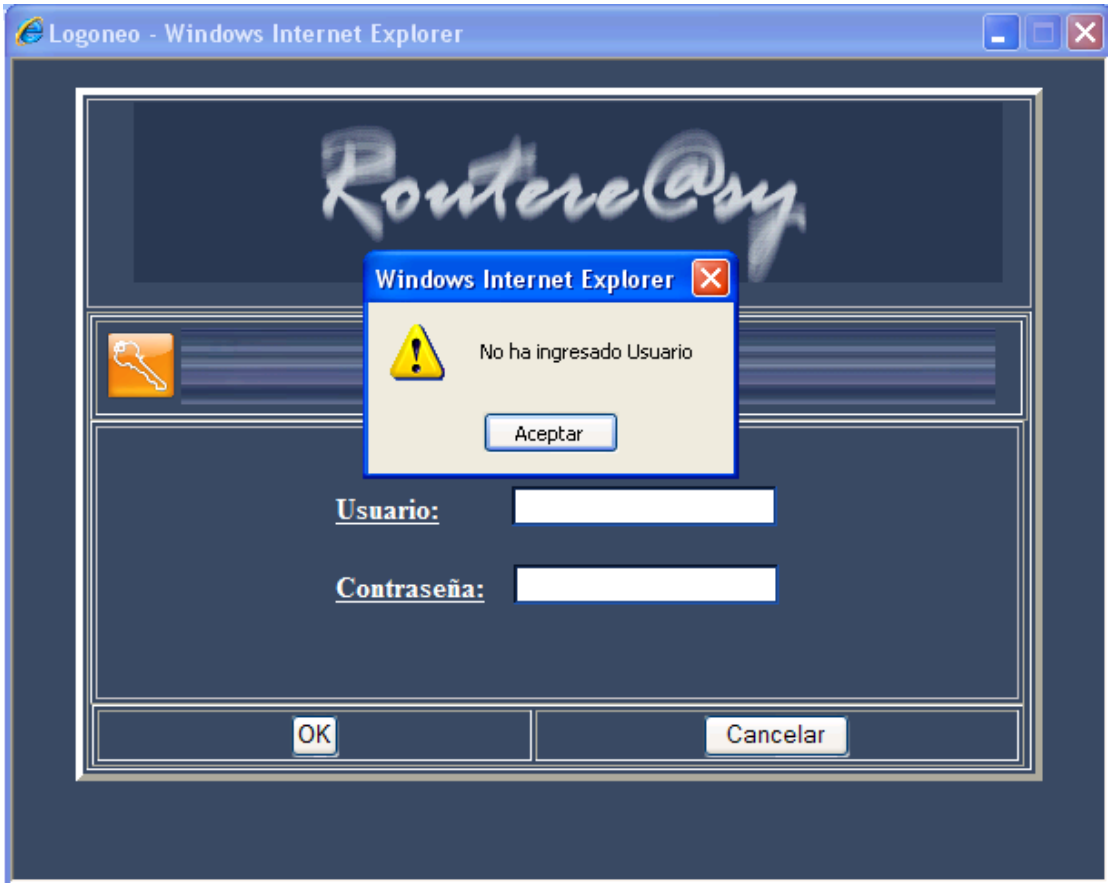

Ilustración 2-3 Alerta de ingreso de Usuario

En este claro ejemplo, se presiono el botón "OK", no se ha ingresado contraseña.

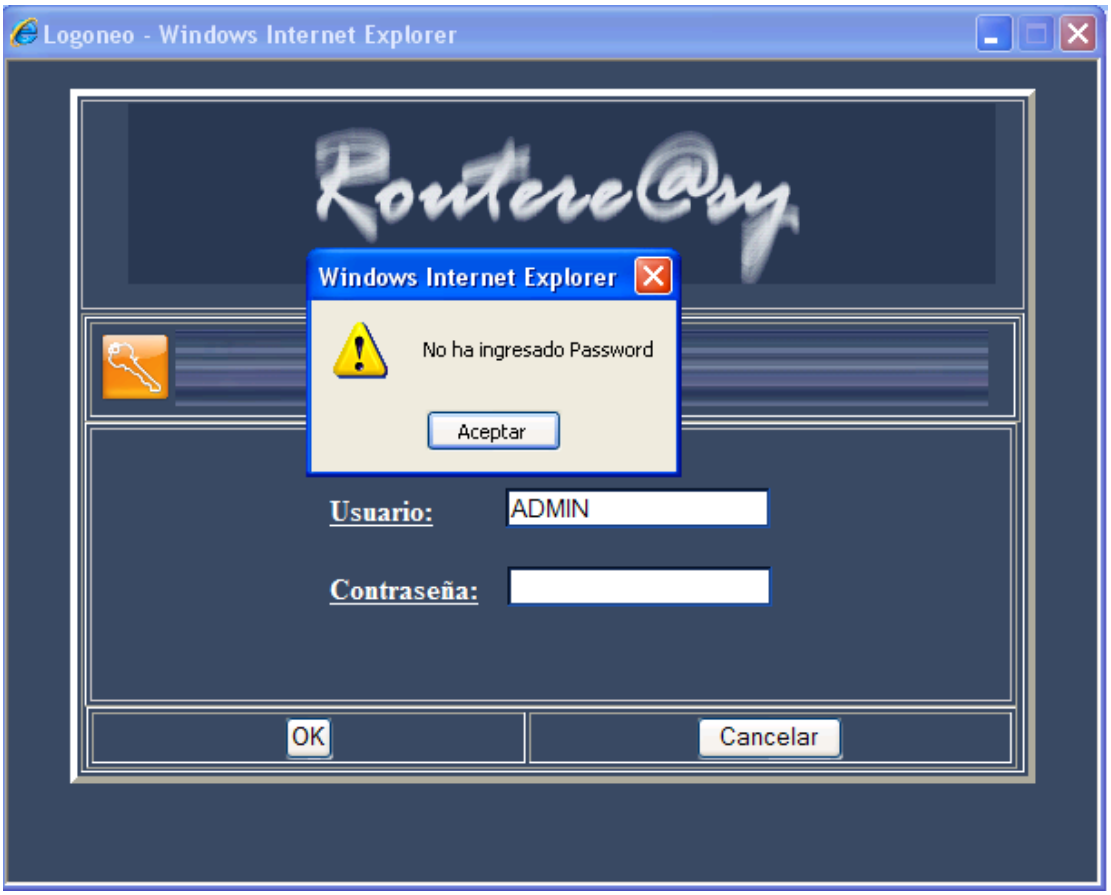

Ilustración 2-4 Alerta de ingreso de Password

Se validará que el Usuario a logonearse es el correcto caso contrario se presentará la siguiente:

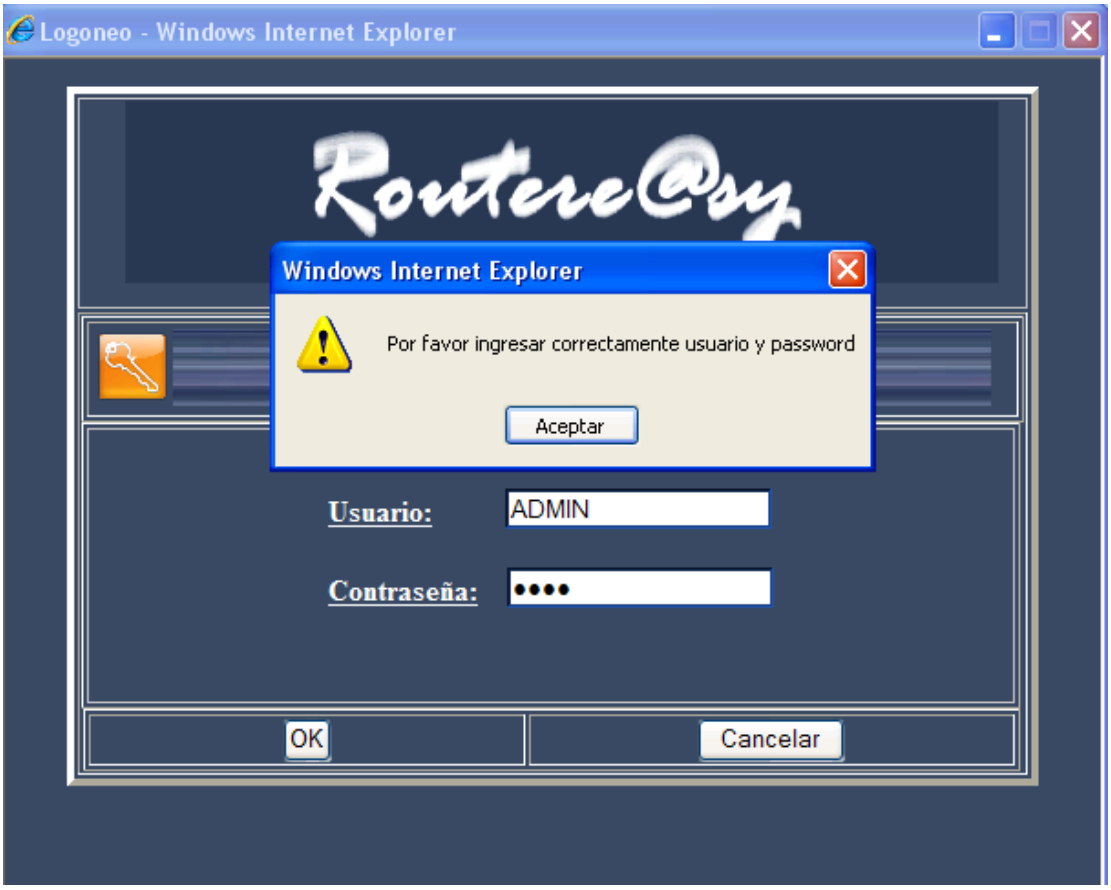

Ilustración 2-5 Alerta de ingreso correcto de Usuario y Password

El usuario tendrá 3 intentos para ingresar a la aplicación caso contrario presentará una alerta, seguidamente cerrará la aplicación.

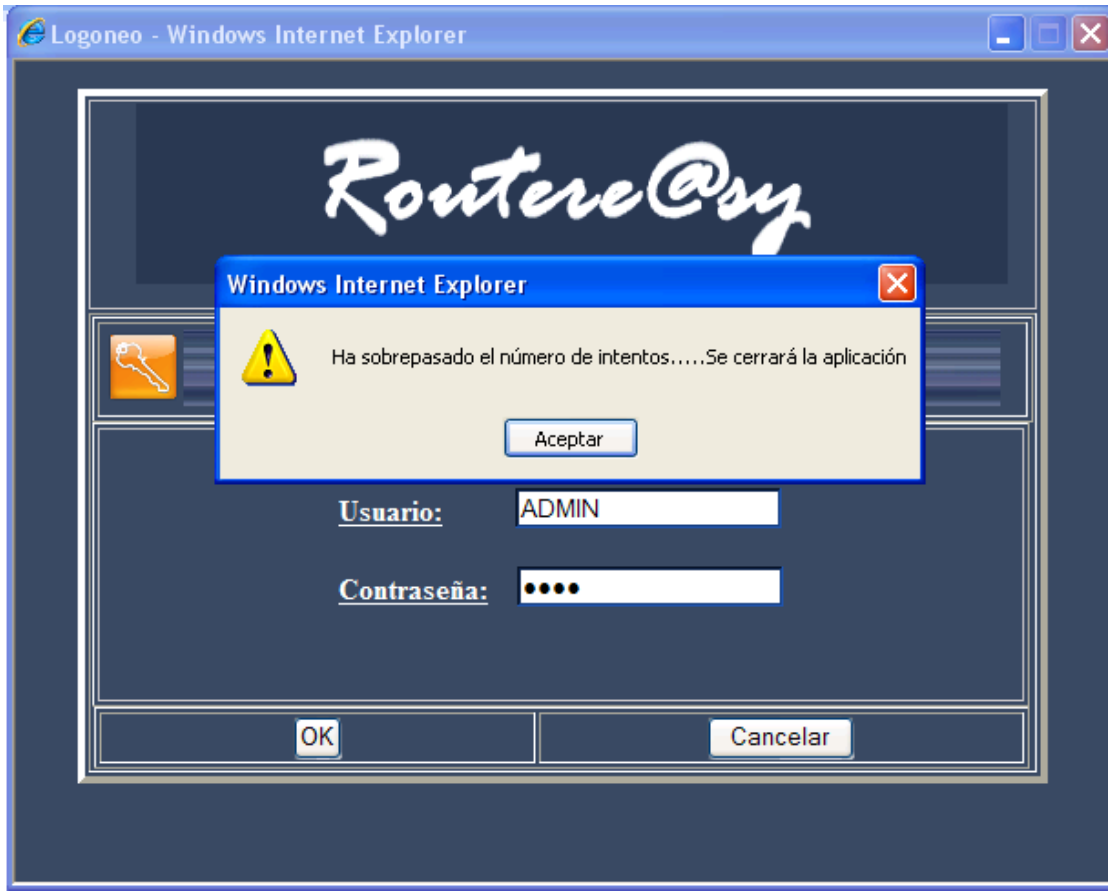

Ilustración 2-6 Alerta de haber sobrepasado el número de intentos

En el caso de no utilizar la aplicación existe la opción de Cancelar y cerrarla directamente.

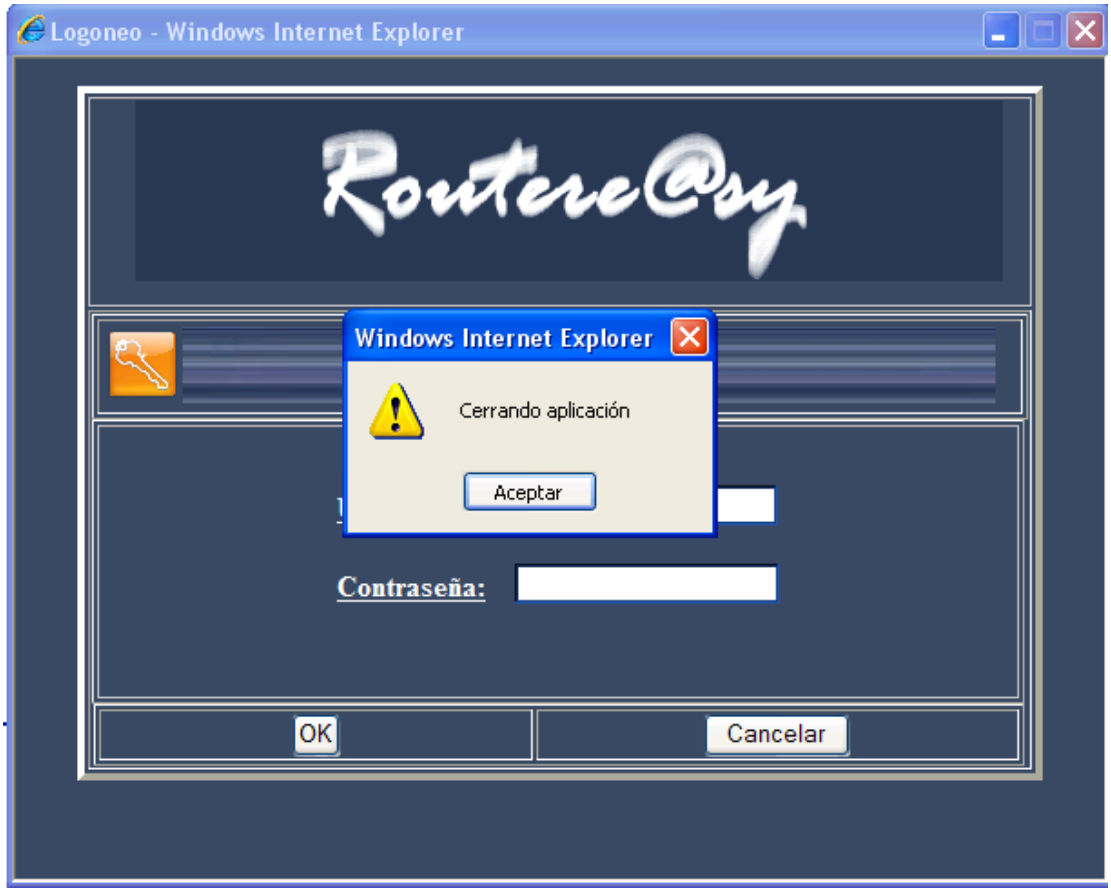

Ilustración 2-7 Alerta de cerrado de aplicación

2.1. Inicio de sesión de administrador

Cuando el Administrador ingresa a la Aplicación tendrá habilitado todas las opciones en el:

1.- Bienvenido

2.- Usuarios

3.- Bitácora de Control

4.- Configuración Básica

5.- Router Estático

6.- Router Dinámico – Rip

7.- Router Dinámico – Ospf

8.- Cerrar Aplicación

Ilustración 2-8 Pantalla de Bienvenida

2.1.1. Bienvenido

2.1.1.1. Introducción

Dentro del Menú de Bienvenido esta el Submenú de Introducción, en el cual estarán especificados conceptos básicos, y una breve descripción de lo concerniente a Implementar una máquina Linux como Router.

Ilustración 2-9 Pantalla de Breve Introducción

2.1.2. Usuarios

Dentro del Menú de Usuarios están los Submenús detallados anteriormente.

2.1.2.1. Crear Usuario

El Administrador tendrá la potestad de realizar esta acción, mas no el operador, para él será restringido.

Ilustración 2-10 Menú de Usuarios

Al hacer clic en el link de Crear usuario en la pantalla anterior, esta nos dirigirá a la pantalla de creación.

Ilustración 2-11 Crear Usuarios

Si usted desea presionar el botón de Grabar, y no ha ingresado nada en las cajas de texto, se presentará una alerta para que ingrese los campos.

Ilustración 2-12 Alerta de ingreso de toda la información

El Administrador debe de ingresar los datos del operador, según el estándar manejado será la primera letra el nombre seguido del apellido.

Cuando el operador sea ingresado correctamente, el deberá utilizar la aplicación para cambiar su clave

Ilustración 2-13 Ingreso de Usuario

En caso de que la clave ingresada y su confirmación sean diferentes, se presentará una alerta, ellas deben de coincidir.

Ilustración 2-14 Alerta de Clave y confirmación no coinciden

Cuando los datos ingresados exitosamente se presentara una alerta indicando que todo esta correcto.

Ilustración 2-15 Alerta de usuario creado exitosamente

2.1.2.2. Cambio de clave

Es aconsejable que la primera vez en se Logonee el usuario Administrador cambie su clave, ya que por default es admin.

Si se presiona el botón de grabar sin haber ingresado nada en los campos de texto se presentará un alerta "Debe ingresar toda la información que se pide".

Ilustración 2-16 Cambiar clave

La nueva clave y su confirmación deben de coincidir, caso contrario aparecerá un alerta indicándolo.

Ilustración 2-17 Cambiar clave

Cuando todo este correcto, se precederá a cambiar la clave presentando así el alerta de que todo fue un éxito.

Ilustración 2-18 Confirmación de clave cambiada

2.1.2.3. Eliminar Usuario

En este caso el Administrador solo podrá eliminarlos. Él solo tendrá que colocar en la caja de texto el usuario a eliminar y listo.

Ilustración 2-19 Pantalla Principal de eliminar usuario

Si no se ha ingresado nada en la caja de texto y se presiona el botón de Eliminar, se presentará una alerta mostrando el mensaje: "Debe ingresar el usuario a eliminar".

Ilustración 2-20 Alerta ingrese usuario a eliminar

Cuando ya es ingresado el usuario y este no existe, se presentará un alerta.

"No puede eliminar un usuario que no existe".

Ilustración 2-21 Alerta no puede eliminar usuario inexistente

En caso de que se ha ingresado de forma correcta y este existe, se presentará un alerta: "Usuario eliminado exitosamente"

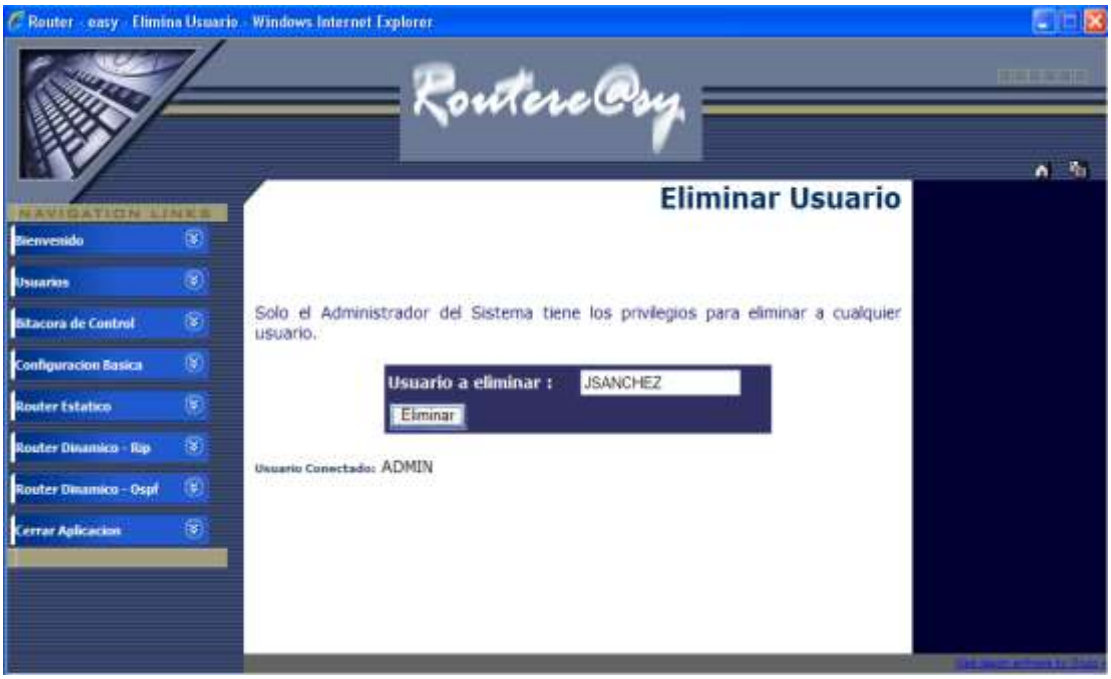

Ilustración 2-22 Eliminar Usuarios

Ilustración 2-23 Alerta de Usuario eliminado exitosamente

2.1.3. Bitácora de Control

Esta opción servirá para monitorear el inicio y cierre de sesión de los usuarios que se encuentren registrados.

2.1.3.1. Usuarios Registrados

Cuando presionemos sobre este link, nos dirigirá a un pantalla informativa, aquí encontraremos los usuarios que se encuentran registrados y habilitados para utilizar esta aplicación.

Ilustración 2-24 Consulta de Usuarios

2.1.3.2. Usuarios Logoneados

Aquí se visualizarán los usuarios que se encuentran utilizando la aplicación en ese momento.

Ilustración 2-25 Usuarios Logoneados

2.1.3.3. Historial Sesiones

El administrador encontrará información de todas las ocasiones en que los usuarios han iniciado y finalizado sesión. Con su fecha y hora correspondiente para los dos eventos.

Ilustración 2-26 Historial de Sesiones

2.1.3.4. Requerimientos

Información de requerimientos o sugerencias de los usuarios que deberán analizarse antes de realizar los cambios solicitados.

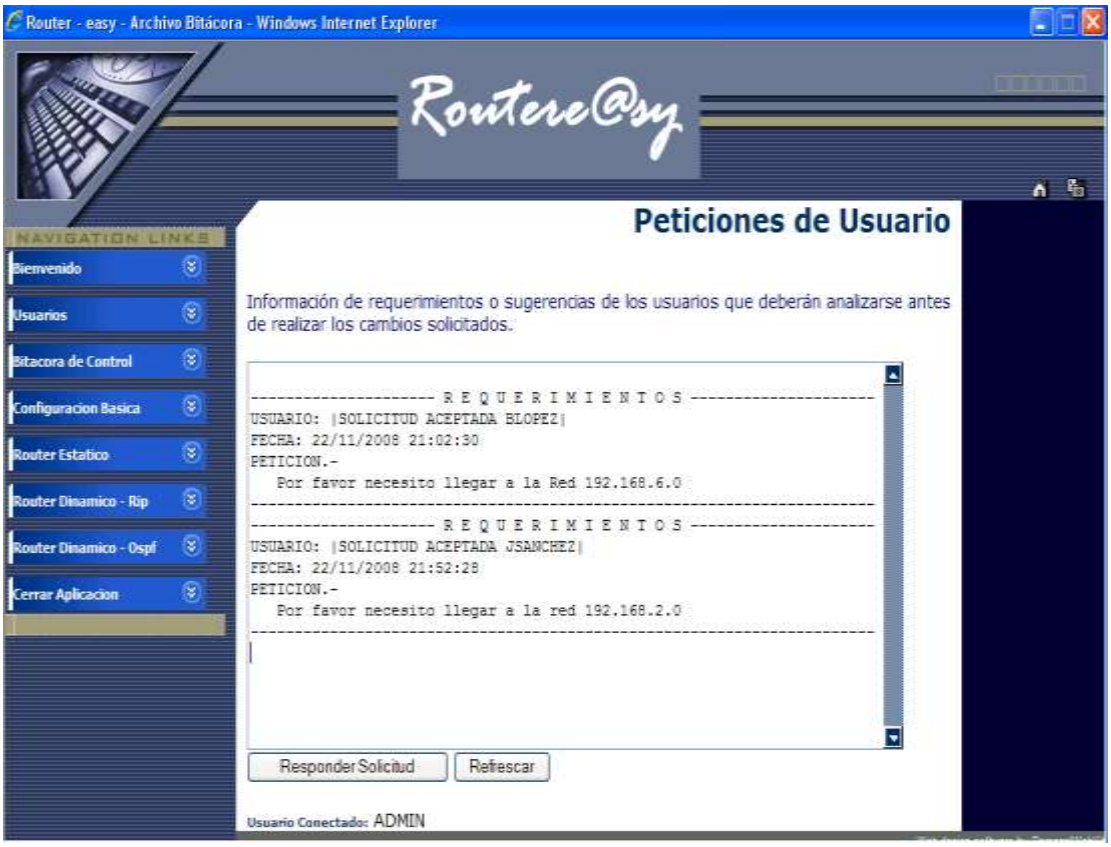

Ilustración 2-27 Peticiones de Usuarios

2.1.4. Configuración Básica

Este llamará la página donde se configurará cada interfaz de red se encuentre en ese momento.

2.1.4.1. Configuración de Tarjetas

Aquí se realizará la configuración de cada una de la interfaces que posee el equipo, automáticamente nuestra aplicación las reconocerá.

Ilustración 2-28 Configuración de Tarjeta de Red

En caso de no haber ingresado la Red completa. Se presentará una alerta para que el Administrador la concluya.

Ilustración 2-29 Alerta de ingreso completo de Red

Ilustración 2-30 Información de Tarjeta de red correcta

En caso de que todo este correcto se emitirá una alerta de que la configuración de tarjeta ha sido un éxito.

Ilustración 2-31 Alerta de configuración de red exitosa

2.1.5. Router Estático

Al momento de ir a este Link se presentará la tabla de Ruteo actual, se podrá observar que existen tres botones en la aplicación.

2.1.5.1. Configuración de Ruteo Estático

El primero botón es Añadir Nueva Ruta.

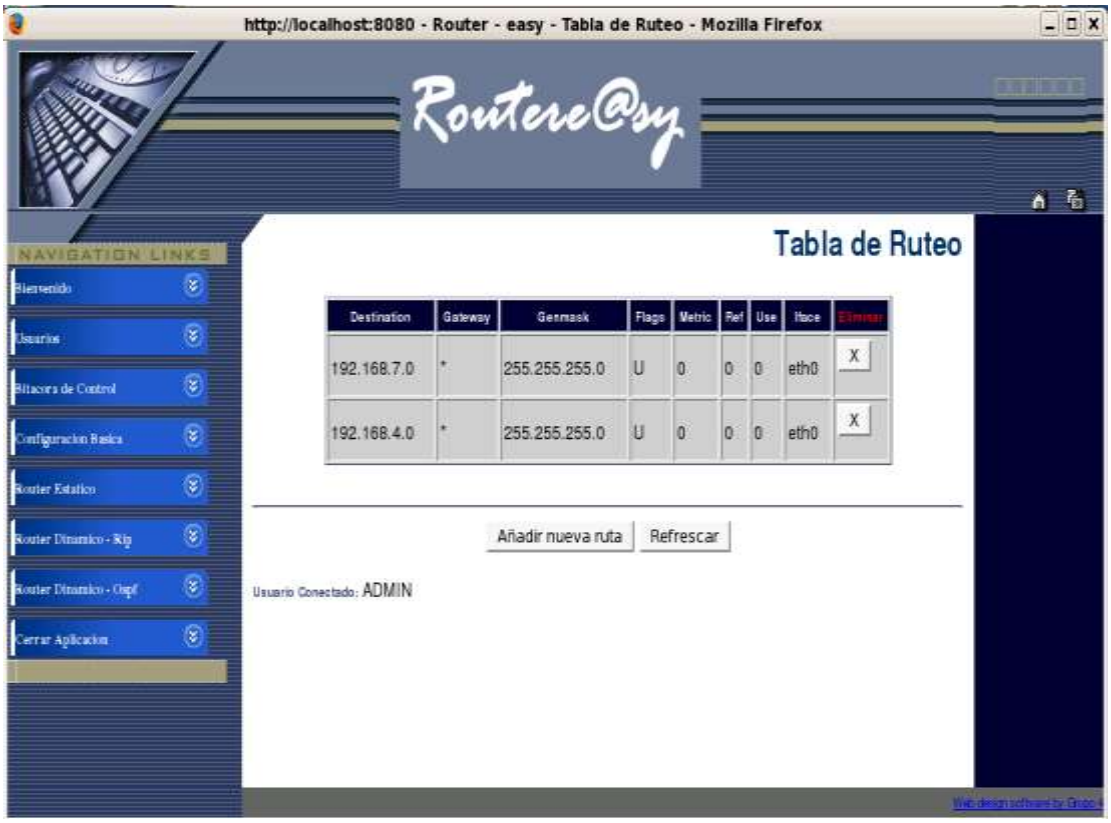

Ilustración 2-32 Tabla de Ruteo

Al momento de presionarlo presentará otra pantalla, en la cual se configurará la red destino.

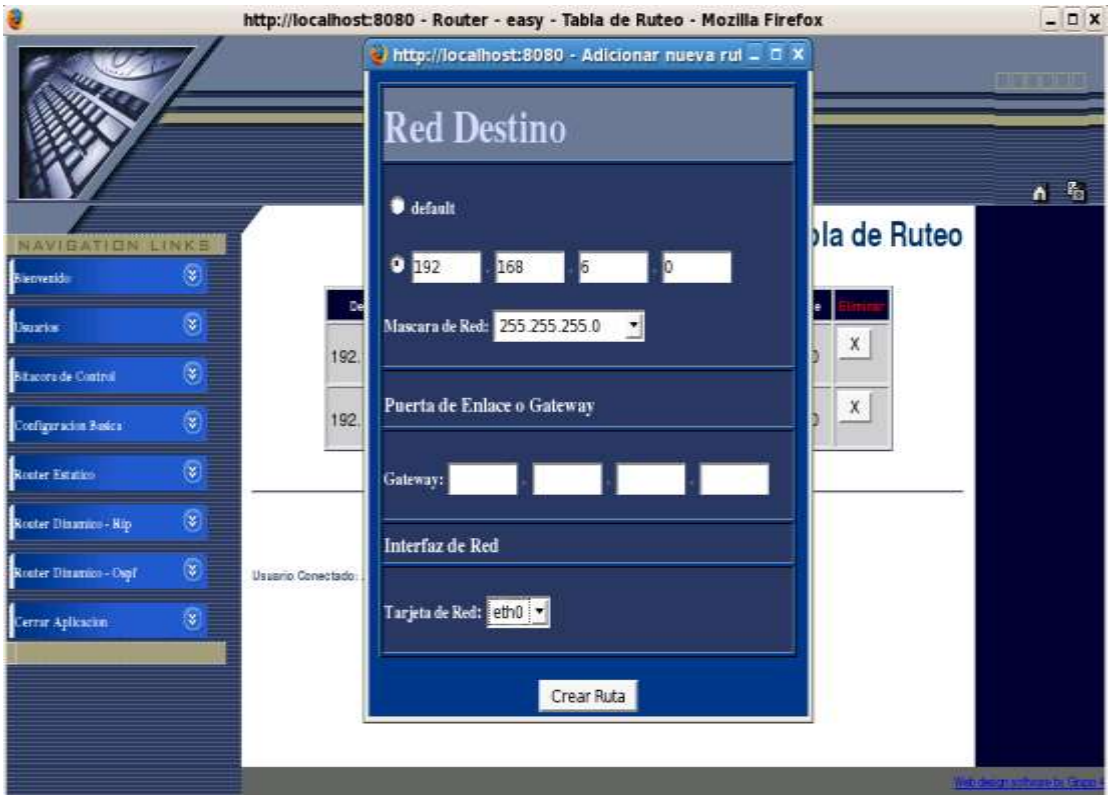

Ilustración 2-33 Adicionar Nueva ruta

Al presionar Crear Ruta en el paso anterior aparecerá una nueva ventana, esta es de confirmación para que el Administrador pueda visualizar cual es la Ruta que agregará si esta de acuerdo presionará si, caso contrario no y saldrá de esta ventana.

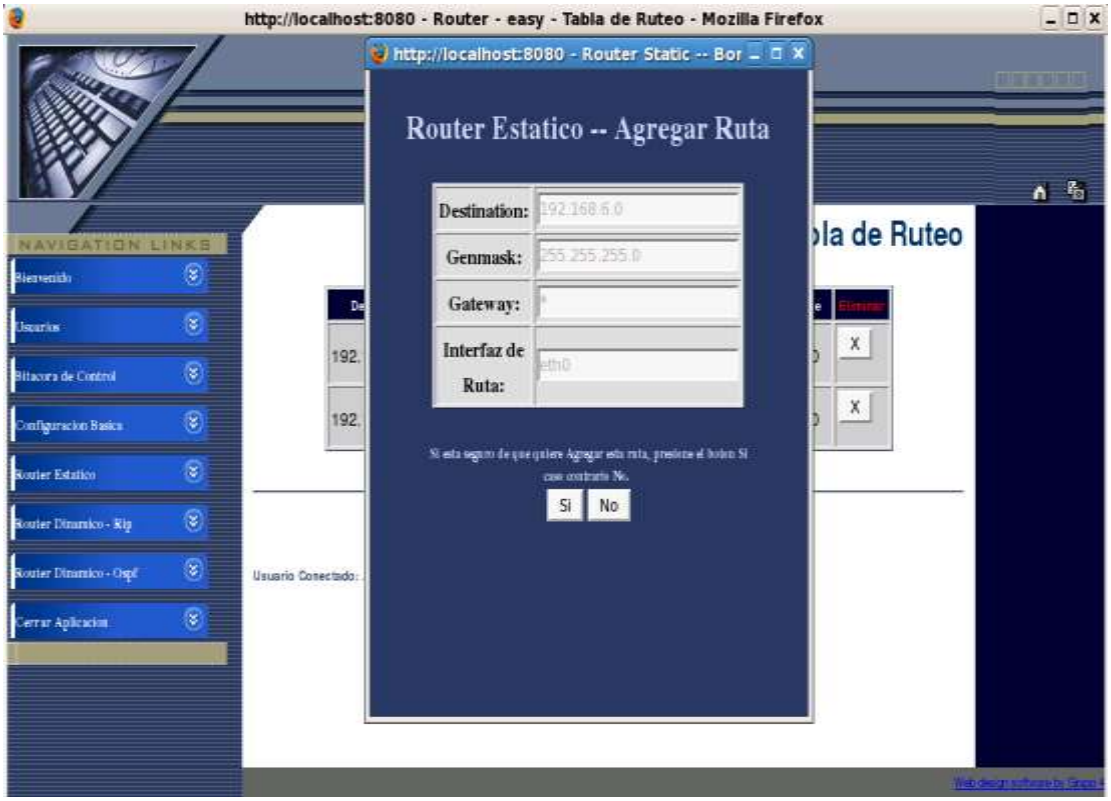

Ilustración 2-34 Confirmación de Ruta

Cuando se ha ingresado todo de manera correcta se presentará una alerta la cual indicará el éxito de la misma.

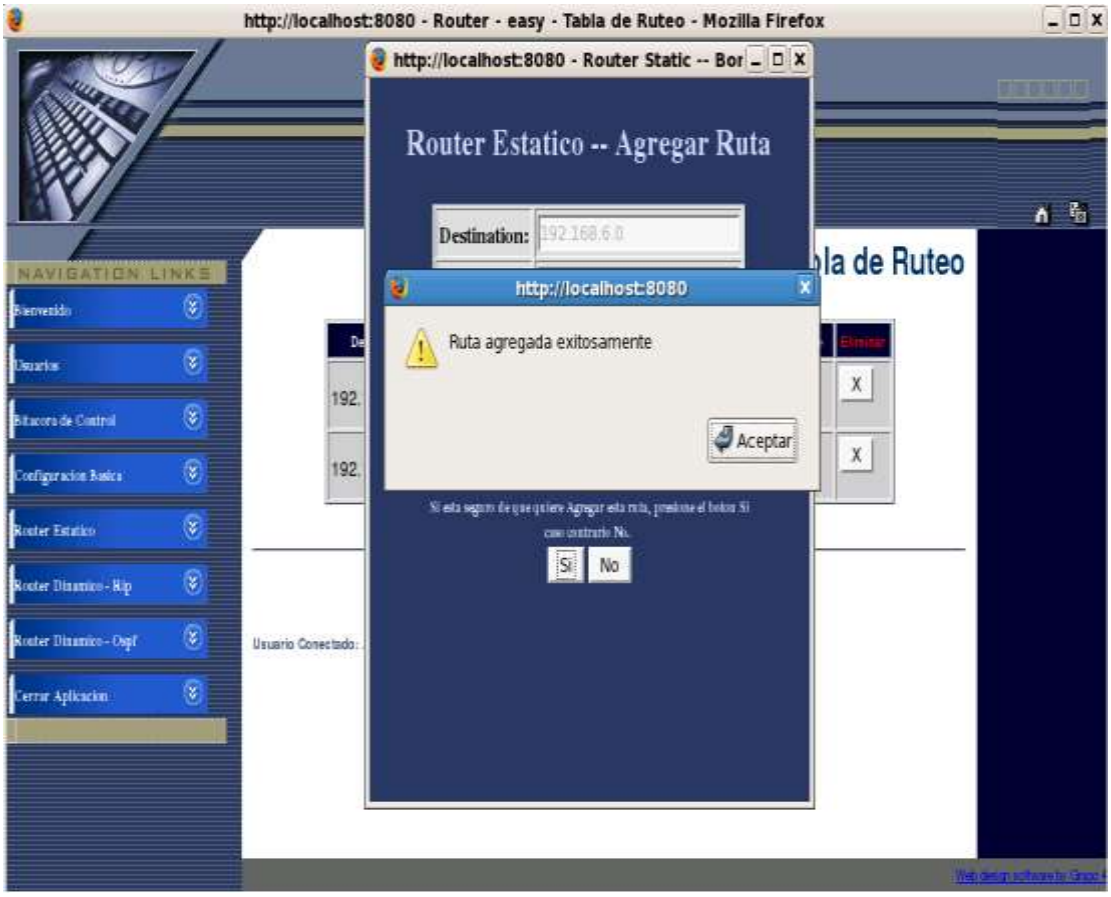

Ilustración 2-35 Alerta de Configuración exitosa

Se regresará a la pantalla inicial, para observar que se adiciono la nueva ruta, para esto se presionará el botón de refrescar, aparecerá en la tabla la nueva ruta.

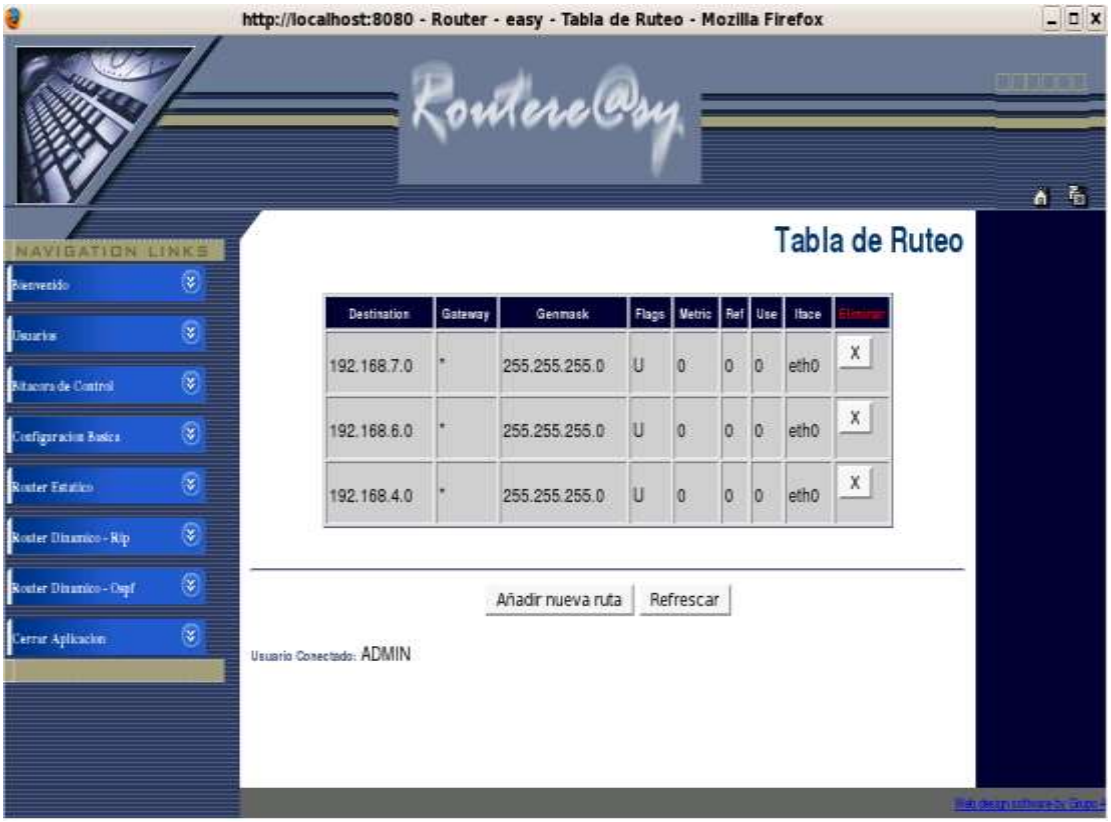

Ilustración 2-36 Tabla de Ruteo con ruta nueva

Para eliminar una ruta solo debemos de presionar en la botón con X que tiene a su derecha, aparecerá una nueva pantalla.

En la nueva pantalla tendremos la ruta especifica a eliminar, esta pantalla solo es de confirmación, si no deseamos borrar esa ruta lo único que tenemos que hacer es regresar a la página donde se encuentra la tabla de Ruteo y eliminar la que deseemos.

Ilustración 2-37 Borrando Ruta

Para eliminar una ruta solo debemos de presionar en la botón con X que tiene a su derecha, aparecerá una nueva pantalla.

En la nueva pantalla tendremos la ruta especifica a eliminar, esta pantalla solo es de confirmación, si no deseamos borrar esa ruta lo único que tenemos que hacer es regresar a la página donde se encuentra la tabla de Ruteo y eliminar la que deseemos.

Ilustración 2-38 Alerta de Ruta borrada exitosamente

Al refrescar la ventana podemos visualizar que la Ruta a sido eliminada

Ilustración 2-39 Ruta borrada

2.1.6. Router Dinámico – Rip

Lo que corresponde a router dinámico, por medio de estas opciones se configurara los archivos ripd.conf y zebra.conf.

2.1.6.1. Control de Servicios Rip

Esta opción servirá para levantar los servicios de ripd y zebra, además desde aquí también se las podrá bajar.

Se la coloco esta opción, porque los servicios pueden estar levantados desde el inicio y es necesario bajarlos.

Ilustración 2-40 Subir servicios Ripd y Zebra

Ilustración 2-41 Bajar servicios Ripd y Zebra

2.1.6.2. Cambio de Host y Pass Rip

Esta opción servirá para cambiar el nombre de Host, además de su password y el enable password, esto se realizará siempre y cuando antes se haya realizado su configuración total.

Se creo esta opción con el fin de que el usuario desee solo cambiar estos ítems, sin realizar ninguna modificación al resto de estos archivos de configuración.

Ilustración 2-42 Alerta de ingreso de toda la configuración

Ilustración 2-43 Alerta tienen que coincidir password y su confirmación

Ilustración 2-44 Alerta no coinciden enable password y su confirmación

Ilustración 2-45 Configuración Completa: host, pass y enable cambiados

2.1.6.3. Configuración de Rip

Esta opción servirá para la Configuración total del rip y zebra.

Se la coloco esta opción, porque los servicios pueden estar levantados desde el inicio y es necesario bajarlos.

Ilustración 2-46 Pantalla ingreso Host, pass, enable, interfaces y neighbor

Cuando todo lo ingresado este correcto, nuestra aplicación nos presentará una alerta de confirmación del evento realizado.

La misma nos mostrara que se actualizo opción servirá para la Configuración total del rip y zebra.

Se la coloco esta opción, porque los servicios pueden estar levantados desde el inicio y es necesario bajarlos.

Ilustración 2-47 Actualización de los archivos Rip y Zebra

El Administrador tendrá la opción de Consultar configuración, el podrá ver el contenido de los archivos reciente de rip y zebra.

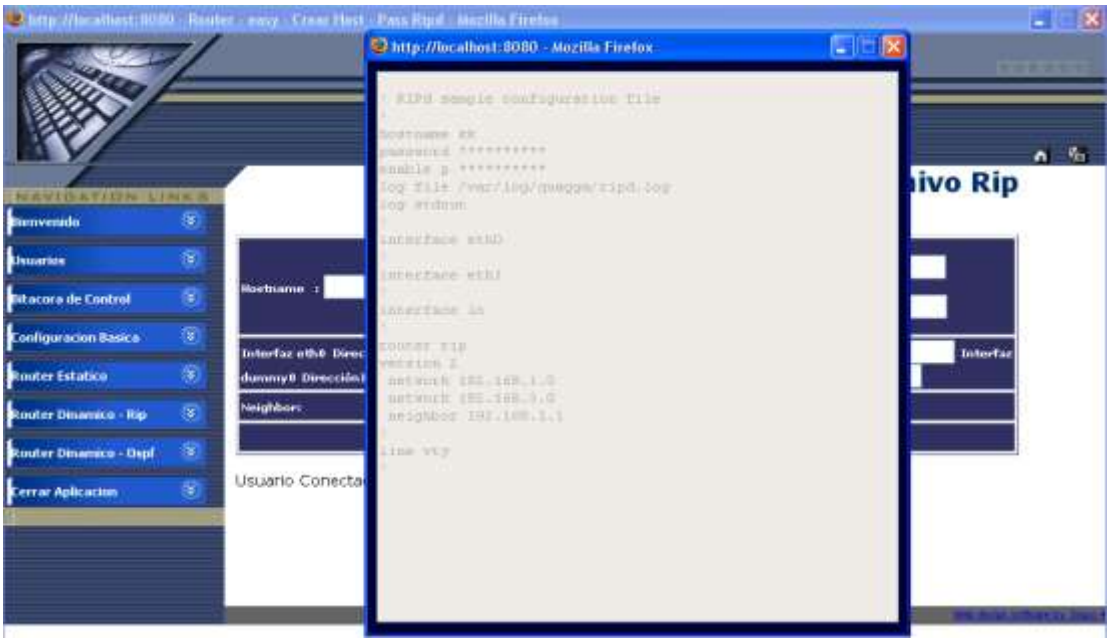

Ilustración 2-48 Consulta archivo ripd

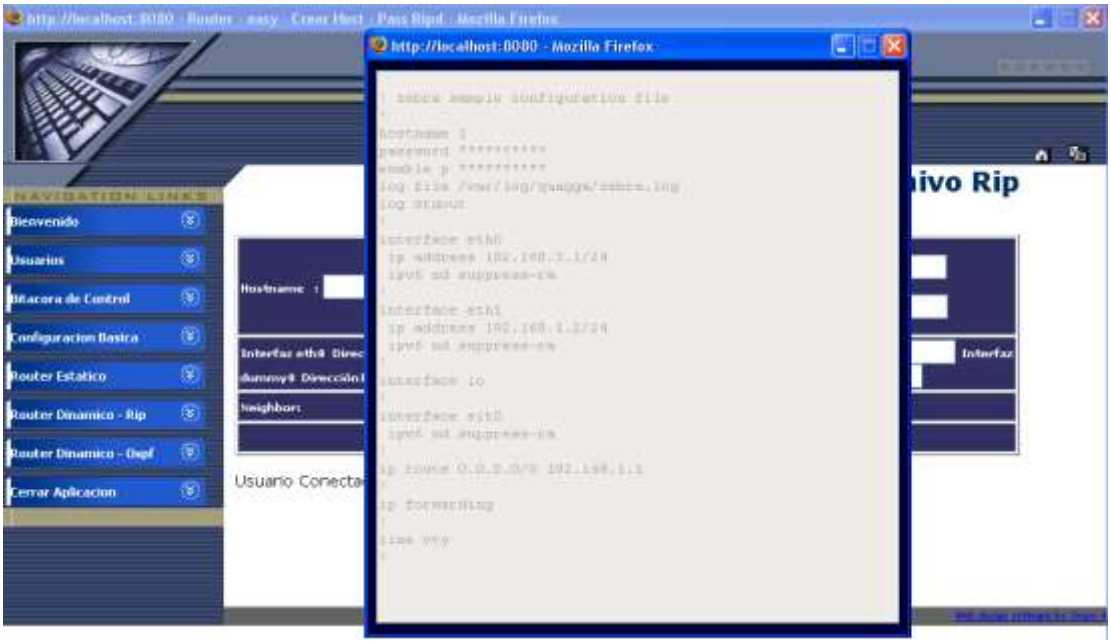

Ilustración 2-49 Consulta archivo zebra

2.1.6.4. Consulta Tabla de Ruteo

Esta opción servirá para ver las rutas.

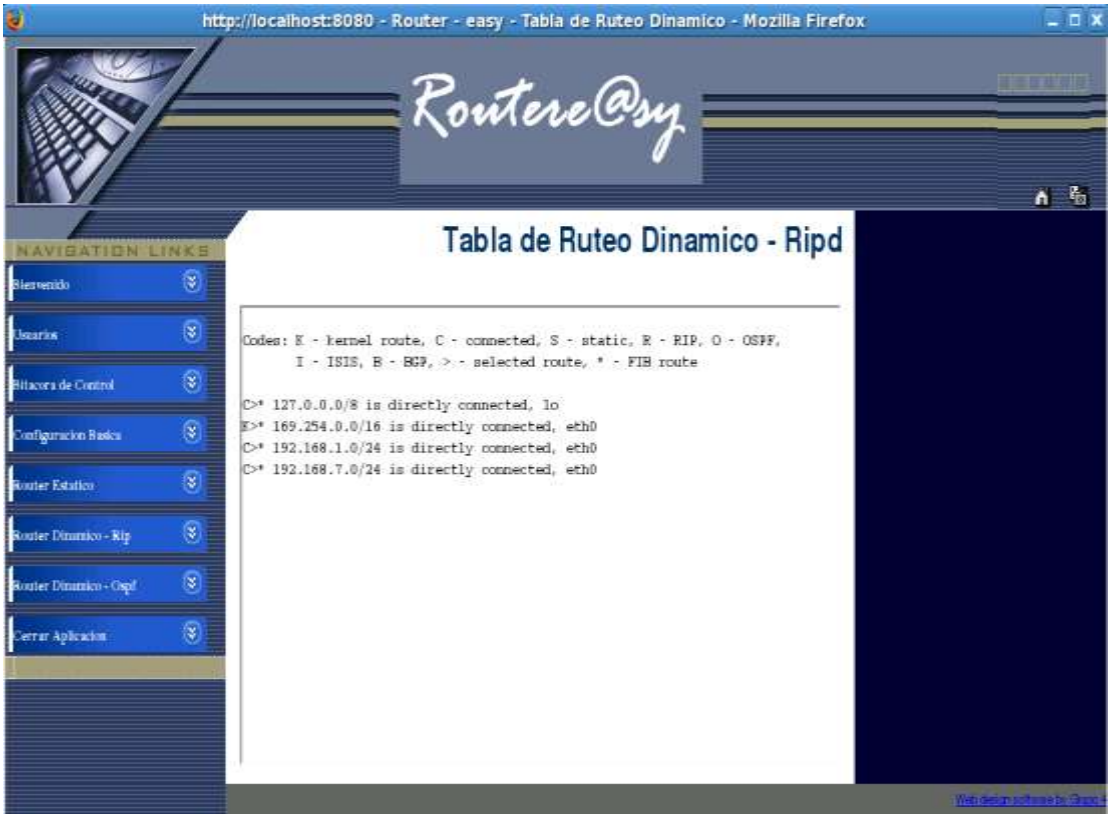

Ilustración 2-50 Consulta Tabla de Ruteo

2.1.7. Router Dinámico – Ospf

Lo que corresponde a router dinámico, por medio de estas opciones se configurara los archivos ospfd.conf y zebra.conf.

2.1.7.1. Control de Servicios Ospf

Esta opción servirá para levantar los servicios de ospf y zebra, además desde aquí también se las podrá bajar.

Se la coloco esta opción, porque los servicios pueden estar levantados desde el inicio y es necesario bajarlos.

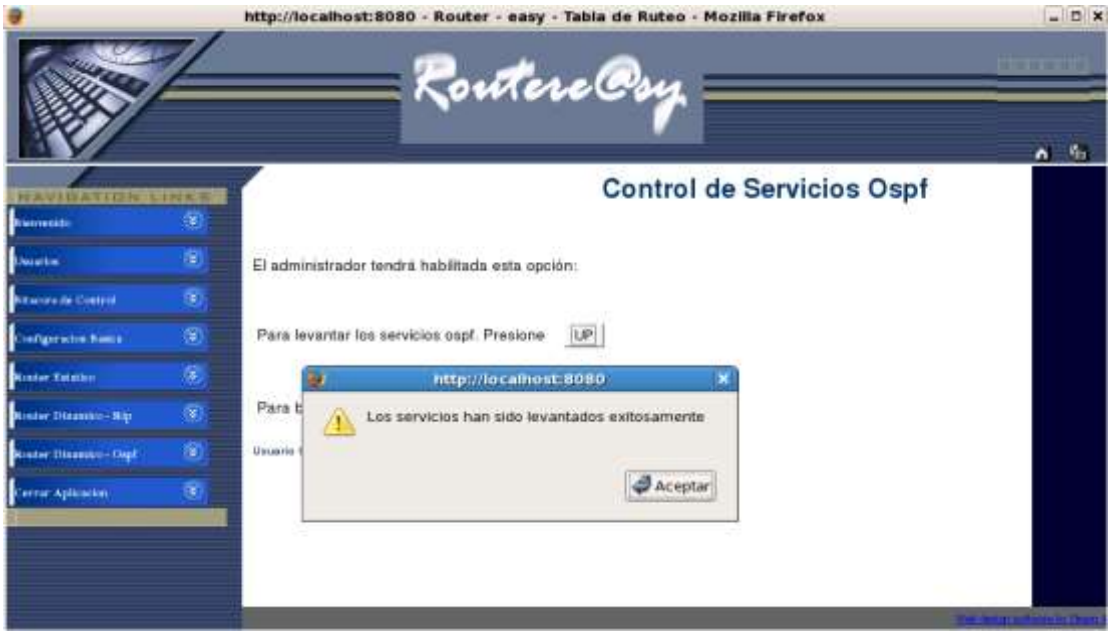

Ilustración 2-51 Subir servicios Ospfd y Zebra

Ilustración 2-52 Subir servicios Ospfd y Zebra

2.1.7.2. Cambio de Host y Pass

Esta opción servirá para cambiar el nombre de Host, además de su password y el enable password, esto se realizará siempre y cuando antes se haya realizado su configuración total.

Se creo esta opción con el fin de que el usuario desee solo cambiar estos ítems, sin realizar ninguna modificación al resto de estos archivos de configuración.

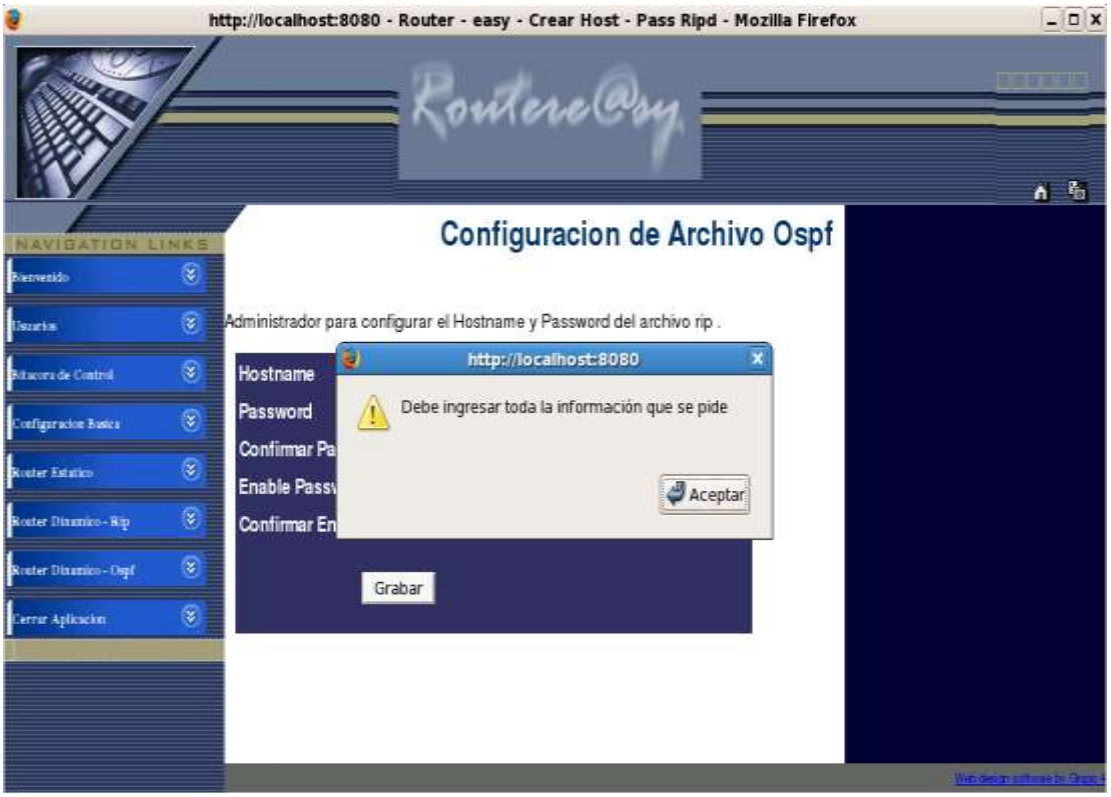

Ilustración 2-53 Alerta de ingrese toda la información

Se presentará una alerta que será informativa, esta se visualizará al momento de que el password y su confirmación no coincidan.

El mismo caso sucederá cuando se ingresa el enable password y su confirmación, también se verá la respectiva alerta.

Ilustración 2-54 Alerta password y su confirmación deben de coincidir

Ilustración 2-55 Ingreso de la configuración

En el momento de que lo ingresado en la caja de texto de la aplicación sea correcto y válido, se verá la siguiente alerta: "configuración completa".

Indicando así que todo ha sido satisfactorio.

Ilustración 2-56 Alerta de configuración completa exitosa

2.1.7.3. Configuración completa

El administrador podrá realizar la configuración completa de los archivo ospfd.conf y zebra.conf mediante esta pantalla.

Podrá configurar el nombre de Host, además del password y el enable password, las interfaces serán visualizadas automáticamente.

Ilustración 2-57 Configuración de los archivos ospfd y zebra

Al presionar el botón Guardar y aplicar, se actualizarán los archivos, para que Administrador se cercioré que se realizo todo sin errores, se le mostrará una alerta.

Ilustración 2-58 Alerta de actualización correcta en los archivos

Luego de configurados los archivos mencionados con anterioridad, el Administrador podrá con solo presionar el botón de Consultar Configuración visualizarlos, para constatar que lo ingresado esta correcto.

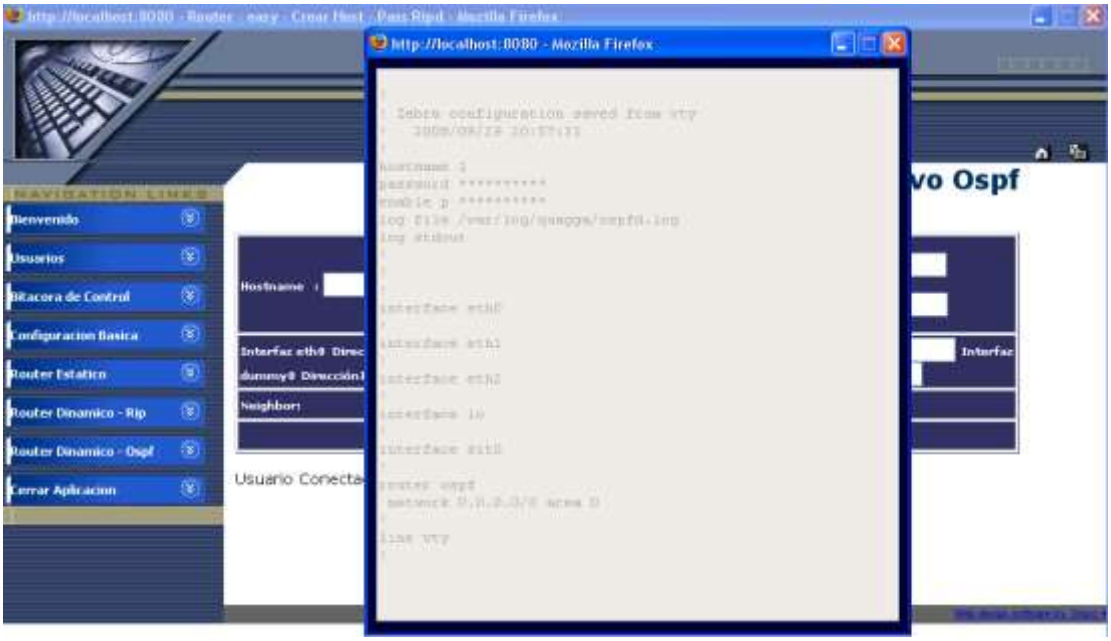

Ilustración 2-59 Visualización del archivo ospfd

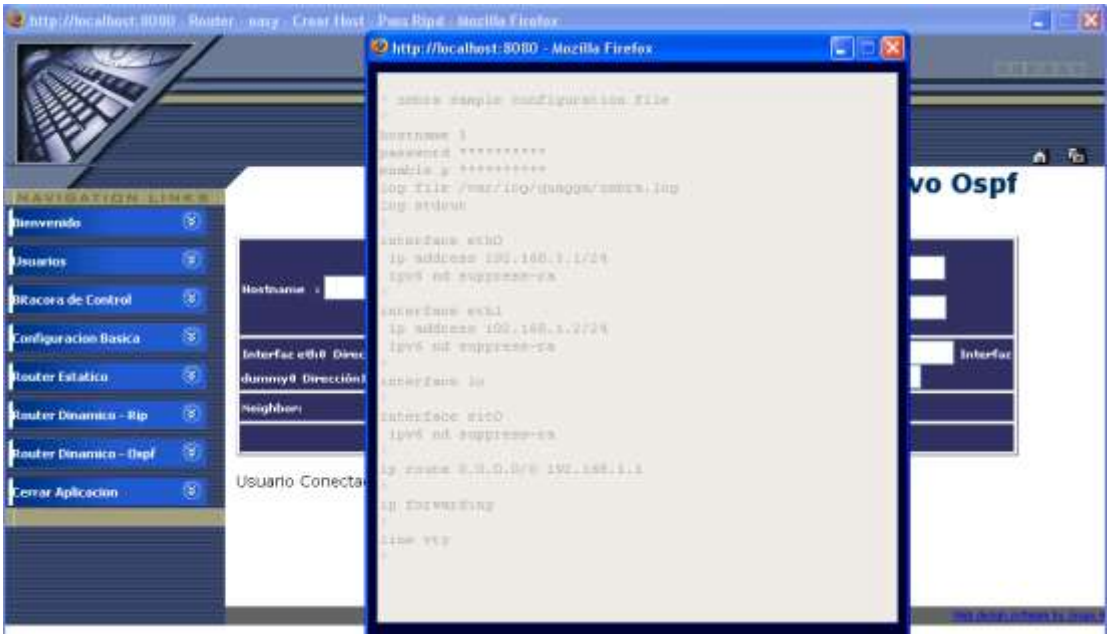

Ilustración 2-60 Visualización del archivo zebra

2.1.7.4. Consulta tabla de Ruteo

El administrador podrá visualizar las rutas actuales, la que comiencen con la letra O , son las que utilizan ospf.

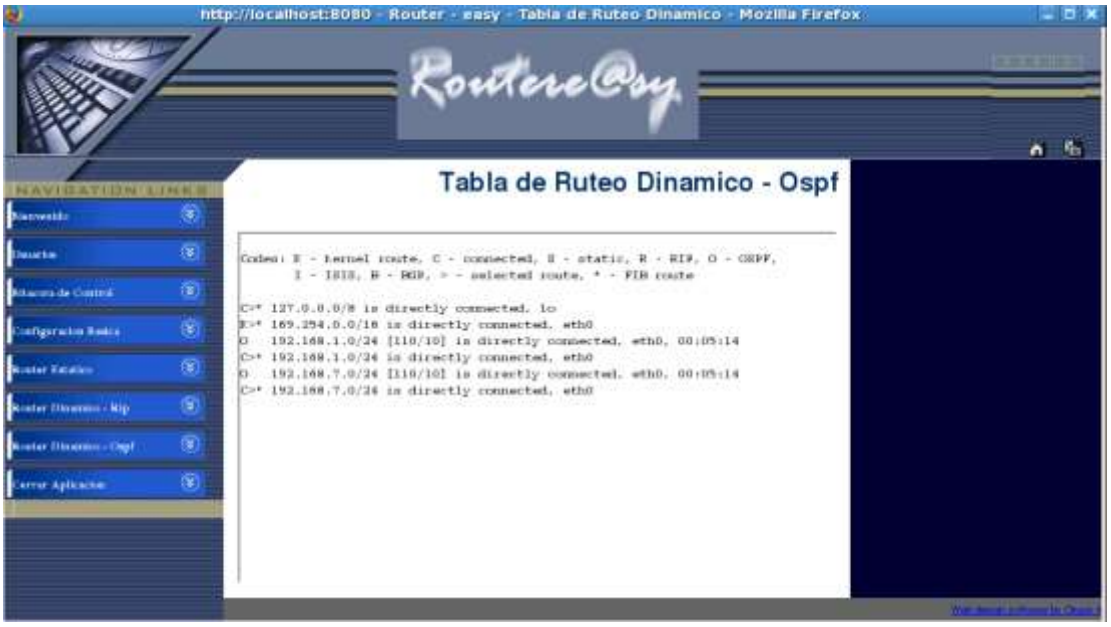

Ilustración 2-61 Tabla de Ruteo Dinámico – Ospf

2.1.8. Cerrar Aplicación

Con esta opción se culmina lo que concierne al menú, como su nombre lo indica, este link sirve para salir de la aplicación.

2.2. INICIO DE SESION DE OPERADOR

El administrador tiene la potestad de crear usuarios, los cuales sería operadores, estos no tienen habilitadas todas las opciones.

La primera pantalla es la de Bienvenida al Operador, el solo podrá visualizar las tablas de Ruteo, además de cambiar su clave y realizar peticiones al administrador.

Se crearan operadores solo para que monitoreen las rutas y si todo esta correcto.

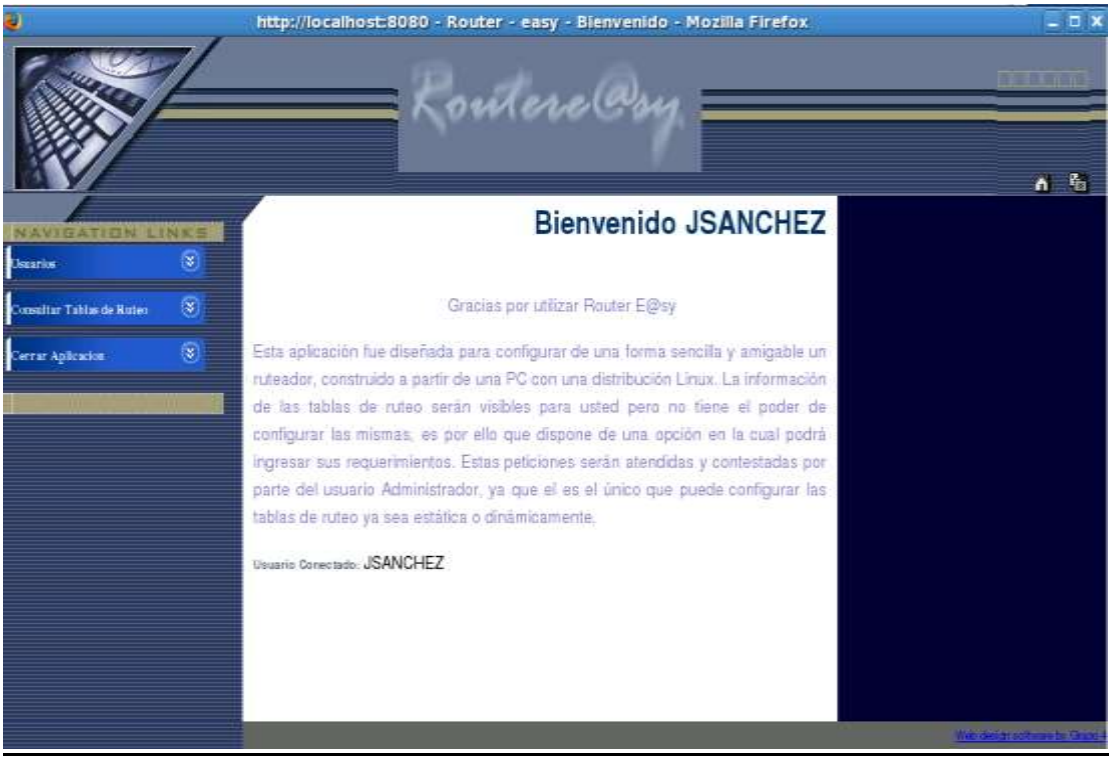

Ilustración 2-62 Pantalla de Inicio de Operador

El operador en su Menú tendrá:

2.2.1. Usuarios

El operador tendrá habilitada las opciones de:

2.2.1.1. Cambiar Clave

Ilustración 2-63 Cambio de clave Operador

Si se presiona Grabar sin haber ingresado nada en las cajas de texto se presentará una alerta para que sean llenados.

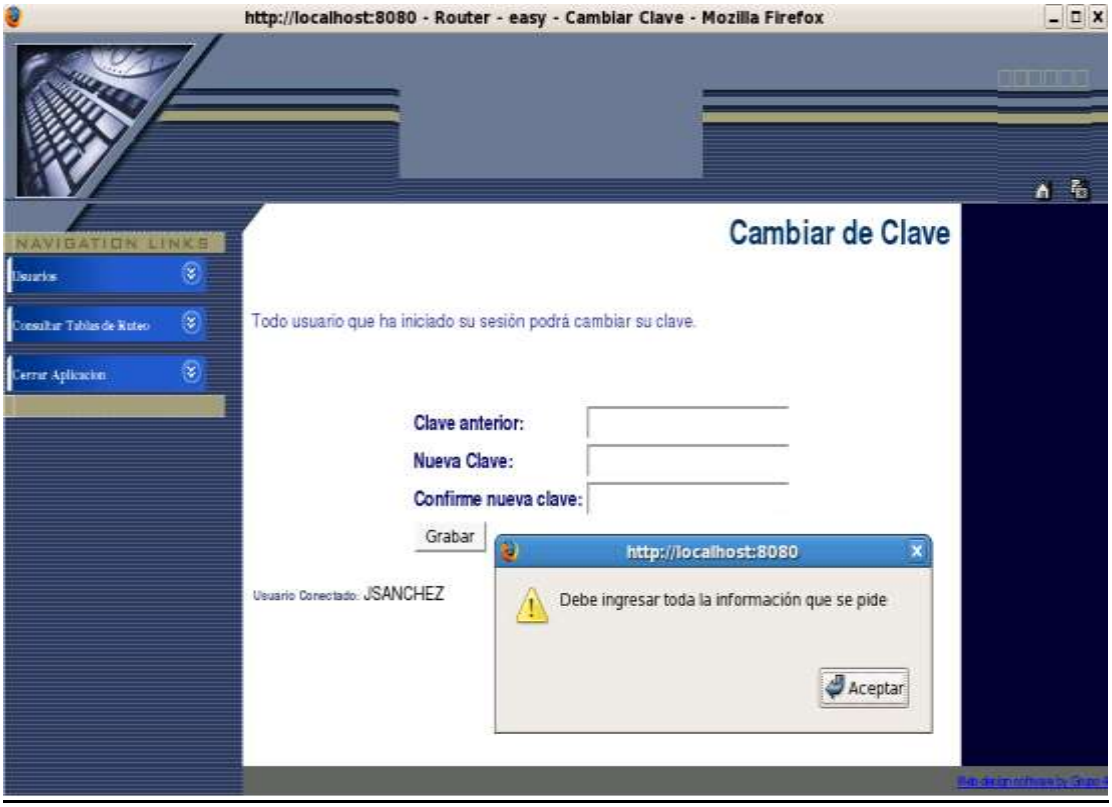

Ilustración 2-64 Alerta de ingreso de información

Si al cambiar la clave se equivoca en colocar la clave anterior, también se le mostrará una alerta descriptiva.

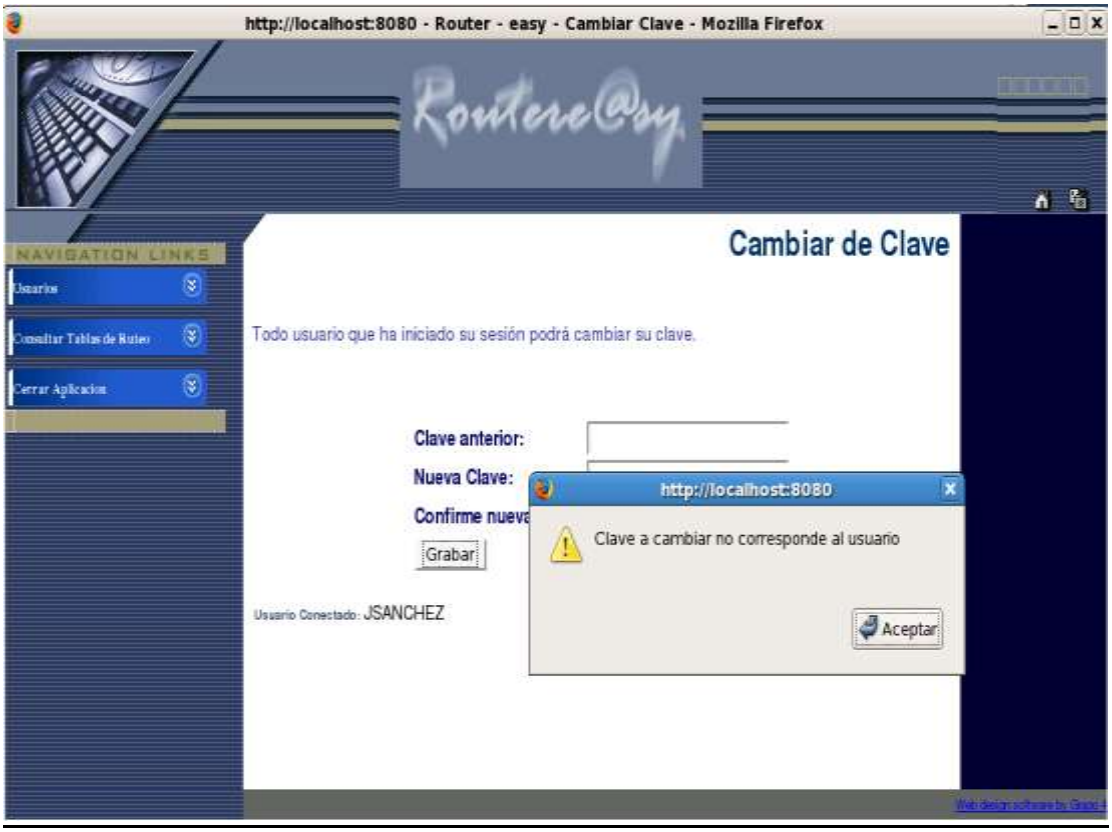

Ilustración 2-65 Alerta de clave anterior no corresponde al usuario

También se mostrará al operador una alerta cuando su nueva clave y confirmación, no coincidan.

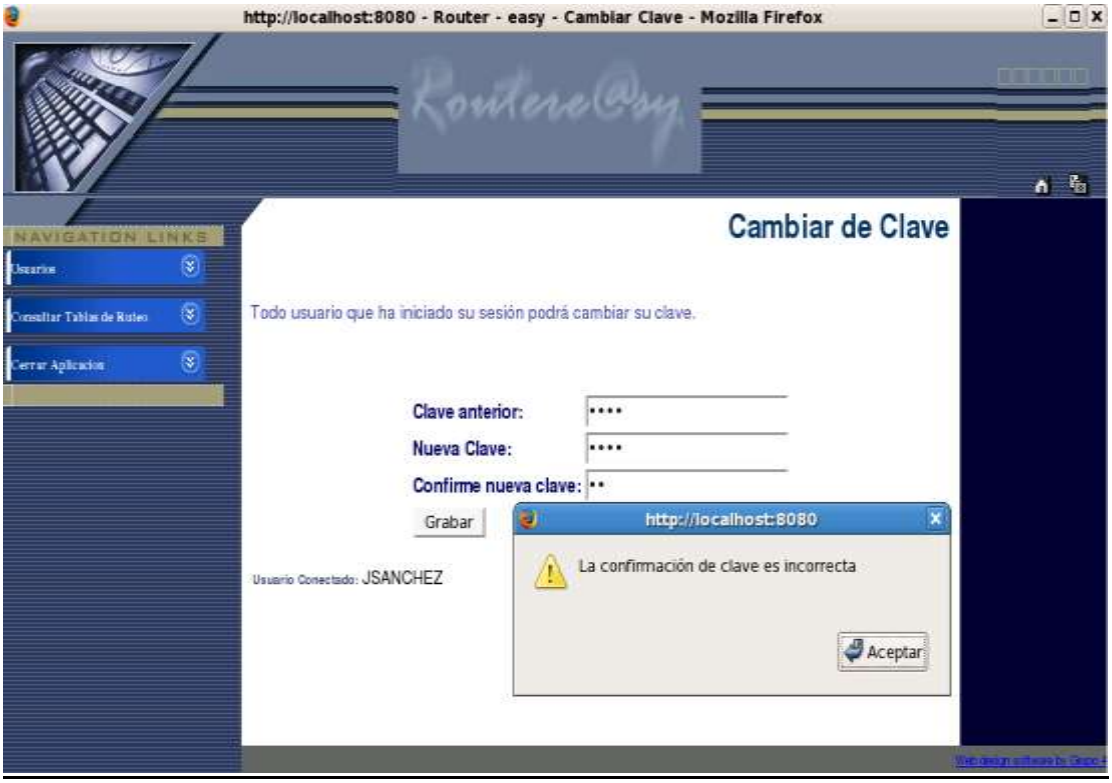

Ilustración 2-66 Alerta nueva clave y su confirmación no coinciden

Cuando todo esta correcto, se mostrará un alerta en la cual indicará que los cambios se han realizado exitosamente.

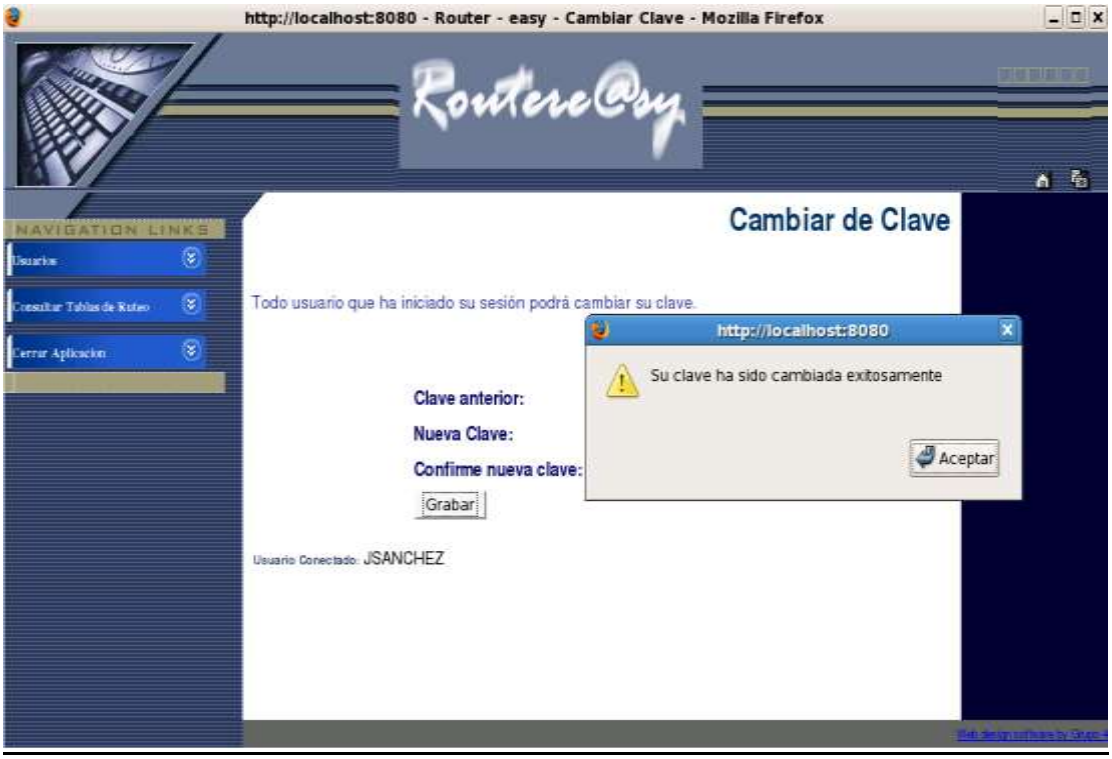

Ilustración 2-67 Alerta de cambio de clave exitosa

2.2.1.2. Peticiones

Cuando el operador necesite algo, o tenga algún inconveniente podrá enviar peticiones al administrador, como por ejemplo llegar a otras rutas.

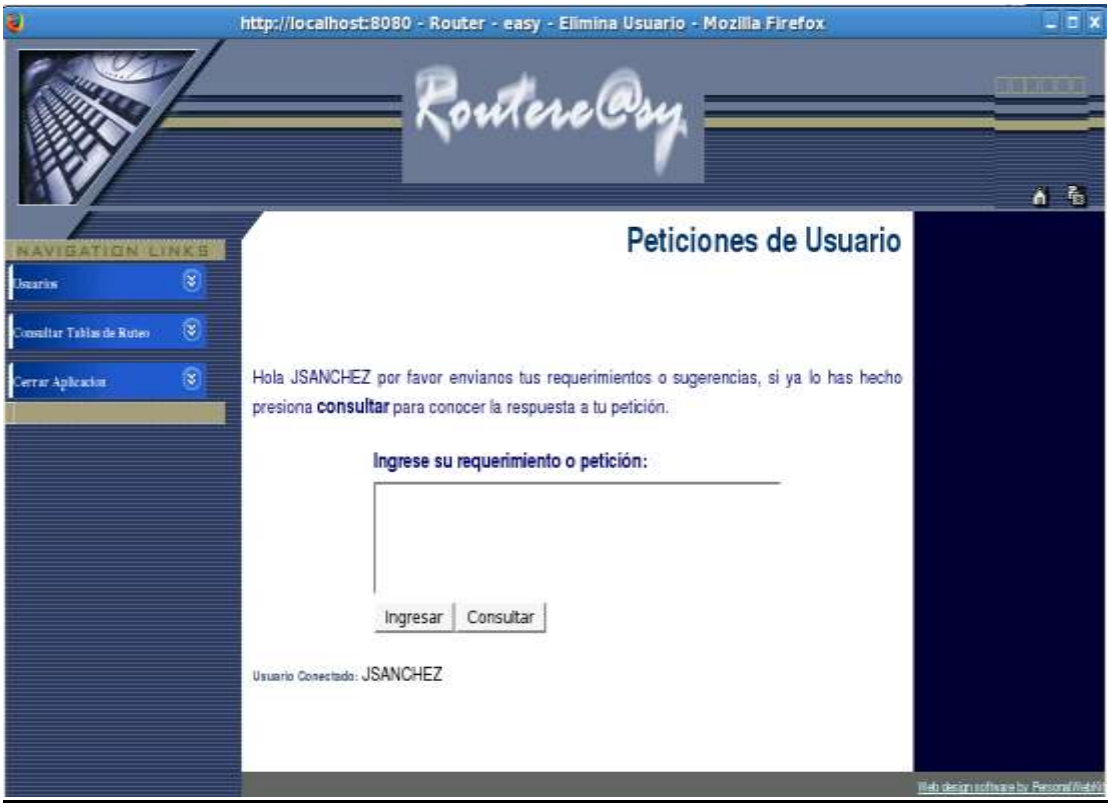

Ilustración 2-68 Ingreso de Requerimiento

Cuando el operador presione el botón ingresar y no ha puesto ningún requerimiento se presentará la siguiente alerta

Ilustración 2-69 Alerta de ingreso de requerimiento

Ilustración 2-70 Petición enviada exitosamente

Cuando el operador haya enviado la petición, podrá visualizar. Aquí observará si ha sido aceptada o rechazada mas una breve descripción.

Ilustración 2-71 Respuesta de Peticiones

2.2.2. Consulta de Tablas de Ruteo

El operador podrá observar la rutas actuales, solo podrá verlas mas no cambiarlas o alterarlas para esto tiene que enviar peticiones al Administrador para que el lo haga.

A continuación en las figuras posteriores observaremos las tablas de ruteo estático y dinámico.

2.2.2.1. Tabla de Ruteo Estático

Dentro de Consultar Tablas de Ruteo se encuentra esta opción, el operador será capaz de visualizar las rutas actuales

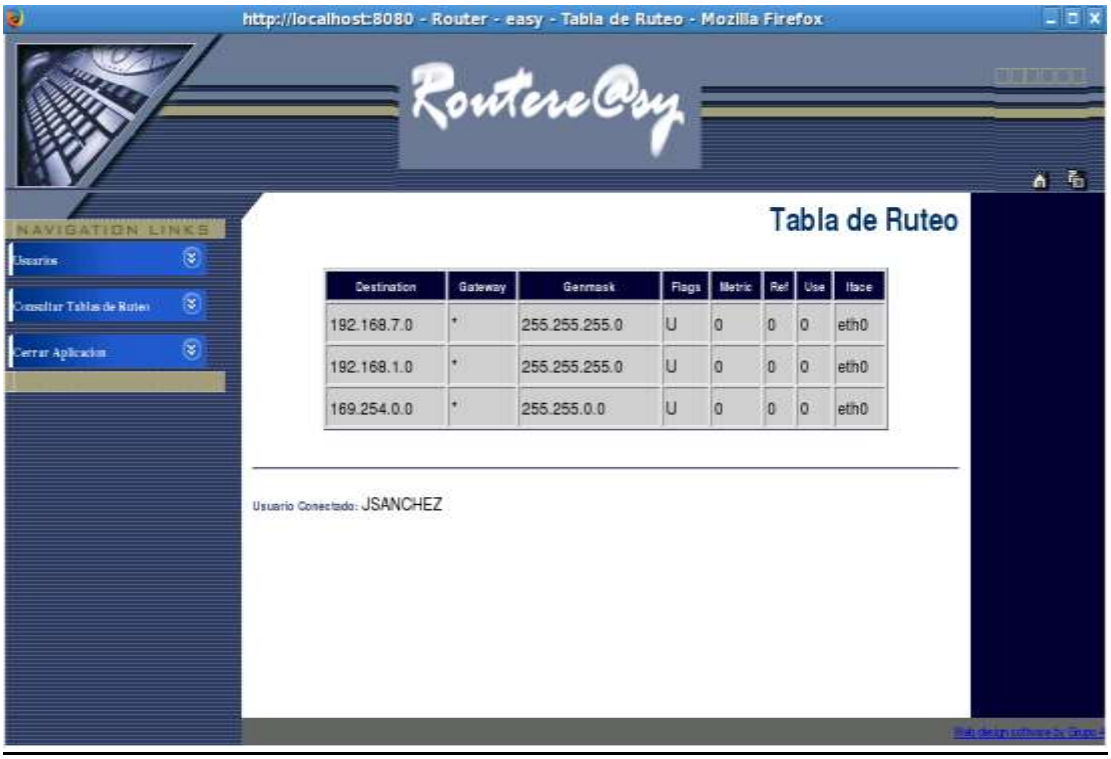

Ilustración 2-72 Tabla de Ruteo Estático

2.2.2.2. Tabla de Ruteo Dinámico

Dentro de Consultar Tablas de Ruteo se encuentra esta opción, el operador será capaz de visualizar las rutas actuales. Ya sea por rip o por ospf.

Ilustración 2-73 Tabla de Ruteo Dinámico