Desarrollo de Aplicaciones Web Utilizando Software Libre: Arquitectura y Recomendaciones

Saúl López Santoyo ls_saul@yahoo.com.mx

Víctor Manuel Jáquez Leal ceyusa@yahoo.com

4 de septiembre de 2001

Índice General

Índice de Figuras

Índice de Tablas

List of Algorithms

Introducción

Hace algunos años, las organizaciones y empresas que contaban con áreas de operación en locaciones geográficamente distantes, tenían que invertir grandes cantidades de dinero y esfuerzo humano para interconectar sus sistemas de información. Esto lo realizaban por medio de costosos enlaces dedicados. Con la masificación de Internet, los organismos fueron adoptando esta estructura como mecanismo para intercambiar información, aunque, en un inicio, con muchas limitaciones. Sin embargo gracias al gran crecimiento de la cantidad de usuarios alrededor del mundo, se han desarrollado diversas tecnologías que tratan de solucionar las necesidades de la mayor cantidad de usuarios posible, permitiendo dar solución a diversas problemáticas, tales como la transferencia de archivos, la comunicación entre personas en tiempo real, administración y uso remoto de sistemas de cómputo, etc., todo esto a un costo relativamente bajo.

Uno de los servicios mas importantes y más usados de Internet en la actualidad es el World Wide Web, éste ha evolucionado a pasos agigantados, primero incorporando en el contenido de las páginas diferentes tipos de letras, imágenes, sonidos etc. y posteriormente, por medio del CGI, lenguajes script y otros métodos, el

usuario interactúa cada vez más con el servidor Web: desde el envío de opiniones acerca de una página determinada hasta el acceso y administración de grandes bases de datos empresariales.

En la actualidad el Web tiene un gran número de usos: Las empresas ponen a disposición del mundo información acerca de los servicios y/o productos que ofrecen, diferentes tipos de organismos informan acerca de sus actividades y establecen comunicación con las personas. El Web se ha convertido en un escaparate de alcance mundial, donde las organizaciones pueden mostrar sus objetivos a practicamente cualquier persona en el mundo a un costo muy bajo.

Uno de las características del Web que tienen más aceptación en la actualidad es el uso de Aplicaciones Web. Las aplicaciones Web son una extensión del protocolo HTTP, el cual permite interactuar con el usuario, tal como si fuera una aplicación de escritorio que se ejecutara en red.

En la actualidad existen diversos modelos de desarrollo de aplicaciones Web cada uno con diversas ventajas y desventajas. Estos modelos utilizan diferentes tecnologías tanto de hardware como de software, y tienen costos de licencias de uso muy variados: desde las propuestas desarrolladas con software que se puede utilizar de manera gratuita, hasta los modelos que usan software con costo de varios miles de dólares.

En este trabajo se propone el uso de una arquitectura que utiliza tecnologías de software de uso libre, del cual, su desempeño y confiabilidad ha sido corroborado por millones de usuarios al rededor del mundo gracias a Internet. Sin embargo,

no existe un compendio de documentación acerca de las tecnologías e implementaciones existentes, así como de la manera de integrarlas.

En el capítulo primero se hablará de las tecnologías que están involucradas en el desarrollo de aplicaciones Web: Arquitectura Cliente/Servidor, el HTTP, HTML, lenguaje de consulta SQL y cifrado de datos. También se mencionarán los diversos paradigmas que existen para el desarrollo de aplicaciones Web: CGI, Co-Servidor, e intérprete incrustado.

En el capítulo segundo, se describirá la estructura de una aplicación Web y la problemática que se presenta en su desarrollo. Se habla de la autenticación de usuarios, manejo de sesiones, el uso de bitácoras, acceso a datos, y el manejo de plantillas.

En el tercer capítulo se habla acerca del diseño de una aplicación Web, desde el análisis de requerimientos, se describen algunos lineamientos para el diseño de la interfaz de usuario, y la creación de un prototipo.

En el capítulo cuarto, se habla de las herramientas de software necesarias, desde el manejador de bases de datos, el lenguaje Perl, etc. Además se muestran ciertos lineamientos para el diseño de la base de datos y el acceso a ella. Por último se describen los procedimientos principales que están involucrados en las aplicaciones Web: la autenticación de usuarios, el manejo de sesiones y el despachador de mensajes.

En el quinto capítulo se muestra una aplicación Web completa, desarrollada bajo los lineamientos del documento. En ella se incluye todo el código fuente así como

la manera de configurar el servidor HTTP para que pueda servir a la aplicación. Por último se habla acerca de algunas herramientas que podemos utilizar para probar el desempeño de la aplicación bajo fuertes cargas de trabajo.

Antecedentes

A principios de los años sesenta, con la aparición de las computadoras digitales de uso general, la tensión producida por la guerra fría y el acelerado crecimiento en los medios masivos de comunicación, comenzó a germinar la idea de transmitir datos a través de los primeros *mainframes* a muy largas distancias. Fue al final de 1969 cuando, bajo el auspicio de DARPA[1], se puso en funcionamiento el primer nodo de *arpanet* en la UCLA (Universidad de Los Ángeles California). La idea de unir a las computadoras y medios de transmisión de los datos a altas velocidades y de manera confiable, comenzó a extenderse de manera insospechada: en menos de dos décadas el *arpanet* dejó su lugar a los que ahora conocemos como Internet, para masificarse a tal punto, que podría compararse con otros medios de comunicación de mayor arraigo como la televisión.

En 1991, Tim Berners-Lee, en los laboratorios del CERN[2], saca a la luz pública una nueva forma de transmitir información escrita a través de computadoras: el

[1]Defense Advanced Research Projects Agency (Agencia de la Defensa para la Investigación de Proyectos Avanzados)

[2]Organisation Européenne pour la Recherche Nucléaire (Organización Europea para la Investigación Nuclear).

hipertexto. Aunque no totalmente de su autoría, sí propuso un protocolo (HTTP[3]) y un lenguaje de marcado (HTML[4]) muy específicos, que permitieron el desarrollo de software orientados hacia esta aplicación. A la información que circulaba por Internet mediante este protocolo y este lenguaje se le conoce como World Wide Web, o simplemente WWW.[1]

El crecimiento de Internet, en gran parte gracias al empuje ejercido por el World Wide Web, fue exponencial. Simplemente observemos que en 1993 había únicamente 130 sitios WWebeb o servidores Web, y para octubre del 2000 se contabilizaban 22,287,727 sitios Web; es decir un crecimiento del 17,000% en sólo 7 años.[2]

El mismo World Wide Web no se ha estancado en su idea original: mostrar información estática, previamente escrita; ha evolucionado hacia la generación de información a partir de una solicitud no determinada, como lo podemos observar en los motores de búsqueda, y en la banca electrónica.

Internet fue desarrollado dentro de una comunidad académica, entre universidades y centros de investigación. En estos lugares, el desarrollo de software se comporta de manera muy diferente a lo que estamos acostumbrados dentro del entorno comercial: el software no se escribe con el fin específico de obtener una ganancia económica. Se escriben programas para resolver los problemas de investigación y se comparten estos programas con los colegas para ayudarles a resolver los suyos y viceversa.

[3]Hipertext Transfer Protocol (Protocolo de Transferencia de Hipertexto).
[4]HyperText Markup Language (Lenguaje de Marcado de HiperTexto).

El software con el cual se erige Internet, es descendiente directo de éste software académico y de investigación, fundamentado en licencias poco restrictivas y respetuosas de la libertad de los usuarios de dicho software[3]. Con el avance del tiempo y de Internet, esta idea no se ha perdido ni debilitado, sino todo lo contrario, el Software Libre (como se le conoce al software liberado bajo las licencias anteriormente citadas) ha tomado importancia en casi todos los ámbitos donde el software en general es utilizado. El Software Libre ha tenido un gran desarrollo relacionado con el área de servidores: aplicaciones de servidores de nombres, servidores de HTTP, de correo electrónico han tenido muchísimo éxito tanto en el desarrollo académico como en el comercial.

Como ya se señaló, el WWW ha evolucionado de maneras inicialmente insospechadas. Originalmente el HTTP y el HTML fueron desarrollados con el objetivo de que los científicos pudieran compartir sus reportes y documentos en general con otros colegas y asistentes. Pero con la creciente popularidad, rápidamente comenzaron a aparecer nuevas formas de utilizar el WWW, en base a las distintas posibles solicitudes de un cliente permitidas por el protocolo.

Actualmente existen dos tipo de sitios Web: las que se comportan como revistas, donde uno simplemente lee la información ahí contenida; y las que se comportan como el software, donde uno hace un grupo de tareas específicas[4]. Estas últimas se conocen como aplicaciones Web, y son desarrolladas por grupos de desarrollo de software, tal como en las aplicaciones de escritorio.

Una de las formas más populares para usar el WWW es a través de las aplicacio-

nes Web. Es la razón de la aparición de un nuevo modelo de negocios en base a estas aplicaciones: los ASP[5], los cuales ofrecen masivamente el uso de sus aplicaciones a través del WWW; y, por el otro lado el crecimiento de e-commerce, en su orientación tanto B2B[6] como B2C[7].

La creciente popularidad de las aplicaciones Web se debe a sus múltiples ventajas, entre las cuales podemos citar:

- **Multiplataforma**: Con un solo programa, un único ejecutable, las aplicaciones pueden se utilizada a través de múltiples plataformas, tanto de hardware como de software.
- **Actualización instantánea**: Debido que todos los usuarios de la aplicación hacen uso de un sólo programa que radica en el servidor, los usuarios siempre utilizarán la versión más actualizada del sistema.
- **Suave curva de aprendizaje**: Los usuarios, como utilizan la aplicación a través de un navegador, hacen uso del sistema tal como si estuvieran navegando por Internet, por lo cual su acceso es más intuitivo.
- **Fácil de integrar con otros sistemas**: Debido a que se basa en protocolos estándares, la información manejada por el sistema puede ser accedida con mayor facilidad por otros sistemas.

[5]Application Service Provider (Proveedor de Servicios de Aplicación).
[6]Business to Business (Empresa a Empresa). V.gr. Manufacturera a Proveedores.
[7]Business To Consumer (Negocio a Consumidor). V.gr. Expendio de CD-Comprador Individual.

- **Acceso móvil**: El usuario puede acceder a la aplicación con la única restricción de que cuente con un acceso a la red privada de la organización o a Internet, dependiendo de las políticas de dicha organización; puede hacerlo desde una computadora de escritorio, una laptop o desde un PDA; desde su oficina, hogar u otra parte del mundo.

Definición del problema

Ya se ha dejado asentado que el desarrollo de aplicaciones Web está siendo utilizado en muchas organizaciones y que esta situación va ir creciendo indefinidamente. Es por ello que día a día se requieran más programadores capacitados para desarrollos basados en el WWW.

Sin embargo, el número de tecnologías que se integran para hacer este tipo de desarrollos, es muy alto y muy independientes entre sí. Para lograr un buen desarrollo es necesario conocer profundamente la mayoría de éstas y dominar las herramientas que se cuentan para manejarlas.

Las herramientas que ofrece el Software Libr*e* para implementar, en las aplicaciones, las tecnologías involucradas en el desarrollo de aplicaciones Web, han demostrado poseer una gran calidad y ser muy completas. Sin embargo, siguiendo la filosofía del diseño de Unix, estas herramientas están construidas para interactuar con otras herramientas a través de interfaces basadas en texto, permitiendo crear aplicaciones complejas a través de estos componentes o herramientas[5]. Sin embargo esta interacción no siempre es clara para el programador, y cuando se busca

documentación al respecto, casi siempre es escasa, motivo por el cual el programador tiene que recurrir a los foros de discusión, deteniendo o ralentizando así el ciclo de desarrollo.

Por otro lado, el desarrollo de aplicaciones Web es una nueva forma de desarrollo, lo cual implica retos no enfrentados previamente por los programadores, administradores de proyectos, analistas de sistemas, etc. Las aplicaciones Web cambian profundamente los conceptos preestablecidos acerca de la programación de aplicaciones para escritorio. Cuestiones de seguridad, multiprocesamiento, presentación de datos, etc., toman un giro diferente, que puede tomar desprevenido a la persona que desea participar en este tipo de desarrollos.

Objetivo general

Con el presente trabajo, se documenta y se estructura el desarrollo de aplicaciones Web utilizando Software Libre, más específicamente Perl, Apache y PostgreSQL.

Objetivos específicos

- Proporcionar un panorama de los estándares, protocolos y lenguajes de programación involucrados en el desarrollo de aplicaciones Web.
- Proponer procedimientos y pequeñas metodologías para diseñar aplicaciones Web.
- Definir los problemas que tiene que solucionar el programador en estos desarrollos tales como autenticación de usuarios, manejo de sesiones, cifrado de datos, entre otros.
- Se ejemplificará las soluciones a los problemas con fragmentos de código de sistemas reales.

Justificación

El desarrollo de aplicaciones Web crece día con día, existe gran cantidad de tecnologías que resultan abrumadoras para el desarrollador al momento de elegir cual usar y cómo hacerlo. Con el uso de herramientas de Software Libre, se obtienen muchos beneficios, entre ellos el acceso al código fuente, por tanto su funcionamiento está bajo el escrutinio de toda una comunidad de usuarios alrededor del mundo; si se detecta algún error o problema de seguridad en algún programa, éstos son corregidos en un tiempo muy corto; y los requerimientos de hardware son relativamente bajos.

Además de la libertad y flexibilidad que se logra usando Software Libre, otro factor de peso que orienta a la elección de este tipo de software es que la mayoría de los programas desarrollados bajo este paradigma, son gratuitos, por lo que al momento de realizar la instalación de ellos en una computadora, no hay que preocuparse por los costos de licencia de uso (que en muchos casos son prohibitivos) que tiene el software propietario.

Sin embargo, la documentación específica para el desarrollo de aplicaciones Web

es muy escasa. A pesar de que existe documentación para las herramientas que pueden ser utilizadas, es muy independiente de las demás, no existe un compendio que nos diga cómo integrar cada una de ellas en un desarrollo Web.

Con este documento se pretende mostrar de manera formal, la arquitectura de una aplicación Web en general, además de ciertos lineamientos para la integración de las tecnologías involucradas.

Hipótesis

Sistematizando el desarrollo de aplicaciones Web con herramientas de Software Libre, se pueden mostrar y entender los factores que participan en este tipo de desarrollos y la manera como son integrados, reduciendo la complejidad y simplificando su desarrollo.

A su vez, distinguiendo la arquitectura propia de Web y siguiendo los lineamientos presentados en este trabajo, se pueden desarrollar aplicaciones Web confiables, seguras, funcionales y con un costo de desarrollo e implementación más bajo del que se tendría usando software propietario, además de que se asegura la compatibilidad en diferentes plataformas, tanto de lado del servidor como del cliente.

Capítulo 1

Generalidades

Este capítulo habla acerca de las tecnologías y protocolos involucrados en el desarrollo de aplicaciones Web. Primero se revisará los conceptos de la arquitectura Cliente/Servidor, con lo cual se tiene una mejor comprensión de cómo funcionan las aplicaciones Web. Se analizará algunos de los protocolos y lenguajes que son el fundamento de las aplicaciones Web: el HTML, y su medio de transporte: el HTTP. También se hablará acerca del lenguaje SQL, el cual es imprescindible si es que se desea desarrollar aplicaciones con acceso a bases de datos. Por último, se enumerará algunos paradigmas que han surgido para la programación para el Web, los cuales tienen diferentes mecanismos y metodologías para realizar este tipo de desarrollos.

El objetivo de este capítulo no es hacer una descripción minuciosa de cada tecnología, sino simplemente dar una descripción general, ya que se puede ahondar

más en los temas utilizando la bibliografía señalada.

1.1 Arquitectura cliente/servidor

En la década de los ochenta las organizaciones tenían que elegir entre computadoras personales o grandes *mainframes* para su procesamiento de datos. Las computadoras personales, aunque económicas y cada vez más poderosas, carecían de los servicios que un *mainframe* o una minicomputadora podía ofrecer; uno de los principales fue compartir información de manera instantánea entre los diferentes usuarios del sistema.

Para hacer que las computadoras personales compartieran información se crearon las redes, a las que también se integraron los mainframes y se comenzó a utilizar, a finales de la década de los ochentas, el término "cliente/servidor", refiriéndose al software que utiliza, ya sea de manera real o metafórica, de computadoras físicamente separadas para realizar sus tareas. La arquitectura cliente/servidor promete facilidad de uso, flexibilidad, interoperabilidad y escalabilidad.

De manera formal, se define el término en dos componentes: un cliente, como un solicitante de servicios y un servidor, como el proveedor de servicios. Una sola máquina puede ser tanto un cliente como un servidor dependiendo de la configuración del software[6].

En la arquitectura de software cliente/servidor ha evolucionado en pos de la resolución de distintos problemas. Entre los modelos de esta arquitectura de software

más comunes y en orden de complejidad se pueden encontrar:

- Arquitectura de servidor de archivos
- Arquitecturas de dos capas
- Arquitecturas de tres capas
- Arquitectura de tres capas con tecnología de monitoreado en el procesamiento de transacciones
- Tres capas con servidor de mensajes
- Tres capas con un servidor de aplicaciones
- Tres capas con arquitectura ORB[1]
- Arquitectura de empresa distribuida/colaborativa

En las aplicaciones Web se utiliza desde las aplicaciones de dos capas en adelante. Con el fin de acotar el alcance de este trabajo, se manejará únicamente las arquitecturas de dos y tres capas.

1.1.1 Arquitecturas de software de dos capas

Las arquitecturas de dos capas consisten de tres componentes distribuidos en dos capas: cliente (solicitante de servicios) y servidor (proveedor de servicios)[7]. Los tres componentes son:

[1]Object Request Broker (Corredor de Peticiones de Objetos).

1. **Interfaz de usuario al sistema**. Tales como una sesión, entradas de texto, desplegado de menús, etc.

2. **Administración de procesamiento**. Tales como la ejecución de procesos, el monitoreado de los mismos y servicios de procesamiento de recursos.

3. **Administración de bases de dato**s. Tales como los servicios de acceso a datos y archivos.

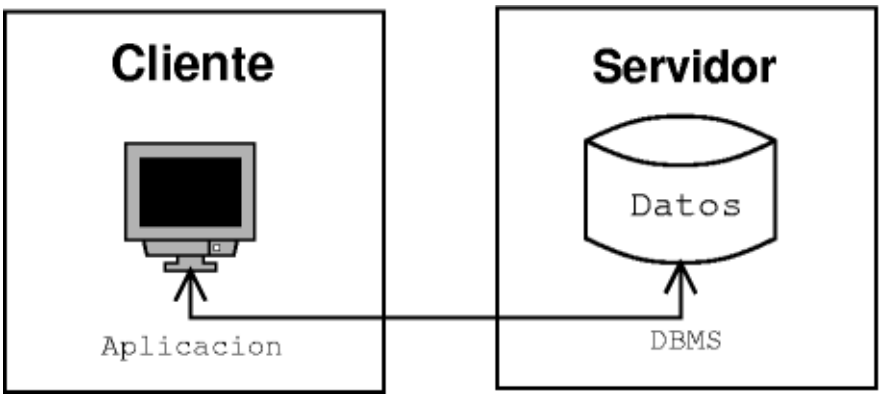

Figura 1.1: Aplicaciones de dos capas.

El diseño de dos capas coloca la interfaz de usuario exclusivamente en el cliente. Coloca la administración de base de datos en el servidor y divide la administración de procesos entre el cliente y/o el servidor, creando únicamente dos capas.

1.1.2 Arquitecturas de software de tres capas

La arquitectura de software de tres capas emergió en la década de los noventas para solventar las limitaciones de la arquitectura de dos capas. La tercera capa (capa

de servicios) se localiza entre la interfaz de usuarios (cliente) y el administrador de datos (servidor). Esta capa intermedia provee de servicios para la administración de procesos (tal como el desarrollo, monitoreado y alimentación de procesos) que son compartidos por múltiples aplicaciones.[8]

Figura 1.2: Aplicaciones de tres capas

El servidor de la capa intermedia (también conocido como servidor de aplicaciones) centraliza la lógica de las aplicaciones, haciendo que la administración de cambios sea más sencilla. En arquitecturas más simples, cualquier cambio en la lógica, implica reescribir todas las aplicaciones que dependan de ésta.

En algunas ocasiones, la capa intermedia se divide en dos o más unidades con diferentes funciones, en estos casos la arquitectura es referida como multinivel. En la figura 1.3 se muestra este caso.

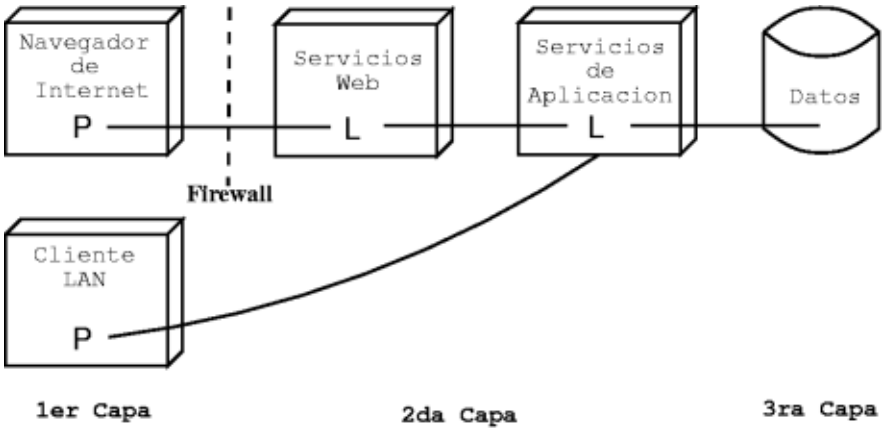

Figura 1.3: Arquitectura multinivel

1.2 El protocolo HTTP

El Protocolo de Transferencia de Hipertexto[2] es un protocolo de nivel de aplicación para sistemas de información distribuidos y colaborativos de hipermedios[3]. Es un protocolo genérico y sin manejo de estados[4], el cual puede ser utilizado para varias tareas más allá de su uso para la transferencia de hipertexto, tal como sistemas para la administración de objetos distribuidos[9].

[2]Hipertexto es la organización de unidades de información en asociaciones conectadas que un usuario puede escoger.

[3]Hipermedios es un término derivado de hipertexto, que extiende la noción de una liga de hipertexto para incluir ligas a un conjunto de objetos multimedia, incluyendo sonido, vídeo y realidad virtual.

[4]Se dice que un protocolo de red puede manejar un estado si mantiene un conjunto de variables entre las múltiples conexiones de un mismo cliente, tal como el FTP.

1.2.1 Descripción de su operación

El protocolo HTTP es un protocolo basado en el esquema solicitud/respuesta. El servidor envía una respuesta al cliente basado en la solicitud hecha por éste, cerrando inmediatamente la conexión entre ambos; por ello se le conoce como un protocolo sin manejo de estados; no existe el concepto de sesión como ocurre en otros protocolos.

Una conexión bajo el protocolo HTTP puede expresarse de la siguiente manera:

1. Un cliente envía una solicitud al servidor en forma de un *método* de solicitud, una *URI*[5], y una *versión del protocolo* utilizado, seguido de una *mensaje* tipo MIME[6] conteniendo los modificadores de la solicitud, la información del cliente, y un posible cuerpo de contenido a través de la conexión con el servidor.

2. El servidor responde con una línea de *estado*, incluyendo la *versión del protocolo* del mensaje y un *código* de éxito o error, seguido por un *mensaje* tipo MIME conteniendo la información del servidor, entidades de metainformación, y posible contenido cuerpo-entidad.

La mayor parte de las comunicaciones HTTP es iniciada por un agente del usua-

[5]Uniform Resource Identifier (Identificador Uniforme de Recursos) es una cadena que indica el recurso sobre el cual va a ser aplicado el método solicitado.

[6]Multi-Purpose Internet Mail Extensions (Extensiones Multipropósito para el Correo de Internet) es una extensión al protocolo original de correo por Internet, que permite usar el protocolo para intercambiar diferentes tipos de archivos de datos: audio, vídeo, imágenes, programas de aplicación, etc. Anteriormente únicamente se podían transmitir archivos basados en ASCII-7.

rio[7] y consiste en una solicitud que será aplicada a un recurso en algún servidor origen. En el caso más simple, puede ser realizada a través de una sola conexión entre el agente del usuario y el servidor origen. Una situación más complicada ocurre cuando uno o más intermediarios esta presentes en la cadena de solicitud/respuesta.

1.2.2 Mecanismo para el manejo de estados en el HTTP

Los servidores HTTP responden a cada solicitud del cliente sin relacionar tal solicitud con alguna solicitud previa o subsecuente. Para solucionar este problema Netscape introdujo el concepto de *cookie*, el cual es un grupo de cabeceras que se agrega a las solicitudes de los clientes y a las respuestas de los servidores, llevando consigo información.

Esta técnica permite a los clientes y a los servidores intercambiar información sobre el estado y colocar las solicitudes y respuestas del HTTP dentro de un contexto más largo, lo cual llamaremos *sesión*.

No obstante existen diferentes contextos potenciales y por lo tanto hay varios tipos potenciales de sesión. El diseño del mecanismo para el manejo de estados a través del intercambios de *cookies* tiene los siguientes atributos[14]:

1. Cada sesión tiene un inicio y un fin.

[7]Agente del usuario: Programa de computadora o dispositivo electrónico que es usado para acceder a los servicios ofrecidos por una aplicación Web. Éste puede ser un navegador de Internet, un teléfono celular habilitado para acceso a Internet, un software convertidor de texto en audio, entre otros.

2. Cada sesión tiene un tiempo de vida relativamente corto.

3. Ya sea el agente del usuario o el servidor original puede terminar una sesión.

4. La sesión está implícita en el intercambio de la información del estado.

1.3 Los lenguajes de marcado HTML/XHTML

El HTML es una aplicación SGML[8] conforme al Estándar Internacional ISO 8879, y es ampliamente considerado como el lenguaje estándar para la publicación en el World Wide Web.

El SGML es un lenguaje para describir lenguajes de marcado. El HTML es un ejemplo de lenguaje definido en SGML.

El HTML, fue concebido originalmente, para ser un lenguaje para el intercambio de documentos científicos y técnicos. Por lo tanto tenía el objetivo de reducir el problema de la complejidad del SGML, especificando un pequeño conjunto de etiquetas[9] estructurales y semánticas para escribir documentos relativamente simples[12].

El XML[10] fue resultado de la intención de recuperar el poder y la flexibilidad del SGML sin la mayor parte de su complejidad. Aunque es una forma restringida del SGML, el XML preserva la mayor parte de su poder y riqueza, y sigue manteniendo todas las características más comúnmente utilizadas.

[8] Standard Generalized Markup Language (Lenguaje Estándar de Marcado Generalizado)
[9] Una etiqueta (tag) es un término genérico que designa a un elemento descriptor de un lenguaje.
[10] Extensible Markup Language (Lenguaje de Marcado Extensible).

El XHTML es una familia de tipos de documentos actuales y futuros y módulos que reproducen, subconjuntan y extienden al HTML 4. Los tipos de documentos de la familia XHTML están basados en XML, y últimamente diseñados para trabajar en conjunción con agentes de usuario basados en XML[11].

El XHTML es una reformulación de los tres tipos de documentos HTML 4 (strict, loose y frames) como aplicaciones XML. Tiene la intención de ser usado como un lenguaje para contenido que tanto, se ajuste al XML y que opere en un agente de usuario que cumpla el HTML 4.

1.3.1 Las hojas de estilo CSS

Uno de los objetivos iniciales del HTML es que la información descrita en éste lenguaje pudiera ser desplegada de manera independiente al dispositivo, sin que se perdiera su estructura original. Es decir, que independientemente de cómo el agente del usuario le presentara la información, ya sea de manera visual, auditiva, braile, etc., el documento debería ser presentado sin perder el contenido que se deseaba transmitir.

Sin embargo con el éxito alcanzado con el navegador gráfico Mozilla a mediados de los noventas, Netscape Inc. comenzó a integrar nuevas características al lenguaje de manera que los autores de documentos HTML pudieran explotar las capacidades gráficas y multimedia del navegador.

Esto hizo que el espíritu inicial del HTML se comenzara a perder, hasta que el

World Wide Web Consortium desarrolló el concepto de CSS[11]. El mecanismo CSS es un mecanismo de definición de estilo, que permite a los autores y a los lectores de documentos HTML adjuntar un estilo a éstos[13]; por ejemplo, tipo de letra, color, espaciado, etc.

Así los autores pueden adjuntar su hoja de estilo preferido, mientras que los lectores pueden tener su hoja de estilo personal que se ajuste a su discapacidad técnica o humana.

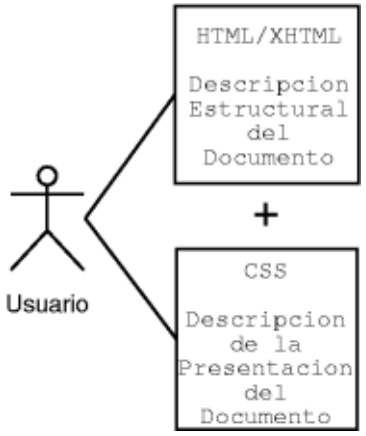

Figura 1.4: HTML + CSS

Otra característica de los CSS es su capacidad para definir objetos que indican comportamientos genéricos, que pueden ser aplicados a distintas estructuras del HTML/XHTML. Además estos objetos pueden ser sobrecargados y heredados, siguiendo un modelo orientado a objetos.

[11] Cascade Style Sheet (Hoja de Estilo en Cascada).

1.3.2 Scripts de lado del cliente

Un script del lado-del-cliente es un programa que puede acompañar a un documento HTML o estar embebido directamente en él. El programa se ejecuta en la máquina del cliente cuando el documento se carga, o en algún otro momento, como cuando una liga es activada. El HTML soporta el manejo de scripts independientemente del lenguaje (JavaScript, VBScript, TCL, etc.)[12].

Los scripts ofrecen al los autores una forma de extender los documentos HTML en formas altamente activas e interactivas:

- Los scripts puede modificar el contenido de documento de manera dinámica.
- Pueden procesar el contenido de una forma HTML.
- Pueden ser ejecutados al dispararse un evento específico tal como la carga y descarga de elementos, movimientos del ratón, etc.
- Pueden ligar controles de formas (botones por ejemplo) para producir elementos de GUI[12].

Netscape Inc. fue el primero en incluir un lenguaje de script en su navegador: el JavaScript. Posteriormente Microsoft sacó el JScript, su versión para el navegador Internet Explorer, la cual, aunque sintácticamente es afín con Java Script,

[12]Graphic User Interface (Interfaz Gráfica de Usuario).

cuenta con muchas incompatibilidades. Además Microsoft le agregó soporte a su navegador para VBScript, otro lenguaje de script basado en su VisualBasic.

Para dirimir el problema de la incompatibilidad de lenguajes soportados por los navegadores, ECMA[13] desarrolló el estándar ECMAScript, basado en los lenguajes JavaScript y JScript.

El ECMAScript es un lenguaje de programación orientado a objetos para realizar cálculos y manipular objetos computacionales dentro de un ambiente anfitrión (Navegador). El ECMAScript no tiene la intención de ser computacionalmente autosuficiente; depende de los objetos provistos por el sistema anfitrión[15].

1.4 El lenguaje SQL

El SQL[14] es un lenguaje de programación interactivo y estándar para obtener y actualizar información de una base de datos, además es útil para definición de los datos sobre los cuales se trabajará.

El SQL fue desarrollado originalmente en la década de los setenta por investigador de la IBM, E. F. Codd. En la década de los ochentas tanto ANSI[15] como ISO[16] desarrollaron un lenguaje de base de datos relacional estándar, basándose en los

[13]ECMA es una asociación industrial internacional dedicada a la estandarización de sistemas de información y comunicación.

[14]Structured Query Language (Lenguaje Estructurado de Consultas)

[15]American National Standards Institute (Instituto Americano de Estándares Nacionales).

[16]International Organization for Standarization (Organización Internacional para la Estandarización)

conceptos del SQL. Actualmente la última revisión del estándar se conoce como SQL3, ocurrida en 1999[16].

A pesar de estos esfuerzos de estandarización, diversos manejadores de base de datos soportan SQL pero añaden extensiones propietarias al lenguaje por lo que las migraciones entre manejadores distintos suelen ser difíciles.

En SQL las consultas toman la forma de un lenguaje de comandos en idioma inglés, que permiten realizar selecciones, inserciones, actualizaciones y búsquedas de datos.

Un buen tutorial sobre el uso del lenguaje SQL puede encontrarse en SQL Course `http://www.sqlcourse.com`.

1.5 Cifrado de información

Todas las comunicaciones a través de Internet utilizan el protocolo TCP/IP[17]. El TCP/IP permite que la información sea enviada de una computadora a otra a través de una serie de computadoras intermedias y de redes distintas antes de alcanzar su destino.

La gran flexibilidad del TCP/IP ha conducido a su aceptación mundial como el protocolo de comunicación básico para Internet e Intranets. Al mismo tiempo, el hecho de que el TCP/IP permite el paso de información a través de computadoras intermedias, hace posible que terceras personas interfieran en las comunicaciones.

[17]Transmission Control Protocol/Internet Protocol (Protocolo de Control de Transmisiones/Protocolo de Internet).

1.5.1 Algoritmos criptográficos

Existen dos clases de algoritmos de cifrado basados en el uso de llaves: algoritmos **simétricos** (o de llave secreta) y **asimétricos** (o de llave pública). La diferencia es que los algoritmos simétricos usan la misma llave tanto para el encriptado como para el desencriptado (o la llave de desencriptado es fácilmente derivada de la llave de encriptado), en tanto que los algoritmos asimétricos usan una llave diferente para el encriptado y desencriptado, y la llave de desencriptado no puede ser derivada de la llave de encriptado.

Los métodos de cifrado asimétricos (también llamados algoritmos de llave pública o generalmente criptografía de llave pública) permiten que la llave de encriptado sea pública (puede ser publicada incluso en un periódico), permitiendo que cualquiera encripte con esa llave, donde únicamente el propio receptor (quien conoce la llave privada de desencriptación) puede desencriptar el mensaje. La llave de encriptado también se le llama llave pública y la llave de desencriptado llave privada o llave secreta.

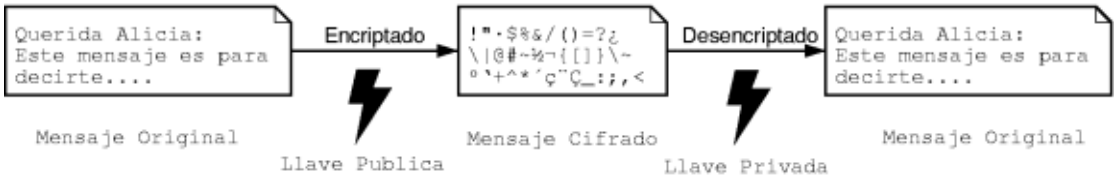

Figura 1.5: Cifrado por llave pública

Generalmente los algoritmos simétricos son mucho más rápidos de ejecutar en una computadora que los asimétricos. En la práctica estos son usados frecuente-

mente juntos, así que los algoritmos de llave pública son usados para encriptar una llave de encriptado generada aleatoriamente, y la llave aleatoria es usada para el encriptamiento del mensaje usando un algoritmo simétrico. A esto algunas veces se le conoce como encriptado híbrido.

Algoritmos Simétricos	**Algoritmos Asimétricos**
OTP	RSA
DES	Diffle-Hellman
AES	ElGamal
Blowfish	DSS
RC4	LUC
IDEA	XTR

Tabla 1.1: Algoritmos Criptográficos

1.5.2 Firmas digitales

Una firma digital es una pequeña cantidad de información que fue creada usando alguna llave secreta, y existe una llave pública que puede ser usada para verificar que la firma fue realmente generada usando la correspondiente llave privada. El algoritmo para generar la firma debe ser tal que sin conocer la llave secreta no es posible crear una firma que pudiera verificarse como válida.

Las firmas digitales son usadas para verificar que un mensaje realmente proviene del remitente declarado (asumiendo que el emisor conocer la correspondiente llave privada para su llave pública).

También las firmas digitales pueden ser utilizadas para certificar que una llave pública pertenece a una persona en particular. Esto se logra usando la combinación

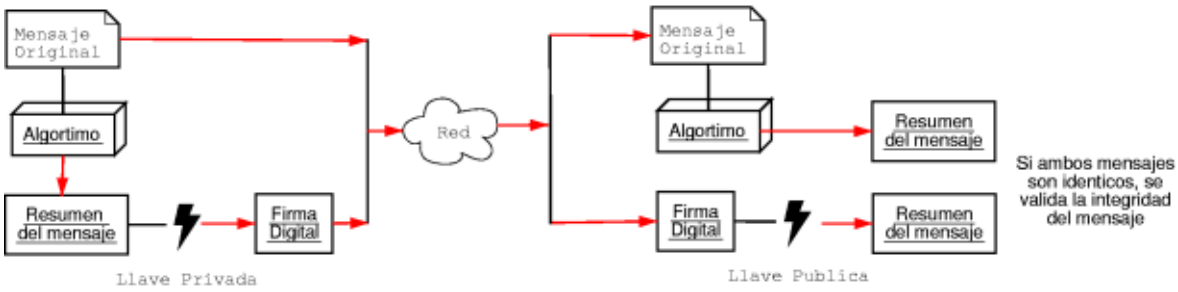

Figura 1.6: Uso de una forma digital para verificar la integridad de los datos

de la llave y la información sobre su dueño por una llave que sea de confianza. La firma digital por una tercera persona (dueño de la llave de confianza), la llave pública y la información sobre el dueño de la llave pública son llamados **certificados**.

Una firma digital de un documento arbitrario es típicamente creado calculando un mensaje resumen del documento, y concatenándolo con información acerca del firmante, un sello de tiempo, etc. La cadena resultante es luego cifrada utilizando la llave privada del firmante. El bloque de bits resultante es la firma.

1.5.3 El protocolo SSL

El protocolo SSL trabaja sobre el TCP/IP y debajo de los protocolos de alto de nivel tales como el HTTP o el IMAP. Usa el TCP/IP en nombre de los protocolos de alto nivel, y en el proceso permite a un servidor SSL autenticar por sí mismo a un cliente SSL, permite al cliente autenticar por sí mismo al servidor, y permite a ambas máquinas establecer una conexión encriptada.

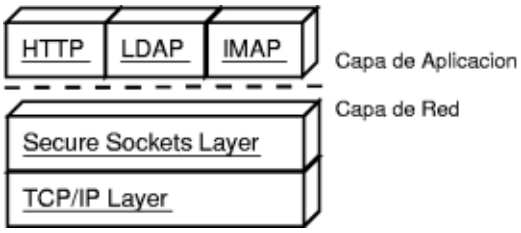

Figura 1.7: Protocolo SSL

Estas capacidades apuntan a las preocupaciones fundamentales sobre las comunicaciones sobre Internet y otras redes TCP/IP:

- La **autenticación del servidor SSL** permite al usuario confirmar la identidad del servidor. Los clientes SSL pueden utilizar técnicas de criptografía de llave pública para checar que el certificado de un servidor y su identificación pública son válidas y han sido avalados por una autoridad certificadora (CA[18]) aceptada en la lista de CA confiables del cliente.

- La **autenticación del cliente SSL** permite a un servidor confirmar la identidad de un usuario. Usando las mismas técnicas que las usadas por la autenticación del servidor, un servidor SSL puede checar que un certificado del cliente y su identificación pública son válidas y certificadas por un CA de confianza.

- Una **conexión SSL encriptada** requiere que toda la información enviada entre un cliente y un servidor sea encriptada por el software emisor y desencriptada por el software receptor, proveiendo un alto grado de confiden-

[18]Certificate Authority (Autoridad Certificadora)

cialidad. Además, toda la información enviada sobre una conexión SSL encriptada es protegida con un mecanismo para detectar si ésta fue alterada durante su tránsito en la red.

El protocolo SSL incluye dos sub-protocolos: el protocolo *SSL record* y el protocolo *SSL handshake*. El protocolo SSL record define el formato usado en la transmisión de los datos. El protocolo SSL handshake involucra el uso del protocolo SSL record para intercambiar una serie de mensajes entre el servidor SSL y el cliente SSL cuando inician una conexión SSL. Este intercambio de mensajes esta diseñado para facilitar las siguientes acciones:

- Que el cliente autentique al servidor.
- Permitir al cliente y al servidor seleccionar los algoritmos criptográficos que son soportados entre ambos.
- Opcionalmente que el servidor autentique al cliente.
- Utilizar técnicas de encriptación de llave publicar para generar una llave secreta compartida.
- Establecer una conexión SSL encriptada.

1.6 Arquitecturas de los servidores de aplicaciones Web

Esta sección describe, a grandes rasgos, los diferentes enfoques que se ha adoptado para que un servidor HTTP ejecute la aplicación Web desarrollada, cada una con sus ventajas y desventajas. Se comenzará con el estándar CGI, que es fundamento de los siguientes; hablaremos sobre el modelo de co-servidor, donde un nuevo servicio se levanta al igual que el servidor HTTP y se comunica con este para transmitirse tantos las peticiones como las respuestas. Finalmente se verá el modelo de un lenguaje intérprete incrustado de manera persistente en el mismo servidor HTTP, por lo tanto el mismo proceso de servidor HTTP se encarga de ejecutar la aplicación Web.

1.6.1 CGI

El CGI[19] es un estándar para tender interfaces entre aplicaciones externas y servidores de información, tales como los servidores HTTP. Un documento HTML enviado por un servidor HTTP a un cliente es estático, el texto que contiene esta en un estado constante, no cambia. Por otro lado, un programa CGI se ejecuta en tiempo real, por lo tanto puede recibir información del usuario, procesarla y arrojar datos formateados en HTML, texto, o algún otro formato plano o binario. en base a esta información. De ahí que los CGI pueden producir información de

[19]Common Gateway Interface (Interfaz Común de Puerta de Enlace).

manera dinámica[18].

El CGI provee de una manera consistente para pasar datos a partir de la solicitud de un agente del usuario a un programa que puede ser localizado y ejecutado por el servidor de HTTP.

Un programa CGI puede ser escrito en cualquier lenguaje que se pueda compilar y/o ejecutar en el sistema donde se implemente:

- C/C++
- Java
- Perl
- Tcl
- Algún Shell de Unix
- Visual Basic

1.6.1.1 Diseño del CGI

Un programa CGI puede adquirir los datos de entrada por diferentes medios: Por medio de variables de entorno (insertadas por el servidor HTTP al momento de generar el nuevo proceso que ejecutará el programa); y por medio de la entrada estándar, este flujo de información puede estar formateada en un estilo MIME y

debe ser analizada sintácticamente para dividirla en sus distintas variables. Cuando los datos se reciben por la entrada estándar, es por que la información se envió por un método POST o PUT del HTTP.

Dentro de las variables de entorno las que contienen la información proveída por el usuario son:

- **QUERY_STRING**. Es cuando la información se suministra al final del URL y se identifica por comenzar con el símbolo '?'. Puede ser resultado de un método GET sobre una forma HTML, por un un documento ISINDEX o directamente descrita en una hiperliga HTML. Normalmente tendrá el patrón sintáctico de variable=valor&variable=valor...

- **PATH_INFO**. Es cuando los datos suministrador van al final del URL, pero continúa como si fuera otra ruta, tal como /otra/ruta.

1.6.1.2 Desventajas del CGI

Debido a que los CGI son programas ejecutables, para que cumplan su función, necesitamos permitir a todo el mundo que se ejecuten en el servidor, lo cual no es lo más seguro, ya que cualquier error en el CGI pdría desembocar en un punto débil en la seguridad del sistema, que puede ser aprovechado para comprometerlo.

Los programas CGI pueden presentar agujeros de seguridad[20] en dos maneras[20]:

[20]Los agujeros de seguridad se pueden definir como puntos vulnerables de un sistema de cómputo, que pueden ser explotables para ganar accesos no autorizados al sistema.

- De manera intencional o no, pueden mostrar información sobre el sistema que puede ayudar a terceros a comprometerlo.
- Pueden realizar tareas dentro del sistema con la información proveída por el usuario remoto, permitiendo que con esta información le deje ejecutar comandos del sistema.

Por otro lado la ejecución de un CGI implica que el servidor HTTP genere otro proceso en el sistema, el cual consume recursos del sistema. El número de solicitudes a un CGI puede llegar a ser tal que consuma todos los recursos y puede desde ralentizar sensiblemente el desempeño del sistema, hasta bloquearlo por completo. Además cada proceso CGI que se levanta, puede solicitar más recursos como una conexión a una base de datos, establecer comunicación con otros procesos o crear nuevos, realizar conexiones TCP/IP a otras máquinas, etc., y todo eso consume el número de recursos disponibles.

Esta última desventaja es la que ha motivado el desarrollo de alternativas al modelo del CGI, para mejorar el desempeño y la escalabilidad de las aplicaciones Web.

1.6.2 Modelo de co-servidor

En este modelo otro proceso servidor, además del servidor HTTP, se levanta al mismo tiempo que éste. El conjunto peticiones definido por el servidor HTTP son transferidas al otro proceso servidor, que actúa como un mecanismo capaz de

abstraer la información HTTP en una serie de facilidades computacionales, tales como manejo automático de sesiones, validación y corrección de datos de entrada y salida, etcétera, y ofrece una API[21] con la que se pueden desarrollar las aplicaciones. Estas aplicaciones que se comunican con la capa de software intermedia se les conoce como Servlets, en contraposición a los Applets que se ejecutan en el software cliente, y que, al igual que los Servlets, fueron popularizados por Sun Microsystem utilizando su lenguaje Java, aunque el modelo puede ser implementado en casi cualquier lenguaje de alto nivel.

Figura 1.8: Estructura del Modelo de Co-servidor

Por lo tanto, y debido a su diseño, este modelo tiene las siguientes ventajas sobre los CGI:

- El Servlet no corre como un proceso separado, corre dentro del proceso "motor". Esto elimina el sobreprocesamiento y disminuye el consumo de recursos.

- El Servlet permanece en memoria y solamente existe una instancia de él. Esto ahorra memoria y permite el manejo de datos persistentes.

[21] Application Program Interface (Interfaz para la Programación de Aplicaciones). Es un método específico prescrito por el sistema operativo de una computadora o por otra aplicación para escribir una aplicaciones haciendo peticiones al sistema operativo o a la otra aplicación.

- El “motor de Servlets” puede actuar como una c*aja de arena*[22], aumentando la seguridad del sistema.

1.6.3 Intérprete incrustado

Normalmente los servidores HTTP proveen de una API, para que los programadores puedan desarrollar extensiones al servidor HTTP. Estas extensiones pueden ser para modificar algún comportamiento específico del servidor HTTP, tal como la manera de autenticar a los usuarios, la validación de accesos a los recursos, el manejo de bitácoras, o la generación de las respuestas. Esto presenta las siguientes ventajas:

- Se puede intervenir en cualquier etapa del procesamiento de la petición hecha al servidor.
- La extensión se ejecuta dentro del proceso del servidor HTTP, evitando así el sobreprocesamiento.

No obstante esta API generalmente se encuentra en C o C++, lenguajes con el que se desarrolla el servidor. Sin embargo para muchos programadores no resulta productivo desarrollar programas de aplicación en tales lenguajes, ya que son difíciles y tardados de programar y depurar, así como la carencia de herramientas intuitivas para un desarrollo rápido. Es por ello que se han desarrollado capas intermedias

[22] Es una aplicación que restringe las posibles operaciones que un módulo del programa puede ejecutar.

de software donde la API del servidor HTTP se mapea a otro lenguaje para que sea más sencilla su utilización. Estos lenguajes suelen ser lenguajes interpretados tales como Visual Basic, Perl, Python, Tcl, etc. Aunque también existe, en estas capas intermedias de software, mapeos a lenguajes con un mayor grado de abstracción, que permite el procesamiento de plantillas con código incrustado dentro del HTML, tales como el PHP o el ASP.

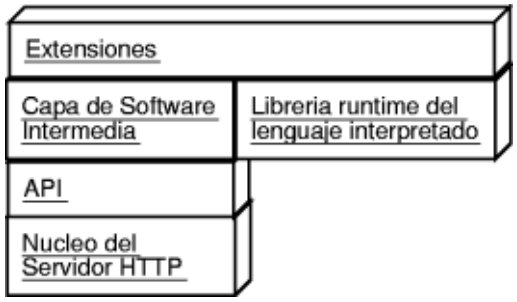

Figura 1.9: Estructura del Intérprete Incrustado

1.7 Modelos para la programación de aplicaciones Web

En esta sección describiremos algunas manera en las que se pueden desarrollar aplicaciones Web. Se incluirán ejemplos ilustrativos.

Se comenzará hablando del CGI, el mecanismo inicial que se implementó para la ejecución de programas en un servidor Web; se verá el mecanismo implementado por los Servlets; por otro lado se hablará sobre un enfoque parecido al CGI pero

implementado de manera más eficiente: los módulos, finalizando con el modelo de incrustación de código ejecutable dentro de documentos HTML.

1.7.1 CGI nativo

Comúnmente cuando se refiere a un CGI, se hace referencia a un programa que utiliza la interfaz propuesta por el estándar CGI. Estos programas normalmente son pequeños y son resultado de una solución rápida a algún planteo de automatización, ya que como interactúan con el estándar a bajo nivel, cada programa debe implementar sus propias herramientas para el procesamiento tanto de los datos de entrada como de salida.

Algorithm 1 Programa CGI en lenguaje C

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv)
{
   char *your_add;

   your_add = getenv("REMOTE_ADDR");

   printf("Content-type: text/html%c%c", 10, 10);
   printf("<html>");
   printf("<head>")
   printf("<title>Hola Mundo</title>");
   printf("</head>");
   printf("<body bgcolor=\"#FFFFFF\">");
   printf("<h1 align=\"center\">Hola Mundo</h1>");
   printf("<p align=\"center\">Tu IP es: %s", your_add);
   printf("</body>");
   printf("</html>");
}
```

1.7.2 Servlets

Los Servlets son módulos de código, normalmente en lenguaje Java, pero pueden encontrarse en algún otro como Python (Zope `http://www.zope.org`), que se ejecutan dentro de un servidor de aplicaciones (motor de Servlets) para responder a las peticiones de los sistemas cliente.

Algorithm 2 Servlet en lenguaje Java

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

public class HelloClientServlet extends HttpServlet
{
   protected void doGet(HttpServletRequest req,
                        HttpServletResponse res)
   throws ServletException, IOException
   {
      res.setContentType("text/html");
      PrintWriter out = res.getWriter();
      out.println("<HTML><HEAD><TITLE>Hello Client!</TITLE>" +
                  "</HEAD><BODY>Hello Client!</BODY></HTML>");
      out.close();
   }

   public String getServletInfo()
   {
   return "HelloClientServlet 1.0 by Stefan Zeiger";
   }
}
```

1.7.3 Módulos del servidor Web

Existen módulos que utilizan el API de los servidores Web para extender su funcionamiento. Estos módulos pueden hacer uso de los manejadores de contenido del servidor HTTP y convertirse así en aplicaciones Web.

Cómo ya comentamos existen integraciones entre lenguajes interpretados y servidores HTTP para simplificar la programación de estos módulos.

Algorithm 3 Módulo para Apache en mod_perl

```
package HolaMundo;

use Apache::Constants ':common';

sub handler {
    my $r = shift;

    $r->content_type('text/html');
    $r->no_cache(1);
    $r->send_http_header;

    $r->print("<HTML>");
    $r->print("<HEAD>");
    $r->print("<TITLE>Hola Mundo</TITLE>");
    $r->print("</HEAD>");
    $r->print("<BODY><H1>Hola Mundo</H1>");

    my $c = $r->connection;

    $r->print("<P>Tu IP es: $c->remote_ip</P>");
    $r->print("</BODY>");
    $r->print("</HTML>");

    return OK;
}

1;
```

Las desventajas que se presentan en este modelo está en la complejidad inherente en el desarrollo de extensiones directas al servidor HTTP: nos comunicamos directamente con éste, y por consecuencia el nivel de abstracción que tenemos es de muy bajo nivel, en comparación con el siguiente modelo.

1.7.4 Código incrustado en el HTML

Este modelo, incrusta el código ejecutable en dentro de un documento HTML, de esta manera, el servidor, al recibir una petición, ejecuta el código incrustado y produce un documento que contiene únicamente código HTML y es enviado al agente del usuario que realizó la petición. A continuación se muestra un breve ejemplo de como se incrusta código ejecutable dentro de un documento HTML.

Dicho de otra manera, en los modelos anteriores, el desarrollador de la aplicación Web escribía código en lenguaje de programación, que como salida, producía HTML. Esta tarea llega a ser laboriosa y muchas veces aburrida. Por lo cual se definió un nuevo modelo, donde el desarrollador no se fija de manera única en el código en algún lenguaje de programación, sino que se dedica a escribir HTML y únicamente donde se deben colocar elementos dinámicos, el desarrollador utiliza marcas que define el sistema del intérprete incrustado utilizado, y escribe código.

Algorithm 4 Ejemplo de un ASP

```
<%@Language = VBScript %>
<html>
<body>
<% For i = 3 to 7 %>
     <FONT SIZE = <% = i %>>
     Hello world! <br>
<% Next %>
</body>
</html>
```

Es posible generar páginas de código incrustado en el HTML con varios lenguajes:

- VBScript (las Active Server Pages -ASP- de Microsoft)
- Perl (Apache::ASP, Mason, ePerl, etc.)

- Java (JSP)
- PHP

Las ventajas de este modelo de desarrollo radican en que no se necesita mucha formación técnica para escribir una aplicación Web. De la misma manera en que se escribe una página Web, se desarrolla una aplicación Web.

Actualmente los lenguajes y ambientes de desarrollo para este modelo proveen de componentes reutilizables muy potentes que ofrecen mayor productividad al programador.

Sin embargo, por su misma naturaleza, su uso se limita a la presentación de documentos basados en texto (HTML, XML, etc.), además de la total flexibilidad que nos ofrecería un lenguaje de programación de propósito general.

Capítulo 2

Estructura general de una aplicación Web

En este capítulo se hablará acerca de los problemas o situaciones con los que tiene que tratar el desarrollador de aplicaciones Web. Pero antes es necesario conocer cómo es que funciona una aplicación Web. En la figura 2.1 se observa un diagrama que muestra la estructura general de una aplicación Web. En primera instancia, lo que ve un usuario al ingresar a un sitio que alberga una aplicación Web es la página de contenido inicial, ésta es la página de bienvenida, generalmente en ella existe un espacio dedicado a la autenticación[1] del usuario para su acceso a áreas restringidas del sistema. Una vez que el usuario es autenticado, la aplicación

[1]Proceso para determinar si algo o alguien es realmente quien declara que es. Este proceso es comúnmente implementado con el uso de un nombre de usuario y una contraseña secreta. Si la entidad que se desea autenticar conoce la contraseña, se asume que es el realmente quien dice ser y se le otorga acceso a los recursos del sistema definidos por el administrador.

Web debe de poner en marcha un mecanismo que le permita determinar, en todo momento, cual es el éstatus del usuario, es decir, en qué parte de la aplicación se encuentra, qué variables se han establecido, etc. Lo anterior se logra con el manejo de sesión. A partir de este punto, todas las acciones o peticiones del usuario serán enviadas al despachador de mensajes, el cual se encarga de validarlas y, si aplica, mandar ejecutar el procedimiento correspondiente al mensaje de solicitud, y al estado encontrado.

Las operaciones que se realizan en una aplicación Web (el almacenamiento de los correos de un usuario o la lista de artículos de un pedido a un almacén por ejemplo) requieren de un medio de almacenamiento persistente, comúnmente esto es realizado por medio de un manejador de bases de datos (DBMS por sus siglas en inglés); la aplicación Web debe de tener un medio para comunicarse con el DBMS con el fin de leer y escribir la información en la base de datos.

Un aspecto importante en el diseño de una aplicación Web es el uso de plantillas. Las plantillas son una serie de archivos que definen la apariencia de una aplicación Web, de esta manera, si se desea cambiar la presentación de la aplicación, sólo se tiene que modificar las plantillas sin necesidad de modificar el código que le da funcionalidad a la aplicación.

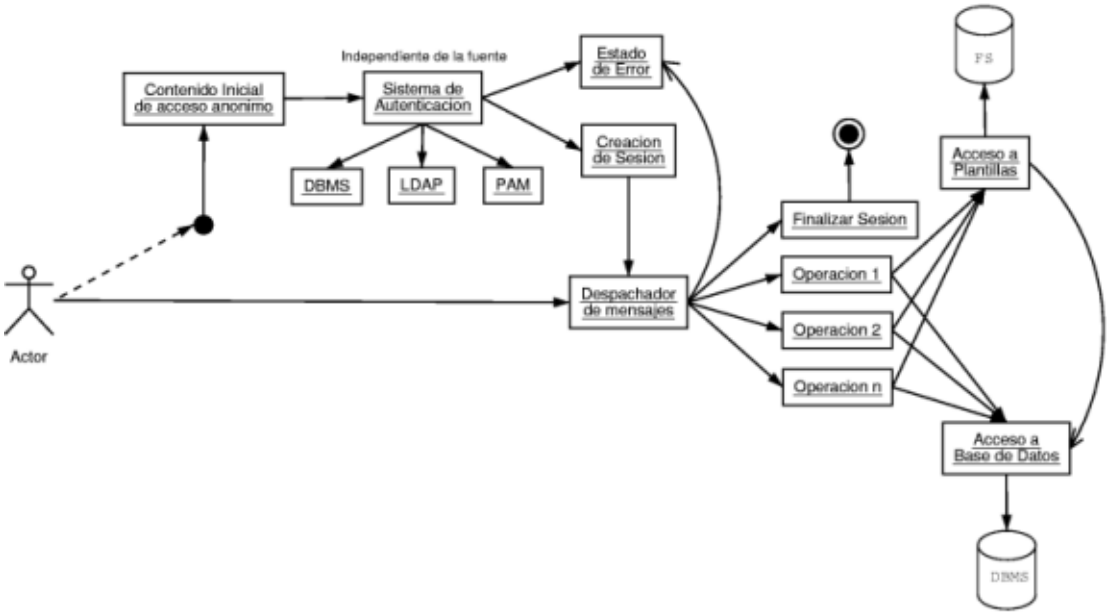

Figura 2.1: Estructura General de una Aplicación Web

2.1 Validación de datos

En cualquier momento que un programa interactúa con un cliente conectado a través de una red, existe la posibilidad de que ese cliente ataque el programa para obtener un acceso no autorizado al sistema. Hasta el más inocente programa puede ser muy peligroso para la integridad del sistema.

Una aplicación Web, por su naturaleza, permite la lectura de archivos, la adquisición de accesos a programas del sistema, etc.. Estos accesos pueden ser utilizados de manera no autorizada

- Explotando las suposiciones del programa

- Explotando las debilidades en la configuración del servidor
- Explotando las debilidades en otros programas y llamadas al sistema

> *La principal debilidad en las aplicaciones Web es la insuficiente validación de los datos de entrada[27].*

Esto quiere decir que para cada dato que el usuario a mandado a la aplicación Web, antes de procesarla debemos tener control sobre ella; es decir, sanearla de datos no autorizados. Jamás se deben hacer suposiciones acerca de esta información.

El CERN[2] nos aconseja seguir la regla número uno en la construcción de *firewalls* pero aplicada a la programación: lo que no está expresamente permitido, está prohibido. Es decir, definir un dominio de valores aceptables por cada elemento de información recibida (variable) y bajo esta verificación reemplazar cualquier caracter no aceptable por algún otro de control, como el guión bajo ("_") o su completa eliminación. No se debe hacer al revés (identificando lo ilegal y escribir código para rechazar dichos casos) por que es muy probable olvidar manejar algún caso importante de entrada ilegal.

Este saneado de los datos de entrada debe ser tanto léxico como sintáctico, haciendo que ésta se ajuste a lo que se debe procesar. El uso de expresiones regulares[3]

[2]Originalmente de las siglas Computer Emergency Response Team (Equipo para la Respuesta de Emergencias Computacionales). Actualmente es el mayor informador de problemas de seguridad en Internet.

[3]Una expresión regular es una manera por la cual un usuario puede decirle a una programa de computadora cómo buscar determinado patrón en una cadena de texto. El usuario puede determinar que hacer cuando determinado patrón se encuentre, tal como eliminarlo o sustituirlo por algún otro.

se vuelve fundamental en esta tarea, ya que éstas nos permiten crear esqueletos con los cuales deben coincidir los datos a procesar, en caso contrario se rechazan o se sanean. Además estas pueden ser reutilizables tanto en el lado del servidor como en el lado de cliente ya que son independientes del lenguaje[4] . Así mismo es importante limitar la longitud máxima (y si es posible mínima) de cada variable.

La validación puede ser hecha tanto de lado del agente del usuario (si y sólo si éste provee de herramientas para ello como lenguajes script de-lado-del-cliente) y naturalmente en el lado del servidor. Es en este punto donde viene la pregunta ¿dónde debemos de validar?.

La validación del lado del servidor asegura que, independientemente del agente del usuario utilizado por el cliente, los datos recibidos serán aceptados únicamente si pasan por el filtro de validación, además de los provistos por los DBMS cuando los datos son serializados. Por el otro lado, la validación en el lado del cliente hace que la aplicación sea más amigable para el usuario, y no tenga que esperar la respuesta del servidor con mensaje de error, para volver a reescribir los datos; sin embargo no se puede saber qué agente del usuario utilizará quien acceda al sistema, y por lo tanto no sabremos con qué capacidades de procesamiento contará.

Es por las anteriores razones que creemos que se debe procurar realizar la validación en ambos lados, para asegurar la confiabilidad de los datos procesados, sin sacrificar la facilidad y amigabilidad de la aplicación. Aunque esto implica la posibilidad de programar más. Se acepta que por norma el 80% del código escrito

[4]Varios lenguajes de programación proveen de expresiones regulares: Perl, Java Script, Python, VBScript, entre otros.

tiene como objetivo la validación de datos y el manejo de estados de excepción, mientras que el 20% restante maneja el funcionamiento de la aplicación.

2.2 Autenticación y autorización de usuarios

La manera predeterminada de trabajar del WWW es anónima, lo único que se puede saber, y no siempre con toda seguridad, es el número IP que accede a un determinado recurso. Casi siempre, y debido a características de personalización, políticas de restricción, o ambas, las aplicaciones Web deben saber qué usuario las están utilizando; es por ello que se debe solicitar la identidad del usuario, comúnmente por medio de un nombre de usuario y una comprobación de la misma, a través de una palabra o frase secreta. A este proceso de identificación se le conoce como autenticación.

Por otro lado está la autorización, donde la aplicación que ha identificado al usuario que desea acceder a ella, ahora lo reconoce como usuario válido, así como sus restricciones en su uso. Para realizar la autorización y autenticación del usuario, la aplicación Web requiere de una fuente de datos que contenga la lista de usuarios y sus restricciones de uso. Esta información se encuentra normalmente en una base de datos y requiere que se accede a ella a través de algún método específico:

- PAM[5]
- Radius/Tacacs

[5]Pluggable Authentication Modules (Módulos Enchufables de Autenticación)

- SQL, ODBC, DBI, etc.
- API del sistema operativo
- etcétera

Existen múltiples esquemas para la autenticación de usuarios al momento de querer acceder a un recurso Web. El protocolo HTTP provee, de manera interna[29], un mecanismo para solicitar la autenticación de un usuario para su acceso al recurso. Finalmente los agentes del usuario o navegadores, se encargan de manejar esa petición HTTP, presentando un cuadro de diálogo solicitando el nombre de usuario y la contraseña. Sin embargo este enfoque presenta varias desventajas, entre ellas están la incapacidad de terminar la sesión hasta que se termine el proceso del agente del usuario, así como la incapacidad de modificar la presentación al usuario que solicita la autenticación, ya que este formulario es manejado por el agente del usuario autónomamente. Además, si se necesita utilizar algún otro método que no sea la utilización de archivos planos (htpasswd), es imprescindible la utilización de módulos externos agregados al servidor HTTP.

La otra alternativa es que la aplicación Web se haga cargo de la autenticación, integrándose a la autorización del usuario y el mecanismo de sesiones. Así la presentación del formulario para la tarea se hace a través de formularios HTML, otorgando más flexibilidad para modificar el método de autenticación cuando se necesite.

El desarrollador tiene que establecer desde la etapa de diseño el método de auten-

ticación que va a utilizar y que le permita la escalabilidad y flexibilidad necesaria.

2.3 Manejo de sesiones

Como ya se estableció en el capítulo 1.2, el protocolo HTTP no maneja un estado de cada conexión realizada por un agente del usuario, es por ello que se deben establecer mecanismos ajenos al servidor de HTTP para llevar el control de la sesión.

Lo anterior se logra mediante el uso de algún esquema almacenamiento persistente de una estructura de datos por cada cliente concurrente; este almacenamiento puede ser tanto en:

- Memoria RAM compartida
- Archivo plano o con registros del sistema de archivos
- Tabla en la base de datos utilizada
- Registros de DNS dinámico[31]
- *Cookies*
- Controles HTML ocultos

Cada uno de los anteriores tienen sus ventajas y desventajas. Por ejemplo, la opción de utilizar memoria RAM es muy compleja; además la memoria RAM es un

recurso muy valioso y muy utilizado por las aplicaciones Web, ya que cada acceso concurrente tendrá sus propias instanciaciones de algunos procesos y variables, por lo cual el consumo de memoria RAM puede incrementarse altamente; el manejo de archivos para el seguimiento de sesiones complica la confiabilidad de los datos ya que el sistema de archivos no provee de los mecanismos apropiados para asegurar la integridad de los datos y el rápido acceso a éstos; y así sucesivamente. Las *cookies* pueden ser leídas por terceros lo cual compromete la seguridad de la aplicación y los controles HTML ocultos se vuelven más complicados de mantener cuando la información persistente crece en tamaño. Pocas aplicaciones Web hacen uso exclusivo de un tipo de almacenamiento persistente, generalmente se prefiere mezclarlos, obteniendo las ventajas de cada uno y tratando de evitar sus desventajas.

Para que la aplicación Web identifique cada petición HTTP, las peticiones deberán contener de manera intrínseca un identificador. Esto se logra pasando ese identificador a través de cualquiera de los siguiente métodos:

- URI
- Parámetro por método GET ó POST
- *Cookie*

De esta manera se evita que el usuario se autentique y autorice en cada petición, así como la retransmisión constante de su nombre de usuario y su contraseña, lo cual es potencialmente peligroso. Debe recalcarse que los identificadores de la

sesión deben de ser únicos y difíciles de adivinar, ya que puede existir la posibilidad de que agentes externos quieran simular accesos válidos y entrar de manera fraudulenta al sistema. Es por ello que se debe de valer de algún mecanismo que provea de identificadores aleatorios y con un gran periodo en su repetición, tales como generadores de cadenas aleatorias (MD5, DES, etc) o la utilización del dispositivo de entropía existente en algunos sistemas operativos (*/dev/random* en Linux). A este esquema de trabajo se le conoce como protección basado en *tickets*[32].

La estructura persistente que se encuentra en el lado del servidor, deberá almacenar únicamente la información necesaria para llevar el seguimiento de la sesión, como identificador de la sesión, identificador del usuario en sesión, tiempo de expiración de la sesión, dirección donde se encuentra localizado el cliente[6], y otras variables que deberán mantenerse durante la sesión así como variables temporales que posteriormente se serializarán en la base de datos en una transacción atómica.

Una última alternativa, aplicada en los modelos de co-servidor, consiste en generar un servicio alterno al del servidor HTTP y que pueda comunicarse con él; este proceso alterno generará ya sea un proceso o un hilo para cada cliente concurrente y será el encargado de controlar su sesión.

Así mismo hay que hacer notar que por la misma naturaleza del HTTP es im-

[6]Esta variable resulta controvertida ya que actualmente, por la escasez de direcciones IP en Internet, se han construido NAT (Network Address Translators) y servidores Proxy, los cuales comparten su dirección IP común con la LAN a la que sirven, por lo tanto varios clientes pueden estar en una misma dirección y nuestro esquema podría fallar si tomamos esta variable como discriminante.

posible asegurar la existencia o la ausencia de una sesión, por lo tanto debemos de establecer un proceso que revise periódicamente los tiempos de expiración de cada proceso y los elimine de la base de datos si ya excedió de lo establecido.

2.4 Bitácoras

Las aplicaciones Web se comportan de igual manera que un servicio dentro del sistema operativo (daemon[7] en el dialecto del Unix). Un daemon es un programa que no se invoca directamente para que ejecute su tarea, si no que en lugar permanece inactivo en espera de que suceda un evento determinado en el sistema para entrar en acción.

Las aplicaciones Web heredan lo anterior del servidor HTTP ya que es un daemon. Como este tipo de procesos no están en constante supervisión de los administradores, es necesario que implementen un mecanismo para guardar todos los mensajes relevantes, que mencionen los eventos sucedidos en el proceso.

Normalmente los servidores de HTTP proveen un sistema de bitácoras, el cual pueden utilizar las aplicaciones Web, no obstante, cuando la aplicación es algo grande y compleja, se prefiere mantener una bitácora diferente al del servidor HTTP para disminuir el nivel de ruido (información no concerniente a la aplicación directamente). Para lograr esto las aplicaciones Web pueden utilizar diferentes enfoques, entre los principales podemos notar:

[7] Disk And Execution Monitor (Monitor de Disco y Ejecución)

- Utilizar el sistema de bitácoras del sistema operativo (syslog en Unix)
- Emplear un archivo plano donde se guarden los registros
- Emplear una tabla en una base de datos
- Utilizar el correo electrónico para enviar los últimos errores.

Tradicionalmente se han categorizado los registros en las bitácoras para jerarquizar la gravedad de los mismos, y poder darle prioridad a los problemas a resolver. El Unix se manejan estas categorías:

1. emergencia
2. alerta
3. crítico
4. error
5. advertencia
6. noticia
7. info
8. depuración

Bajo este esquema surge un problema cuando una misma bitácora recibe registros de varios daemons. En este caso se pueden crear otras categorías, tal como lo maneja Windows NT.

2.5 Modelo de flujo de datos

Desarrollar una aplicación a través de una conexión sin estado con el usuario va en contra del paradigma habitual del desarrollo de aplicaciones, ya que el programador está acostumbrado a que el proceso que se crea, al correr alguna aplicación, se mantiene en el sistema hasta que el usuario decida terminar o suceda algún evento de excepción que no se puede controlar (como un fallo en el sistema operativo, en el hardware, etc.). Cualquier programa puede visualizarse como una máquina de estado finito, donde el procesador brinca de una línea de código a otro, dependiendo de su lógica. En una aplicación Web, después que se generó una respuesta a la petición del cliente, el servidor HTTP pierde la identificación del cliente hasta su siguiente petición, es por ello que resulta complicado conocer su estado actual en el sistema.

Por ello la manera habitual con la cual se lleva el control de la máquina de estado finito es a través de la transmisión de mensajes entre el cliente y el servidor, donde el cliente avisa al servidor en qué estado se encuentra y el servidor le indica a qué estado ha pasado. Por lo tanto, a cada solicitud HTTP el cliente envía un mensaje al servidor y el servidor le responde con una actualización o con un seguimiento de su estado actual.

Es por lo anterior que en la arquitectura que recomendamos se debe revisar, a cada solicitud HTTP, los mensajes que envía el cliente, los valide la aplicación Web, y en caso de no hallar excepciones, mande ejecutar los procedimientos correctos de su sesión. Esto se cristaliza en la codificación en un *despachador de mensajes*,

que deberá ejecutarse a cada petición HTTP que llegue. Este *despachador* tiene la obligación de recibir los mensajes del cliente, ejecutar operaciones de validación y autenticación y posteriormente mandar ejecutar el procedimiento de acuerdo al mensaje solicitado y al estado encontrado.

La buena operación de este *despachador* es vital para el correcto funcionamiento de la aplicación Web, y cualquier falla o mala implantación del mismo, repercutiría en la inestabilidad del sistema o hasta en la presencia de agujeros de seguridad que llevarían desde la corrupción de la integridad en la base de datos, hasta el mal uso y/o destrucción de la información.

2.6 Acceso a base de datos

Gran parte de las aplicaciones Web sirven como interfaz hacia un almacén de datos, ya que permiten obtener y escribir datos de una manera intuitiva. Para aplicaciones pequeñas que no requieran de gran cantidad de procesamiento de datos, es viable almacenarlos en estructuras específicas que se pueden bloquear y desbloquear según las necesitades de la aplicación.

Sin embargo, para aplicaciones que requieren manipulación compleja de los datos, control de acceso concurrente de usuarios, mayor estabilidad, control de integridad de datos, entre otros, es imprescindible el uso de un manejador de bases de datos o DBMS con soporte para SQL.

Según la definición de Elmasri y Navathe en *Fundametals of Database Systems*,

un DBMS es "*un sistema de software de propósito general que facilita el proceso de definir, construir, y manipular bases de datos para diferentes aplicaciones*".

El DBMS empleado debe tener las siguientes características[33]:

- **Los datos se guardan en relaciones (o tablas).** Estas tablas deben estar organizadas de manera que cada una de sus columnas pueda ser identificada por su nombre, la manera en la que los renglones estén organizados no es relevante.

- **Las operaciones deben ser relacionales.** Las operaciones que el sistema provea, deben de generar nuevas tablas a partir de otras existentes. Por ejemplo: *SELECT nombre, id FROM customer* debe de crear una nueva tabla como resultado (la nueva tabla no tiene nombre) y ésta es un subconjunto de la tabla existente.

- **Debe de soportar uniones.** Una base de datos relacional debe soportar operaciones de unión, es decir que permita la obtención de datos relacionados de diferentes tablas y ponerlos en una misma tabla de resultados.

Independientemente de qué DBMS se elija y de cómo trabaje, lo que se busca es que el manejador provea de una API independiente del manejador. Entre las APIs independientes de manejador más comunes están ODBC, JDBC y DBI.

El papel de una API independiente del manejador es crear una interfase común para los diferentes manejadores. De esta manera, una aplicación programada que

usa una API independiente para comunicarse con la base de datos, puede migrarse fácilmente a otro manejador, siempre y cuando éste cuente con soporte para la API independiente para la que fue creada la aplicación.

Recomendamos la utilización de un lenguaje de programación que provea de mecanismos de conexión a manejadores de base de datos en un grado de abstracción que permita una fácil migración y se adhiera a los estándares de la industria. Esto se logra por medio de la librería DBI (2.2).

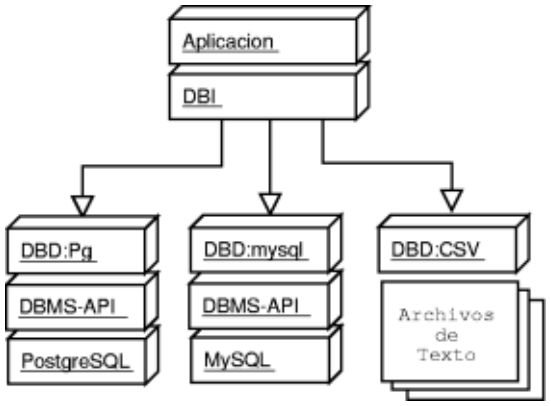

Figura 2.2: Acceso a datos con independencia del DBMS

2.6.1 Conexión persistente a la base de datos

Un problema muy importante que se presenta en el desarrollo de aplicaciones Web es la persistencia a la conexión. Como ya se estableció (capítulo 1.2), las conexiones HTTP terminan cuando el servidor responde al cliente, por tanto si antes de esa respuesta se ejecuta un proceso que acceda a una base de datos, este proceso debe abrir una conexión con el DBMS para finalmente cerrarla al momento

de terminar el proceso. Este esquema produce una sobrecarga de procesamiento cuando el acceso simultáneo de usuarios crece (*overhead*) y realentiza sensiblemente el procesamiento del servidor y la atención a las solicitudes de los clientes. Por ello se debe exigir que la conexión a la base de datos sea persistente y segura. Esta característica debe ser provista por el marco de trabajo y se basa en la creación de una zona de memoria compartida donde se establecerán las variables y procedimientos que accederán a la base de datos, y los procesos que atiendan a las solicitudes Web deberán interactuar con esta zona de memoria compartida y no directamente con el DBMS. Así mismo está la alternativa de crear un proceso alterno al servidor de HTTP que se encargue de la conexión persistente y provea de una comunicación confiable con el servidor de HTTP, sin embargo, como ya se explicó (capítulo 1.6.2), desarrollar este servicio es una tarea compleja, aunque más flexible.

2.7 Separando la capa de presentación

En el desarrollo de aplicaciones Web hay una gran tendencia a separar la presentación, la información en HTML/XHTML y la lógica de la aplicación [50]. Esto se hace con el fin de delegar roles de trabajo, dentro del desarrollo del sistema: los diseñadores gráficos que se darán a la tarea de elaborar la capa de presentación de la aplicación utilizando HTML/XHTML y hasta técnicas de HTML dinámico (DHTML) en aras de la facilidad de uso. Los programadores y administradores de las bases de datos escribirán la lógica de la aplicación y las rutinas de acceso a

datos.

En este punto hay que escoger entre dos alternativas: se le otorga el control de la lógica a los documentos en HTML y utilizando el modelo de desarrollo de *código incrustado en el HTML,* o se emplea el concepto de plantillas en la programación. Una plantilla es un documento completo en el cual existen partes que se pueden llenar con información variable para adaptarlo a la situación necesitada. Estas plantillas son leídas y procesadas por la aplicación para luego ser enviadas al agente del usuario. Las plantillas pueden ser constantemente modificadas por los diseñadores gráficos sin el temor de corromper el funcionamiento de la aplicación.

La diferencia entre la programación de código incrustado y el uso de plantillas radica en el programa procesador de documentos. Si éste interactúa directamente con el servidor HTTP y ofrece gran flexibilidad y así como una API completa y acceso a una gran colección de objetos, entonces estaremos bajo el modelo de código incrustado. Por el contrario, si el programa procesador de documentos es limitado, simple y puede ser manipulado desde cualquier programa de aplicación, entonces estaremos trabajando con el esquema de plantillas.

2.7.1 Presentación interactiva

El HTML dinámico puede permitir a los documentos Web tener una apariencia y actuar como una aplicación de escritorio o una producción multimedia. Ejemplos de lo anterior puede citarse el cambio del formato de un texto cuando el usuario pase el ratón sobre él; o permitir al usuario hacer “arrastra y suelta” de una imagen

en otro lugar de la página Web.

El HTML dinámico no es más que la conjunción creativa del HTML, CSS, un lenguaje del lado de cliente como ECMA y un agente de usuario que soporte lo anterior, además de la especificación DOM[8][37]. Alternativamente puede utilizar tecnología como la Flash de Macromedia, o Applets Java o componentes ActiveX, pero están fuera del espectro del presente trabajo.

La capacidad de cambiar el contenido de un documento, de acuerdo a la interacción con el usuario, es muy importante para que el usuario se sienta contento utilizando la aplicación; se le puede proveer de retroalimentación inmediata, sin necesidad de esperar una respuesta del servidor. Al ofrecerle una interfaz parecida a las de las aplicaciones de escritorio, el usuario se sentirá más cómodo utilizando la aplicación Web.

No obstante, se corre el riesgo de hacer crecer los documentos HTML, y hay que tener en cuenta que el ancho de banda es un recurso muy limitado y la paciencia de los usuarios poca, por lo que debemos programar con sobriedad pero sin perder la elegancia. Además hay que tener en mente que no todos los agentes de los usuarios soportan las mismas características, y que por lo tanto se debe asegurar siempre que la aplicación Web se despliegue correctamente en cada agente posible del usuario, ya que no existe normalmente la capacidad de exigirle alguno en específico al usuario.

[8]Document Object Model (Modelo de Documento Objeto)

Capítulo 3

Aspectos importantes en el análisis y diseño de una aplicación Web

En los capítulos anteriores, se ha analizado la estructura de una aplicación Web así como algunos puntos que se deben de considerar al desarrollar este tipo de aplicaciones. Con estos fundamentos, se está en posibilidades de introducirse en el mundo del desarrollo de software para el Web. El primer paso para el desarrollo de una aplicación Web robusta es definir el problema que deseamos resolver y hasta qué punto una aplicación Web puede ayudar a resolverlo, posteriormente hay que definir las necesidades que serán cubiertas por ella, el siguiente paso es comenzar a diseñar el esquema general de la aplicación incluyendo y realizar un prototipo que se usará para corroborar que el cumplimiento de las necesidades sea completo.

3.1 Análisis del sistema

El análisis de sistema es la etapa en la que se realiza una investigación acerca de el problema que se pretende resolver y cómo puede ser resuelto. En esta fase del desarrollo, la comunicación entre los analistas, los programadores y los usuarios es de gran importancia para que el sistema cumpla con las expectativas planteadas.

El primer paso en la etapa de análisis es definir el problema. Una vez hecho, se indica qué necesita ser resuelto, independientemente de si la solución se obtendrá por medio de un sistema de cómputo u otros medios. El problema debe de ser definido en conjunto por los usuarios, los diseñadores del sistema, y los desarrolladores.

Una vez definido el problema, el siguiente paso es analizarlo y obtener la lista de requerimientos. Uno de los aspectos más importantes para que una aplicación Web sea exitosa, es que ésta cubra (o supere) las necesidades para la que fue planeada.

El análisis de requerimientos es: “el proceso de analizar las necesidades de información de los usuarios finales, el ambiente organizacional y de cualquier sistema actualmente utilizado con el fin de determinar las características de un sistema que pueda satisfacer esas necesidades”[35]. Es muy importante la participación de los usuarios finales en esta etapa para asegurar que el sistema funcione adecuadamente y cumpliendo sus expectativas. Además, hay que asegurar que los requerimientos sean registrados en un documento para ser consultados durante el proceso de desarrollo del sistema con el fin de verificar que se cumpla con las

necesidades planteadas.

La definición de los requerimientos es importante porque[36]:

- El no desarrollar y documentar una buena especificación de requerimientos es la principal causa de que los proyectos de software fracasen.
- Un sistema elaborado en base a un análisis de requerimientos de poca calidad, decepcionará al usuario y le traerá muchos problemas al desarrollador.
- Es muy costoso corregir los errores cometidos en la obtención de requerimientos en etapas de desarrollo posteriores.

3.2 Diseño de la interfaz

Una vez que se cuenta con el análisis del sistema y la lista de requerimientos, el siguiente paso es la creación de la interfaz de usuario. Esta es una etapa importante en el desarrollo de aplicaciones, la interfaz de usuario es la que va a determinar que tan utilizable va a ser un sistema.

En ocasiones los programadores subestiman la importancia de la interfaz de usuario y concentran sus esfuerzos en la funcionalidad del sistema y en optimizar su desempeño. Sin embargo, si la interfaz de usuario es pobre, el usuario no se sentirá cómodo al usar la aplicación o, peor aún , dejará de usarla.

Una buena interfaz de usuario debe de ser diseñada según ciertos lineamientos[37]:

1. **Consistencia.** Todas las pantallas de la aplicación Web deben de tener una distribución consistente de imágenes, texto y controles gráficos[1]. Ante acciones de usuario comunes (elegir un elemento de una lista desplegable, dejar un campo en blanco donde no es permitido) el sistema debe de mostrar mensajes con la misma estructura sin importar en que parte del sistema ocurra el evento. La única manera de lograr una interfaz de usuario consistente es la implementación de estándares de interfaz. Con ello, los grupos de desarrollo sabrán como acomodar los controles gráficos, dónde poner las etiquetas, que lineamientos de justificación de texto seguir, entre otras cosas.

2. **Dar soporte diferentes niveles de usuarios.** Una aplicación pueda ser usada tanto por usuarios neófitos como por expertos, bajo este tenor, la aplicación debe de ser configurable para mostrar toda la gama de opciones que un usuario experto puede requerir o solo mostrar las indispensables para que un usuario neófito pueda hacer su trabajo sin verse agobiado por una gran cantidad de opciones.

3. **Flujo de pantallas.** El paso de una pantalla a otra debe de ser coherente con el trabajo que intente realizar el usuario. De esta manera, no es nada conveniente hacer que el usuario tenga que pasar por una pantalla de configuración de colores en pantalla para llegar a la pantalla de configuración de

[1]Elementos que permiten la creación de una Interfaz gráfica de usuario tales como: botones, cuadros de texto, listas desplegables, casillas de verificación, entre otros. En inglés son conocidos como *widgets*

márgenes de impresión.

4. **No sobrepoblar las pantallas.** Al presentar gran cantidad de controles gráficos en una misma pantalla dificultan la comprensión de la misma. Si para realizar una tarea determinada se requiere de la obtención de mucha información por parte del usuario, es muy recomendable dividir en diferentes pantallas sucesivas la captura de información y/o establecimiento de opciones. A esto último se le conoce como asistente o "wizard".

5. **Agrupar elementos relacionados.** Una buena práctica en la creación de interfaces de usuario es la agrupación de elementos que estén relacionados entre sí. Por ejemplo, podemos delimitar por un rectángulo o algún otro tipo de imagen los datos generales de un empleado tales como nombre, dirección , edad, etc.; En otro cuadro podemos agrupar otros datos tales como: Numero de cuenta bancaria, fecha de expiración, saldo, etc.

Los anteriores puntos son aplicables a todos los tipos de aplicaciones, ya sea Web o de escritorio. Sin embargo, existen lineamientos que son específicos al desarrollo de aplicaciones Web, una buena referencia es "*Los siete pecados mortales de un sitio Web (y por qué deben de evitarse a toda costa)*" de Jesse Berst [47]. Los grandes pecados o errores que se pueden cometer en el diseño de un sitio Web son:

1. **Navegación inconsistente.** A veces se hace click en una barra de menú lateral, otras es un menú despegable. La ubicación de los títulos, gráficos,

vínculos, etc. debe de ser consistente en todas las páginas del sitio.

2. **Vínculos rotos.**

3. **Sitios que requieren de un navegador específico.** Es muy molesto toparse con sitios que están diseñados para un navegador específico, así como los que requieren que se baje un *plug-in*[2] determinado sin ofrecer una versión del documento que no requiera de dicho plug-in.

4. **No poner información de contacto.** Muchos diseñadores de sitios de empresas importantes olvidan poner la dirección de la empresa, números telefónicos, números de atención a clientes, etc.

5. **Uso de marcos.** También conocidos como *frames*, traen consigo problemas con el uso de los botones Anterior y Siguiente de los navegadores, muchas veces son creados de un tamaño fijo y la información contenida en ellos no es visible completamente, etc.

6. **Sitios que abren otras ventanas de navegación.** Es molesto para el navegador ingresar a un sitio y que éste abra nuevas ventanas de navegación con vínculos a otros sitios.

7. **Símbolos de "En construcción".** La mayoría de los sitios en el Web están en constante evolución, Es poco profesional indicar que el sitio está en

[2]Componente de software que agrega funcionalidad a un navegador sin necesidad de recompilarlo. Un ejemplo típico de plug-in es el de Macromedia Flash, el cual es usado para visualizar animaciones vectoriales en páginas Web. Éste plug-in puede ser instalado para el Internet Explorer y Netscape Navigator.

construcción.

3.3 Creación de un prototipo

Un prototipo es una implementación básica de la interfaz de usuario que incluye todos los componentes visuales de la aplicación así como el código necesario para permitir al usuario desplazarse a través de todas las pantallas. Como el prototipo sólo es un modelo de prueba, no tiene integrada ninguna de las reglas de negocios ni tiene acceso a la base de datos. El prototipo tiene la finalidad de presentar a los usuarios finales las pantallas con las que contará el sistema final para que éstos puedan determinar si el sistema cumple con sus necesidades.

La elaboración del prototipo es un proceso iterativo descrito a continuación:

1. **Determinar la necesidades.**En esta etapa se define qué es lo que los usuarios esperan realizar con el sistema, qué reportes esperan que arroje, etc.

2. **Construcción del prototipo.** Para la elaboración del prototipo es muy recomendable el uso de herramientas de alto nivel, tales como un editor de documentos HTML ya sea uno estándar o uno “Lo que ves es lo que obtienes” (WYSIWYG, What You See Is What You Get, por sus siglas en inglés). Con el editor HTML se crean las diferentes pantallas de las que se compondrá el sistema estableciendo los vínculos entre ellas. En esta etapa es conveniente la creación de un diagrama de flujo de interfaz. Este diagrama muestra en un diagrama de bloques las pantallas de las que se compone

el sistema y como se puede llegar a ellas. Con este diagrama se puede tener una mejor comprensión del sistema y se puede verificar fácilmente que el flujo de pantallas sea coherente.

3. **Evaluar el prototipo.** Una vez que se tiene una versión del prototipo, el siguiente paso es evaluarla con los usuarios para determinar si se cumple con sus expectativas. Es importante determinar lo que es correcto, lo que está mal y lo que falta. Si se descubre que hay errores en el prototipo o que éste tiene características faltantes tenemos que regresar al paso 1.

4. **Determinar si el prototipo está terminado.** Una vez que en el paso anterior no se determinan nuevos requerimientos o estos son de poca importancia.

Para ejemplificar la creación del prototipo, se comienza por crear un diagrama de flujo de pantallas para un sistema de una compañía de paquetería. En la figura 3.1 se muestra el diagrama que ilustra cual será la secuencia de pantallas que el usuario podrá utilizar.

En la parte de las pantallas para la elaboración de un nuevo envío se muestra que para realizar la tarea es necesario pasar por varias ventanas, en la primera se capturan los datos del remitente, en la segunda los datos del destinatario y en la tercera los datos pertinentes al paquete que se enviará, por último se muestra una pantalla que contiene un resumen con los datos previamente capturados. Este flujo de pantallas que piden información para realizar una tarea específica es conocido

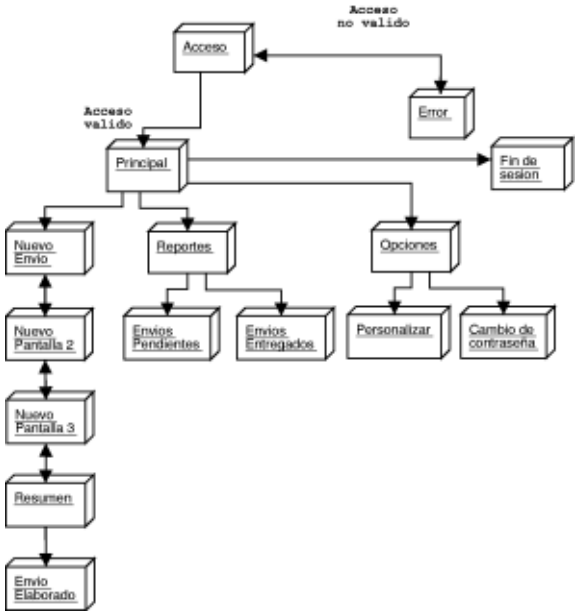

Figura 3.1: Diagrama de flujo de pantallas

como *Asistente.* Se puede observar en la figura 3.1 que en las pantallas para la captura de un envío, las flechas apuntan en ambas direcciones, lo que indica que el usuario puede avanzar por las pantallas y retroceder, en caso de que hubiese cometido algún error.

El paso siguiente es crear el esqueleto del prototipo en HTML[3]. Se deben de crear todas las pantallas de las que se compondrá el sistema, incluidos los reportes. En las figuras siguientes se muestran algunos ejemplos de las pantallas creadas.

[3]Para la elaboración de los ejemplos de este trabajo se utilizó Bluefish (http://bluefish.openoffice.nl). Un editor de HTML liberado bajo GPL muy versátil.

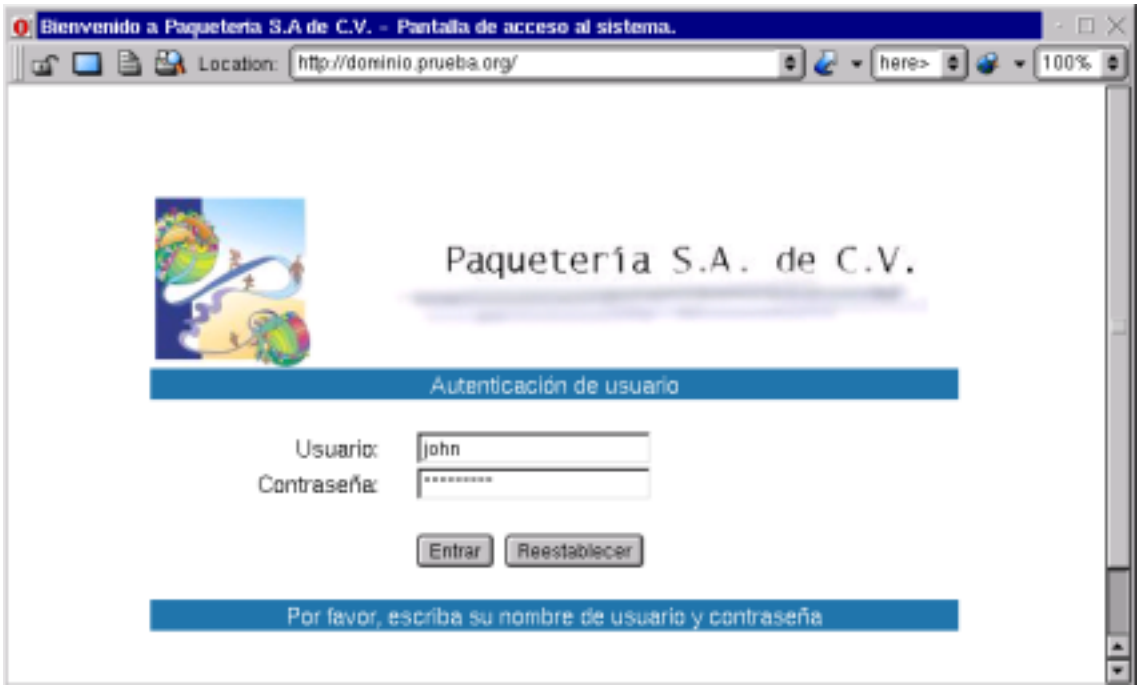

Figura 3.2: Pantalla de acceso al sistema

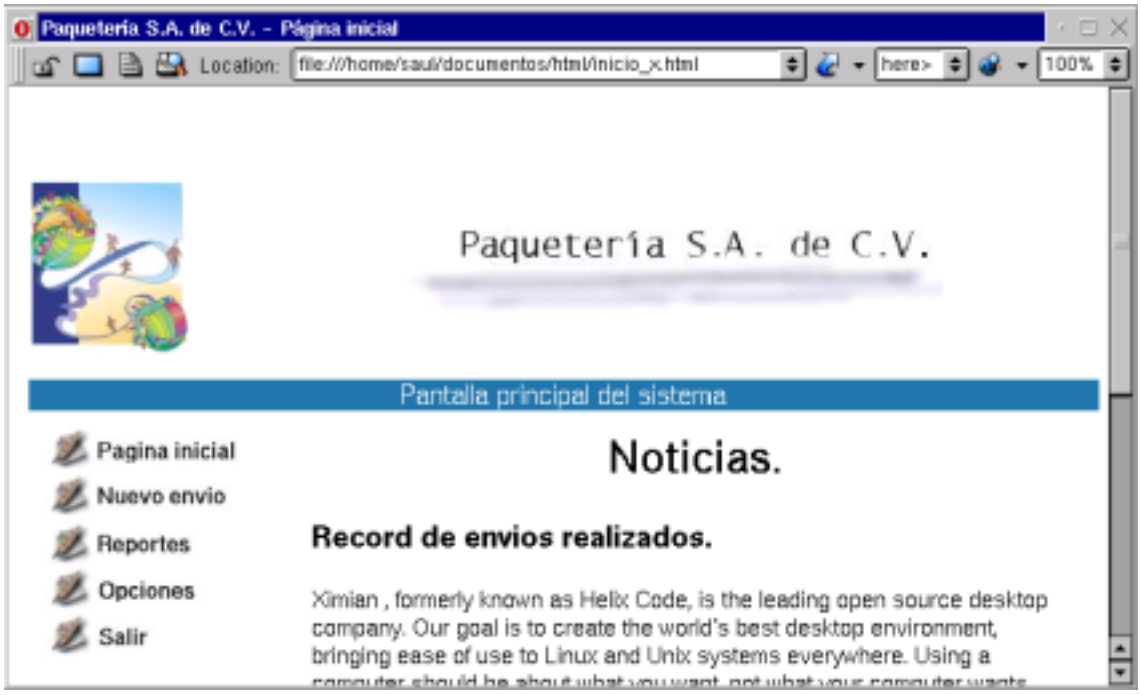

Figura 3.3: Pantalla Principal

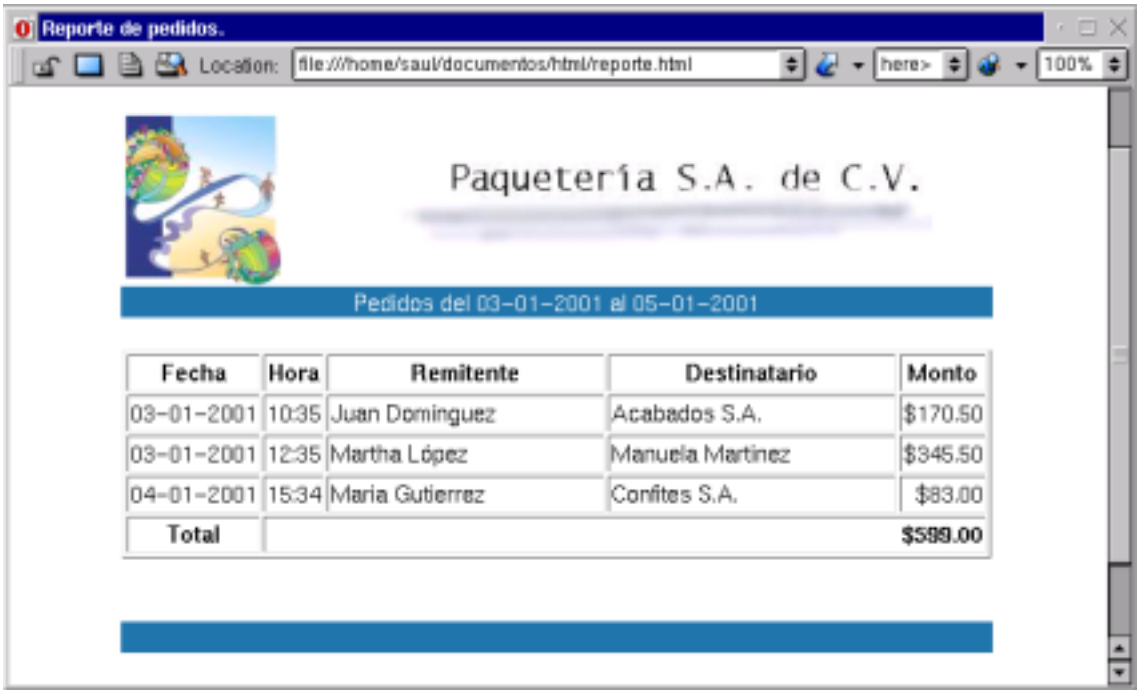

Figura 3.4: Pantalla de Reporte

Capítulo 4

Ejemplo práctico: Cómo desarrollar una aplicación Web

En el presente capítulo, se hablará acerca de las herramientas necesarias para el desarrollo e implementación de una aplicación Web: manejador de base de datos, servidor Web, lenguaje de programación Perl, entre otros (figura 4.3). Posteriormente se menciona algunos aspectos importantes para el diseño de la base de datos y por último, la ilustración por medio de ejemplos las etapas que se deben de cubrir para la elaboración de una aplicación Web.

4.1 Herramientas necesarias

Para el desarrollo de una aplicación Web siguiendo los lineamientos presentados en este documento, es necesaria una computadora que tenga instalados los paquetes que se mencionan a continuación.

4.1.1 PostgreSQL

PostgreSQL es un sofisticado manejador de bases de datos relacionales-Orientadas a objetos. Soporta casi todas las instrucciones SQL, incluyendo subconsultas, transacciones, funciones y tipos de datos definidos por el usuario.

Los sistemas de administración de base de datos (DBMS) relacionales tradicionales soportan un modelo de datos consisten en una colección de nombres relacionados, conteniendo atributos de una tipo específico. En los sistemas comerciales actuales, los tipos posibles incluyen números de punto flotante, enteros, cadenas de caracteres, monetario, y fechas. Es comúnmente reconocido que este modelo es inadecuado para futuras aplicaciones de procesamiento de datos. El modelo relacional ha reemplazado exitosamente modelos previos en parte por su "simplicidad espartana". Sin embargo, como se mencionó, esta simplicidad muchas veces hace que la implementación de ciertas aplicaciones sea muy difícil. Postgres ofrece un poder adicional para incorporar los siguientes cuatro conceptos en manera que los usuarios puedan fácilmente extender el sistema:

- clases

- herencia
- tipos
- funciones

Otras características proveen poder y flexibilidad adicional:

- restricciones
- *triggers*
- reglas
- integridad de transacciones

Estas características ponen a Postgres en la categoría de base de datos referidas como *objeto-relacionales*. Hay que notar que este concepto es distinto al referido como las *orientadas a objetos*, que en general no están bien dispuestas para soportar los lenguajes tradicionales de base de datos relacionales. Así, aunque Postgres tiene algunas características orientadas a objetos, está firmemente afianzado en el mundo de las base de datos relacionales. De hecho, algunas bases de datos comerciales han recientemente incorporado características implementadas inicialmente por Postgres[46].

4.1.2 Lenguaje de programación Perl

Perl (Practical Extraction and Report Language por sus siglas en inglés) es un lenguaje de programación que fue creado en 1986 por Larry Wall. Perl es un lenguaje optimizado para buscar en archivos de texto arbitrarios, extraer información de esos archivos de texto, e imprimir reportes basado en esa información. También es un buen lenguaje para muchas tareas de administración de sistemas. El lenguaje tiene la intención de ser práctico (fácil de usar, eficiente y completo) más que bien parecido (pequeño, elegante, mínimo).

Perl combina algunas de las mejores características de C, sed, awk y sh, así la gente familiarizada con esos lenguajes tendrán poca dificultad para utilizarlo. Perl no impone arbitrariamente el tamaño de los datos que manejes; utiliza sofisticadas técnicas para la búsqueda de patrones sobre textos y datos binarios; implementa mecanismo para el seguimiento del flujo de datos, previniendo agujeros de seguridad[39].

Además, desde la versión 5, provee de los siguientes beneficios adicionales:

- Modularidad y reusabilidad de innumerables módulos disponibles a través del CPAN[1]
- Empotrable y extensible
- Las subrutinas pueden ser sobrecargadas, autocargadas y prototipeadas

[1]Comprehensive Perl Archive Network (Red Global de Archivos Perl). Es una serie de servidores espejos que contienen módulos reutilizables en Perl libremente distribuidos.

- Estructuras de datos arbitrariamente anidadas y funciones anónimas
- Programación orientada a objetos
- Compatibilidad con código en C o en Perl-Bytecode
- Soporte para procesos ligeros (hilos)
- Soporte para internacionalización, localización y Unicode
- Manejo del ámbito a nivel léxico
- Ambientes de depuración interactivos
- Se adhiere al estándar POSIX por completo

4.1.2.1 DBI/DBD

El DBI es un módulo para el acceso a base de datos del lenguaje de programación Perl. Define un conjunto de métodos, variables y convenciones que proveen de una interfaz consistente a bases de datos, independientes de la base de datos que se utilice.

Es importante recordar que el DBI es únicamente una interfaz. El DBI es una capa aglutinante entre una aplicación y uno más módulos manejadores de base de datos. Es el módulo manejador el cual hace la mayoría del trabajo real. El DBI provee una interfaz estándar y una marco de trabajo para los manejadores que se operarán.

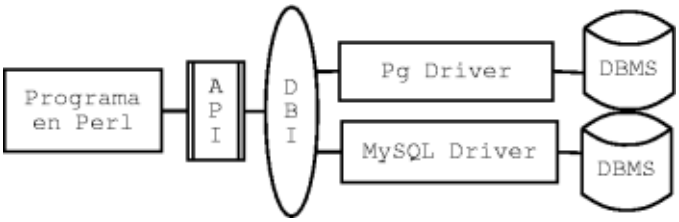

Figura 4.1: Arquitectura de una Aplicación DBI

El API define las interfaces a llamadas a servicios y a variables del DBMS para los programas en Perl que las utilizarán. El API es implementado por la extensión Perl DBI.

El DBI envía las llamadas a servicios al driver apropiado para la ejecución actual. El DBI es también responsable de la carga dinámica de los controladores, verificación y manejo de errores, provisión de implementaciones a métodos por defecto, y muchas otras tareas no específicas de base de datos.

Cada controlador contiene implementaciones para los métodos del DBI usando la interfaz de funciones privadas del DBMS correspondiente [40]. Este controlador es conocido como DBD, que es el acrónimo reservado de DBI database driver.

4.1.2.2 Template Toolkit

El Template Toolkit es una colección de módulos que implementan un sistema rápido, flexible, poderoso y extensible de procesamiento de plantillas. Fue originalmente diseñado y permanece principalmente útil para general contenido Web dinámico, pero puede ser igualmente utilizado para procesar cualquier tipo de documento de texto.

En su nivel más simple provee de una fácil manera de procesar archivos plantilla, llenando las referencias a variables incrustadas con sus valores equivalentes.

Una de las características del Template Toolkit, además del procesamiento de plantillas, es su habilidad para ligar variables de la plantilla con cualquier tipo de dato de Perl: escalares, listas, tablas, arreglos, subrutinas y objetos.

Provee de un número de directivas adicionales para procesamiento avanzado. También soporta una arquitectura de carga dinámica de módulos de Perl, extendiendo así su funcionalidad.

4.1.2.3 Apache Session

El Apache::Session en un módulo Perl que actúa como marco de trabajo persistente el cual es particularmente útil para rastrear la información de sesión entre solicitudes HTTP.

El módulo consiste de tres componentes: la interfaz, el almacén de objetos y el administrador de seguros.

4.1.3 El servidor de HTTP Apache

El servidor HTTP Apache es un servidor Web poderoso y flexible. Implementa los último protocolos incluyendo el HTTP/1.1 (RFC2616). Es altamente configurable y extensible con módulos de terceros. Puede ser personalizable escribiendo módulos a través de una API. Provee acceso completo al código fuente, y utiliza

una licencia no restrictiva. Corre bajo Windows NT/9x, Netware 5.x, OS/2, y en la mayoría de las versiones de Unix, así como en varios otros sistemas operativos.

Todas estas características lo convierten en el servidor HTTP más utilizado en Internet, con un 59% de presencia entre los servidores Web a nivel mundial[2], en febrero del 2001 (figura 4.2). Este porcentaje supera al representado por la suma de los todos los sitios que usan otros servidores.

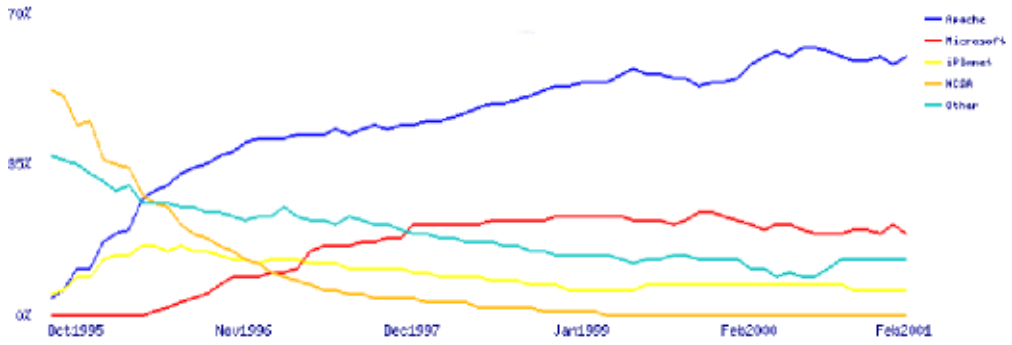

Figura 4.2: Participación del servidor Web Apache

4.1.3.1 mod_perl

De forma simple, mod_perl es un paquete de software que incrusta un intérprete de Perl en el servidor HTTP Apache, otorgando al desarrollador acceso a la API de Apache desde el mismo Perl. Esto permite la construcción de módulos que extienden al Apache en lenguaje Perl, en lugar de C.

El entorno de mod_perl es muy diferente al de CGI habitual. Desde que el intérprete de Perl es parte del proceso servidor, no hay necesidad de iniciar o detener

[2]Información obtenida en http://www.netcraft.com/survey/

este intérprete por cada solicitud. Y, desde que los programas en mod_perl son en realidad módulos Perl que extienden la funcionalidad del servidor, no hay necesidad de recompilar el código hecho en Perl por cada solicitud. Lo anterior hace que cada ejecución de código en Perl dentro el entorno de mod_perl, sea extremadamente rápido.

Por otro lado, y como ya se mencionó en esta sección, mod_perl es una API en Perl para el servidor Apache. Esto significa que cualquier facilidad que el Apache provee, puede ser accedible y extendida usando Perl. Observe la siguiente lista que describe las diferentes fases de un proceso de servidor Web [42]:

Fase del Proceso	**Descripción**
PerlChildHandler	Cuando un nuevo proceso hijo es creado
PerlPostReadRequestHandler	Se ejecuta una vez por cada transacción
PerlTransHandler	Traducción de URI. Emplado en proxies
PerlInitHandler	PostReadRequest o HeaderParser
PerlHeaderParserHandler	Cuando el URI ha sido mapeado a ruta. Robots.
PerlAccessHandler	Control de acceso: ¿cualquiera puede entrar?
PerlAuthenHandler	Autenticación: ¿nombre de usuario y contraseña válidos?
PerlAuthzHandler	Autorización: ¿el usuario tiene permitido accesar?
PerlTypeHandler	Asginación de un tipo MIME provicional
PerlFixupHandler	Cambios de último minuto en las variables de entorno
PerlHandler	Realiza la respuesta
PerlLogHandler	Hace la escritura a bitácoras de la transacción
PerlCleanupHandler	Limpia. Envía correo.
PerlChildExitHandler	Se ejecuta antes de que el proceso hijo termine

Tabla 4.1: Fases de un proceso del servidor Web Apache [44]

Con la habilidad de agregar lógica a algunos o a todas estas fases se crea un ambiente extremadamente robusto que permite la construcción de aplicaciones Web avanzadas y de alto rendimiento.

Tipo de Código	Hits/sec/MHz	OS	Servidor	Aplicación
PHP	1,565	Linux	Apache	mod_php 4a
PHP	1,42	Linux	Apache	mod_php 3
ModPerl Handler	1,236	Linux	Apache	mod_perl
ASP VBScript	1,015	WinNT	IIS	asp/vbscript
Java Servlet	0,335	Linux	Apache	jrun

Tabla 4.2: Aprovechamiento de los Servidores de Aplicaciones Web más comunes [49]

4.1.3.2 mod_ssl

El proyecto mod_ssl provee al servidor Web Apache 1.3 de una criptografía fuerte a través de los protocolos Secure Socket Layer (SSL v2/v3) y el Transport Layer Security (TLS v1) con la ayuda del proyecto de Software Libre OpenSSL.

Alguna de sus características son:

- Es Software Libre bajo una licencia parecida a la BSD.
- Disponible tanto para Unix como para plataformas Win32.
- Soporta los protocolos SSLv2, SSLv3 y TLSv1
- Soporta tanto los algoritmos de cifrado RSA y Diffie-Hellman
- Integrado al Apache mediante una API extendida (EAPI).
- Manejo avanzado de contraseñas para llaves privadas
- Autenticación basada en certificados X.509 tanto para el cliente como para el servidor

- Soporte para la revocación del certificados X.509
- Soporte para la renegociación de parámetros SSL por URL
- Facilidades adicionales de control de acceso basado en expresiones booleanas
- Compatibilidad hacia atrás con otras soluciones basadas en Apache SSL
- Poderoso sistema de bitácoras
- Asistencia para la generación de certificados X.509v3 (tanto en RSA como en DSA)

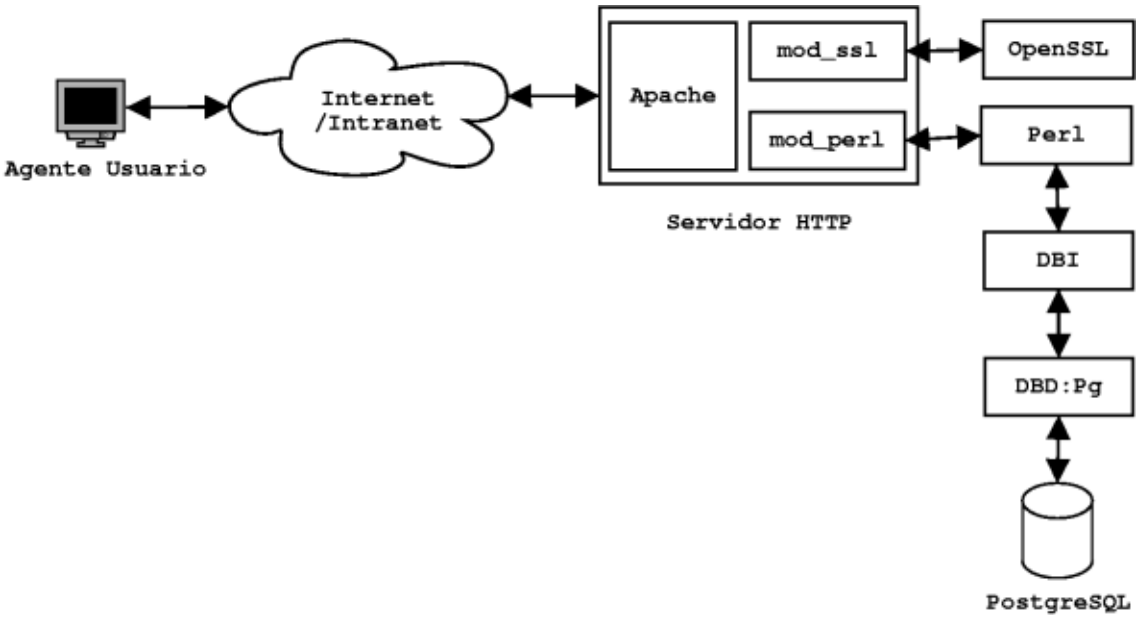

Figura 4.3: Diagrama del Software Involucrado

4.2 Diseño de la base de datos

El diseño de la base de datos es crucial para el buen funcionamiento de una aplicación Web , si no se cuenta con un diseño de calidad, la operación de la aplicación Web se puede ver entorpecida o de plano anulada.

Con el fin de obtener el mejor desempeño y flexibilidad de la aplicación Web, es recomendable seguir ciertas guías para el diseño de la base de datos. Bajo estos lineamientos, es posible desarrollar un sistema de base de datos escalables y que pueden manejar desde la más sencilla consulta hasta la más compleja.

Es muy importante cotejar los requerimientos con el diseño de la base de datos, ¿el diseño de la base de datos permite realizar todas las consultas que se necesitan?. Además, conforme se progresa en la fase de desarrollo de la aplicación o incluso después de que ésta es terminada, es posible que surjan nuevos requerimientos; la base de datos debe de ser lo suficientemente flexible para permitir cambios que satisfagan esos requerimientos.

Antes de comenzar a crear las tablas se debe de crear un diccionario de datos. El diccionario de datos es una referencia que describe las entidades y sus cada uno de sus atributos. En él se especifican el nombre del campo, su descripción, el tipo de dato (entero, cadena de texto, monetario, etc.) y el tamaño, si aplica.

El siguiente paso es crear la base de datos. Para la creación de la base de datos en PostgreSQL es necesario ejecutar el comando *createdb* como el superusuario de PostgreSQL

```
$createdb libreria
```

Es recomendable definir el esquema de la base de datos en un archivo de texto que contenga todas las sentencias SQL que se requieran para definir las tablas, sus atributos, los tipos de datos, los índices, etc. La razón para hacerlo de esta manera y no tecleando las instrucciones directamente en el *pgsql*[3] es que durante el desarrollo de la aplicación, frecuentemente es necesario borrar la base de datos y crearla de nuevo y teniendo el archivo de texto nos facilita mucho las cosas. Como ejemplo sencillo de un archivo para crear una base de datos de una librería véase el código 5.

Una vez que se tiene el archivo con la definición del esquema, hay que ejecutar el comando:

```
psql -f libreria.sql libreria
```

4.3 Programación

4.3.1 Acceso a datos

Como mencionamos en el capítulo 4.1.2.1, el acceso a los datos se hace por medio de la API independiente del manejador DBI.

[3]El pgsql es la herramienta interactiva de consulta en modo texto de PostgreSQL, desde él se pueden realizar consultas, modificar registros, crear y eliminar tablas, etc.

Algorithm 5 Archivo con definición de esquema de una base de datos

```
-- ================================================================
--Database name: libreria
--DBMS name: PostgresSQL
--Created on: 25/01/2001
--Last modification: 09/03/2001
--Author: Saul Lopez Santoyo
-- ================================================================
DROP TABLE LIBRO;
DROP TABLE AUTOR;
DROP TABLE AUTOR_DE;
DROP SEQUENCE SEQ_AUTOR;

-- ================================================================
-- AUTOR
-- TABLA QUE CONTIENE LOS DATOS DE UN AUTOR
-- ================================================================
CREATE SEQUENCE SEQ_AUTOR
INCREMENT 1
MINVALUE 1
MAXVALUE 100000
START 1
CACHE 1
CYCLE;

CREATE TABLE AUTOR (
       ID_AUTOR    INT4 DEFAULT NEXTVAL ('SEQ_AUTOR') PRIMARY KEY,
       APELLIDOS   VARCHAR(50) NOT NULL,
       NOMBRES    VARCHAR(30) NOT NULL
);

-- ================================================================
-- AUTOR
-- INFORMACION DE UN LIBRO
-- ================================================================
CREATE TABLE LIBRO (
       ISBN    VARCHAR(50) PRIMARY KEY,
       TITULO    VARCHAR(60) NOT NULL,
       EDICION    INTEGER
);

-- ================================================================
-- AUTOR_DE
-- TABLA QUE SIMPLIFICA LA RELACION DE MUCHOS A MUCHOS ENTRE LIBRO Y AUTOR
-- ================================================================
CREATE TABLE AUTOR_DE (
       FK_ID_AUTOR  INTEGER NOT NULL,
       FK_ISBN     VARCHAR(50) NOT NULL,
       CONSTRAINT   _FK_ID_AUTOR    FOREIGN KEY (FK_ID_AUTOR)
    REFERENCES AUTOR(ID_AUTOR),
       CONSTRAINT   _FK_ISBN     FOREIGN KEY (FK_ISBN)
    REFERENCES LIBRO(ISBN)

);
```

Para usar el DBI es necesario establecer una conexión al servidor de bases de datos. Una vez que se tiene la conexión, se pueden ejecutar comandos SQL y procesar los datos regresados.

4.3.1.1 Conexión a la base de datos.

Para conectarse a la base de datos es necesario indicar el nombre del origen de datos, un nombre de usuario y una contraseña.

Algorithm 6 Conexión a la base de datos

```
use DBI;

my $datasource = 'dbi:mysql:db:localhost';
my $username = 'dba';
my $password = 'secreto';
my $attr = { RaiseError => 1, AutoCommit => 1 }

my $dbh = DBI->connect($datasource, $username, $password, $attr);

# Verificar si se realizó la conexión
unless (defined $dbh) {
die $DBI::errstr;
} else {
print "Conexión realizada\n";
}

# Desconectarse de la db
$dbh->disconnect;
```

4.3.1.2 Ejecución de instrucciones SQL

Para ejecutar una instrucción SQL se utiliza el método:

Algorithm 7 Instrucciones SQL

```
$rc = $dbh->do("Insert into employee values ('Martin', 'Dominguez', 'Fresno 234')");
```

El valor de $rc nos indica el resultado de la instrucción, si se ejecutó con éxito su valor será diferente de cero. Si ocurrió un error al ejecutar la instrucción, el valor será igual a cero. Se puede ver el mensaje de error en la variable

```
$DBI::errstr
```

Si una instrucción será ejecutada muchas veces, es mejor indicarle al DBI que prepare la instrucción antes de ejecutarla. Al preparar una instrucción, el servidor de bases de datos, si es compatible con esta característica, precompilará la instrucción, lo que puede resultar en una mejora en el desempeño.

Es posible especificar al DBI que se le mandarán parámetros a una instrucción preparada. Por ejemplo la instrucción siguiente:

Algorithm 8 Instrucciones SQL iterativas

```
my $sth = $dbh->prepare("INSERT INTO EMPLOYEE (FNAME, SNAME, ADDRESS) VALUES (?, ?, ?)");
```

Los signos de interrogación indican que se mandarán parámetros a la instrucción. Así, nos regresan la "manija" de instrucción en `$sth`. Posteriormente, podemos ejecutar la instrucción que preparamos únicamente indicando los parámetros:

```
$sht->execute('Martin', 'Dominguez', 'Fresno 234');
```

La instrucción anterior es equivalente a:

```
INSERT INTO EMPLOYEE (FNAME, SNAME, ADDRESS) VALUES ('Martin', 'Dominguez', 'Fresno 234');
```

El uso de la instrucción preparada es más cómodo para los programadores, además de que se puede lograr una mejora en el desempeño si esta instrucción se ejecutara muchas veces.

4.3.1.3 Manejando los resultados de las consultas.

El DBI es muy flexible con respecto al manejo de los resultados. Se pueden manipular el conjunto de registros que resultan de una consulta con varias instrucciones:

fetchrow_array() Obtiene el siguiente renglón como un arreglo de valores:

Algorithm 9 Uso de fetchrow_array

```
while (my @values = $sth->fetchrow_array) {
    my ($fname, $lname, $address) = @values;
    print "$fname $lname vive en $address.\n";
}
```

fetchrow_arrayref() Obtiene el siguiente renglón como una referencia a un arreglo. Esta es la manera mas efectiva de obtener datos.

Algorithm 10 Uso de fetchrow_arrayref

```
while (my $values_ref = $sth->fetchrow_arrayref) {
    my ($fname, $lname, $address) = @$values_ref;
    print "$fname $lname vive en $address.\n";
}
```

fetchrow_hashref() Obtiene el siguiete renglón como un hash con los datos usando el nombre de columna como llave. Este método no tiene un buen desempeño comparado con las otras alternativas.

Algorithm 11 Uso de fetchrow_hashref

```
while (my $row = $sth->fetchrow_hashref) {
     foreach $column (keys $row) {
         print "$column: $$row($column), ";
     }
     print "\n";
}
```

Algorithm 12 Uso de fetchall_arrayref

```
my $rows = $sth->fetchall_arrayref();
     foreach $array_ref (@$rows) {
     my ($fname, &lname, $address) = @$array_ref;
     print "$fname $lname vive en $address.\n";
}
```

fetchall_arrayref() Este método obtiene todos los renglones y los pone como una referencia a un arreglo. Cada elemento de ese arreglo contiene un arreglo de referencias a los valores de cada renglón.

Una vez que se termine de usar el sth, hay que usar el método

```
$sth->finish();
```

4.3.1.4 Conexión persistente

Como ya habíamos comentado previamente, un requerimiento esencial para un servidor de HTTP, es que posea un mecanismo para realizar conexiones a base de datos de manera persistentes, reutilizables por sus procesos.

El Apache junto con su módulo mod_perl lo provee a través del módulo Perl conocido como Apache::DBI. Su forma de inicializarlo es el siguiente.

Algorithm 13 Script de inicialización del Apache

```
use strict;

$ENV{GATEWAY_INTERFACE} =~ /^CGI-Perl/
      or die "GATEWAY_INTERFACE not Perl!";

use Apache;
use Apache::Constants qw(:common);
use Apache::DBI ();
use Apache::Session::DBI;
use DBI ();
use CGI ();
CGI->compile(':all');

my $datasource = 'dbi:mysql:db:localhost';
my $username = 'dba';
my $password = 'secreto';
my $attr = { RaiseError => 1, AutoCommit => 1 }

Apache::DBI->connect_on_init($datasource, $username, $password, $attr);

1;
```

4.3.2 Estableciendo comunicación con el servidor Web.

La herramienta básica para el desarrollo de aplicaciones Web con Perl es el módulo Perl de Apache. Este módulo provee de una interfase entre el intérprete de Perl y el servidor Web Apache. Su principal uso es en aplicaciones desarrolladas en mod_perl aunque se puede usar con otros fines.

La unidad básica de trabajo del módulo Apache es el objeto request (solicitud). Este contiene todo lo que el servidor necesita saber para responder a una petición. Los manejadores de Perl recibirán como parámetro una referencia a este objeto y podrán modificarlo o usarlo de diferentes maneras.

La manera de obtener la referencia al objeto request es la siguiente:

```
my $r = Apache->request;
```

Una vez que se tiene la referencia al objeto request, es posible obtener detalles acerca de la solicitud que elaboró el agente usuario.

```
$r->method
```

Regresa el método de solicitud que el cliente indicó. Puede regresar algo como "GET" o "PUT".

```
$r->protocol
```

Regresa una cadena que indica que protocolo HTTP usa el cliente. Puede regresar algo similar a "HTTP/1.0" o "HTTP/1.1".

Los anteriores son solo algunos de los métodos que provee el modulo de Perl Apache, para mayor información consulte su pagina de manual Manual del modulo de Apache `http://www.perldoc.com/cpan/Apache.html`.

4.3.2.1 Manteniendo el estatus con *cookies*

Como se mencionó en el capítulo 1.2 el protocolo HTTP es incapaz de mantener el contexto de una página a otra. Una forma de mantener el contexto entre el servidor HTTP y el cliente se logra con las *cookies*, estas son paquetes de información establecida por el servidor que se guardan en el cliente y que son enviados al servidor cada que éste lo solicite. De esta manera el servidor puede determinar los movimientos que se han realizado desde un cliente, que variables se han establecido, etc.

Para nuestra aplicación Web, manejaremos varios métodos :

Algorithm 14 Manejo de *cookies*

```
sub fetch_cookie {
   my $self = shift;

   my $cookie = $self->APACHE->header_in('Cookie');
   return () if (!$cookie);
   return $self->parse_cookie($cookie);
}

sub set_cookie {
   my $self = shift;
   my $unset = shift;
   my $session = $unset ? '' : $self->SESSION->{_session_id}; # kill session?

   my $offset = $self->CONFIG->{session_expire};
   my $GMT = 21600; # 6 horas de diferencia para el GMT
   my $now = time + $GMT;
   my $exp = $unset ? $now : $now + $offset;
   my $expires = &Time::CTime::strftime('%a, %d-%b-%Y %X GMT', localtime($exp));
   my $domain = $self->CONFIG->{domain};
   my $path = $self->CONFIG->{path};

   my $cookie = "session_id=$session; domain=$domain; path=$path; expires=$expires";
   $self->APACHE->header_out('Set-Cookie' => "$cookie");

sub parse_cookie {
   my $self = shift;
   my $cookie = shift;
   my %results;

   my (@pairs) = split(';', $cookie);
   foreach (@pairs) {
      s/\s*(.*?)\s*/$1/;
      my ($key, $value) = split('=');
      my (@values) = split('&', $value);
      $results{$key} = \@values;
   }
   return \%results;
}
```

4.3.3 Plantillas

El API del módulo Perl Template es orientado a objetos. Una instancia de este objeto puede procesar múltiples plantilas. Para crear una instancia se declara de la siguiente manera:

Algorithm 15 Instanciación de la clase Template

```
use Template;
my $template = Template->new({
      INCLUDE_PATH => '/var/www/templates',
      DEFAULT => 'nofound.tml',
      OUTPUT => $r
});
```

Las opciones enviadas al método constructor definen tanto el directorio donde se extraerán las plantillas, la plantilla predeterminada, la definición de la salida, que en este caso es la instanciación de el descriptor del proceso HTTP, etc.

Para que la instancia procese las plantillas y genere una salida en base a éstas, hay que decirle el nombre de la plantilla a procesar y una tabla hash con las variables, objetos y procedimientos necesarios para vaciar los valores requeridos por la plantilla:

Algorithm 16 Procesamiento de una plantilla

```
$template->process('miplantilla.tml', {
                      a => (1, 2, 3, 4),
                      b => $otro_objeto,
                      c => &procedimiento,
                     })
      || die $template->error;
```

Siguiente los valores de configuración de defecto, un ejemplo de plantilla sería el siguiente:

Algorithm 17 Ejemplo de una plantilla

```
[% INCLUDE body.html %]
[% INCLUDE date.html %]
<script language="javascript">
<!--
d = new Date();
document.dateform.dia.selectedIndex = [% p.dia %] - 1;
document.dateform.mes.selectedIndex = [% p.mes %] - 1;
document.dateform.anio.selectedIndex = [% p.anio %] - 2000;

document.dateform.dia1.selectedIndex = [% p.dia1 %] - 1;
document.dateform.mes1.selectedIndex = [% p.mes1 %] - 1;
document.dateform.anio1.selectedIndex = [% p.anio1 %] - 2000;
//-->
</script>
[% i = 0 %]
<table border="0" width="100%" cellspacing="0" cellpadding="5">
<tr bgcolor="#96b284">
<th> </th><th>Inicio</th><th>Fin</th><th>IP</th><th>Duraci&oacute;n</th>
<th>MBytes enviados</th><th>MBytes recibido</th>
<th>Causa de desconexi&oacute;n</th></tr>
[% WHILE (tuple = sth.fetch()) %]
[% IF (i % 2 == 0) %]
[% color = '#f6ffdb' %]
[% ELSE %]
[% color = '#ffffff' %]
[% END %]
[% i = i + 1 %]
<tr bgcolor="[% color %]">
<td>[% i + p.skip * limit %]</span></td>
<td align="center"><span class="mini">[% tuple.0 %]</span></td>
<td align="center"><span class="mini">[% tuple.1 %]</span></td>
<td align="center"><span class="mini">[% tuple.2 %]</span></td>
<td align="center"><span class="mini">[% tuple.3 %]</span></td>
<td align="center"><span class="mini">[% tuple.4 %]</span></td>
<td align="center"><span class="mini">[% tuple.5 %]</span></td>
<td align="center"><span class="mini">[% tuple.6 %]</span></td>
[% END %]
</tr>
</table>
<table border="0" width="100%">
<tr><td align="left">
[% IF ((p.skip - 1) > -1) %]
<a class="li" href="[% config.serverurl %]/usage?
dia=[% p.dia %]&mes=[% p.mes %]&anio=[% p.anio %]&
dia1=[% p.dia1 %]&mes1=[% p.mes1 %]&anio1=[% p.anio1 %]&
skip=[% p.skip - 1 %]&report=detailed">&lt;&lt; P&aacute;gina Anterior</a>
[% END %]
</td>
<td align="center">[% total %] Conexiones</td>
<td align="right">
[% IF (((p.skip + 1) * limit) < total) %]
<a class="li" href="[% config.serverurl %]/usage?
dia=[% p.dia %]&mes=[% p.mes %]&anio=[% p.anio %]&
dia1=[% p.dia1 %]&mes1=[% p.mes1 %]&anio1=[% p.anio1 %]&
skip=[% p.skip + 1 %]&report=detailed">Siguiente P&aacute;gina &gt;&gt;</a>
[% END %]
</td>
</tr>
</table>
[% INCLUDE footer.html %]
```

4.3.4 Autenticación de usuarios

La autenticación de usuarios es simplemente un procedimiento que recibe las credenciales del usuario y devuelve si son válidos o lo contrario. Las credenciales más simples son un nombre de usuario y una contraseña. La comprobación de la validez de las contraseñas normalmente se basa en la comparación de éstas con algún almacenador de confianza de estos datos. La interacción con los almacenadores de credenciales puede ser de muchos tipos: LDAP, base de datos, NIS, Radius, Kerberos, etc. Así que debemos codificar la autenticación dependiendo de la interacción con el almacenador de credenciales. Aquí tenemos dos ejemplos sencillos:

Algorithm 18 Autenticación utilizando Radius

```
use Authen::Radius;

sub authorizate {
my $self = shift;
my ($username, $password) = @_;

my $r = new Authen::Radius(Host   => $self->CONFIG->{radiusserver},
   Secret => $self->CONFIG->{radiussecret})
|| $self->fatal_error(Authen::Radius::strerror);

Authen::Radius->load_dictionary;

$r->add_attributes ({ Name =>'1', Value =>$username },
{ Name =>'2', Value =>$password },
{ Name =>'4', Value =>$self->CONFIG->{radiusserver} },
{ Name =>'6', Value =>'2' },
{ Name =>'7', Value =>'1' });

$r->send_packet(1);
my $s = $r->recv_packet;

return 0 if ($s ne ACCESS_ACCEPT);
return 1;
}
```

Algorithm 19 Autenticaicón utilizando una tabla de base de datos

```
sub authorizate {
my $self = shift;
my ($username, $password) = @_;

my $sql = "SELECT 1 FROM radcheck
               WHERE UserName = '$username' AND Attribute = 'Password'
                     AND Value = '$password'";
my $res = $self->DBH->selectall_arrayref($sql)->[0][0] ;
$self->LOG->warning("$username/$password = $res");
return ($res) ? 1 : 0;
}
```

4.3.5 Manejo de sesiones

Esta es una de las secciones de infraestructura de una aplicación Web más compleja e importante. Por lo tanto es importante que sea lo más simple y clara posible. Para lograr esto se ha puesto un diagrama con el flujo de las tareas a realizar en punto (Figura 4.4).

get_session Este método extrae la *cookie* emitida por el agente del usuario, y una vez obteniendo el identificador de la sesión, la información persistente de la sesión es extraida de una tabla de la base de datos a través del módulo Perl Apache::Session. En caso de algún error se devuelve un número negativo correspondiente al error.

multipe_sessions Este método recibe como parámetro el nombre del usuario que solicita una sesión válida. Posteriormente extrae todas las sesiones existentes en la tabla y compara cada una de ellas con el identificador recibido. Si alguna pertence al mismo usuario que solicita la sesión se devuelve el número de sesión encontrado. Si no se devuelve vacío.

Algorithm 20 get_session

```
sub get_session {
        my $self = shift;
        my $cookie = $self->fetch_cookie;
        my %session;
        return -1 unless $cookie;
        return -2 unless $cookie->{session_id}->[0];
        unless(eval{
         tie %session,'Apache::Session::MySQL',$cookie->{session_id}->[0],
          { Handle => $self->DBH, LockHandle => $self->DBH };} )
        { return -3; }
        $self->{session} = \%session;
        return 0;
     }
```

Algorithm 21 multiple_sessions

```
sub multiple_sessions {
       my $self = shift;
       my $username = shift;
       my %session;
       my $sth = $self->DBH->prepare("select id from sessions");
       $sth->execute;
       while (my $sessid = $sth->fetch) {
         tie %session, 'Apache::Session::MySQL', $sessid->[0],
          { Handle => $self->DBH, LockHandle => $self->DBH };
         if ($session{username} eq $username) {
           $sth->finish; return $sessid->[0];
         }
       }
       return undef;
     }
```

destroy_session Recibe como parámetro un identificador de sesión y lo elimina de la tabla de sesiones de la base de datos.

Algorithm 22 destroy_session

```
sub destroy_session {
       my $self = shift;
       my $sessid = shift;
       my %session;
       eval {
         tie %session, 'Apache::Session::MySQL', $sessid,
           { Handle => $self->DBH, LockHandle => $self->DBH };
       };
       eval { tied(%session)->delete; } if (!@);
     }
```

create_session Recibe como parámetros el nombre de usuario que solicita la sesión. La crea mediante la interfaz del Apache::Session y le agrega en la información persistente tanto como el nombre de usuario como el número IP del usuario solicitante y lo guarda en la estructura de la sesión. Finalmente esta estructura se guarda en la estructura que define la aplicación y devuelve el identificador de la sesión.

Algorithm 23 create_session

```
sub create_session {
       my $self = shift;
       my $username = shift;
       my %session;
       unless ( eval {
         tie %session, 'Apache::Session::MySQL', undef,
           { Handle => $self->DBH, LockHandle => $self->DBH }; } )
       {  $self->fatal_error('Can\'t create session in server'); }
       $session{username} = $username;
       $session{ipaddress} = $self->APACHE->connection->remote_ip;
       $self->{session} = \%session;
       return $session{_session_id};
     }
```

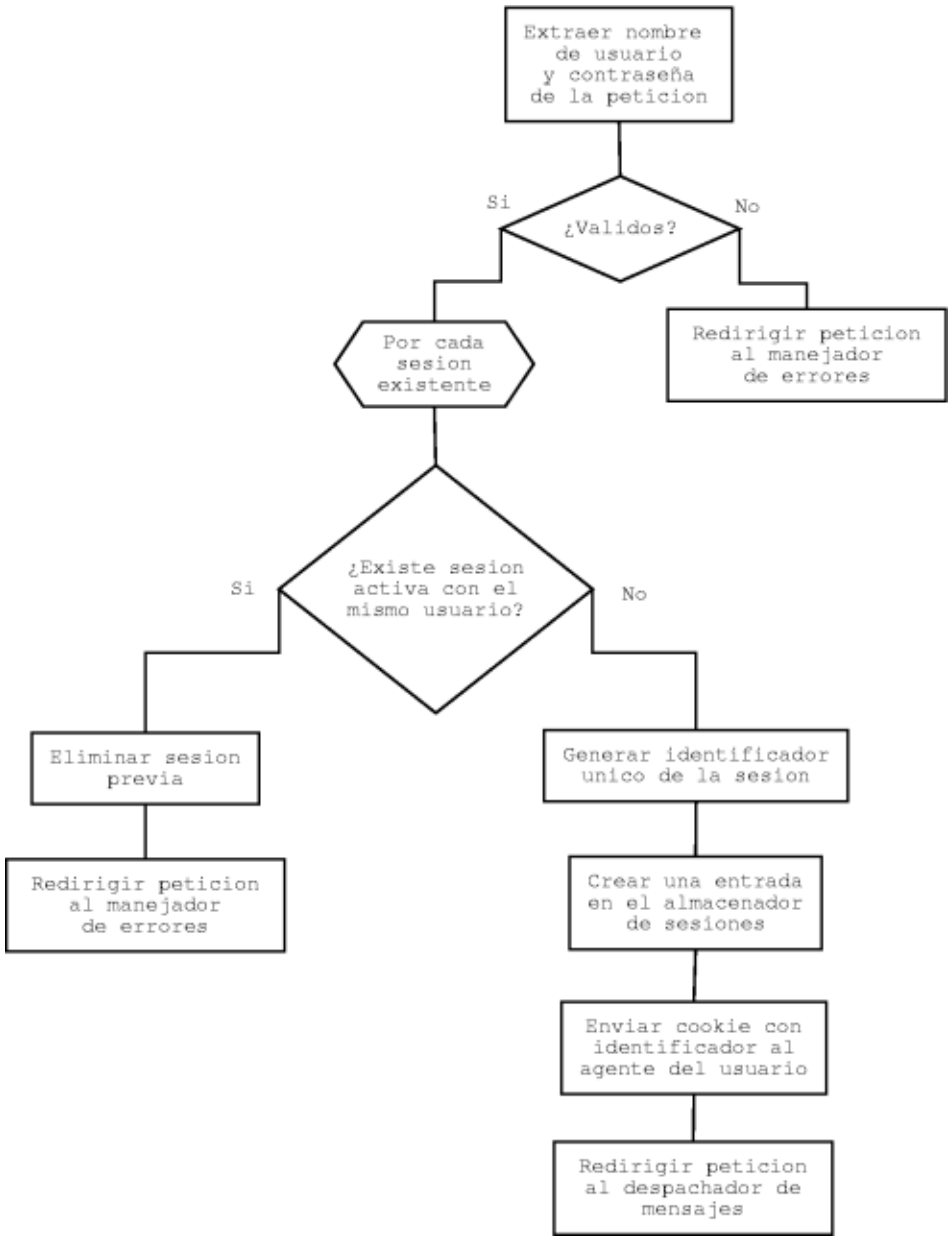

Figura 4.4: Diagrama de flujo del proceso de autenticación y creación de sesión

4.3.6 Despachador de mensajes

A continuación se muestra un diagrama de flujo con los aspectos generales que debe realizar un despachador de mensajes (Figura 4.5).

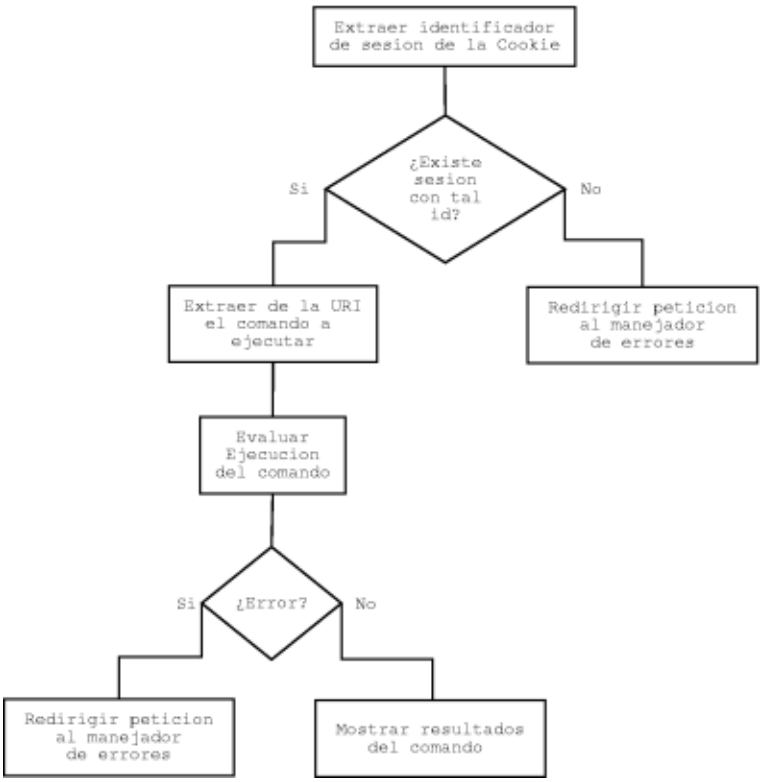

Figura 4.5: Diagrama de flujo del despachador de mensajes

Un ejemplo de despachador de mensajes como módulo en Perl (24).

4.4 Aplicación de ejemplo.

4.4.1 Análisis de requerimientos.

Panorama de las necesidades del negocio

El sistema tiene la finalidad de mejorar la calidad del servicio que se presta a los clientes de Coral Tecnología y Sistemas S.A. de C.V.

Algorithm 24 Despachador de mensajes

```
sub handler {
   my $S = corus->new;
   $S->initialize;

   my $path = $S->APACHE->path_info;
   my ($null, $cmd, @args) = split('/', $path);
   $S->LOG->debug("$path: $cmd");
   $cmd = "cmd_$cmd";

   if ($S->get_session) {
      if ($cmd eq 'cmd_login') {
         my %p = $S->APACHE->content;
         if ($S->authorizate($p{username}, $p{password})) {
            while (my $sessid = $S->multiple_sessions($p{username})) {
               $S->destroy_session($sessid);
            }
           $S->LOG->debug("user $p{username}");
           $cmd = 'cmd_init';
           push @args, $p{username};
         } else {
           $S->print_template('login.html');
           $S->end;
           return OK;
         }
      } elsif ($cmd =~ /cmd_\w+/) {
         $S->print_error('Tu navegador no soporta cookies');
         $S->end;
         return OK;
      } else {
         $S->print_template('login.html');
         $S->end;
         return OK;
      }
   } else {
      $S->signout if ($cmd eq 'cmd_'); # a forced login
      if ($S->SESSION->{ipaddress} ne $S->APACHE->connection->remote_ip) {
         $S->print_error('You\'re trying to forge a session');
         $S->end;
         return OK;
      }
   }

   eval { $S->$cmd(@args); };
   if ($@ =~ /^Can't locate object method/) {
      $S->print_error('Method not implemented');
   } elsif ($@) {
      $S->print_error('RunTime Error');
   }

   $S->end;
   return OK;
}
```

Se elaborará un aplicación Web que permita a los clientes:

- Ver sus datos de facturación para corroborar su validez
- Ver el plan de conexión que poseen
- Consultar su estado de cuenta actual
- Listar su histórico de pagos
- Consultar su tiempo de conexión dentro de un periodo determinado
- Consultar el ancho de banda consumido dentro de un periodo determinado
- Listar cada conexión realizada dentro de un periodo determinado
- Listar los rechazos de desconexión dentro del mes en curso debido a error en la contraseña o solicitud de conexión múltiple.

El producto

Acceso a Internet a través de dial-up

El cliente

La persona física o moral que tiene una cuenta de usuario a Coral Tecnología y Sistemas S.A. de C.V., y paga periódicamente un monto por su cuenta.

Entidades de datos

El sistema debe desplegar ciertas entidades, cada una de las cuales tiene diferentes atributos.

- El usuario
  - Nombre de usuario
  - Razón Social
  - Domicilio Fiscal
  - Colonia Fiscal
  - CP Fiscal
  - Ciudad Fiscal
  - Estado Fiscal
  - RFC
  - Plan de conexión
  - Frecuencia de pago
  - Estado de la cuenta (habilitado/no habilitado - facturado/no facturado)
  - Fecha de creación
  - Siguiente fecha de pago
- Registro de accesos
  - Tiempo de inicio de la conexión
  - Tiempo de finalización de la conexión
  - Número IP asignado
  - Tiempo total de sesión

  - Octetos de entrada
  - Octetos de salida
  - Causa de la desconexión

- Registro de pagos y cobros
  - Monto
  - Tipo (cargo/abono)
  - Fecha
  - Descripción

- Errores de conexión
  - Fecha
  - Razón del rechazo

Requerimientos para la captura de datos

El sistema debe proveer diversos menús y formularios para que la aplicación obtenga datos del usuario.

- Menú principal
  - Datos de facturación del cliente (página de inicio por defecto)
  - Estado de cuenta

  - Estadísticas de acceso
  - Errores de conexión
  - Salir de la aplicación

- Estadísticas de conexión
  - Fecha inicial del reporte
  - Fecha final del reporte
  - Reporte resumido / Reporte detallado

4.4.2 Diseño de la interfaz

Para el sistema se crearon diferentes pantallas que dan la funcionalidad deseada.

En la figura 4.6 se muestra la pantalla con los datos de un usuario de Coral.

4.4.3 Esquema general de la aplicación

En este capitulo se muestra una aplicación completa desarrollada bajo los lineamientos descritos en éste documento. Esta aplicación tiene como fin mostrar el estado de cuenta de los clientes de un PSI. La estructura de directorios de la aplicación se muestra en la figura 4.7:

Directorio de la aplicación: Es el directorio bajo el cual residirán los módulos de la aplicación.

Figura 4.6: Pantalla de datos del usuario

Directorio corus: Contiene los módulos de autenticación, despachador de mensajes, manejo de errores, etc.

Directorio template: Contiene las plantillas html que definen la presentación de las pantallas.

apache.pm: Éste módulo contiene el despachador de peticiones.

app.pm: Desde éste módulo se realizan las funciones propias de la aplicación, tales como imprimir los estados de cuenta, estadísticas de conexión de los clientes del ISP, etc.

auth.pm: Encargado de las funciones de autenticación de usuarios.

error.pm: Es un pequeño manejador de errores.

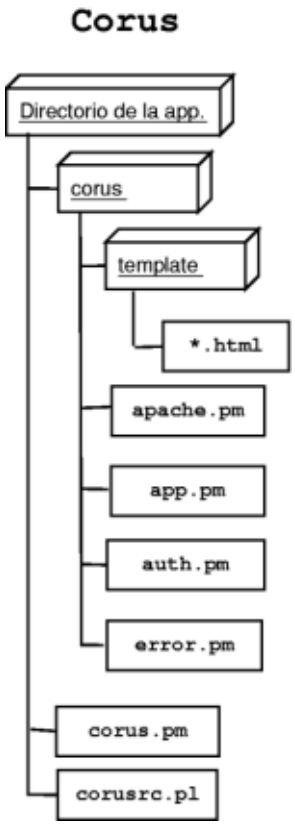

Figura 4.7: Estructura de directorios de la aplicación corus

corus.pm: Módulo principal que tiene las rutinas de manejo de sesión, lectura y escritura de *cookies*, entre otras.

Los módulos de la aplicación pueden residir en diferentes ubicaciones en el árbol de directorios del sistema; Una buena opción es ubicarla bajo el directorio lib/perl dentro del directorio raíz del servidor HTTP[4].. Por ejemplo, si el directorio raíz de l servidor HTTP es /opt/httpd, los módulos deberán ubicarse en la carpeta:

[4]Cuidado: no confundir el directorio raíz del servidor HTTP con el directorio raíz de documentos.

```
/opt/httpd/lib/perl
```

4.4.3.1 Configuración del servidor HTTP

Una vez que se tienen los módulos de la aplicación en el directorio mencionado, es necesario configurar el servidor HTTP para que pueda servir las páginas de la aplicación. Para ésto hay que modificar el archivo httpd.conf que reside en el subdirectorio conf/ y agregar las siguientes líneas:

```
Perlmodule corus
<Location /corus/>
    SetHandler perl-script
    PerlHandler corus::apache
</Location>
```

De esta manera, le indicamos al servidor HTTP que cualquier petición a corus/ sea manejada por el módulo apache, el cual es el despachador de peticiones.

4.4.4 El código fuente

4.4.4.1 Módulo de configuración

```
#/usr/bin/perl

my $my_config = {
dbname         => 'mydb',
dbuser         => 'admin',
dbpass         => 'naranjas',
```

```
dbhost         => 'localhost',
logfile        => '/var/log/httpd/corus.log',
loglevel       => 'info',
templatepath   => '/opt/httpd/lib/perl/corus/template',
domain         => '.pepito.com.mx',
path           => '/corus',
session_expire => 3600,
radiusserver   => 'mi.num.ip',
radiussecret   => 'naranjas'
};

$corus::conf = $my_config;
```

4.4.4.2 Módulo de control de la aplicación Web

```
package corus;

use strict;

use Apache::Constants qw(:common REDIRECT);
use DBI;
use Log::Dispatch;
use Log::Dispatch::File;
use Template;
use Time::CTime;
use Socket;

use corus::error;
use corus::auth;
use corus::app;

require "corusrc.pl";

sub new {
my $self = {};
bless($self);
return $self;
}

sub initialize {
my $self = shift;

$self->_set_apache_request;
$self->_set_config;
```

```
$self->_set_dbh;
$self->_set_logger;
$self->_set_template;
# $self->_set_command;

$self->LOG->debug("A CORUS object has been created");
}

sub _set_apache_request {
my $self = shift;
my $r = Apache->request;
$self->{apache} = $r;
return $self;
}

sub _set_config {
my $self = shift;
my $dir_config = $self->APACHE->dir_config();
foreach (keys %$dir_config) {
$corus::conf->{$_} = $dir_config->{$_};
}
$self->{config} = $corus::conf;
$self->{config}->{serverurl} = $self->server_url . "/corus";
return $self;
}

sub _set_dbh {
my $self = shift;

my $dbname   = $self->CONFIG->{dbname};
my $username = $self->CONFIG->{dbuser};
my $password = $self->CONFIG->{dbpass};
my $dbhost   = $self->CONFIG->{dbhost};

# Just for Apache::DBI
# my $data_source = "DBI:mysql:database=$dbname:host=$dbhost";
# my $dbh = DBI->connect($data_source, $username, $password) ||
# $self->fatal_error("Can't connect to the database: $@");

my $dbh = DBI->connect("dbi:mysql:radius:localhost", "root", "mYs91",
   { RaiseError => 1, AutoCommit => 1 });

$self->{dbh} = $dbh;
return $self;
}

sub _set_logger {
```

```
my $self = shift;

my $filename  = $self->CONFIG->{logfile};
my $min_level = $self->CONFIG->{loglevel};

my $set_timestamp = sub {
my %param = @_;
my $time = localtime;
return "[$time] $param{level}: $param{message}\n";
};

my $dispatch = Log::Dispatch->new(callbacks => $set_timestamp);
$dispatch->add(Log::Dispatch::File->new(name      => 'coruslog',
min_level => $min_level,
filename  => $filename,
mode      => 'append'));
$self->{logger} = $dispatch;
return $self;
}

sub _set_template {
my $self = shift;

my $include_path = $self->CONFIG->{templatepath};
my $default = 'nofound.html';
my $eval_perl = 1;
my $output = $self->APACHE;

my $template = Template->new({
INCLUDE_PATH => $include_path,
DEFAULT      => $default,
EVAL_PERL    => $eval_perl,
OUTPUT       => $output
});

$self->{template} = $template;
return $self;
}

sub _set_command {
my $self = shift;

my $path = $self->APACHE->path_info;
my ($null, $cmd, @args) = split('/', $path);
$self->LOG->debug("$path: $cmd");
$self->CONFIG->{command} = { cmd => "cmd_$cmd", args => \@args };
return $self;
```

```
}

sub end {
my $self = shift;

$self->DBH->disconnect;
$self->LOG->debug('A CORUS object has been finished correctly');
}

sub DBH { return $_[0]->{dbh}; }
sub LOG { return $_[0]->{logger}; }
sub CONFIG { return $_[0]->{config}; }
sub APACHE { return $_[0]->{apache}; }
sub TEMPLATE { return $_[0]->{template}; }
sub SESSION { return $_[0]->{session}; }

sub fetch_cookie {
my $self = shift;

my $cookie = $self->APACHE->header_in('Cookie');
$self->LOG->debug("Found this cookie = $cookie");
return () if (!$cookie);
return $self->parse_cookie($cookie);
}

sub set_cookie {
my $self = shift;
my $unset = shift;
my $session = $unset ? '' : $self->SESSION->{_session_id}; # kill session?

my $offset = $self->CONFIG->{session_expire};
my $GMT = 21600; # 6 horas de diferencia para el GMT
my $now = time + $GMT;
my $exp = $unset ? $now : $now + $offset;
my $expires = &Time::CTime::strftime('%a, %d-%b-%Y %X GMT',
 localtime($exp));
my $domain = $self->CONFIG->{domain};
my $path = $self->CONFIG->{path};

my $cookie = "session_id=$session; domain=$domain; path=$path; expires=$expires";
$self->LOG->debug("Setting COOKIE=$cookie");
$self->APACHE->header_out('Set-Cookie' => "$cookie");
}

sub parse_cookie {
my $self = shift;
my $cookie = shift;
```

```
my %results;

$self->LOG->debug("----> $cookie");
my (@pairs) = split(';', $cookie);
foreach (@pairs) {
s/\s*(.*?)\s*/$1/;
my ($key, $value) = split('=');
$self->LOG->debug("Cookie pair: $key/$value");
my (@values) = split('&', $value);
$results{$key} = \@values;
}
return \%results;
}

sub get_session {
my $self = shift;

my $cookie = $self->fetch_cookie;
$self->LOG->debug("Session_id in cookie: $cookie->{session_id}->[0]");
return -1 unless $cookie; # Cookie doesn't exist
return -2 unless $cookie->{session_id}->[0]; # Session Cookie doesn't exist

my %session;
unless (eval {
tie %session, 'Apache::Session::MySQL', $cookie->{session_id}->[0], {
Handle     => $self->DBH,
LockHandle => $self->DBH
};
}) {
$self->LOG->debug('forged or expired session id sended');
return -3;
}

$self->{session} = \%session;
return 0;
}

sub multiple_sessions {
my $self = shift;
my $username = shift;
my %session;

my $sth = $self->DBH->prepare("select id from sessions");
$sth->execute;
while (my $sessid = $sth->fetch) {
tie %session, 'Apache::Session::MySQL', $sessid->[0], {
```

```
Handle     => $self->DBH,
LockHandle => $self->DBH
};
if ($session{username} eq $username) {
$self->LOG->info('Found a multiple session');
$sth->finish;
return $sessid->[0];
}
}
return undef;
}

sub destroy_session {
my $self = shift;
my $sessid = shift;
my %session;

eval {
tie %session, 'Apache::Session::MySQL', $sessid, {
Handle     => $self->DBH,
LockHandle => $self->DBH
};
};
if (!$@) {
$self->LOG->info("Killing session $sessid");
eval { tied(%session)->delete; };
}
}

sub create_session {
my $self = shift;
my $username = shift;
my %session;

$self->LOG->debug('Creating the session in server');
unless (
eval {
tie %session, 'Apache::Session::MySQL', undef, {
Handle     => $self->DBH,
LockHandle => $self->DBH
};
}) { $self->fatal_error('Can\'t create session in server'); }

# my %p = $self->APACHE->content;
$self->LOG->debug("username in session : $username");
$session{username} = $username;
$session{ipaddress} = $self->APACHE->connection->remote_ip;
```

```
$self->{session} = \%session;
return $session{_session_id};
}

sub server_url {
my $self = shift;
my $hostname = shift;

my ($port) = sockaddr_in($self->APACHE->connection->local_addr);
my $scheme = 'http';
if ($port == 443) {
$scheme = 'https';
$port = '';
} elsif ($port == 80) {
$port = '';
} else {
$port = ":$port";
}
my $hostname ||= $self->APACHE->server->server_hostname;
return "$scheme://$hostname$port";
}

sub dont_cache {
my $self = shift;

$self->APACHE->no_cache(1);
$self->APACHE->err_header_out(Pragma => 'no-cache');
$self->APACHE->err_header_out('Cache-Control' => 'no-cache');
}

sub set_content_type {
my $self = shift;
my $type = shift || 'text/html';

$self->APACHE->content_type($type);
$self->APACHE->send_http_header;
}

sub redirect {
my $self = shift;
my $url = shift;

$self->APACHE->header_out(Location => $url);
$self->APACHE->status(REDIRECT);
$self->APACHE->send_http_header;
return OK;
}
```

```
sub print_template {
my $self = shift;
my $template = shift;
my $var = shift || {};
my $dont_cache = shift || 1; # don’t cache by default

$var->{config} = $self->CONFIG;

if ($self->SESSION) {
$var->{username} = $self->SESSION->{username};
$var->{user} = $self->DBH->selectall_arrayref("SELECT razonsocial FROM usuario WHERE usuario = ’$var->{username}’")->[0][0];
}



$self->dont_cache if ($dont_cache);
$self->set_content_type;
$self->TEMPLATE->process($template, $var)
|| $self->fatal_error($self->TEMPLATE->error);
}

sub print_error {
my $self = shift;
my $message = shift;

my $error = $@;
$self->LOG->error("$message\n$error");
$error =~ s/ at.*$//;
$self->print_template(’error.html’,
  { message => $message, error => $error });
}

sub make_request {
my $self = shift;
my $method = shift;

my $serverurl = $self->CONFIG->{serverurl};
return $self->redirect("$serverurl/$method");
}

sub signout {
my $self = shift;

$self->set_cookie(1);
$self->destroy_session($self->SESSION->{_session_id});
$self->make_request;
}
```

```
1;
```

4.4.4.3 Despachador de peticiones

```
package corus::apache;

use strict;

use Apache::Constants;

sub handler {
my $S = corus->new;
$S->initialize;

my $path = $S->APACHE->path_info;
my ($null, $cmd, @args) = split(’/’, $path);
$S->LOG->debug("$path: $cmd");
$cmd = "cmd_$cmd";

if ($S->get_session) {
if ($cmd eq ’cmd_login’) {
my %p = $S->APACHE->content;
if ($S->authorize($p{username}, $p{password})) {
while (my $sessid = $S->multiple_sessions($p{username})) {
$S->destroy_session($sessid);
}
$S->LOG->debug("user $p{username}");
$cmd = ’cmd_init’;
push @args, $p{username};
} else {
$S->print_template(’login.html’,
   { message => ’Nombre de usuario o contrase&ntilde;a no v&aacute;lida’ });
$S->end;
return OK;
}
} elsif ($cmd =~ /cmd_\w+/) {
$S->print_error(’Tu navegador no soporta o tiene deshabilitado el manejo de <em>cookies</em>.<br>O tu tiempo de sesi&oacute;n ha terminado’)
$S->end;
return OK;
} else {
$S->print_template(’login.html’);
$S->end;
return OK;
```

```
}
} else {
$S->signout if ($cmd eq 'cmd_'); # a forced login
$S->LOG->debug('Check out the ip connection');
if ($S->SESSION->{ipaddress} ne $S->APACHE->connection->remote_ip) {
$S->print_error('You\'re trying to forge a session');
$S->end;
return OK;
}
}

eval { $S->$cmd(@args); };
if ($@ =~ /^Can't locate object method/) {
$S->print_error('Method not implemented');
} elsif ($@) {
$S->print_error('RunTime Error');
}
$S->end;
return OK;
}

1;
```

4.4.4.4 Métodos de la aplicación

```
package corus;

use strict;

# CONST
my $LIMITSIZ = 30;
my $RADIUSLOG = "/var/log/radius.log";
my @VALIDPOPS = ('sji', 'apaseo', 'celaya', 'cortazar');

sub cmd_init {
my $self = shift;
my $username = shift;

$self->create_session($username);
```

```
$self->set_cookie;
return $self->make_request('main');
}

sub cmd_main {
my $self = shift;

my $user = $self->SESSION->{username};
my $sql = "SELECT razonsocial, domicilio_fiscal, colonia_fiscal,
                     cp_fiscal, ciudad_fiscal, estado_fiscal, rfc,
                     plan, frecuencia, habilitado, facturar,
                     DATE_FORMAT(fecha_creacion, \"%d-%m-%Y\"),
                     DATE_FORMAT(fecha_facturacion, \"%d-%m-%Y\")
              FROM usuario WHERE usuario='$user'";

my $tuple = $self->DBH->selectall_arrayref($sql)->[0]
|| $self->fatal_error($self->DBH->errstr);

$tuple->[7] = 'Ilimitado' if ($tuple->[7] eq 'ilimi');
$tuple->[7] = 'Diurno'    if ($tuple->[7] eq 'diurn');
$tuple->[7] = 'Nocturno'  if ($tuple->[7] eq 'noctu');
$tuple->[7] = '40 Horas'  if ($tuple->[7] eq '40hor');
$tuple->[7] = '20 Horas'  if ($tuple->[7] eq '20hor');

$tuple->[8] = 'Anual'      if ($tuple->[8] eq 'a');
$tuple->[8] = 'Semestral'  if ($tuple->[8] eq 's');
$tuple->[8] = 'Trimestral' if ($tuple->[8] eq 't');
$tuple->[8] = 'Mensual'    if ($tuple->[8] eq 'm');

$tuple->[12] = ($tuple->[10] == 1) ? $tuple->[12] : '-----';

$tuple->[9] = ($tuple->[9] == 1) ? 'Habilitado' : 'Deshabilitado';
$tuple->[10] = ($tuple->[10] == 1) ? 'Facturado' : 'No Facturado';

$self->print_template('main.html', { tuple => $tuple });
}

sub cmd_signout {
my $self = shift;

$self->signout;
}

sub cmd_balance {
my $self = shift;

my $pop = $self->get_user_pop;
```

```
foreach (@VALIDPOPS) {
if ($_ eq $pop) {
return $self->print_balance;
}
}
return $self->is_not_valid_pop;
}

sub cmd_usage {
my $self = shift;

my %p = $self->APACHE->args;

if (!%p) {
$self->print_template('selectdate.html');
} else {
if ($p{report} eq 'detailed') {
$self->print_conection_stats(%p);
} else {
$self->print_conection_resume(%p);
}
}
}

sub print_conection_stats {
my ($self, %p) = @_;

my $from = "$p{anio}-$p{mes}-$p{dia}";
my $to = "$p{anio1}-$p{mes1}-$p{dia1}";
my $user = $self->SESSION->{username};
my $limit = $p{skip} * $LIMITSIZ;

my $sql = "SELECT COUNT(*) FROM radacct
               WHERE AcctStartTime >= '$from' AND AcctStopTime <= '$to' AND
                     AcctStopTime != 0 AND UserName = '$user'";
my $total = $self->DBH->selectall_arrayref($sql)->[0][0]
|| $self->die($self->DBH->errstr);


$sql = "SELECT DATE_FORMAT(AcctStartTime, \"%d-%m-%Y %r\"),
                  DATE_FORMAT(AcctStopTime, \"%d-%m-%Y %r\"),
                  FramedIPAddress,
                  SEC_TO_TIME(AcctSessionTime),
                  AcctInputOctets / (1024 * 1024),
                  AcctOutputOctets / (1024 * 1024),
                  AcctTerminateCause
```

```
            FROM radacct
            WHERE AcctStartTime >= '$from' AND AcctStopTime <= '$to' AND
                  AcctStopTime != 0 AND UserName = '$user'
            LIMIT $limit, $LIMITSIZ";

my $sth = $self->DBH->prepare($sql) || $self->die($self->DBH->errstr);
    $sth->execute || $self->die($self->DBH->errstr);

    $self->print_template('conection.html', { sth => $sth,
  p => \%p,
      limit => $LIMITSIZ,
      total => $total });
}

sub print_conection_resume {
my ($self, %p) = @_;

my $from = "$p{anio}-$p{mes}-$p{dia}";
my $to = "$p{anio1}-$p{mes1}-$p{dia1}";
my $user = $self->SESSION->{username};

my $sql = "SELECT COUNT(*),
                     SEC_TO_TIME(SUM(AcctSessionTime)),
                     SUM(AcctInputOctets) / (1024 * 1024),
                     SUM(AcctOutputOctets) / (1024 * 1024)
               FROM radacct
               WHERE AcctStartTime >= '$from' AND AcctStopTime <= '$to' AND
                     AcctStopTime != 0 AND UserName = '$user'";

my $tuple = $self->DBH->selectall_arrayref($sql)->[0]
|| $self->die($self->DBH->errstr);

$self->print_template('conresume.html', { tuple => $tuple,
  p => \%p });
}

sub cmd_errorlist {
my $self = shift;
my @lines;

my $username = $self->SESSION->{username};
open FH, "<$RADIUSLOG";
my $re = "\\[\\s*$username\\s*(\\/.*)?\\s*\\]";
while (<FH>) {
if ($_ =~ /$re/) {
next if ($_ =~ /Login OK/);
s/Password should be '.*'//;
```

```
push @lines, $_;
}
}
close FH;
$self->print_template("errorlist.html", { lines => \@lines });
}

sub print_balance {
my $self = shift;

my $user = $self->SESSION->{username};

my $sql = "SELECT SUM(importe) FROM pagos WHERE usuario = '$user' AND tipo = 'c'";
my $debit = $self->DBH->selectall_arrayref($sql)->[0][0];

$sql = "SELECT SUM(importe) FROM pagos WHERE usuario = '$user' AND tipo ='a'";
my $salary = $self->DBH->selectall_arrayref($sql)->[0][0];

my $balance = sprintf("%.2f", $debit - $salary);

$self->LOG->debug("Usuario: $user - Balance: $balance");

$sql = "SELECT pagos.fecha, precios.descripcion, pagos.importe, pagos.id
            FROM pagos, precios
            WHERE pagos.concepto = precios.id AND
                  pagos.usuario = '$user'
            ORDER BY pagos.fecha ASC, pagos.id ASC";

my $sth = $self->DBH->prepare($sql) || $self->die($self->DBH->errstr);

$sth->execute || $self->die($self->DBH->errstr);

$self->print_template('balance.html', { balance => $balance,
sth => $sth });
}

sub is_not_valid_pop {
my $self = shift;

$self->print_template('notvalidpop.html');
}

sub get_user_pop {
my $self = shift;

my $user = $self->SESSION->{username};
```

```
my $sql = "SELECT pop.nombre FROM pop, usuario
               WHERE usuario.usuario = ’$user’ AND
                     usuario.id_pop = pop.id";
my $resp =  $self->DBH->selectall_arrayref($sql)->[0][0]
|| $self->fatal_error($self->DBH->errstr);
return $resp;
}

1;
```

4.4.4.5 Módulo para el manejo de errores

```
package corus;

use strict;

sub fatal_error {
my $self = shift;

my $message = shift;

$self->LOG->emergency($message);
$self->end;
$self->die($message);
}

sub die {
my $self = shift;
my $message = shift;

die $message;
}

1;
```

4.5 Herramientas para las pruebas de desempeño

Uno de los factores más importantes en una aplicación Web es su desempeño. Debido a que la misma aplicación (parte de ella) es instanciada por cada solicitud HTTP, se pueden tener cientos de ellas en un momento dado, generando así una competencia por los recursos computacionales, que pueden derivar en una degradación general de los tiempos de respuesta, degradación que resulta muy molesta para los usuarios del sistema.

Hay que considerar el tiempo en tarda en llegar la petición del usuario, el procesamiento de la misma y el tiempo consumido para hacerle llegar la respuesta. Es por ello que se debe tipificar el ancho que banda a que tienen acceso el grueso de los usuarios, y en base a esto diseñar una interfaz que vaya acorde. Así mismo, hay que monitorear el tiempo consumido por cada una de las respuestas de petición; de ser posible, censarlas y así poder decidir sobre qué se debe optimizar y que no. Estos censos también servirán para medir la utilidad del hardware, podremos saber si estamos utilizando el correcto, si necesitamos más poder o estamos sobrados para el alcance que va a tener la aplicación.

Muy probablemente se deba programar herramientas de medición propias. Sin embargo existen varias disponibles para los desarrollos en Perl y mod_perl, tal como se puede apreciar en la siguiente lista[48]:

- ApacheBench

  Es una herramienta para medir al servidor Web Apache. Esta diseñado para

dar una idea del desempeño de la instalación actual. Muestra en particular cuántas peticiones por segundo el servidor es capaz de servir.

El ApacheBench (ab) viene con el código fuente del Apache.

- httperf

  Es otra herramienta que mide el despeño del servidor Web. Este es el URI para bajarlo httperf `http://www.hpl.hp.com/personal/DavidMosberger/httperf.html`.

- http_load

  Es otra utilería que prueba la carga de un servidor Web. Puede simular una conexión de 33.6kbps (como un módem) y permite un archivo con una lista de URLs que serán tomados aleatoriamente. Se puede especificar el número de conexiones paralelas a ejecutar, y el número de peticiones generadas por segundo. HttpLoad `http://www.acme.com/software/http_load/`

- crashme

  Es un conjunto de scripts que imitan al ApacheBench, pero además emite variables como la **latencia** (segundos por petición), además de simular completamente al Netscape. lwp-bench `http://perl.apache.org/guide/lwp-bench.pl`

- Apache::Timeit

  Es un módulo que mide el desempeño de módulos que actuarán como ma-

nejadores Perl (PerlHandler). Los resultados se envían al sistema del bitácoras del servidor Web. Apache::Timeit `http://perl.apache.org/dist/contrib/Timeit.pm`

Conclusiones y recomendaciones

El desarrollo de aplicaciones Web es un campo relativamente joven para el cual existen diversas propuestas. En el presente trabajo se examinaron los diferentes protocolos, estándares y lenguajes usados para el desarrollo de aplicaciones Web en general. Además se presentó un panorama acerca del estado actual del desarrollo de aplicaciones Web y la problemática que hay que resolver para la implementación de este tipo de aplicaciones. Finalmente se presentó una propuesta de desarrollo de aplicaciones usando Software Libre.

Como se expuso en este trabajo, el Software Libre rompe los viejos paradigmas a los que estábamos acostumbrados: los sistemas de información ya no dependen de una sola empresa de software que provee de todas las soluciones en una "caja mágica" de la cual sabemos qué es lo que hace pero no cómo lo hace y más importante aún, si fue desarrollado considerando las medidas de seguridad básicas.

Al usar las alternativas propuestas en el presente trabajo para el desarrollo de aplicaciones Web, se obtienen sistemas de información robustos de gran desempeño, estabilidad, escalabilidad y un costo menor al que se tendría al usar software pro-

pietario. Muestra de ello son las diferentes aplicaciones Web que los autores de este documento, en conjunto con otras personas, hemos desarrollado.

Bajo el paradigma de Software Libre hemos desarrollado diversas aplicaciones Web dentro de las que destacan:

- Sistema de Control de Estados de Cuenta y Acceso de los usuarios de Coral Tecnología y Sistemas S.A. de C.V. En producción desde Febrero de 2001.
- Sistema Para Captura de Calificaciones del Instituto Tecnológico de Celaya. En producción desde Agosto de 2000.
- Sistema Para Transferencia de Dinero Latin Trans S.A. de C.V. - Latin Trans Incorporated. En producción desde Junio de 2000.
- Coradm, Sistema administracion de clientes dial up, buzones de correo electrónico y manejo de estado de cuenta de Coral Tecnología y Sistemas S.A. de C.V. En producción desde Mayo de 2000.
- Sistema de outsorcing de correo electrónico SuCorreo de Coral Tecnología y Sistemas S.A. de C.V. En producción desde hace dos años.

Hemos transitado por un camino que apenas comienza a pavimentarse; la reglas finales aún no están escritas, y con la efervescencia de las nuevas tecnologías es probable que la balanza se incline para diferentes modelos o alternativas. Sin embargo, el desarrollo de aplicaciones Web es una tendencia incontenible, hacia

allá caminan las organizaciones al implementar Intranets y Extranets; hacía allá avanzan millones de internautas al utilizar Internet como su lugar de trabajo.

A medida que los tomadores de decisiones observan la gama de ventajas que el uso del Software Libre trae consigo, su uso en los sectores empresarial y de gobierno de muchos países, crece día con día. Actualmente el servidor Web Apache es el más utilizado; el lenguaje Perl ha dominado la escena de la programación Web desde los inicios del mismo, aún antes del Java, del ASP, o cualquier otro a excepción del C. Las bases de datos como PostgreSQL y MySQL están ganando popularidad día con día en el ambiente de negocios. Recientemente la empresa de software SAP, liberó el manejador de base de datos con el cual trabaja todo su sistema, bajo la licencia GPL: el SAPDB SAPDB `http://www.sapdb.org`; muestra de la popularidad y confianza que se esta ganando el Software Libre. El software de cifrado de datos (OpenSSL), ha sido desarrollado bajo una constante auditoría en su código, lo cual le ha resultado en una reputación y confianza de millones de usuarios al rededor del mundo.

Sin embargo existen muchos retos. El buen funcionamiento de una red, depende únicamente de sus usuarios, de la política del buen vecino. Es por ello que como desarrolladores Web, debemos apegarnos a los estándares recomendados por las organizaciones representativas de las tecnologías involucradas (la W3 Consortium, la IETF, etc.) y no refugiarnos en tecnologías emergentes de naturaleza restrictiva y cerrada, encaminada solamente hacía la retención forzada de un nicho tecnológico por un pequeño grupo de intereses. Debemos defender nuestra libertad en el uso de la información, y defender el mismo derecho a nuestros usuarios.

Ignorar esto en aras de la comodidad y facilidad, acumularía fuerza necesaria para destruir la sinergia generada por el trabajo de los usuarios a través de la red. Otra tarea es la seguridad, debemos programar de manera defensiva, con únicamente un supuesto: siempre van a existir personas mal intencionadas. No debemos suponer que la aplicación será utilizada de una manera, debemos obligar que únicamente se pueda utilizar de manera correcta. Pero no únicamente debemos tener cuidado en nuestra programación, es importante de igual manera verificar la configuración de todo el software que utilizamos: el sistema operativo, el manejador de base de datos, el servidor Web, y todo lo que intervenga en el funcionamiento de la aplicación.

A continuación y para finalizar nuestro trabajo haremos un breve y consciso repaso de las recomendaciones que se dieron a lo largo del documento:

1. Acatarse lo más posible a los estándares respaldados por la W3C
2. Utilizar los más posible software libre en el desarrollo e implementación de nuestras aplicaciones.
3. Poner mayor esfuerzo en nuestro código de validación de datos de entrada.
4. Hacer que nuestro mecanismo de autenticación sea lo suficientemente flexible para poder cambiar nuestro fuente de datos independiente del protocolo que utilize
5. Utilizar un manejo de sesiones basado en *tickets*, así como su seguimiento en el servidor.

6. Llevar una bitácora del uso de la aplicación
7. Programar un despachador de mensajes basado en la URI.
8. La conexión a la base de datos debe ser persistente
9. Separar lo más posible la capa de presentación con la lógica del programa utilizado plantillas
10. Utilizar DHTML únicamente cuándo sea realmente necesario (v.gr. retroalimientación inmediata al usuario)
11. Evitar el sobrecargado de información en una página para evitar la confusión al usuario y el consumo innecesario de ancho de banda
12. Realizar prototipos de páginas estáticas para que sean evaluadas por el solicitante de la aplicación
13. Cifrar toda la información que sea transmitida entre el agente del usuario y el servidor
14. Desarrollar con la orientación a mensajes

Apéndice A

Glosário de términos.

Apache. Servidor web de libre distribución creado bajo una licencia Open Source.

Agente del usuario Programa de computadora o dispositivo electrónico que es usado para acesar a los servicios ofrecidos a través de Internet. Puede ser un navegador de internet, un telefono celular u otro dispositivo.

ANSI American National Standards Institute - Instituto Americano de Estándares Nacionales.

API Application Program Interface -La interfaz de programas de aplicación describe a detalle los métodos de un sistema operativo o aplicación que pueden ser usados por un programador y realizar llamadas a ellos.

ASP Aplication Service Provider - Proveedor de Servicios de Aplicación. Giro

comercial donde una empresa ofrece una aplicación a un conjunto de clientes y su uso es a través de Internet.

B2B Business to Businness - Empresa a Empresa. Tipo de aplicación utilizada entre empresas por medio de Internet.

B2C Business to Consumer - Empresa a Consumidor. Tipo de aplicación donde interactúan los clientes finales con la empresa a través de Internet.

CERN Organisation Européene pour la Recherche Nucléaire - Organización Europea para la Investigación Nuclear.

CA Certificate Authority - Autoridad certificadora, es una entidad en una red que emite y maneja credenciales de seguridad y una llave pública para cifrado de mensajes.

CGI Common Gateway Interface - Interfaz Común de Puerta de Enlace. Estándar la para transmisión de información entre el servidor y otras aplicaciones independientes del servidor.

Cookie Pedazo de información enviada por el servidor HTTP al agente del usuario para almacenar de manera presistente cualquier información.

CORBA Common Object Request Broker Architecture - Arquitectura de Corredor Común para la Solicitud de Objetos. Es una arquitectura y especificaicón para crear, distribuir y administrar objetos de programa distribuidos en una red.

CPAN Comprehensive Perl Archive Network - Red Global de Archivos Perl.

CSS Cascade Style Sheet - Hoja de Estilo en Cascada. Es un conjunto de especificaciones que definen o redefinen el compartamiento de una marca en un documento HTML.

DARPA Defense Advanced Research Projects Agency - Agencia de la Defensa para la Investigación de Proyectos Avanzados (EUA).

DBMS DataBase Management System - Sistema de Manejo de Bases de Datos, es un programa de computadora que permite a uno o mas usuarios crear y acceder datos en una base de datos.

DNS Domain Name Server - Servidor de Nombres de Domino. Protocolo tanto para la resolución de nombre de dominio como su viceversa.

DOM Document Object Model - Modelo de Documento Objeto.

Extranet Red privada de una empresa o institución que utiliza los protocolos de Internet y un sistema de telecomunicaciones público para compartir, de manera segura, parte de la información de negocios con clientes, proveedores, socios, u otros negocios.

GPL General Public Licence. Es la licencia que utiliza la Free Software Foundation para proteger el software distribuido por el proyecto GNU, así como otros programas que no forman parte del proyecto GNU, pero que sus desarrolladores han adoptado.

GUI Graphic User Interface - Interfaz Gráfica del Usuario.

Hipertexto Es la organización de unidades de información en asociaciones conectadas que un usuario puede escoger. Una instancia de esa asociación es llamada vínculo o hipervínculo.

HTTP HyperText Transfer Protocol - El Protocolo de Transferencia de Hipertexto es un conjunto de reglas para intercambiar archivos (texto, gráficos, imágenes, sonido, video y otros archivos multimedia) en el World Wide Web.

IETF Internet Engineering Task Force - Es la entidad encargada de definir el protocolo de operación estándar de Internet, el TCP/IP.

Internet Es una red de redes de alcance mundial que utiliza los protocolos TCP/IP, también se le conoce como “La Red”.

internet Es cualquier red de redes que esta limitada a cierto número de redes o a ciertas localidades geográficas.

Intranet Es una red privada que utiliza los protocolos de Internet pero reside en una empresa u organización.

ISO International Standards Organization - Organización Internacional para la Estandarización.

Kerberos Es un método seguro para autenticar una solicitud a un servicio en una red de computadoras.

LDAP Lightweight Directory Access Protocol - Protocolo ligero de acceso a directorios, es un protocolo que permite ubicar organizaciones, individuos y otros recursos tales como archivos y dispositivos en una red, ya sea Internet o una intranet corporativa.

Llave privada Véa private key.

Llave pública Vea public key .

MIME Multipurpose Internet Mail Extensions - Extensiones Multipropósito para el Correo de Internet. Extensión al protocolo original del correo electrónico para la transformación de cadenas ASCII-8 a ASCII-7.

Open Source Código abierto. Es una modalidad en el desarrollo de software en la cual los desarrolladores hacen disponible el código fuente del mismo a terceras personas.

ORB Object Request Broker - Corredor de Peticiones de Objetos. Componente de software que se encarga de transportar mensajes entre objetos que utilizan el estándar CORBA.

PAM Pluggable Authentication Modules - Módulos Enchufables de Autenticación.

Perl Practical Extraction and Report Language - lenguaje práctico de extracción y reportes.

POSIX Portable Operating System Interface - Interfaz portable de sistemas operativos es un conjunto de estándares de interfaces de sistemas operativos basados en el sistema operativo UNIX.

PostgreSQL Manejador de bases de datos Objeto-Relacionales.

Private Key Llave privada - Es la llave utilizada para descifrar los datos cifrados con la llave pública correspondiente.

Public Key Llave pública - Es un valor proveido por una autoridad certificadora que puede usarse para cifrar mensajes.

RFC Request for Comments - Solicitud de Comentarios, es un documento formal de la IETF que es el resultado de la definición de un anteproyecto acerca de un tópico determinado y está sujeto a revisión por entidades interesadas.

RSA Sistema de cifrado y autenticación para Internet que utiliza el algoritmo creado en 1977 por Ron Rivest, Adi Shamir y Leonard Adleman.

SGML Standard Generalized Markup Language - Lenguaje Estándar de Marcado Generalizado. Estándar para definir un lenguaje de marcado y su conjunto de marcas.

SQL Structured Query Language - Lenguaje Estructurado de Consultas. Lenguaje estándar interactivo para la consulta y actualización de bases de datos.

SSL Secure Socket Layer - Capa de Sockets Seguros. Protocolo posterior al TCP/IP que cifra la infromación transmitida.

SQL. Structured Query Language - El lenguaje estructurado de consulta es un lenguaje de programación que permite obtener y actualizar una base de datos.

TCP/IP Transmission Control Protocol / Internet Protocol - Protocolo para el Control de Transmisiones / Protocolo de Internet. Identifica a un conjunto de protocolos que operan en Internet.

UNIX. Sistema operativo desarrollado en los Laboratorios Bell en 1969 como un sitema interactivo de tiempo compartido.

URI Uniform Resource Identifier - Identificador Uniform de Recursos. Cadena que especifíca al recurso que se desea solicitar.

Vínculo Es una conexión entre una palabra, imágen u objeto de información a otro documento.

WWW Véa World Wide Web.

Web Véa World Wide Web.

World Wide Web La telaraña de cobertura mundial, también conocido como WWW o Web, es el conjunto de recursos y usuarios en la Internet que hacen uso del protocolo de transferencia de hipertéxto (HTTP). Según la W3C: “El World Wide Web es el universo de información accesible por red, es una representación del conocimiento humano”.

W3C Es un consorcio industrial que busca promover estándares para la evolución del Web y la interoperabilidad entre productos para el WWW al poducir especificaciones y software de referencia.

XML EXtensible Markup Language - Lenguaje de Marcado Extensible. Es lenguaje para definir el formato de la información a compartir a través de una red.

Apéndice B

Licencia GFDL

GNU Free Documentation License
   Version 1.1, March 2000

 Copyright (C) 2000 Free Software Foundation, Inc.
     59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

0. PREAMBLE

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the

software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document’s overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is

not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve

the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.
E. Add an appropriate copyright notice for your modifications
   adjacent to the other copyright notices.
F. Include, immediately after the copyright notices, a license notice
   giving the public permission to use the Modified Version under the
   terms of this License, in the form shown in the Addendum below.
G. Preserve in that license notice the full lists of Invariant Sections
   and required Cover Texts given in the Document’s license notice.
H. Include an unaltered copy of this License.
I. Preserve the section entitled "History", and its title, and add to
   it an item stating at least the title, year, new authors, and
   publisher of the Modified Version as given on the Title Page. If
   there is no section entitled "History" in the Document, create one
   stating the title, year, authors, and publisher of the Document as
   given on its Title Page, then add an item describing the Modified
   Version as stated in the previous sentence.
J. Preserve the network location, if any, given in the Document for
   public access to a Transparent copy of the Document, and likewise
   the network locations given in the Document for previous versions
   it was based on. These may be placed in the "History" section.
   You may omit a network location for a work that was published at
   least four years before the Document itself, or if the original
   publisher of the version it refers to gives permission.
K. In any section entitled "Acknowledgements" or "Dedications",
   preserve the section’s title, and preserve in the section all the
   substance and tone of each of the contributor acknowledgements
   and/or dedications given therein.
L. Preserve all the Invariant Sections of the Document,
   unaltered in their text and in their titles. Section numbers
   or the equivalent are not considered part of the section titles.
M. Delete any section entitled "Endorsements". Such a section
   may not be included in the Modified Version.
N. Do not retitle any existing section as "Endorsements"
   or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or
appendices that qualify as Secondary Sections and contain no material
copied from the Document, you may at your option designate some or all
of these sections as invariant. To do this, add their titles to the
list of Invariant Sections in the Modified Version’s license notice.
These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains
nothing but endorsements of your Modified Version by various
parties--for example, statements of peer review or that the text has
been approved by an organization as the authoritative definition of a
standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for

verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document’s Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this

License will not have their licenses terminated so long as such
parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions
of the GNU Free Documentation License from time to time. Such new
versions will be similar in spirit to the present version, but may
differ in detail to address new problems or concerns. See
http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number.
If the Document specifies that a particular numbered version of this
License "or any later version" applies to it, you have the option of
following the terms and conditions either of that specified version or
of any later version that has been published (not as a draft) by the
Free Software Foundation. If the Document does not specify a version
number of this License, you may choose any version ever published (not
as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of
the License in the document and put the following copyright and
license notices just after the title page:

      Copyright (c) YEAR YOUR NAME.
      Permission is granted to copy, distribute and/or modify this document
      under the terms of the GNU Free Documentation License, Version 1.1
      or any later version published by the Free Software Foundation;
      with the Invariant Sections being LIST THEIR TITLES, with the
      Front-Cover Texts being LIST, and with the Back-Cover Texts being LIST.
      A copy of the license is included in the section entitled "GNU
      Free Documentation License".

If you have no Invariant Sections, write "with no Invariant Sections"
instead of saying which ones are invariant. If you have no
Front-Cover Texts, write "no Front-Cover Texts" instead of
"Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we
recommend releasing these examples in parallel under your choice of
free software license, such as the GNU General Public License,
to permit their use in free software.

Bibliografía

[1] W3 Consortium; A Little History of the World Wide Web http://www.w3.org/History.html; 15 febrero 2001

[2] Robert H'obbes' Zakon; Hobbes' Internet Timeline http://www.ieft.org/rfc/rfc2235.txt; 15 febrero 2001

[3] Richard Stallman; The GNU Project http://www.gnu.org/gnu/thegnuproject.html; 15 febrero 2001

[4] Dave Winer;What is a Web Application? http://davenet.userland.com/2000/03/12/whatIsAWebApplication;15 febrero 2001

[5] Eric Raymond;The Art of Unix Programming http://www.tuxedo.org/~esr/writings/taoup/chapter1.html; 15 febrero 2001

[6] Darleen Sadoski;Client/Server Software Architectures – An Overview http://www.sei.cmu.edu/str/descriptions/clientserver.html; 15 febrero 2001

[7] Darleen Sadoski;Two Tier Software Architectures http://www.sei.cmu.edu/str/descriptions/twotier.html; 15 febrero 2001

[8] Darleen Sadoski;Three Tier Software Architectures http://www.sei.cmu.edu/str/descriptions/threetier.html; 15 febrero 2001

[9] R. Fielding, J. Gettys, J. Mogul, et all.; Hypertext Transfer Protocol – HTTP/1.1 http://www.ietf.org/rfc/rfc2616.txt; 15 febrero 2001

[10] N. Borenstein, N. Freed; MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Messages Bodies http://www.ietf.org/rfc/rfc1521.txt; 15 febrero 2001

[11] W3C HTML working group;XHTML 1.O: The Extensible HyperText Markup Language http://www.w3.org/TR/xhtml1; 15 febrero 2001

[12] W3C HTML working group; HTML 4.01 Specification http://www.w3.org/TR/html4; 15 febrero 2001

[13] Hakom Wium Lie y Bert Box;Cascading Style Sheets, level 1 http://www.w3.org/TR/REC-CSS1; 21 febrero 2001

[14] D. Kristol y L. Montulli;HTTP State Management Mechanism http://www.ietf.org/rfc/rfc2109.txt; 21 febrero 2001

[15] ECMA;ECMAScript Language Specification. Standard ECMA-262 ftp://ftp.ecma.ch/ecma-st/Ecma-262.pdf; 22 febrero 2001

[16] JCC's SQL Std. Page http://www.jcc.com/SQLPages/jccs_sql.htm; 22 febrero 2001

[17] Sun Microsystems, Inc. y Netscape Communications Corp.;Introduction to Public-Key Cryptography http://docs.iplanet.com/docs/manuals/security/pkin/index.htm; 22 febrero 2001

[18] NCSA HTTPd Development Team;Common Gateway Interface http://hoohoo.ncsa.uiuc.edu/cgi/intro.html; 1 Marzo 2001

[19] SSH Communication Security;Introductions to Cryptography http://www.ssh.fi/tech/crypto/intro.html; 1 Marzo 2001

[20] Lincon D. Stein;The World Wide Web Security FAQ http://www.w3.org/Security/Faq/; 1 Marzo 2001

[21] Dr. Subrahmanyam Allamaraju; Java Servlets: Design Issues http://www.subrahmanyam.com/articles/servlets/ServletIssues.html; 2 Marzo 2001

[22] Stefan Zeiger;Servlet Essentials http://www.novocode.com/doc/servlet-essentials/index.html; 2 Marzo 2001

[23] Cary Griffith; ASP Technology Prons and Cons http://www.pscu.com/articles/2000/February/article626.htm; 3 Marzo 2001

[24] Aaron Skonnard; Understanding the IIS Architecture http://www.microsoft.com/mind/1099/inside/inside1099.htm; 5 Marzo 2001

[25] fdc@cliwe.ping.de;ModPerl FAQ http://perl.apache.org/faq/; 5 Marzo 2001

[26] Apache HTTP Server Documentation Project; Apache HTTP Server Documentation http://httpd.apache.org/doc; 14 Marzo 2001

[27] Gregory Gillies (Phrack Magazine); CGI Security Holes http://www.phrack.com/search.phtml?view&article=p49-8; 17 Marzo 2001

[28] CERT Coodination Center; How To Remove Meta-characters From User-Supplied Data In CGI Scripts http://www.cert.org/tech_tips/cgi_metacharacters.html; 17 Marzo 2001

[29] J. Franks, P. Halam-Baker, J. Hostetler, et all; HTTP Authenticacion: Basic and Digest Access Authenticacion http://www.ietf.org/rfc/rfc2617.txt; 21 marzo 2001

[30] David A. Wheeler; Secure Programming for Linux and Unix HOWTO http://www.linuxdoc.org/HOWTO/Secure-Programs-HOWTO; 21 marzo 2001

[31] Ask Slashdot: Wildcard DNS, Session Management and Prior Art http://slashdot.org/askslashdot/00/03/05/2345247.shtml; 21 marzo 2001

[32] Ken Williams; Documentacion del modulo Perl Apache::AuthCookie http://search.cpan.org/doc/KWILLIAMS/Apache-AuthCookie-2.011/AuthCookie.pm; 22 marzo 2001

[33] Brian Jepson, Joan Peckham, Ram Sadasiv; *Database Application Programming with Linux*; 1ra. Edition; Wiley Computer Publishing;

[34] Eric S. Raymond; JARGON http://tuxedo.org/jargon; 27 Marzo 2001

[35] System Development Life Cycle http://www.commerce.virginia.edu/rrn2n/teaching/sldc.htm; 28 Marzo 2001

[36] Dr. Owen Molloy; Software Engineering Lecture 2 http://www.it.nuigalway.ie/~o_molloy/courses/ct858/Lecture3/tsld001.htm; 28 Marzo 2001

[37] Scott W. Ambler; User Interface Design http://www.ambysoft.com/userInterfaceDesign.pdf; 28 Marzo 2001

[38] W3C DOM Working Group; Document Object Model http://www.w3.org/DOM/; 4 Abril 2001

[39] Larry Wall;Perl http://www.perldoc.com/perl5.6/pod/perl.html; 10 Abril 2001

[40] Tim Bunce, J. Douglas Dunlop, Jonathan Leffler; DBI http://www.perldoc.com/cpan/DBI.html; 10 Abril 2001

[41] Andy Wardley; Template Toolkit http://www.template-toolkit.org; 10 Abril 2001

[42] Kip Hampton; What is modperl? http://take23.org/whatis_mod_perl.xml; 11 Abril 2001

[43] Netcraft; Netcraft Web Server Survey http://www.netcraft.com/survey; 11 Abril 2001

[44] Nathan Torkington; Introduction to modperl http://prometheus.frii.com/~gnat/mod_perl/mod_perl.pdf; 11 Abril 2001

[45] mod_ssl Project; About de modssl Project http://www.modssl.org/about; 11 Abril 2001

[46] The PostgreSQL Development Team; PostgreSQL Documentation http://www.postgresql.org/users-lounge/docs/7.0/postgres/; 11 Abril 2001

[47] Jesse Berst; Seven Deadly Website Sins http://www.zdnet.com/anchordesk/story/story_1716.html; 17 Abril 2001

[48] Stas Bekman; modperl Guide http://perl.apache.org/guide/; 8 Mayo 2001

[49] Chamas Corp.;Web Applications Benchmarks http://www.perlmonth.com/features/benchmarks/benchmkars.html?issue=4&id=934820831; 10 Mayo 2001

[50] Michel Pelletier and Amos Latteier; The Zope Book http://www.zope.org/Members/michel/ZB; 12 Mayo 2001