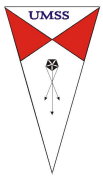

UNIVERSIDAD MAYOR DE SAN SIMÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA
CARRERA DE LICENCIATURA EN INGENIERÍA INFORMÁTICA

CREACIÓN DE UN SISTEMA DE GESTIÓN DE CONTENIDOS FLEXIBLE Y DINÁMICO

Proyecto de Grado, Presentado Para Optar al Diploma Académico
de Licenciatura en Ingeniería Informática.

Presentado por : JALDIN LEDEZMA RICHARD
Tutor : M.Sc. Costas Jáuregui Vladimir Abel

Cochabamba - Bolivia
Julio, 2013

“*Dedicado a mis padres Grover y Lucía*”

Agradecimientos

No habría sido posible culminar este proyecto sin la colaboración de muchas personas, docentes y amigos.

En primer lugar quiero agradecer a Dios, por darme la oportunidad de vivir y ser siempre una luz a la cual seguir durante el desarrollo de todo este proyecto.

También agradecer a mis padres, Grover Jaldin y Lucía Ledezma, por su amor y apoyo incondicional. A mis hermanos Kevin, Jonathan, Abigail, Gabriel y Daniel. A mis primas Norma y Sandra. Agradecer a toda mi familía por su apoyo, por su paciencia, por la confianza depositada en mi y sobre todo por sus consejos.

De igual manera, quiero agradecer a las personas que hicieron posible este proyecto, quienes depositaron su confianza en mi persona, me ofrecieron sus valiosos consejos y me brindaron todo el apoyo académico. Agradecer a mi tutor M.Sc. Vladimir A. Costas J. y al Dr. Pablo R. Azero A.

Por ultimo, también estoy en deuda con mis amigos y compañeros de estudio. Aquellos que con sus consejos, comentarios y sugerencias, ayudaron en la culminación de este proyecto: Lic. Dennys Panozo Montero, Ing. Gabriel García Marquéz, Daniel Saguez Tezanos P., Ing. Miguel A. Ferrufino F., Lic. Carlos Gomez A., Hugo O. Arias S., Ing. Jaqueline Jaimes M., Ing. Paul D. Soliz T., Ing. Antonio Mamani, Ing. Leonel Heredia M., Abel Prado C., Ing. Yamila Chávez M. y Ing. Andrea Cruz G.

Resumen

El desarrollo de un Sistema de Gestión de Contenidos es un proyecto grande, ya que deben ser implementados distintos módulos para dar soporte a las crecientes demandas de los usuarios, así como también las demandas de los desarrolladores.

Los Sistemas de Gestión de Contenidos en la actualidad tienen procesos y configuraciones específicos para la manipulación de los contenidos (artículos, secciones, noticias, menús, etc.). Por otro lado, la gestión de las plantillas y principalmente su personalización es un proceso que requiere de mucho tiempo y dedicación.

Estos dos características hacen que se consuma mucho tiempo de desarrollo en procesos que pueden ser simplificados o incluso automatizados. En el presente proyecto se ha desarrollado un CMS que cumple con los siguientes puntos:

- Las configuraciones se hacen en su mayoría de forma automática.
- La personalización de las plantillas son realizadas directamente desde el CMS de forma gráfica.
- El contenido es personalizado directamente sobre la plantilla.
- Por el uso de librerías personalizadas *i*CMS tiene un tiempo de respuesta menor en comparación con otros CMSs.
- Soporte para la renderización extensiones (Módulos, Bloques y Plantillas).
- Soporte para la instalación de nuevas plantillas.

*i*CMS tiene una arquitectura basada en un motor de renderizado de plantillas como elemento base, adicionalmente se sirve de un conjunto de renderizadores especializados para las demás extensiones (Módulos y Bloques) soportados por *i*CMS.

Cada módulo sigue el patrón MVC (Modelo-Vista-Controlador), esto es muy útil para tener un renderizador estándar para todos los módulos.

Para la mayor parte del *i*CMS se ha utilizado las tecnologías AJAX y JavaScript, con lo cual se ha obtenido gran dinamicidad en la personalización tanto de la plantilla como del contenido.

En conclusión, se encontró que el uso de tecnologías como AJAX, JavaScript, JSON y XML combinado con librerías personalizadas hacen que se pueda mejorar la experiencia del usuario.

Índice general

Índice de Códigos

Índice de figuras

Capítulo 1

INTRODUCCIÓN

En el ambiente de la tecnología y principalmente en el ambiente del desarrollo de software los cambios y actualizaciones son constantes, es de la misma forma que este proyecto propone un nuevo enfoque a los sistemas de gestión de contenidos o CMS's.
Los CMS's en la actualidad siguen patrones bien definidos para la creación de contenidos, muchas veces estos procesos requieren de una formación específica lo cual dificulta la utilización de los mismos. La aplicación *i*CMS desarrollada durante este proyecto se presenta como una alternativa frente a los CMS's tradicionales (Joomla, Drupal, Zikula, etc.), *i*CMS es un CMS de fácil utilización tanto para desarrolladores como para usuarios finales, *i*CMS logra esta cualidad gracias a que muestra gran parte de los controles para la creación del contenido desde la misma pantalla inicial.
Otra de las características principales de *i*CMS es la herramienta de personalización de la plantilla, este trabaja sobre la plantilla base, permitiendo crear nuevas secciones para el contenido, definir o cambiar la sección del contenido principal, entre otras.
Todos los CMS's son diseñados para cumplir con un objetivo bien marcado (portales web, e-commerce, e-learning, etc.) e *i*CMS no es la excepción, *i*CMS ha sido definida para la creación de portales web, aunque con la agregación de extensiones (modulos, bloques, temas) la funcionalidad básica puede ser ampliada enormemente.

Parte I

CONCEPTOS

Capítulo 2

MARCO REFERENCIAL

2.1. Introducción

El marco referencial contiene una descripción de los aspectos fundamentales, técnicas y herramientas utilizadas durante el desarrollo del presente proyecto.

Entre los aspectos fundamentales se menciona elementos como los CMS's, que finalidad tienen, ventajas y desventajas de los CMS; también se hace mención de algunos elementos fundamentales de los lengaujes de programación web.

2.2. ¿Qué son los CMS?

"*Los CMS o Sistemas de Gestión de Contenidos son aplicaciones web utilizadas para crear, editar, gestionar y publicar contenido digital multimedia en diversos formatos. El gestor de contenidos genera páginas web dinámicas interactuando con el servidor web para generar la página web bajo petición del usuario, con el formato predefinido y el contenido extraído de alguna base de datos en el servidor*".

Un aspecto muy importante en los CMS es la personalización del formato al cual se hace mención en el parrafo anterior, es *la plantilla*, la cual es considerada como una extensión (módulo o elemento que puede ser agregado en el CMS para ampliar sus capacidades) que provee un mecanismo para cambiar la apariencia del contenido que es mostrado en el navegador (Robertson, 2003).

2.3. Caraterísticas de los CMS's

Permiten gestionar bajo un formato estandarizado, la información contenida en una base de datos en el servidor, reduciendo el tamaño de las páginas para su descarga en el navegador del cliente y por consiguiente reduciendo el tiempo de espera de carga de la página en el navegador, también reduciendo el costo de gestión del portal con respecto a un sitio web estático, en el que cada cambio de diseño debe ser realizado en todas las páginas web, de la misma forma que cada vez que se agrega contenido tiene que maquetarse una nueva página HTML y subirla al servidor web.

Todas las páginas de un CMS se muestran utilizando la misma plantilla.

En la actualidad, aparte de la ampliación de las funcionalidades de los CMS a través de extensiones, uno de los campos más interesantes es la incorporación de estándares que mejoran la compatibilidad de componentes, facilitan el aprendizaje al cambiar de sistema y aportan calidad y estabilidad.

Algunos de estos estándares son CSS, que permite la creación de hojas de estilo; XML, un lenguaje de marcas que permite estructurar un documento; XHTML, que es un subconjunto del anterior orientado a la presentación de documentos vía web; WAI, que asegura la accesibilidad del sistema; y RSS, para sindicar contenidos de tipo noticia.

También las aplicaciones que rodean los CMS acostumbran a ser estándar (de facto), como los servidores web Apache e ISS; los lenguajes PHP, Perl y Python; y las bases de datos MySQL

y PostgreSQL. La disponibilidad para los principales sistemas operativos de estas aplicaciones y módulos, permite que los CMS puedan funcionar en diversas plataformas sin muchas modificaciones.

Cada CMS tiene una finalidad concreta para la que fue diseñada, aunque en la práctica real no se sigue de manera estricta, los CMS se usan principalmente en en la creacion de paginas web, wikis, foros, blogs, e-commerce, e-learning, etc.

Por ejemplo, Joomla fue diseñado para la creación de páginas web, WordPress inicialmente fue diseñado para la creación de Blogs, MediaWiki, como dice su nombre, fue diseñado para la creación de Wikis.

2.4. AJAX

El término AJAX es un acrónimo de Asynchronous JavaScript + XML, que se puede traducir como "JavaScript asíncrono + XML". Ajax no es una tecnología en sí mismo. En realidad, se trata de varias tecnologías independientes que se unen en formas nuevas y sorprendentes. (Pérez, 2011a)

Las tecnologías que forman AJAX son:

- *XHTML y CSS*, para crear una presentación basada en estándares.
- DOM, para la interacción y manipulación dinámica de la presentación.
- XML, XSLT y JSON, para el intercambio y la manipulación de información.
- XMLHttpRequest, para el intercambio asíncrono de información.
- JavaScript, para unir todas las demás tecnologías.

En el siguiente esquema, la imagen de la izquierda muestra el modelo tradicional de las aplicaciones Web. La imagen de la derecha muestra el nuevo modelo propuesto por AJAX.

En las aplicaciones web tradicionales, las acciones del usuario en la página (clic en un botón, seleccionar un valor de una lista, etc.) desencadenan llamadas al servidor. Una vez procesada la petición del usuario, el servidor devuelve una nueva página HTML al navegador del usuario.

AJAX permite mejorar completamente la interacción del usuario con la aplicación, evitando las recargas constantes de la página, ya que el intercambio de información con el servidor se produce en un segundo plano.

Las aplicaciones construidas con AJAX eliminan la recarga constante de páginas mediante la creación de un elemento intermedio entre el usuario y el servidor. La nueva capa intermedia de AJAX mejora la respuesta de la aplicación. Además que otorga cierta elegancia a la forma de responder de la pagina frente a las acciones del usuario.

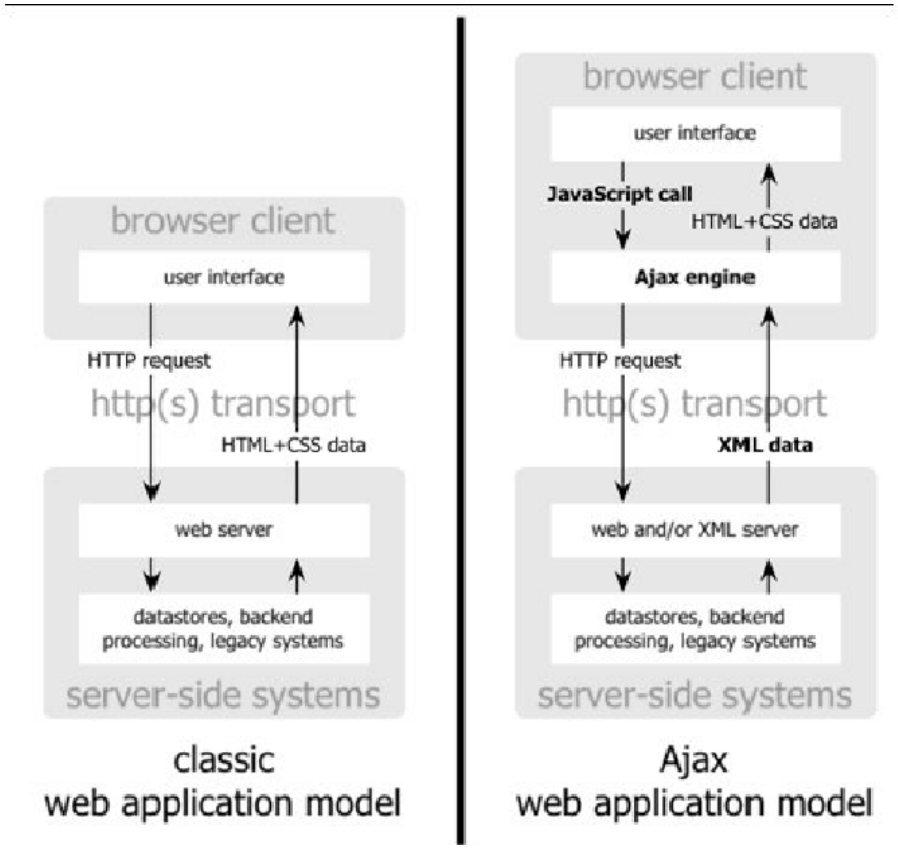

Figura 2.1: Tecnologías agrupadas bajo el concepto de AJAX. Fuente: Javier Eguíluz Pérez; *Introducción a AJAX.*

2.5. Hyper Text Markup Language (HTML)

Hyper Text Markup Language (Lenguaje de Marcado de Hipertexto), o HTML, es el lenguaje de marcado predominante para la elaboración de páginas web. Es usado para describir la estructura y el contenido en forma de texto, así como para complementar el texto con otros elementos.

HTML también es usado para referirse al contenido del tipo de MIME text/html o como un término genérico para el HTML, ya sea en forma descendida del XML (como XHTML 1.0 y posteriores) o en forma descendida directamente de SGML (como HTML 4.01 y anteriores).

Este lenguaje será el encargado de convertir un simple archivo de texto inicial en una página web con diferentes tipos y tamaños de letra, con imágenes, animaciones, formularios interactivos, etc (Echevarría, 1995).

2.6. eXtensible Markup Language (XML)

"*Es un metalenguaje de etiquetas extensible desarrollado por la W3C. Consolidado como una simplificación y adaptación del SGML que permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML). Por lo tanto XML*

no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.". (W3C, 2011)

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

La principal que se tiene al utilizar XML es que nos es posible estructurar la información para distintos propósitos.

2.7. Hojas de Estilo (CSS)

CSS es un lenguaje creado para controlar el aspecto o presentación de los documentos electrónicos definidos con HTML y XHTML. CSS es la mejor forma de separar los contenidos y su presentación y es imprescindible para crear páginas web complejas.

"*Separar la definición de los contenidos y la definición de su aspecto presenta numerosas ventajas, ya que obliga a crear documentos HTML/XHTML bien definidos y con significado completo (también llamados "documentos semánticos")*". Además, mejora la accesibilidad del documento, reduce la complejidad de su mantenimiento y permite visualizar el mismo documento en infinidad de dispositivos diferentes (Lie y Bos, 1999).

Al crear una página web, se utiliza en primer lugar el lenguaje HTML/XHTML para marcar los contenidos, es decir, para designar la función de cada elemento dentro de la página: párrafo, titular, texto destacado, tabla, lista de elementos, etc.

Una vez creados los contenidos, se utiliza el lenguaje CSS para definir el aspecto de cada elemento: color, tamaño y tipo de letra del texto, separación horizontal y vertical entre elementos, posición de cada elemento dentro de la página, etc.

Todo este trabajo se logra definiendo características en el HTML conocidas como atributos, pero no cualquier atributo, concretamente el atributo "class" o también el atributo "id", el primero de uso exclusivo para las hojas de estilo (CSS), y el segundo como una alternativa, el atributo "id" también es utilizado frecuentemente en las instrucciones de Javascript.

2.8. JavaScript

El JavaScript es un lenguaje de programación que surgió por la necesidad de ampliar las posibilidades del HTML. En efecto, al poco tiempo de que las páginas web apareciesen, se hizo patente la necesidad de algo más que las limitadas prestaciones del lenguaje básico, ya que el HTML solamente provee de elementos que actúan exclusivamente sobre el texto y su estilo, pero no permite, como ejemplo sencillo, ni siquiera abrir una nueva ventana o emitir un mensaje de

aviso.

JavaScript se utiliza principalmente para crear páginas web dinámicas.

Una página web dinámica es aquella que incorpora efectos como texto que aparece y desaparece, animaciones, acciones que se activan al pulsar botones y ventanas con mensajes de aviso al usuario.

Técnicamente, JavaScript es un lenguaje de programación interpretado, por lo que no es necesario compilar los programas para ejecutarlos. En otras palabras, los programas escritos con JavaScript se pueden probar directamente en cualquier navegador sin necesidad de procesos intermedios. (Pérez, 2011b)

2.9. Hypertext Pre-Processor (PHP)

PHP es un lenguaje de programación usado normalmente para la creación de páginas web dinámicas. Se trata de un lenguaje interpretado.

Su interpretación y ejecución se da en el servidor web, en el cual se encuentra almacenado el script, y el cliente sólo recibe el resultado de la ejecución. Cuando el cliente hace una petición al servidor para que le envíe una página web, generada por un script PHP, el servidor ejecuta el intérprete de PHP, el cual procesa el script solicitado que generará el contenido de manera dinámica, pudiendo modificar el contenido al enviar, y regresar el resultado al servidor, el cual se encarga de regresárselo al cliente.

Permite la conexión a diferentes tipos de servidores de bases de datos tales como MySQL, Postgres, Oracle, ODBC, DB2, Microsoft SQL Server, Firebird y SQLite; lo cual permite la creación de Aplicaciones web muy robustas.

Tiene la capacidad de ser ejecutado en la mayoría de los sistemas operativos, tales como UNIX (y de ese tipo, como Linux o Mac OS X) y Windows, puede interactuar con los servidores de web más populares ya que existe en versión CGI, módulo para Apache, e ISAPI.

PHP es una alternativa a las tecnologías de Microsoft ASP y ASP.NET (que utiliza C#/VB.NET como lenguajes), a ColdFusion de la compañía Adobe (antes Macromedia), a JSP/Java de Sun Microsystems, y a CGI/Perl. Aunque su creación y desarrollo se da en el ámbito de los sistemas libres, bajo la licencia GNU. (Gerner, 2004)

Ventajas

- Es un lenguaje multiplataforma.
- Capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad, destaca su conectividad con MySQL.
- Capacidad de expandir su potencial utilizando la enorme cantidad de módulos (llamados ext's o extensiones).
- Posee una amplia documentación en su página oficial, entre la cual se destaca que todas las funciones del sistema están explicadas y ejemplificadas en un único archivo de ayuda.

- Es libre, por lo que se presenta como una alternativa de fácil acceso para todos.
- Permite las técnicas de Programación Orientada a Objetos.
- Biblioteca nativa de funciones sumamente amplia e incluida.
- No requiere definición de tipos de variables.
- Tiene manejo de excepciones.

Desventajas

- No posee una abstracción de base de datos estándar, sino bibliotecas especializadas para cada motor (a veces más de una para el mismo motor).
- No posee adecuado manejo de internacionalización, unicode, etc.

2.10. Servidor HTTP Apache

Apache es el servidor web hecho por excelencia, su configurabilidad, robustez y estabilidad hacen que cada vez millones de servidores reiteren su confianza en este programa.

La historia de Apache se remonta a febrero de 1995, donde empieza el proyecto del grupo Apache, el cual está basado en el servidor Apache httpd de la aplicación original de NCSA.

El desarrollo de esta aplicación original se estancó por algún tiempo tras la marcha de Rob McCool por lo que varios webmaster siguieron creando sus parches para sus servidores web hasta que se contactaron vía email para seguir en conjunto el mantenimiento del servidor web, fue ahí cuando formaron el grupo Apache. (Foundation, 2011)

Fueron Brian Behlendorf y Cliff Skolnick quienes a través de una lista de correo coordinaron el trabajo y lograron establecer un espacio compartido de libre acceso para los desarrolladores.

Este software libre es grandemente reconocido en muchos ámbitos empresariales y tecnológicos, por los siguientes aspectos:

- Corre en una multitud de Sistemas Operativos, lo que lo hace prácticamente universal.
- Apache es una tecnología gratuita de código fuente abierto. Esto le da una transparencia a este software de manera que si queremos ver que es lo que estamos instalando como servidor , lo podemos saber, sin ningún secreto.
- Apache es un servidor altamente configurable de diseño modular. Es muy sencillo ampliar las capacidades de servidor. Actualmente existen muchos módulos para Apache que son adaptables a este, y están ahí para que los instalemos cuando los necesitemos.
- Apache trabaja con gran cantidad de lenguajes como Perl, PHP y otros lenguajes de script. Teniendo todo el soporte que se necesita para tener páginas dinámicas.
- Apache te permite personalizar la respuesta ante los posibles errores que se puedan dar en el servidor. Es posible configurar Apache para que ejecute un determinado script cuando ocurra un error en concreto.

- Tiene una alta configurabilidad en la creación y gestión de logs. Apache permite la creación de ficheros de log a medida.

2.11. MySQL

MySQL es un sistema gestor de bases de datos relacionales en SQL, esto significa que permite la gestión de los datos de una Base de datos relacional, usando un lenguaje de consulta estructurado. Y, por tanto, que a partir de una oración llamada más propiamente consulta, MySQL llevará a cabo una determinada acción sobre nuestra base de datos. (Corporation, 2012)

Algunas de sus características son:

- Código abierto
- Fácil de instalar y configurar
- Funcionalidad
- Portabilidad
- Velocidad

2.12. Programación orientada a objetos

El desarrollo de software es un proceso muy complejo que requiere de una metodología eficiente y sistemática. El ciclo de vida de un sistema de computación comienza con la formulación de un problema, seguido de análisis, diseño, implementación, verificación y validación del software. El modelo orientado a objetos presenta un enfoque evolucionado para la ingeniería de software.

Una diferencia importante entre la programación tradicional y la programación orientada a objetos es que la programación tradicional separa los datos de las funciones, mientras que la programación orientada a objetos define un conjunto de objetos donde se combina de forma modular los datos con las funciones. (Deitel y Deitel, 2004)

Los aspectos principales de la programación orientada a objetos son:

- Objetos
- Clasificación
- Instanciación
- Generalización
- Abstracción
- Encapsulación
- Modularidad
- Extensibilidad
- Polimorfismo
- Reutilización de Código

2.13. UML

UML (Unified Modeling Language) es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema o software orientado a objetos. Se ha convertido en el estándar de facto de la industria, debido a que ha sido concebido por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh.

Esta notación ha sido ampliamente aceptada debido al prestigio de sus creadores y debido a que incorpora las principales ventajas de cada uno de los métodos particulares en los que se basa.

Booch, OMT y OOSE. UML ha puesto fin a las llamadas "guerras de métodos" que se han mantenido a lo largo de los 90, en las que los principales métodos sacaban nuevas versiones que incorporaban las técnicas de los demás. Con UML se fusiona la notación de estas técnicas para formar una herramienta compartida entre todos los ingenieros software que trabajan en el desarrollo orientado a objetos.

El objetivo principal cuando se empezó a gestar UML era posibilitar el intercambio de modelos entre las distintas herramientas CASE orientadas a objetos del mercado. Hay que tener en cuenta que el estándar UML no define un proceso de desarrollo específico, tan solo se trata de una notación. (Sommerville, 2002)

2.14. Patrones Arquitectónicos

Los patrones arquitectónicos se utilizan principalmente para definir la arquitectura (estructura) de la aplicación de software. El objetivo de utilizar estos patrones es mejorar tanto el rendimiento como la mantenibilidad de los sistemas informáticos(GangOfFour), 1994).

2.14.1. MVC

El patrón MVC "*Modelo Vista Controlador*", define tres elementos básicos de toda aplicación de software:

Modelo , Los modelos representan a todo lo relacionado con las bases de datos.

Vista , Las vistas representan a las pantallas finales para el usuario.

Controlador , Los controladores, como su nombre indica, se concentran en las acciones que se generan desde las vistas para procesarlas.

2.14.2. MVP

El patrón MVP "*Modelo Vista Presentador*", es muy similar al MVC, de hecho surge como una mejora al mencionado patrón, también define tres elementos básicos de toda aplicación de software:

Modelo , Los modelos representan a todo lo relacionado con las bases de datos.

Vista , Las vistas representan a las pantallas finales para el usuario.

Presentador , Los presentadores hacen de controladores, pero a la vez se encargan de presentar la información necesaria a las vistas, para que estas se construyan.

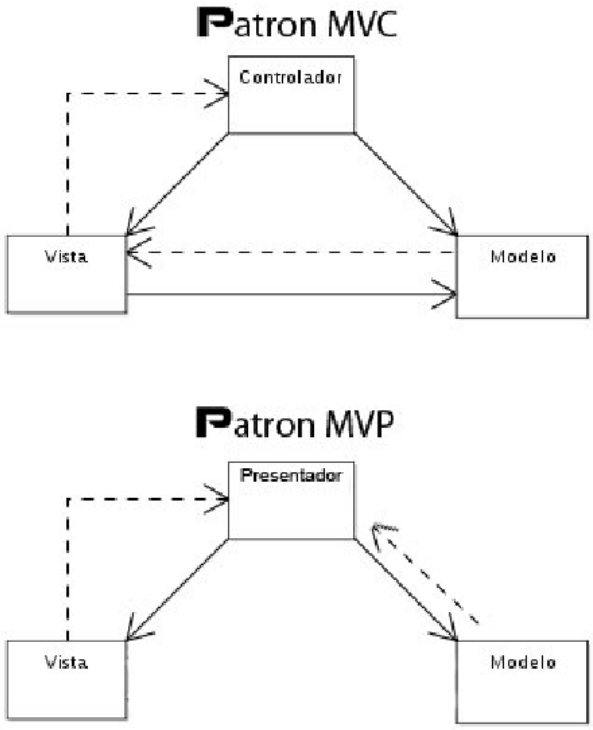

Figura 2.2: Patrónes MVC vs MVP. [Elaboración propia]

2.15. Otros Patrones

Aparte de los patrones de arquitectura, existen muchos otros patrones que proveen soluciones a problemas casi cotidianos en el desarrollo de cualquier sistema tanto web como de escritorio (GangOfFour), 1994).

A continuación se presentan algunos de ellos.

2.15.1. Patrón Singleton

Este patrón es uno de los más básicos y también uno de los más utilizados en los Sitemas de Información. Consiste en tener una sola instancia de alguna clase la cual es global para toda la aplicación.

2.15.2. Patrón Factory Method

Este patrón fue utilizado para complementar el patrón Singleton. Consiste en tener una clase con métodos que se encargan de construir instancias de distintas clases.

2.16. Sistemas de Información Web

La principal característica que tiene un sistema de información web frente a un sistema de información tradicional es que un sistema de información web, está alojado en un servidor web, por tanto es posible acceder a él desde cualquier parte del mundo, las 24 horas del día y desde cualquier equipo que tenga conexión a internet, en cambio con un sistema tradicional accedemos a él, desde un terminal especifico.

Otra ventaja que se tiene en un sistema web es que la información almacenada en el servidor ya tiene sus propios respaldos y demás, gracias a las políticas manejadas por los dueños de los servidores. De esta manera deja de ser una preocupación para el usuario final el cómo evitar la pérdida o corrupción de la información.

Capítulo 3

ÁREA DE APLICACIÓN

3.1. Introducción

El presente proyecto propone una solución a un problema que se presenta con bastante frecuencia a la hora de crear y administrar el contenido de los sitios web creados con CMS's, Sistemas de Gestión de Contenidos por su sigla en inglés.
Con el desarrollo de un Sistema de Gestión de Contenidos que pretende:

- Mejorar en tiempo de desarrollo de las páginas web.
- Reducir las configuraciones necesarias para crear contenidos.
- Crear un panel de administración interactivo e intuitivo.
- Hacer gráfica la manera de manipular el contenido.
- Facilitar la modificación de las plantillas en tiempo de ejecución.

La esencia del sistema se encuentra en lo intuitivo y la flexibilidad que ofrece en la gestión del contenido, lo cual pretende mejorar la creación de contenidos y sobre todo facilitar la manipulación de los mismos, ya sea desde la administración avanzada o sobre las plantillas mismas.
El sistema es un CMS (Sistema de Gestión de Contenidos) orientado principalmente a la creación de sitios Web.

3.2. Antecedentes

La administración de los sitios web siempre ha sido compleja debido a la forma de trabajo de las herramientas de administración, ya sea desde la misma actualización de los contenidos que presenta, hasta la codificación de extensiones, por tal razón surgieron los CMS (Sistemas de Gestión de Contenidos) cuya principal tarea es *"Facilitar la creación, gestión, distribución y publicación de información"*.

Esta tarea se ha facilitado bastante gracias a los CMS's, pero muchas veces la poca experiencia sobre cómo administrarlos hace que la tarea de crear sitios siga siendo tediosa, otra situación que dificulta también la creación y administración de sitios con CMS es que se tienen interfaces gráficas poco amistosas y sobre todo poco intuitivas.

3.3. Identificación del problema

Una de las dificultades a la hora de crear y administrar sitios web diseñados con los CMS actuales es que muchas veces es necesario hacer varias configuraciones para poder gestionar (crear, editar, etc.) los contenidos.

A veces tenemos herramientas que vienen por defecto que muy pocas veces se utiliza, dando lugar a que el trabajo de administrar un sitio hecho con un CMS no sea intuitivo ni sencillo (Drupal, Zikula). Por otro lado esto también provoca que tengamos un panel de administración bastante abarrotado.
Muchas veces es necesario conocer los pasos o la secuencia de creación de nuevos contenidos, y también para mostrar los mismos.

En varios de los CMS's actuales la personalización de las plantillas es un trabajo que requiere demasiado tiempo, además de que no es intuitivo.

3.4. Objetivos

3.4.1. Objetivo General

Desarrollar un sistema de gestión de contenidos que permita crear y administrar los sitios web de manera exible y dinámica.

3.4.2. Objetivos Específicos

- Facilitar una estructura exible de forma tal que las configuraciones para crear los contenidos sean mínimas o se hagan de forma automática.
- Definir una estructura dinámica que facilite el trabajo en la administración de los sitios web creados con el CMS.
- Crear una estructura gráfica que permita la gestión de los contenidos de forma dinámica y exible.
- Implementar un motor de plantillas gráfico para facilitar la personalización de los contenidos sobre dichas plantillas.

3.5. Innovación Tecnológica

La creación de un CMS cuya principal característica es ser gráfico y flexible, es un cambio que invierte de alguna manera la forma convencional de gestionar los contenidos permitiendo gestionar los contenidos de distintas maneras.

3.6. Justificación

Este proyecto facilitará una herramienta que permitirá gestionar los contenidos, crearlos y posteriormente acomodarlo en las plantillas para su visualización en los navegadores.

La herramienta resultante será de gran ayuda para todos los desarrolladores en menor o mayor medida, ya que no serán necesarios tener grandes conocimientos acerca de CMS's para poder crear y administrar sitios web.

Muchas veces hemos oído la frase "El tiempo es oro", pues bien, al reducir el tiempo de desarrollo de los sitios con el CMS resultante de este proyecto maximizamos la puesta en línea de los sitios, facilitándole al cliente la posibilidad de trabajar con su sitio en muy poco tiempo.

3.7. Alcances

El proyecto no contempla de momento, la creación de extensiones para ampliar las funcionalidades del mismo pero si el soporte para las mismas.

Asimismo, entre las cosas que se contemplan en este proyecto cabe mencionar que este será un prototipo.

La herramienta resultante será capaz de facilitar:

- La creación de nuevos contenidos, como también la edición de los mismos.
- La creación de nuevas secciones (Espacios donde se mostrarán los contenidos).
- La gestión de roles tales como Administrador, Editor y Visitante.
- La manipulación de los contenidos y secciones de manera gráfica directamente sobre las plantillas.
- La gestión de contenidos secundarios.

Parte II

DESARROLLO DEL PROYECTO

Capítulo 4

*i*CMS

4.1. Introducción

El proyecto iCMS no sigue una metodología de desarrollo en particular, por lo tanto este capítulo muestra los elementos utilizados tanto para el desarrollo mismo del proyecto como también parte de los resultados obtenidos (modelos).

El desarrollo del proyecto fue basado en una lista de tareas para poder conseguir la arquitectura inicial del *i*CMS.

4.2. Lista de tareas

La lista de tareas que se muestran a continuación sirvio de base para poder desarrollar el *i*CMS.

- Definir estructura de Directorios y Archivos.
- Definir configuraciones básicas.
- Desarrollar renderizador para las plantillas.
- Diseñar estructura de los módulos y bloques.
- Desarrollar renderizador para módulos y bloques.
- Desarrollar clases Singleton (Request, DBO, Factory, Editor WYSIWYG).
- Desarrollar bloque de menus.
- Desarrollar bloque de usuarios (pantalla de acceso - login)
- Desarrollar modulo de contenido.
- Desarrollar modulo de usuarios (formulario de registro, modificacin de datos).
- Crear libreras de “drag and drop” personalizadas y generalizadas para cualquier tipo de contenido (artculos, bloques, etc.).

Durante el desarrollo de estas tareas, han ido surgiendo nuevas tareas más especializadas.

- Desarrollar módulo de menús.
- Desarrollar bloque de navegación (breadcrumb).
- Desarrollar bloque y módulo de noticias.
- Mejorar el renderizador de plantillas para soportar el reordenamiento de bloques.
- Actualizar el renderizador de plantillas para soportar soportar la edición de la plantilla.

4.3. Arquiterctura

Para diseñar la arquitectura del sistema *i*CMS se considerarón dos patrones arquitectónicos, el patrón MVC (Modelo-Vista-Controlador) y el patrón MVP (Modelo-Vista-Presentador).

Mucho de este patrón se ven en los distintos módulos que componen el *i*CMS, también vemos otros patrones presentes en el desarrollo del proyecto (Singleton, Factory Method).

A continuación se presenta un pequeño ejemplo sin el patrón MVC y luego el mismo ejemplo pero aplicando el patrón MVC y el patrón MVP:

Como se puede apreciar en el siguiente ejemplo, todos los elementos de la aplicación estan mezclados en un solo archivo.

```
<?php
$db = new PDO('mysql:host=localhost;dbname=bd', 'root', 'password');
$consulta = $db->prepare('SELECT * FROM items');
$consulta->execute();
$items = $consulta->fetchAll();
?>
<div>
    <table class="adminlist" cellspacing="1">
        <thead>
            <tr>
                <th class="title">ID</th>
                <th nowrap="nowrap">Nombre</th>
                <th nowrap="nowrap">Descripcio^oacute<n</th>
            </tr>
        </thead>
        <tbody>
            <?php foreach ($items as $row) {
            $link = 'index.php?accion=alguna_accion&id='. $row->id;
            ?>
            <tr>
                <td><?php echo $row->id; ?></td>
                <td><a href="<?php echo $link; ?>"><?php echo $row->nombre; ?></a></td>
                <td><a href="<?php echo $link; ?>"><?php echo $row->descripcion; ?></a></td>
            </tr>
            <?php } ?>
        </tbody>
    </table>
</div>
```

Código 1: Ejemplo sin ningún patrón

El patrón MVC separa los elementos de una aplicacion en tres grupos (Modelos, Vistas y Controladores).

El controlador es el elemento que se encarga de gestionar todas los eventos que son generados por el usuario a través de las Vistas, así como también de generar las vistas que son el resultado de dichas acciones.

```
<?php

class Controlador {

    public function mostrar(){
        $vista = new Vista();
        $vista->setPlantilla('principal');
        $vista->mostrar();
```

```
    }
}
?>
```

Código 2: Clase Controlador

El modelo es el elemento encargado de la gestión de los datos (almacenamiento y recuperación).

```
<?php

class Modelo {
    protected $db;
    public function __construct() {
        $this->db = new PDO('mysql:host=localhost;dbname=bd', 'root', 'password');
    }

    public function getDatos(){
        $consulta = $this->db->prepare('SELECT * FROM items');
        $consulta->execute();
        return $consulta->fetchAll();
    }
}
?>
```

Código 3: Clase Modelo

La vista es el elemento encargado de estructurar los datos recuperados para presentarlos al usuario, la vista conoce la existencia del modelo y puede interactuar con el.

```
<?php
class Vista {
    protected $layout;
    public function mostrar(){
        $model = new Modelo();
        $items = $model->getDatos();
        require 'plantillas/' . $layout . '.php';
    }

    public function setLayout($_layout = ''){
        $this->layout = $_layout;
    }
}
?>
```

Código 4: Clase Vista

La plantilla es un elemento auxiliar de la vista, no es una clase, sino un archivo html con el mínimo que sirve para maquetar los datos obtenidos en la vista.

```
<html>
    <head></head>
    <body>
        <div>
            <table class="adminlist" cellspacing="1">
                <thead>
                    <tr>
                        <th class="title">ID</th>
                        <th nowrap="nowrap">Nombre</th>
                        <th nowrap="nowrap">Descripcio^oacute<n</th>
                    </tr>
                </thead>
                <tbody>
                    <?php foreach ($items as $row) {
                    $link = 'index.php?accion=alguna_accion&id='. $row->id;
```

```
16                  ?>
17                  <tr>
18                      <td><?php echo $row->id; ?></td>
19                      <td><a href="<?php echo $link; ?>"><?php echo $row->nombre; ?></a></td>
20                      <td><a href="<?php echo $link; ?>"><?php echo $row->descripcion; ?></a></td>
21                  </tr>
22                  <?php } ?>
23              </tbody>
24          </table>
25      </div>
26  </body>
27  </html>
```

Código 5: Plantilla de la vista

El patrón MVP separa (al igual que el patrón MVC) los elementos de una aplicación en tres grupos (Modelos, Vistas y Presentadores). La mayor diferencia entre MVC y MVP es que en el patrón MVP, la Vista no sabe de la existencia del Modelo, algo que no sucede en el patrón MVC.

El presentador es básicamente un controlador, pero no delega la responsabilidad de procesar la información que llega ya sea desde la vista o del modelo.

```
1  <?php
2  class Presentador {
3      public function mostrar(){
4          $modelo = new Modelo();
5          $items = $modelo->getDatos();
6          require 'principal.php';
7      }
8  }
9  ?>
```

Código 6: Clase Presentador

El modelo trabaja de la misma forma que en el patrón MVC.

```
1  <?php
2
3  class Modelo {
4      protected $db;
5      public function __construct() {
6          $this->db = new PDO('mysql:host=localhost;dbname=bd', 'root', 'password');
7      }
8
9      public function getDatos(){
10         $consulta = $this->db->prepare('SELECT * FROM items');
11         $consulta->execute();
12         return $consulta->fetchAll();
13     }
14 }
15 ?>
```

Código 7: Clase Modelo

La vista es el elemento encargado de estructurar los datos recuperados por el presentador para mostrarlos al usuario.

```
1  <html>
2      <head></head>
3      <body>
4          <div>
5              <table class="adminlist" cellspacing="1">
6                  <thead>
7                      <tr>
```

```
8                       <th class="title">ID</th>
9                       <th nowrap="nowrap">Nombre</th>
10                      <th nowrap="nowrap">Descripcio^oacute<n</th>
11                  </tr>
12              </thead>
13              <tbody>
14                  <?php foreach ($items as $row) {
15                  $link = 'index.php?accion=alguna_accion&id='. $row->id;
16                  ?>
17                  <tr>
18                      <td><?php echo $row->id; ?></td>
19                      <td><a href="<?php echo $link; ?>"><?php echo $row->nombre; ?></a></td>
20                      <td><a href="<?php echo $link; ?>"><?php echo $row->descripcion; ?></a></td>
21                  </tr>
22                  <?php } ?>
23              </tbody>
24          </table>
25      </div>
26  </body>
27 </html>
```

Código 8: Vista

Dadas las enormes similitudes entre estos dos patrones, se decidió hacer uso del patrón MVC, la razón para esta elección es que todas las extensiones tienen caracteristicas similares pero no iguales a las de Joomla, con lo cual se plantea para un futuro brindar el soporte necesario para que *i*CMS sea totalmente compatible con las extensiones de Joomla.

4.4. Modelo de Clases

Como ya fue citado en el marco referencial, el modelo clases es la herramienta que da soporte, de alguna manera a la Programación Orientada a Objetos. Representando a los objetos que son participes del de sistema *i*CMS.
A continuación podemos ver el modelo de clases general del proyecto *i*CMS.

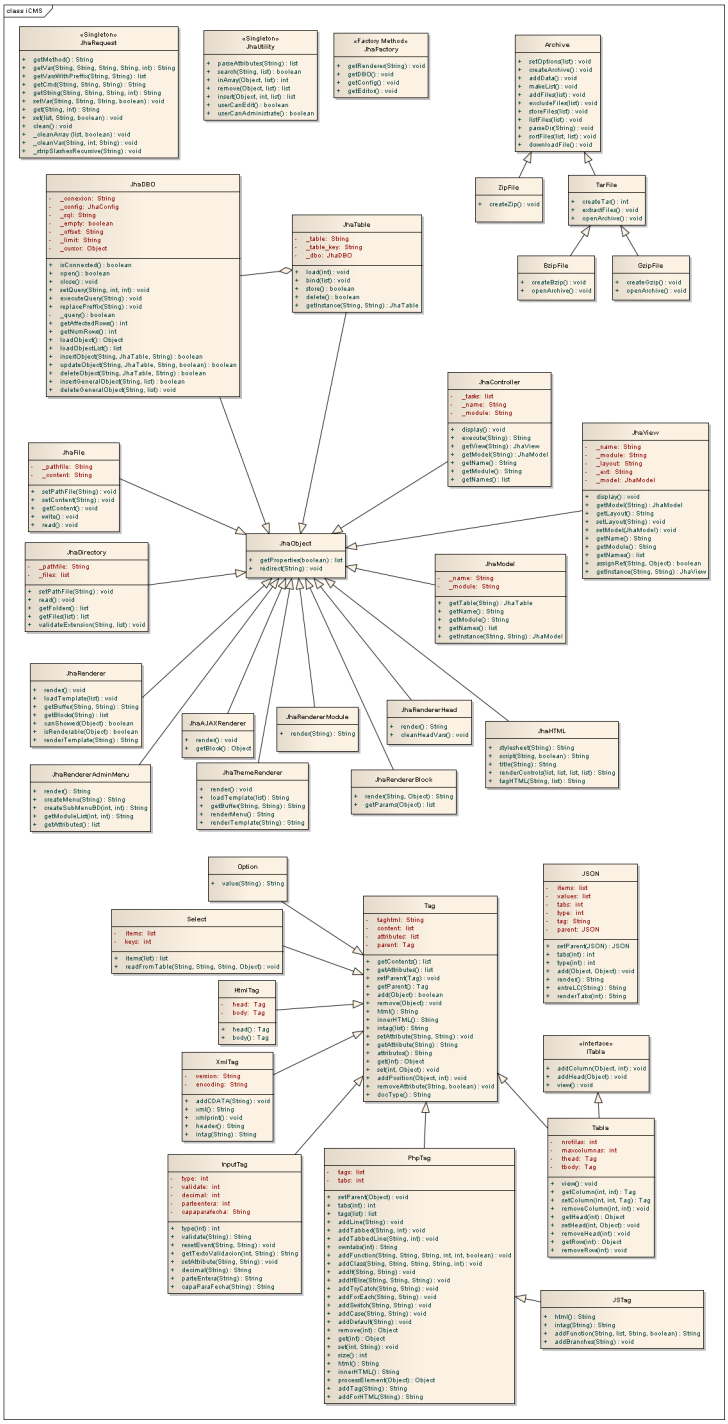

Figura 4.1: Modelo de clases general. [Elaboración propia].

4.4.1. Aplicación de los patrones *Singleton* y *Factory Method*

La aplicación de estos patrones en el sistema se da en tres clases que son utlizadas por todo el sistema, desde el *Framework Jhaley* hasta las extensiones (módulos y bloques).

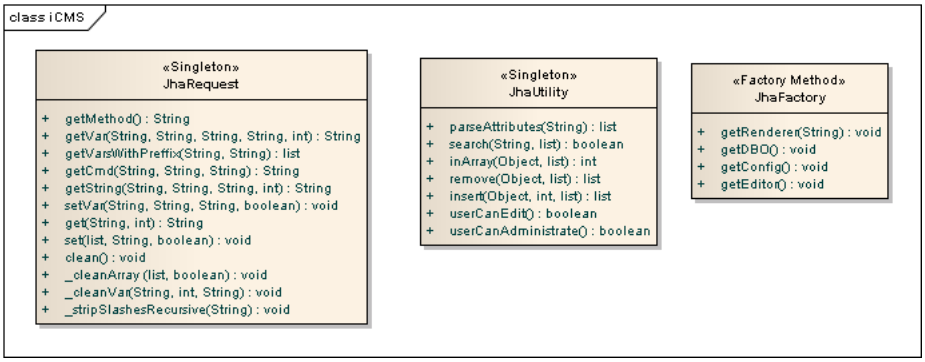

Figura 4.2: Modelo de clases: Patrones Singleton y Factory Method. [Elaboración propia].

La clase *JhaFactory* es la encargada de crear las instancias Singleton, si la instancia ya existe, entonces esta es reutilizada.

```php
<?php
defined( '_JHAEXEC' ) or die( 'Access Denied' );
jhaimport('jhaley.base.object');

class JhaFactory extends JhaObject {
    static function &getRenderer($type = null){
        static $renderer;
        if($type == null){
            if(JhaRequest::getVar('elem') == 'mod_base' && JhaRequest::getVar('controller') == 'theme' &&
                 (JhaRequest::getVar('task') == 'personalizeTheme' || JhaRequest::getVar('task') ==
                 'savePersonalizedChanges')){
                jhaimport('jhaley.html.theme');
                $renderer = new JhaThemeRenderer();
            }
            else {
                jhaimport('jhaley.html.renderer');
                $renderer = new JhaRenderer();
            }
        }
        elseif($type == 'block'){
            jhaimport('jhaley.html.block');
            $renderer = new JhaRendererBlock();
        }
        elseif($type == 'module'){
            .....
        }
        return $renderer;
    }

    static function &getDBO(){
        static $dbo;
        if (!is_object($dbo)) {
            jhaimport('jhaley.db.dbo');
            $dbo = new JhaDBO(JhaFactory::getConfig());
        }
        return $dbo;
    }

    static function &getConfig(){
```

```
38          static $config;
39          if (!is_object($config)) {
40              require_once JHA_CONFIGURATION_PATH . DS . 'config.php';
41              $config = new JhaConfiguration();
42          }
43          return $config;
44      }
45
46      .....
47  }
48  ?>
```

Código 9: Clase Factory Method

Un ejemplo claro del uso del patrón Singleton se puede ver en la instancia de la clase *JhaDBO*, la cual es utilizada constantemente en los modelos de los distintos módulos.

```
1   <?php
2   defined( '_JHAEXEC' ) or die( 'Access Denied' );
3   jhaimport('jhaley.mvc.model');
4
5   class MenuModelMenu extends JhaModel {
6       public function __construct() {
7           parent::__construct();
8       }
9
10      public function getMenus(){
11          $db = &JhaFactory::getDBO();
12          $db->setQuery('SELECT m.id, m.titulo, m.descripcion, u.nombre as creador FROM #__menu as m,
                #__usuario as u WHERE m.usuario = u.id ORDER BY m.id');
13          return $db->loadObjectList();
14      }
15
16      public function getMenu($id){
17          $db = &JhaFactory::getDBO();
18          $db->setQuery('SELECT * FROM #__menu WHERE id = ' . $id);
19          return $db->loadObject();
20      }
21      .....
22  }
23  ?>
```

Código 10: Patrón Singleton en acción

4.4.2. Manejo de Archivos compresos

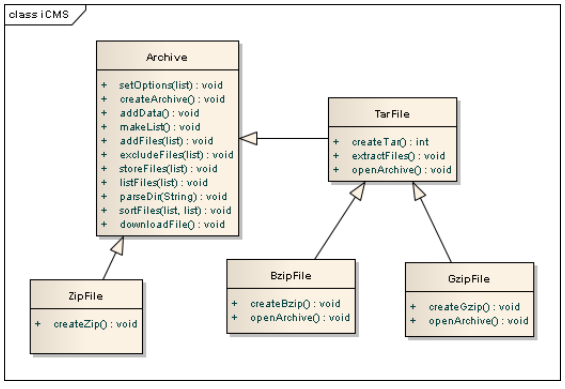

Figura 4.3: Modelo de clases: Manejo de Archivos Compresos. [Elaboración propia].

El manejo de archivos compresos esta dado por un grupo de clases especializadas para este cometido, además estas clases utilizan elementos propios del core de PHP a través de la manipulación de los archivos compresos en forma de datos binarios.

La superclase *Archive* contiene ciertos elementos que merecen ser detallados.

Una de las caractiristicas primordiales es que sus configuraciones están dadas por una serie de opciones:

basedir , describe el directorio donde se comprimirán/descomprimirán los archivos.

name , describe el nombre del archivo compreso.

inmemory , describe se se manipulara el archivo directamente desde el disco duro o en memoria.

level , describe el nivel de compresión.

... entre otros.

```php
<?php
defined( '_JHAEXEC' ) or die( 'Access Denied' );

class Archive {
    function __construct($name) {
        $this->options = array (
            'basedir' => ".",
            'name' => $name,
            'prepend' => "",
            'inmemory' => 0,
            'overwrite' => 0,
            'recurse' => 1,
            'storepaths' => 1,
            'followlinks' => 0,
            'level' => 3,
            'method' => 1,
            'sfx' => "",
            'type' => "",
            'comment' => ""
        );
        $this->files = array ();
        $this->exclude = array ();
        $this->storeonly = array ();
        $this->error = array ();
    }
    .....
}
?>
```

Código 11: Opciones para la configuración de los compresos.

Otra característica es la posibilidad de crear archivos descargables desde los navegadores.

```php
<?php
defined( '_JHAEXEC' ) or die( 'Access Denied' );

class Archive {
    .....
    function downloadFile() {
        if ($this->options['inmemory'] == 0) {
            $this->error[] = "Can only use download_file() if archive is in memory. Redirect to file
                otherwise, it is faster.";
            return;
```

```
10          }
11          switch ($this->options['type']) {
12              case "zip":
13                  header("Content-Type: application/zip");
14              break;
15              case "bzip":
16                  header("Content-Type: application/x-bzip2");
17              break;
18              case "gzip":
19                  header("Content-Type: application/x-gzip");
20              break;
21              case "tar":
22                  header("Content-Type: application/x-tar");
23          }
24          $header = "Content-Disposition: attachment; filename=\"";
25          $header .= strstr($this->options['name'], "/") ? substr($this->options['name'],
                 strrpos($this->options['name'], "/") + 1) : $this->options['name'];
26          $header .= "\"";
27          header($header);
28          header("Content-Length: " . strlen($this->archive));
29          header("Content-Transfer-Encoding: binary");
30          header("Cache-Control: no-cache, must-revalidate, max-age=60");
31          header("Expires: Sat, 01 Jan 2000 12:00:00 GMT");
32          print($this->archive);
33      }
34  }
35  ?>
```

Código 12: Opciones para la descarga de los compresos.

Las clases *ZipFile* y *TarFile* extienden directamente de *Archive*, mientras que *ZipFile* representa a los archivos compresos completamente formados, *TarFile* representa a un archivo de empaquetado.

```
1   <?php
2   defined( '_JHAEXEC' ) or die( 'Access Denied' );
3   jhaimport('jhaley.compress.archive');
4
5   class ZipFile extends Archive {
6       function __construct($name) {
7           parent::__construct($name);
8           $this->options['type'] = "zip";
9       }
10
11      function createZip() {
12          ...
13      }
14  }
15  ?>
```

Código 13: Compresos Zip.

```
1   class TarFile extends Archive {
2       function __construct($name) {
3           parent::__construct($name);
4           $this->options['type'] = "tar";
5       }
6
7       function createTar() {
8           ...
9       }
10  }
11  ?>
```

Código 14: Empaquetadores Tar.

Los compresos *Gzip*, *TGzip*, *Bzip* y *TBzip* son representados por las subclases *GzipFile* y *BzipFile*. Lo que se gana separando en distintas clases el manejo de los compresos son dos cosas. Primero, que se tienen clases especializadas en un solo tipo de compreso a la vez. Segundo, que es mucho más sencillo añadir nuevos tipos de compresos por medio de la herencia y definición del comportamiento del nuevo compreso.

4.4.3. Clases para la Gestión de la Base de Datos

Las clases que se ven en el siguiente gráfico son escenciales para la gestión de la información en la base de datos. La clase *JhaDBO* sigue el patrón Singleton y es utilizado en todas las llamadas a la base de datos, la clase JhaTable es la superclase para todos los DTOs (Data Transfer Object) que representan a las tablas de la base de datos.

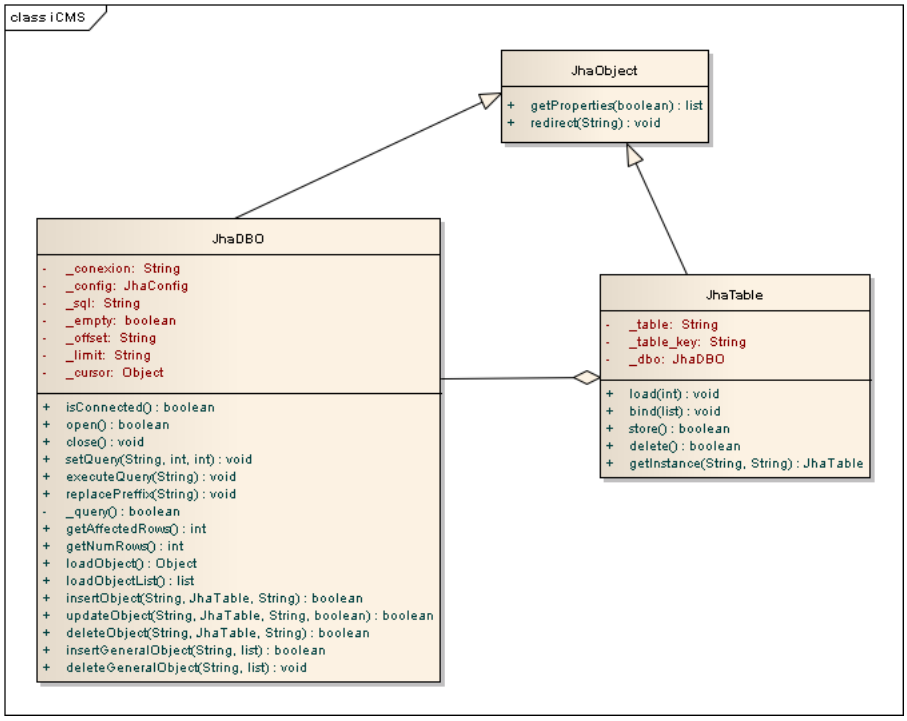

Figura 4.4: Modelo de clases: Gestión de Base de Datos. [Elaboración propia].

A continuación se describen brevemente las clases dedicadas a la gestión de base de datos.

Todo los objetos de *iCMS* tienen la capacidad de redireccionar a otras URLs, esto es bastante útil sobre todo en la sección de administración de *i*CMS, también cuentan con la capacidad de recuperar sus propiedades internas de los objetos.

```
1  <?php
2  defined( '_JHAEXEC' ) or die( 'Access Denied' );
3
4  class JhaObject{
5      public function getProperties( $isPublic = true ) {
6          $vars  = get_object_vars($this);
7          if($isPublic) {
```

```
            foreach ($vars as $key => $value) {
                if ('_' == substr($key, 0, 1)) {
                    unset($vars[$key]);
                }
            }
        }
        return $vars;
    }

    protected function redirect($url = 'index.php'){
        $config = JhaFactory::getConfig();
        if (preg_match( '#^index[2]?.php#', $url )) {
            $host = 'http://' . $_SERVER['HTTP_HOST'];
            $site = $config->site;
            $url = $host . $site . $url;
        }
        header( 'HTTP/1.1 301 Moved Permanently' );
        header( 'Location: ' . $url );
    }
}
?>
```

Código 15: Objeto Padre.

JhaDBO es un objeto creado con la única finalidad de procesar y ejecutar las consultas SQL nativas, adicionalmente contiene funciones auxiliares para realizar el trabajo de procesamiento de las consultas.

```
<?php
...

class JhaDBO extends JhaObject {
    //Conexion a la base de datos
    private $_conexion;

    //Configuracion del sitio
    private $_config;

    //Consulta SQL
    private $_sql;
    private $_empty;

    //Offset y limit, sirven para obtener resultados parciales.
    private $_offset;
    private $_limit;
    private $_cursor;

    function __construct($config = null){
        $this->_config = $config;
        if(!$this->open()){
            throw new Exception ( "Error al conectarse a la Base de Datos." );
        }
    }
    ...

    /**
     * Abre una conexion a la base de datos, esta conexion se deberia abrir una sola vez.
     */
    public function open(){
        ...
    }

  public function close() {
    ...
  }

  public function setQuery($query, $offset = 0, $limit = -1){
```

```
40      $this->_sql = $this->replacePreffix($query);
41      $this->_offset = $offset;
42      $this->_limit = $limit;
43    }
44
45    /**
46     * Establece la consulta SQL y luego la ejecuta
47     */
48    public function executeQuery($query){
49      ...
50    }
51
52    private function replacePreffix($query){
53      return str_replace("#__", $this->_config->db_preffix, $query);
54    }
55
56    /**
57     * Ejecuta la consulta SQL
58     */
59    private function _query(){
60      ...
61    }
62    ...
63
64    /**
65     * Tras la ejecucion de la consulta SQL, este metodo retorna el resultado como un Objeto
66     */
67    public function loadObject() {
68      ...
69    }
70
71    /**
72     * Tras la ejecucion de la consulta SQL, este metodo retorna el resultado como una lista de Objetos
73     */
74    public function loadObjectList( $key = '' ) {
75      ...
76    }
77
78    /**
79     * Inserta los datos del objeto en la base de datos.
80     */
81    public function insertObject( $table, &$object, $keyName = NULL ) {
82      ...
83    }
84
85    /**
86     * Actualiza la informacion de la base da datos con la informacion del objeto
87     */
88    public function updateObject( $table, &$object, $keyName, $updateNulls = true ) {
89      ...
90    }
91
92    /**
93     * Elimina el registro descrito por el objeto
94     */
95    public function deleteObject( $table, &$object, $keyName = NULL ) {
96      ...
97    }
98
99    public function insertid() {
100     return mysql_insert_id($this->_conexion);
101   }
102
103   /**
104    * Inserta registros en la base de datos con mas de un valor como clave primaria.
105    */
106   public function insertGeneralObject( $table, $elems) {
107     ...
```

```
108   }
109
110   /**
111    * Elimina registros de la base de datos que tienen mas de un valor como clave primaria.
112    */
113   public function deleteGeneralObject( $table, $elems) {
114     ...
115   }
116 }
117 ?>
```

Código 16: DBO (DataBase Object).

4.4.4. Clases para la Gestión de Archivos y Directorios

Las clases para la gestión de los archivos y directorios se muestran en el siguiente diagrama. La clase *JhaFile* sirve principalmente para la lectura y escritura de archivos, la clase *JhaDirectory* provee una estructura para gestionar los archivos y subdirectorios de un directorio.

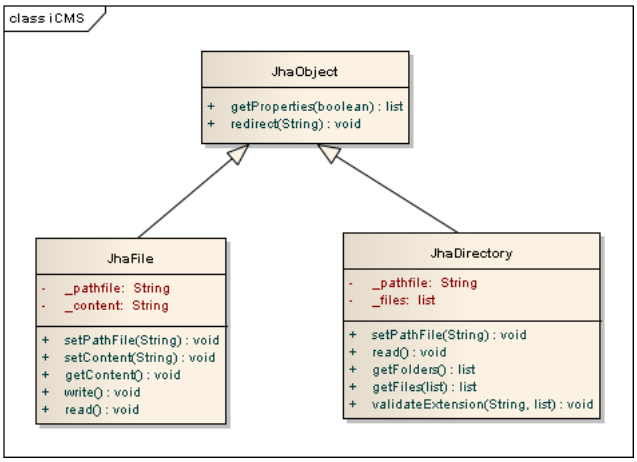

Figura 4.5: Modelo de clases: Gestión de Archivos y Directorios. [Elaboración propia].

A continuación se describen brevemente las clases dedicadas a la gestión de archivos y directorios.

La clase *JhaFile* permite interactuar de forma directa con los diferentes archivos a través de sus métodos *read* y *write*.

```
1  <?php
2  ...
3
4  class JhaFile extends JhaObject {
5      var $_pathfile;
6      var $_content;
7
8      public function __construct($pathfile = '', $content = ''){
9          ...
10     }
11
12     public function setPathFile($pathfile){
13         $this->_pathfile = $pathfile;
```

```
14      }
15
16      public function setContent($content){
17          $this->_content = $content;
18      }
19
20      public function getContent(){
21          return $this->_content;
22      }
23
24      /**
25       * Se encarga de escribir el archivo.
26       */
27      public function write(){
28          ...
29      }
30
31      /**
32       * Se encarga de leer el archivo.
33       */
34      public function read(){
35          ...
36      }
37  }
38  ?>
```

Código 17: JhaFile.

La clase *JhaDirectory* se combina con los objetos de tipo *JhaFile* para permitir interactuar tanto con los diferentes archivos como con los directorios a través de sus métodos *getFolders* y *getFiles*, ademas cuenta con una función extra para validar las extensiones de los archivos *validateExtension*.

```
1   <?php
2   ...
3
4   class JhaDirectory extends JhaObject {
5       var $_pathfile;
6       var $_files;
7
8       public function __construct($pathfile = '.'){
9           ...
10      }
11
12      public function setPathFile($pathfile){
13          $this->_pathfile = $pathfile;
14      }
15
16      public function read(){
17          $this->_files = scandir($this->_pathfile);
18      }
19
20      /**
21       * En funcion al filepath recibido, retorna todos los directorios existentes en el.
22       */
23      public function getFolders(){
24          ...
25      }
26
27      /**
28       * En funcion al filepath recibido, retorna todos los archivos existentes en el.
29       * No toma en cuenta los archivos de los directorios.
30       */
31      public function getFiles($filetypes = array()){
32          ...
```

```
33      }
34
35      /**
36       * Encargado de validar que todos los archivos del directorio tengan la misma extension
37       */
38      private function validateExtension($file, $filetypes){
39          ...
40      }
41  }
42  ?>
```

Código 18: JhaDirectory.

4.4.5. Clases para la Gestión de Elementos Web

Las clases para la gestión de los elementos web se muestran en el siguiente diagrama. Estas clases estan basadas en el Framework Goaamb (desarrollado por el Ingeniero de Sistemas Alvaro Michel Barrera).
Todas estas clases han sido pensadas principalmente para trabajar con generadores de código, pero en el proyecto iCMS se hace uso de ellas principalmente en el manejo de archivos XML y JS.

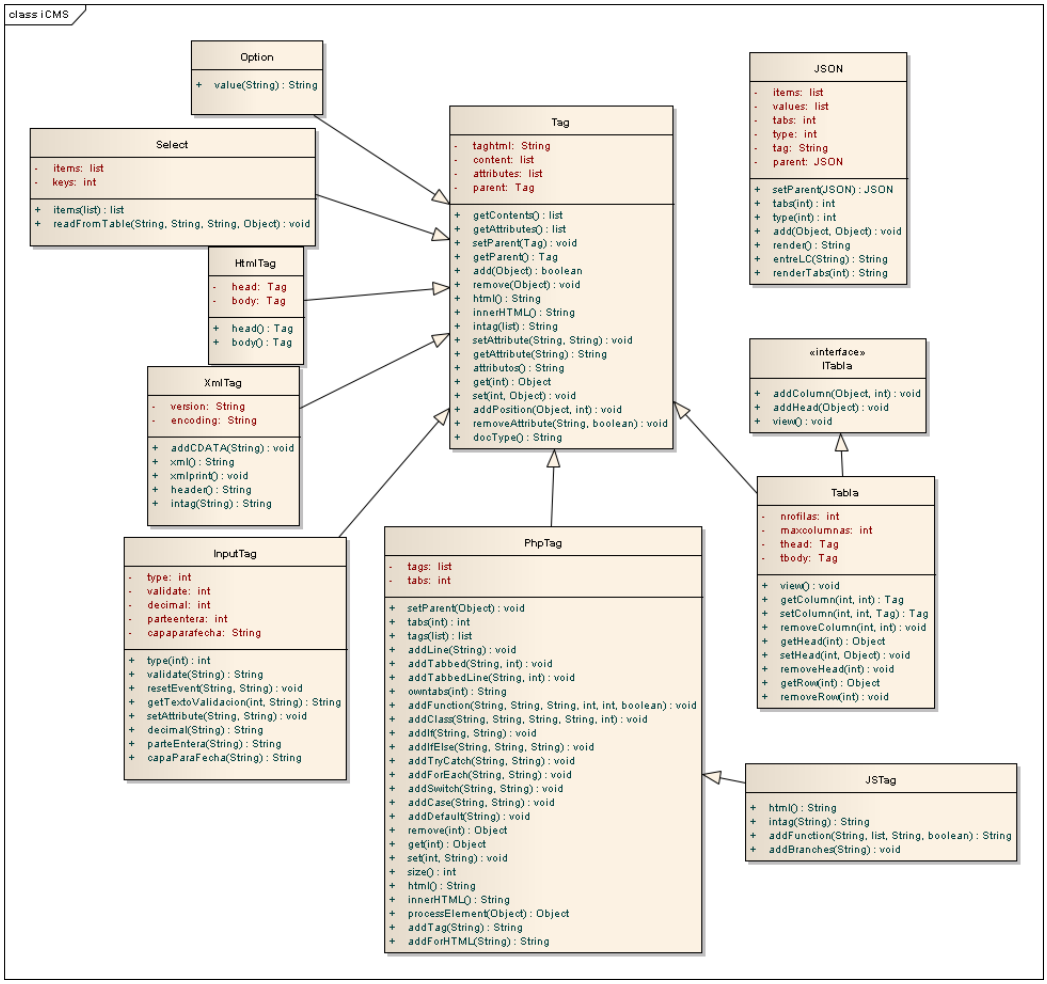

Figura 4.6: Modelo de clases: Gestión de Elementos Web. [Elaboración propia].

Como podemos ver en la figura anterior, la super clase *Tag* contienen la mayor parte de las funciones propias para los elementos HTML y PHP. Asimismo muchos de los elementos HTML están representados por las clases: *HtmlTag*, *InputTag*, *Select*, *Option* y *Tabla*. Por otro lado tenemos a la clase para el manejo de documentos XML: *XmlTag*. Para el manejo de códigos PHP, tenemos la clase *PhpTag*; dado que los elementos JavaScript tienen cierto parecido con los tags php, la clase JSTag extiende de esta última.
Los elementos JSON se comportan de una manera muy distinta a los elementos que tienen "tags", razón por lo cual la clase *JSON* no extiende de ninguna otra.

4.4.6. Clases de la Arquitectura del *i*CMS

Las clases que se muestran a continuación son las superclases que representan al patrón Modelo-Vista-Controlador. Las implementaciones de estas superclases estan presentes en las distintas extensiones (módulos).

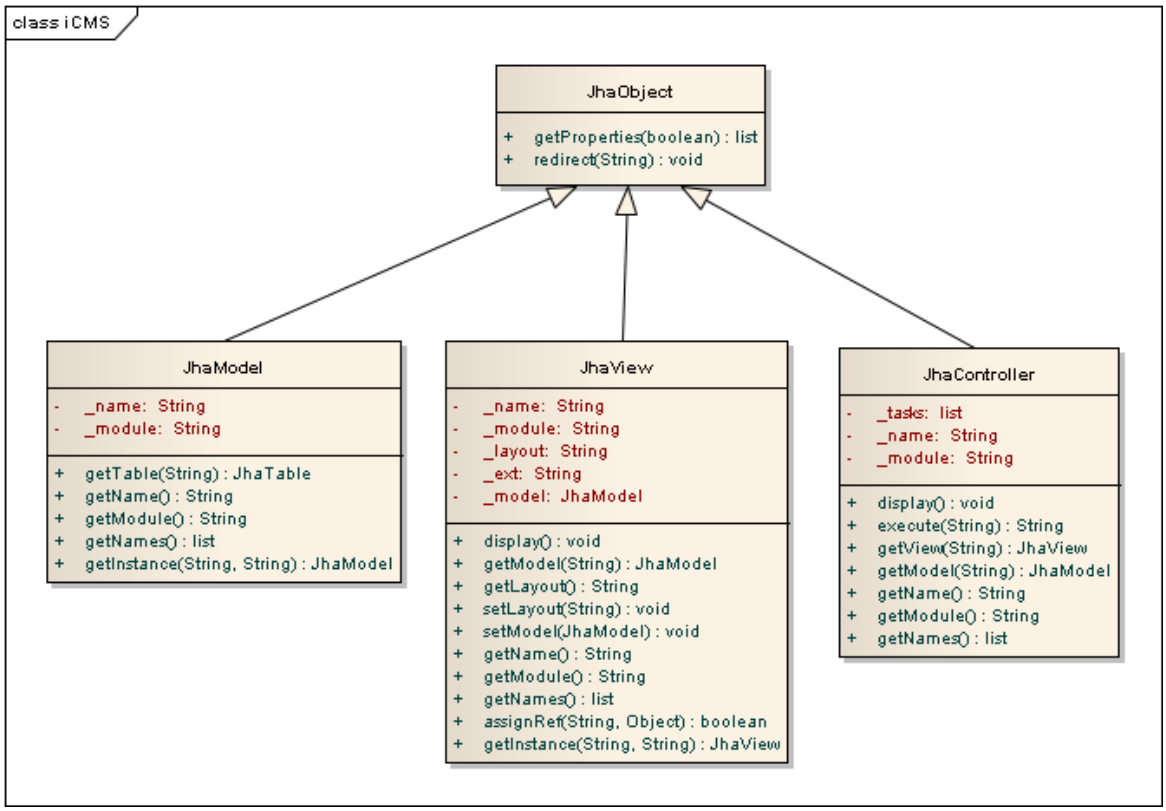

Figura 4.7: Modelo de clases: Modelo-Vista-Controlador. [Elaboración propia].

La clase *JhaModel* describe dos métodos importantes como *getTable* y *getInstance*; *getTable* se encarga de recuperar objetos de tipo *JhaTable* y es principalmente utilizada a la hora de recuperar registros de la base de datos. Por otro lado *getInstance* retorna una instancia del modelo en cuestión, es decir que depende del módulo desde el que se llame a este método.

```
1 <?php
2 ...
3
4 class JhaModel extends JhaObject {
```

```
5
6       protected $_name;
7       protected $_module;
8
9       public function __construct(){
10          if (empty( $this->_name ) || empty( $this->_module )) {
11              $this->_name = $this->getName();
12              $this->_module = $this->getModule();
13          }
14      }
15
16      public function &getTable($name = '') {
17          ...
18      }
19      ...
20
21      public function &getInstance($name, $prefix){
22          ...
23      }
24  }
25  ?>
```

Código 19: JhaModel.

La clase *JhaView* se encarga de renderizar las diferentes plantillas dependiendo del controlador y la acción relacionada a estos, en resumen por cada acción que ejecute el controlador, la vista puede renderizar diferentes pantallas, es por eso que cada vista tiene 'n' plantillas, a continuación se describen los métodos más importantes:

display Se encarga de mostrar plantilla por defecto para la vista.

getModel Retorna el modelo para el módulo al que pertenece la vista.

setLayout Esteblece la plantilla que se utilizara para renderizar el resultado de ejecutar la acción en el controlador.

assignRef Esteblece referencias a los valores recuperados desde el modelo, en síntesis añade nuevos atributos a la vista, de forma que la plantilla los pueda utilizar de forma casi transparente.

getInstance Devuelve una instancia de la vista, al igual que con el modelo, esta acción depende del contexto.

```
1   <?php
2   ...
3
4   class JhaView extends JhaObject {
5
6       protected $_name;
7       protected $_module;
8       protected $_layout;
9       protected $_ext;
10      protected $_model;
11
12      public function __construct(){
13          if (empty( $this->_name ) || empty( $this->_module )) {
14              $this->_name = $this->getName();
15              $this->_module = $this->getModule();
16          }
17          $this->_layout = $this->_name;
18          $this->_ext = '.php';
19      }
```

```
20
21      public function display(){
22          $file = $this->_layout . $this->_ext;
23          $content = '';
24          $urlFile = $GLOBALS['JHA_MODULE_PATH'].'views'.DS.'html'.DS.$file;
25          if(file_exists( $urlFile )){
26              ob_start();
27              require_once $urlFile;
28              $content = ob_get_contents();
29              ob_end_clean();
30              echo $content;
31          }
32          else{
33              throw new Exception("Vista no encontrada");
34          }
35      }
36
37      public function &getModel($model = ''){
38          $model = (empty($model) ? $this->getName() : $model);
39          $prefix = $this->_module . 'Model';
40          jhaimport('jhaley.mvc.model');
41          $this->model = JhaModel::getInstance($model, $prefix);
42          return $this->model;
43      }
44      ...
45
46      function assignRef($key, &$val) {
47          if (is_string($key) && substr($key, 0, 1) != '_') {
48              $this->$key =& $val;
49              return true;
50          }
51          return false;
52      }
53
54      public function &getInstance($name, $prefix){
55          $name = preg_replace('/[^A-Z0-9_\.-]/i', '', $name);
56          $prefix = preg_replace('/[^A-Z0-9_\.-]/i', '', $prefix);
57              $viewClass = $prefix.$name;
58              $result = false;
59              if (!class_exists( $viewClass )) {
60                  $path = $GLOBALS['JHA_MODULE_PATH'].'views'.DS.$name.'.php';
61                  if ($path) {
62                      require_once $path;
63                      if (!class_exists( $viewClass )) {
64                          throw new Exception( 'Vista ' . $viewClass . ' no encontrado.' );
65                          return $result;
66                      }
67                  }
68                  else return $result;
69              }
70              $result = new $viewClass();
71              return $result;
72      }
73  }
74  ?>
```

Código 20: JhaView.

La clase *JhaController* se encarga de ejecutar las acciones trás las cuales se delega la responsabilidad de renderizar el resultado a alguna de las vistas, a continuación se describen los métodos más importantes de esta clase:

display Se encarga de hacer que la vista muestre su plantilla por defecto.

execute Ejecuta el método correspondiente a la acción a ejecutarse, por defecto ejecuta la acción "display".

getView Retorna la vista asociada al controlador.

getModel Retorna el modelo para el módulo al que pertenece la vista.

```php
<?php
...

class JhaController extends JhaObject {

      var $_tasks;
      var $_name;
      var $_module;

    public function __construct(){
          ...
    }

    public function display(){
        $view = &$this->getView();
        $model = &$this->getModel();
        $view->setModel($model);
        $view->display();
    }

    public function execute($task){
        $task = strtolower( $task );
        if (isset($this->_tasks[$task])) {
            $jobToDo = $this->_tasks[$task];
        }
        elseif (isset($this->_tasks['display'])){
            $jobToDo = $this->_tasks['display'];
        }
        else{
            throw new Exception( 'Proceso asociado a la tarea ' . $task . ' no encontrada.' );
        }
        return $this->$jobToDo();
    }

    public function &getView($view = ''){
        $view = (empty($view) ? $this->getName() : $view);
        $prefix = $this->_module . 'View';
        jhaimport('jhaley.mvc.view');
        return JhaView::getInstance($view, $prefix);
    }

    public function &getModel($model = ''){
        $model = (empty($model) ? $this->getName() : $model);
        $prefix = $this->_module . 'Model';
        jhaimport('jhaley.mvc.model');
        return JhaModel::getInstance($model, $prefix);
    }
    ...
}
?>
```

Código 21: JhaController.

4.4.7. Clases para la Gestión de Renderizadores

Las clases que se muestran en el siguiente diagrama, se utilizan para renderizar algún tipo de contenido en particular, se tienen rederizadores para los bloques, los módulos, el menú de administración, las plantillas, etc.
Si bien todos los renderizadores tienen el método "render" cada renderizador tiene un comportamiento totalmente diferente al de otro.

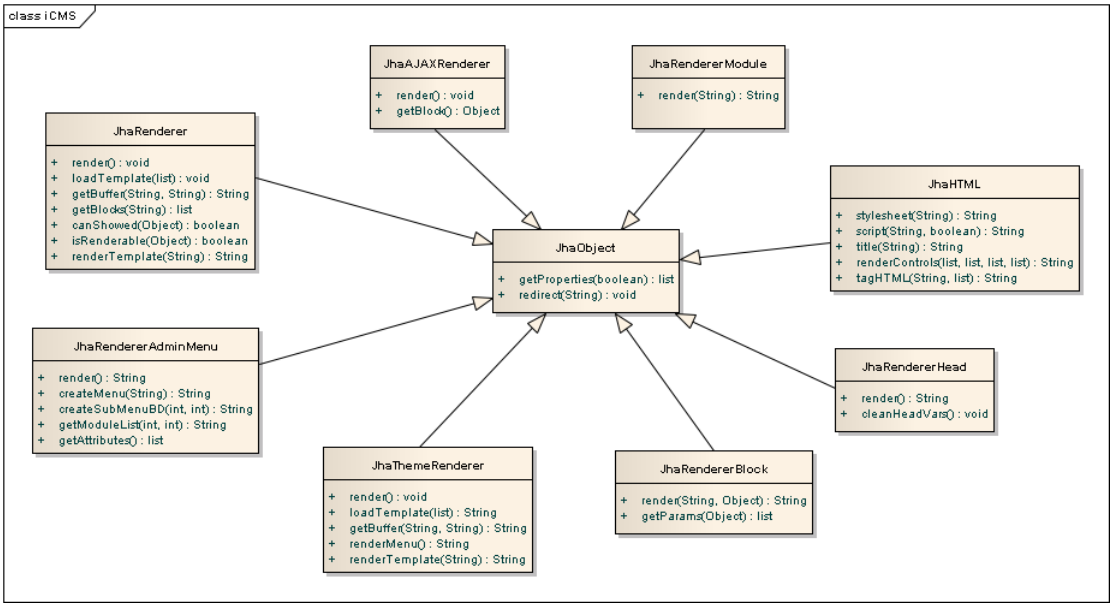

Figura 4.8: Modelo de clases: Renderizadores. [Elaboración propia].

A continuación vemos en detalle cada uno de los renderizadores.

El renderizador *JhaRenderer* hereda de la clase *JhaObject*, este se encarga de renderizar la plantilla entera, es decir que de alguna manera este hace uso de los demás renderizadores.
Los siguientes métodos son parte de este renderizador:

render Básicamente este método es el punto de partida para la renderización.

loadTemplate Retorna la plantilla como si fuera una cadena para hacerla más manejablea nivel de código.

renderTemplate Recibe como parametro el resultado de "loadTemplate", en base a este parametro agrega contenidos que son los resultados de los otros renderizadores.

getBuffer Este método actúa como discriminador, dependiendo del tipo de contenido que debe mostrarse llama a uno u otro renderizador. En el caso de los bloques hace una comprobación adicional que le permite saber si un bloque se debe mostrar o no a través de los métodos "canShowed" e "isRenderable".

getBlocks Es un método auxiliar para "getBuffer".

```
1 <?php
2 ...
```

```
3
4  class JhaRenderer extends JhaObject {
5      public function render(){
6          $db = &JhaFactory::getDBO();
7          $db->setQuery('SELECT * FROM #__plantilla WHERE predeterminado = 1');
8          $row = $db->loadObject();
9
10         $template = $this->loadTemplate($row);
11         echo $this->renderTemplate(str_replace('<jhadoc:include type="maincontent" />', $row->html,
               $template));
12     }
13
14     protected function loadTemplate($row){
15         $template = '';
16         ob_start();
17         require_once JHA_THEMES_PATH.DS.($row ? $row->nombre : 'default').DS.'index.php';
18         $template = ob_get_contents();
19         ob_end_clean();
20         return $template;
21     }
22
23     protected function getBuffer($type = null, $region = null){
24         $content = '';
25         if($type == null){
26             return;
27         }
28         $renderer = &JhaFactory::getRenderer($type);
29         if($type == 'module'){
30             $GLOBALS['JHA_MODULE_PATH'] =
                   JHA_BASE_PATH.DS.'modules'.DS.JhaRequest::getVar('elem','mod_content').DS;
31             $path = $GLOBALS['JHA_MODULE_PATH'].substr(JhaRequest::getVar('elem', 'mod_content'),4).'.php';
32             $content = $renderer->render($path).$content;
33         }
34         elseif ($type == 'block' && $region != null){
35             $blocks = $this->getBlocks($region);
36             if($blocks) {
37                 if(JhaUtility::userCanEdit()){
38                     $content .= '<div class="jhablock"><div class="jhablock-header">Region: ' .
                           ucfirst($region) . '</div>';
39                     $script = '';
40                 }
41                 foreach($blocks as $block) {
42                     if($this->isRenderable($block) && $this->canShowed($block)){
43                         $GLOBALS['JHA_BLOCK_PATH'] = JHA_BASE_PATH.DS.'blocks'.DS.$block->renderizador.DS;
44                         $path = $GLOBALS['JHA_BLOCK_PATH'] . substr($block->renderizador,6) . '.php';
45                         $content .= $renderer->render($path, $block);
46                         if(JhaUtility::userCanEdit()){
47                             $script .= "dragBlock.addTarget(Jha.dom.$('block" . $block->id . "'));\n";
48                         }
49                     }
50                 }
51                 if(JhaUtility::userCanEdit()){
52                     //aumentar final de bloque y tb. la opcion de Add more blocks
53                     //la opcion se abrira en un iframe.
54                     $content .= JhaHTML::script($script, false);
55                     //agregar el enlace para agregar nuevos bloques.
56                     $content .= (JhaUtility::userCanAdministrate() ? '<div><a
                           href="index.php?elem=mod_base&controller=block&task=newblock&region=' . $region .
                           '"><img src="images/new.jpg" title="Agregar nuevo bloque" /></a></div>' : '');
57                     $content .= '</div>';
58                 }
59             }
60             else {
61                 if(JhaUtility::userCanEdit()){
62                     $content .= '<div class="jhablock"><div class="jhablock-header">Region: ' .
                           ucfirst($region) . '</div>' . (JhaUtility::userCanAdministrate() ? '<div><a
                           href="index.php?elem=mod_base&controller=block&task=newblock&region=' . $region .
                           '"><img src="images/new.jpg" title="Agregar nuevo bloque" /></a></div>' : '') .
```

```
                    '</div>';
63              }
64          }
65      }
66      elseif ($type == 'head' || ($type == 'admin-menu' && JhaUtility::userCanAdministrate())){
67          $content = $renderer->render().$content;
68      }
69      return $content;
70  }
71
72  protected function getBlocks($region){
73      $db = &JhaFactory::getDBO();
74      $db->setQuery("SELECT * FROM #__bloque WHERE region = '" . $region . "' ORDER BY orden ASC");
75      return $db->loadObjectList();
76  }
77
78  protected function canShowed($block){
79      if(JhaUtility::userCanEdit()){
80          return true;
81      }
82      return 0 == intval($block->needlogin);
83  }
84
85  protected function isRenderable($block){
86      $db = &JhaFactory::getDBO();
87      $itemid = JhaRequest::getVar('itemid',0);
88      if($itemid == 0){
89          $db->setQuery('SELECT * FROM #__menuitem WHERE home = 1');
90          $row = $db->loadObject();
91          $itemid = $row->id;
92      }
93      $db->setQuery("SELECT * FROM #__menubloque WHERE (idmenu = '" . $itemid . "' OR idmenu = '0' ) AND
            idbloque = '" . $block->id . "'");
94      $row = $db->loadObject();
95      if($row)
96          return true;
97      return false;
98  }
99
100 protected function renderTemplate($template) {
101     if(JhaUtility::userCanEdit()){
102         $GLOBALS['JHA_HEAD_VARS'][] = JhaHTML::script("dragBlock = new
                Jha.drag();\ndragBlock.setType('block');\ndragBlock.ajaxPost = function (objajax, idSource,
                idTarget, isBeforeTarget) { objajax.json = true; res = { elem : 'mod_base', controller :
                'block', src : idSource, trg : idTarget, json : objajax.json, before : isBeforeTarget, task
                : 'reorder'}; return res; };", false);
103     }
104     $replace = array();
105     $matches = array();
106     if(preg_match_all('#<jhadoc:include\ type="([^"]+)" (.*)\/>#iU', $template, $matches)) {
107         $matches[0] = array_reverse($matches[0]);
108         $matches[1] = array_reverse($matches[1]);
109         $matches[2] = array_reverse($matches[2]);
110         $count = count($matches[1]);
111         for($i = 0; $i < $count; $i++) {
112             $attribs = JhaUtility::parseAttributes( $matches[2][$i] );
113             $replace[$i] = $this->getBuffer($matches[1][$i], (isset($attribs['region']) ?
                    $attribs['region'] : null));
114         }
115         $template = str_replace($matches[0], $replace, $template);
116     }
117     return $template;
118 }
119 }
120 ?>
```

Código 22: Renderizador JhaRenderer.

El renderizador *JhaRendererModule* también hereda de la clase *JhaObject*, este se encarga de renderizar los módulos unicamente.
Este renderizador solamente cuenta con el método render ya que no necesita hacer ning'un tipo de comprobación, de esto se encargan los propios módulos.

```php
<?php
...

class JhaRendererModule extends JhaObject {
    public function render($path){
        $content = '';
        if(file_exists( $path )){
            ob_start();
            require $path;
            $content = ob_get_contents();
            ob_end_clean();
        }
        return $content;
    }
}
?>
```

Código 23: Renderizador de modulos.

El renderizador *JhaRendererBlock* al igual que los anteriores, también hereda de la clase *JhaObject*, este se encarga de renderizar unicamente los bloques, algo que cabe destacar es que cada bloque tienen un conjunto de configuraciones para lo cual cuenta con un métdo auxiliar para este menester.
A continuación vemos una descripción de los métodos que son parte de este renderizador:

render En escencia se encarga de devolver el resultado de renderizar algún bloque.

getParams Retorna los parametros para el bloque, esta información se la obtiene de la base de datos juntamente con el contenido del bloque.

```php
<?php
...

class JhaRendererBlock extends JhaObject {
    public function render($path, $block){
        $content = '';
        $params = $this->getParams($block);
        if(file_exists( $path )){
            ob_start();
            require $path;
            if(JhaUtility::userCanEdit()){
                $content .= '<div class="block blockelement' . $params['suffix'] . '" id="block' .
                    $block->id . '"><div class="block-header"><div align="left" style="float:left;"><a
                    href="index.php?elem=mod_base&controller=block&task=editblock&id=' . $block->id . '"
                    title="Editar"><img src="images/edit.jpg" /></a></div><div align="left"
                    onmousedown="javascript:dragBlock.onDragStart(event, Jha.dom.$(\'block' . $block->id .
                    '\'));" style="line-height: 2;">' . $block->titulo . '</div></div>' . ob_get_contents()
                    . '</div>';
            }
            else {
                $content = '<div class="blockelement' . $params['suffix'] . '">' . ob_get_contents() .
                    '</div>';
            }
            ob_end_clean();
        }
        return $content;
    }

```

```
22      public function getParams($block){
23          $params = split("\n",$block->params);
24          $res = array();
25          foreach ($params as $param){
26              $parts = split('=', $param);
27              $res[$parts[0]] = $parts[1];
28          }
29          return $res;
30      }
31  }
32  ?>
```

Código 24: Renderizador de bloques.

El renderizador *JhaRendererHead* se encarga de renderizar unicamente elementos que van en el tag HTML "head".
A continuación vemos una descripción de los métodos que son parte de este renderizador:

render Renderiza los las hojas de estilo (CSS) y los scripts (JS) dependiendo del tipo de sesión.

cleanHeadVars Esto limpia los elementos duplicados, además crea un orden entre los elementos que son parte del tag "tag".

```
1   <?php
2   ...
3
4   class JhaRendererHead extends JhaObject {
5       public function render(){
6           $content = '';
7           $GLOBALS['JHA_HEAD_VARS'] = (isset($GLOBALS['JHA_HEAD_VARS'][0]) ? $GLOBALS['JHA_HEAD_VARS'] :
                array());
8           if(JhaUtility::userCanEdit()){
9               $GLOBALS['JHA_HEAD_VARS'] = array_reverse($GLOBALS['JHA_HEAD_VARS']);
10              $GLOBALS['JHA_HEAD_VARS'][] = JhaHTML::script('libraries/jhaley/js/menu.js');
11              $GLOBALS['JHA_HEAD_VARS'][] = JhaHTML::script('libraries/jhaley/js/popup.js');
12              $GLOBALS['JHA_HEAD_VARS'][] = JhaHTML::script('libraries/jhaley/js/pmenu.js');
13              $GLOBALS['JHA_HEAD_VARS'][] = JhaHTML::script('libraries/jhaley/js/effect.js');
14              $GLOBALS['JHA_HEAD_VARS'][] = JhaHTML::script('libraries/jhaley/js/drag.js');
15              $GLOBALS['JHA_HEAD_VARS'][] = JhaHTML::script('libraries/jhaley/js/ajax.js');
16              $GLOBALS['JHA_HEAD_VARS'][] = JhaHTML::script('libraries/jhaley/js/jha.js');
17              $GLOBALS['JHA_HEAD_VARS'][] = JhaHTML::stylesheet('libraries/jhaley/css/menu.css');
18              $GLOBALS['JHA_HEAD_VARS'][] = JhaHTML::stylesheet('libraries/jhaley/css/base.css');
19              $GLOBALS['JHA_HEAD_VARS'] = array_reverse($GLOBALS['JHA_HEAD_VARS']);
20          }
21          else{
22              $GLOBALS['JHA_HEAD_VARS'] = array_reverse($GLOBALS['JHA_HEAD_VARS']);
23              $GLOBALS['JHA_HEAD_VARS'][] = JhaHTML::script('libraries/jhaley/js/effect.js');
24              $GLOBALS['JHA_HEAD_VARS'][] = JhaHTML::script('libraries/jhaley/js/jha.js');
25              $GLOBALS['JHA_HEAD_VARS'] = array_reverse($GLOBALS['JHA_HEAD_VARS']);
26          }
27          if(isset($GLOBALS['JHA_HEAD_VARS']) && count($GLOBALS['JHA_HEAD_VARS']) > 0){
28              $this->cleanHeadVars();
29              $content .= implode('',$GLOBALS['JHA_HEAD_VARS']);
30          }
31          return $content;
32      }
33
34      protected function cleanHeadVars(){
35          $array = array();
36          foreach ($GLOBALS['JHA_HEAD_VARS'] as $headVar){
37              if (JhaUtility::inArray($headVar, $array) == -1) {
38                  $array[] = $headVar;
39              }
40          }
41          $GLOBALS['JHA_HEAD_VARS'] = $array;
```

```
42      }
43  }
44  ?>
```

Código 25: Renderizador para el tag HTML 'head'.

El renderizador *JhaAJAXRenderer* se encarga de renderizar unicamente bloques y/o módulos. Este renderizador no sigue el flujo normal que es pasar por la clase *JhaRenderer*, sino que como su nombre indica esta diseñado para las peticiones asíncronas.
Los métodos que son parte de este renderizador son:

render Renderiza bloques y módulos dependiendo de la petición AJAX.

getBlock Recupera los datos del bloque, si es que la petición AJAX está relacionada con los bloques.

```
1   <?php
2   ...
3
4   class JhaAJAXRenderer extends JhaObject {
5       public function render(){
6           $content = '';
7           $elem = JhaRequest::getVar('elem');
8           if(substr($elem, 0, 3) == 'mod'){
9               $renderer = &JhaFactory::getRenderer('module');
10              $GLOBALS['JHA_MODULE_PATH'] = JHA_BASE_PATH.DS.'modules'.DS.$elem.DS;
11              $path = $GLOBALS['JHA_MODULE_PATH'].substr($elem, 4).'.php';
12              $content = $renderer->render($path).$content;
13          }
14          else {
15              $block = $this->getBlock();
16              $renderer = &JhaFactory::getRenderer('block');
17              $GLOBALS['JHA_BLOCK_PATH'] = JHA_BASE_PATH.DS.'blocks'.DS.$block->renderizador.DS;
18              $path = $GLOBALS['JHA_BLOCK_PATH'] . substr($block->renderizador,6) . '.php';
19              $content = $renderer->render($path, $block);
20          }
21          echo $content;
22      }
23
24      protected function getBlock(){
25          $db = &JhaFactory::getDBO();
26          $db->setQuery("SELECT * FROM #__bloque WHERE id = '" . JhaRequest::getVar('id') . "'");
27          return $db->loadObject();
28      }
29  }
30  ?>
```

Código 26: Renderizador las llamadas asíncronas.

El renderizador *JhaRendererAdminMenu* al igual que los anteriores, también hereda de la clase *JhaObject*, este renderizador está destinado unicamente a renderizar el menu de administración.
Los métodos que son parte de este renderizador son:

render Renderiza los menus y sub menus de administración, el resto de métodos son auxiliares para renderizar fragmentos del menu: *createMenu*, *createSubMenuBD*, *getModuleList* y *getAttributes*.

```
1   <?php
2   ...
```

```
3
4  class JhaRendererAdminMenu extends JhaObject {
5      public function render(){
6          $content = '';
7          $user = (isset($_SESSION['USER']) ? $_SESSION['USER'] : NULL);
8          $canEdit = $user->rol == 'Super Administrador' || $user->rol == 'Editor';
9          if($canEdit){
10             $xml = simplexml_load_file(JHA_LIBRARIES_PATH.DS.'jhaley'.DS.'xml'.DS.'adminmenu.xml');
11             $content .= '<div id="jha-admin-menu"><div style="width: 950px; margin: 0pt auto; height: 25px;
                   background-color: black;">' . $this->createMenu($xml) . '</div></div><br />';
12             $GLOBALS['JHA_HEAD_VARS'][] = JhaHTML::script('document.menu = null;
13             window.onload = function(){
14                 element = Jha.dom.$(\'jhamenu\');
15                 var menu = new Jha.menu(element);
16                 document.menu = menu;
17             };', false);
18         }
19         return $content;
20     }
21
22     protected function createMenu($xml){
23         $content = '<ul' . $this->getAttributes($xml->attributes()) . '>';
24         foreach ($xml->elements->element as $element) {
25             $att = $element->attributes();
26             $content .= '<li><a' . $this->getAttributes($att) . '>' . $element->text . '</a>';
27             if(!isset($element->elements) && $att['type'] == 'bd'){
28                 $content .= $this->createSubMenuBD(0);
29             }
30             elseif(isset($element->elements)){
31                 $content .= $this->createMenu($element);
32             }
33             $content .= '</li>';
34         }
35         return $content . '</ul>';
36     }
37
38     protected function createSubMenuBD($level, $id = NULL){
39         $content = '';
40         $links = $this->getModuleList($level, $id);
41         if(count($links) > 0){
42             $content .= '<ul>';
43             foreach ($links as $link) {
44                 $href = ($link->link != '' ? ' href="' . $link->link . '&itemid=' . $_SESSION["itemid"] .
                       '"' : '');
45                 $content .= '<li><a' . $href . '>' . $link->nombre . '</a>';
46                 $content .= $this->createSubMenuBD($level + 1, $link->id) . '</li>';
47             }
48             $content .= '</ul>';
49         }
50         return $content;
51     }
52
53     protected function getModuleList($level, $id = NULL){
54         $db = &JhaFactory::getDBO();
55         $db->setQuery('SELECT * FROM #__modules WHERE parent = ' . ($level == 0 ? '0' : $id));
56         return $db->loadObjectList();
57     }
58
59     protected function getAttributes($attributes){
60         $attr = '';
61         if(count($attributes) > 0){
62             foreach ($attributes as $index => $value){
63                 if($index == 'href'){
64                     $attr .= ' ' . $index . '="' . $value . '&itemid=' . $_SESSION["itemid"] . '"';
65                 }
66                 elseif($index != 'type'){
67                     $attr .= ' ' . $index . '="' . $value . '"';
68                 }
```

```
69              }
70          }
71          return $attr;
72      }
73  }
74  ?>
```

Código 27: Renderizador para el menú de administración.

El renderizador *JhaThemeRenderer* también hereda de la clase *JhaObject*, este renderizador está destinado unicamente a renderizar la plantilla en modo de personalización, esto significa que podremos personalizarla desde la interfaz gráfica.
Los métodos que son parte de este renderizador son:

render Renderiza la plantilla de forma muy similar a *JhaRenderer* con la diferencia de que no renderiza el contenido sino que inserta en su lugar elementos para su manipulación desde la interfaz gráfica, es decir para personalizar la plantilla.

```
1   <?php
2   ...
3
4   class JhaThemeRenderer extends JhaObject {
5       public function render(){
6           $db = &JhaFactory::getDBO();
7           $db->setQuery('SELECT * FROM #__plantilla WHERE predeterminado = 1');
8           $row = $db->loadObject();
9
10          $template = $this->loadTemplate($row);
11          $_SESSION['themeHTML'] = isset($_SESSION['themeHTML']) ? $_SESSION['themeHTML'] : $row->html;
12          $_SESSION['themeXML'] = isset($_SESSION['themeXML']) ? $_SESSION['themeXML'] : $row->xml;
13          echo $this->renderTemplate($template);
14      }
15
16      protected function loadTemplate($row){
17          $template = '';
18          ob_start();
19          require_once JHA_THEMES_PATH.DS.($row ? $row->nombre : 'default').DS.'index.php';
20          $template = ob_get_contents();
21          ob_end_clean();
22          return $template;
23      }
24
25      protected function getBuffer($type = null, $region = null){
26          $content = '';
27          if($type == null){
28              return;
29          }
30          $renderer = &JhaFactory::getRenderer($type == 'maincontent' ? 'module' : $type);
31          if($type == 'maincontent') {
32              $GLOBALS['JHA_MODULE_PATH'] =
                    JHA_BASE_PATH.DS.'modules'.DS.JhaRequest::getVar('elem','mod_content').DS;
33              $path = $GLOBALS['JHA_MODULE_PATH'].substr(JhaRequest::getVar('elem', 'mod_content'),4).'.php';
34              $content = $renderer->render($path).$content;
35          }
36          elseif ($type == 'head'){
37              $content = $renderer->render().$content;
38          }
39          elseif ($type == 'admin-menu'){
40              $content = $this->renderMenu().$content;
41          }
42          return $content;
43      }
44
45      private function renderMenu(){
```

```
46          $content = '';
47          if(JhaUtility::userCanEdit()) {
48              $content .= '<div id="jha-admin-menu"><div style="width: 950px; margin: 0pt auto; height: 25px;
                    background-color: black;"><ul id="jhamenu"><li><a
                    href="index.php?elem=mod_base&controller=theme&task=savePersonalizedChanges">Guardar
                    Cambios</a></li><li><a
                    href="index.php?elem=mod_base&controller=theme&task=cancelPersonalizedChanges">Cancelar</a></li></ul></div></
                    />';
49              $GLOBALS['JHA_HEAD_VARS'][] = JhaHTML::script('document.menu = null;
50              window.onload = function(){
51                  element = Jha.dom.$(\'jhamenu\');
52                  var menu = new Jha.menu(element);
53                  document.menu = menu;
54              };', false);
55          }
56          return $content;
57      }
58
59      protected function renderTemplate($template) {
60          $replace = array();
61          $matches = array();
62          if(preg_match_all('#<jhadoc:include\ type="([^"]+)" (.*)\/>#iU', $template, $matches)) {
63              $matches[0] = array_reverse($matches[0]);
64              $matches[1] = array_reverse($matches[1]);
65              $count = count($matches[1]);
66              for($i = 0; $i < $count; $i++) {
67                  $replace[$i] = $this->getBuffer($matches[1][$i]);
68              }
69              $template = str_replace($matches[0], $replace, $template);
70          }
71          return $template;
72      }
73  }
74  ?>
```

Código 28: Renderizador para la personalizacón de plantillas.

La clase *JhaHTML* no es precisamente un renderizador de contenido especializado como el de los módulos o bloques, pero en su lugar es capaz de generar contenido HTML simple como tags 'link', 'script' y 'title'.
Los métodos que son parte de este renderizador son:

stylesheet Renderiza un tag 'link' con alguna hoja de estilos.

script Renderiza un tag 'script' con algún contenido javascript ya sea contenido en linea o desde un archivo.

title Renderiza un tag 'title' con el titulo de la página.

tagHTML Renderiza un tag HTML arbitrario.

renderControls Renderiza el conjunto de controles de administración de contenidos, es decir los controles de Nuevo, Editar, Eliminar, Guardar, Cancelar. En el panel de administración estos controles son muy comunes, así que este método facilita mucho su creación

```
1   <?php
2   ...
3
4   class JhaHTML extends JhaObject {
5
6       public static function stylesheet($url){
7           jhaimport('jhaley.web.tags');
```

```
8           $taghtml = new Tag ('link');
9           $taghtml->setAttribute('type', 'text/css');
10          $taghtml->setAttribute('rel', 'stylesheet');
11          $taghtml->setAttribute('href', $url);
12          return $taghtml->html ();
13      }
14
15      public static function script($src, $isUrl = true){
16          jhaimport('jhaley.web.jstag');
17          $tagjs = new JSTag ();
18          if($isUrl){
19              $tagjs->setAttribute('type', 'text/javascript');
20              $tagjs->setAttribute('src', $src);
21          }
22          else{
23              $tagjs->add($src);
24          }
25          return $tagjs->html ();
26      }
27
28      public function title($title = ''){
29          jhaimport('jhaley.web.tags');
30          $taghtml = new Tag ('title', $title);
31          return $taghtml->html ();
32      }
33
34      //array-> titulos, tasks, linktype**, icons.
35      // ** -> Para la validacion de los checkboxes.
36      public function renderControls($titles, $tasks, $linktypes, $icons){
37          if(count($titles) <= 0 && count($tasks) <= 0 && count($linktypes) <= 0) return '';
38          jhaimport('jhaley.web.tags');
39          $tagtable = new Tag ('table');
40          $tagtable -> add( $tagtr = new Tag('tr') );
41          for($i = 0; $i < count($titles); $i++){
42              $mustValidate = $linktypes[$i] == 'edit' || $linktypes[$i] == 'delete' || $linktypes[$i] ==
                    'save';
43              $onclick = '';
44              if($mustValidate) {
45                  $condicion = ($linktypes[$i] == 'edit' || $linktypes[$i] == 'delete' ?
                        'Jha.html.checkbox.validate()' : 'validateForm()');
46                  $onclick = 'if(' . $condicion . '){Jha.dom.$(\'task\').value = \'' . $tasks[$i] . '\';
                        document.forms.adminForm.submit();}';
47              }
48              else {
49                  $onclick = 'Jha.dom.$(\'task\').value = \'' . $tasks[$i] . '\';
                        document.forms.adminForm.submit();';
50              }
51              if($icons != null && $icons[$i] != null){
52                  $tagicon = new Tag('img');
53                  $tagicon->setAttribute('src', $icons[$i]);
54                  $tagicon->setAttribute('title', $titles[$i]);
55              }
56              else{
57                  $tagicon = new Tag('span');
58                  $tagicon -> setAttribute('class', 'icon-' . $linktypes[$i]);
59                  $tagicon -> setAttribute('title', $titles[$i]);
60              }
61              $tagtr -> add( $tagtd = new Tag('td') );
62              $tagtd -> add( $taga = new Tag('a') );
63              $taga -> setAttribute('href', 'javascript:;');
64              $taga -> setAttribute('onclick', 'javascript:' . $onclick);
65              $taga -> add( $tagicon );
66              $taga -> add( $titles[$i] );
67          }
68          return $tagtable->html ();
69      }
70
71      public function tagHTML($tag = 'p', $extras = array()) {
```

```
72          jhaimport('jhaley.web.tags');
73          $taginput = new Tag ('input');
74          foreach ($extras as $index => $value) {
75              $taginput->setAttribute($index, $value);
76          }
77          return $taginput->html ();
78      }
79  }
80  ?>
```

Código 29: Renderizador para tags HTML.

4.5. Modelo de Base de Datos

El modelo de base de datos o modelo Entidad-Relación, contiene tablas que son propias del motor de renderizado de plantillas, también están presentes algunas tablas que son parte de las extensiones (contenido, menu). Escencialmente son extensiones, pero a la vez son una parte fundamental del *i*CMS.

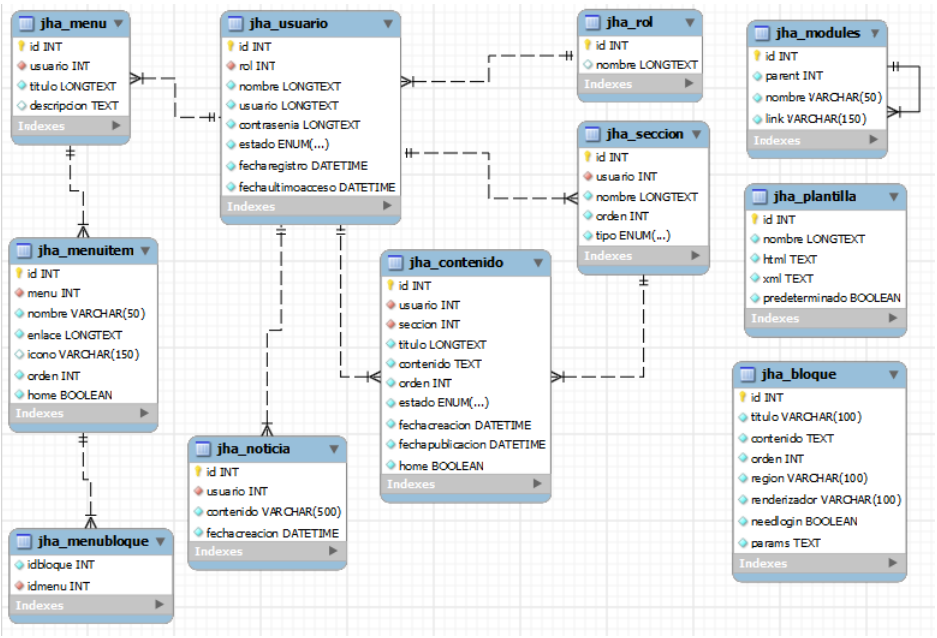

Figura 4.9: Modelo de Base de Datos. [Elaboración propia].

Capítulo 5

MÓDULOS, BLOQUES Y TEMAS

5.1. Introducción

Se presenta una descripción de las distintas extensiones que forman parte del *i*CMS, de forma general las extensiones estan compuestas por módulos, bloques y temas, veremos en detalle la estructura tanto a nivel de clases como a nivel de directorios. Al estar trabajando con un lenguaje de PostScripting como PHP, muchos de los archivos no aparecen en el modelo de clases, justamente por que no son clases como tal.

5.2. Estructura de los módulos

Todos los módulos siguen el patrón Modelo-Vista-Controlador, además maneja un archivo el cual se convierte en punto de entrada para el modulo, la finalidad de este archivo es para ejecutar el método correcto del controlador dependiendo de la acción requerida a través de los parametros del REQUEST.

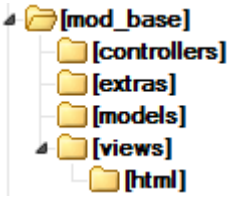

Figura 5.1: Estructura de Directorios: Módulos. [Elaboración propia].

En el figura anterior el nombre del módulo que utilizamos como ejemplo es “base”, esta estructura de directorios es exactamente igual para todos los módulos con la salvedad del nombre. Dentro del directorio “mod_base” tenemos una script php que sirve como punto de entrada al módulo, además podemos ver que la unica finalidad de esta porción de código es recuperar una acción y ejecutarla.

```
1  <?php
2  ...
3
4  $control = JhaRequest::getVar('controller', 'base');
5  require_once($GLOBALS['JHA_MODULE_PATH'].'controllers'.DS.$control.'.php' );
6
7  $classname = 'BaseController' . $control;
8  $controller = new $classname();
9
10 $controller->execute(JhaRequest::getVar('task'));
11 ?>
```

Código 30: Punto de entrada al módulo.

Posteriormente tenemos al directorio “extras”, el cual es utilizado generalmente para archivos auxiliares, tales como: Hojas de estilo y Javascript.

El directorio “controllers” se encarga de contener los controladores, por defecto siempre debe existir un archivo con el mismo nombre del módulo, es decir un archivo llamado “base.php”. Además los controladores siguen un patrón para definir el nombre de la clase, este patrón es: «nombre_modulo»Controller«nombre_controlador», en el caso del controlador por defecto tenemos: “BaseControllerBase”, si existiera un segundo controlador, por ejemplo “block” se tendría:

“BaseControllerBlock”.

El directorio “models” contiene todos los modelos, uno por cada controlador, la nomenclatura de sus nombres es muy similar a la de los controladores: «nombre_modulo¿Model¡nombre_modelo», siendo «nombre_modelo» exactamente el mismo que «nombre_controlador», entonces para el controlador “base” tenemos “BaseModelBase” y para “block” “BaseModelBlock” respectivamente.

El directorio “views” esta destinado a contener las vistas, igual que en el caso anterior una vista por cada controlador, le nomenclatura para los nombres es exactamente la misma que para los controladores y modelos, con la salvedad de que en vez de “Controller” y “Model” se utiliza “View”, es decir: “BaseViewBase” y “BaseViewBlock” respectivamente.

Algo que es necesario mencionar es que una vista esta asociada a muchas plantillas (Plantilla no es lo mismo que Tema) para poder mostrar la información bajo un formato determinado para cada caso, Paraesto utiliza el siguiente directorio.

El directorio “html” se encarga de contener las plantillas HTML, del cual se sirven las vistas para dar formato a la información, estos archivos contienen en su mayoria (por no decir en su totalidad) contenido HTML con fragmentos de código PHP unicamente.

5.3. Estructura de los bloques

Los bloques esencialmente son porciones de contenido fijo en algún lugar de la página, lostemas definen regiones donde se muestra el contenido, algunos tipos de regiones estan destinados a los bloques, donde el renderizador de bloques establece este contenido.

Tomando como ejemplo el bloque “login”, tenemos que de forma muy similar a los módulos, el nombre del directorio de un bloque lleva un prefijo “block_”, dando como resultado “block_login”.

Dentro del directorio del bloque tenemos los siguientes archivos: helper, login, params, info.xml y el directorio html que contiene un archivo logintpl.

Helper, contiene una clase que tiene el siguiente formato para su nombre: Jha«nombre_bloque»Helper, esto a fin de que el renderizador sepa como llamar al Helper. Esta clase solo provee funciones de ayuda para el bloque, algunas veces se encarga de la obtencion de información de la base de datos.

```
1  <?php
2  ...
3
4  class JhaLoginHelper {
5      private $params;
6
7      function __construct($params) {
8          $this->params = $params;
9      }
10
11     public function getTask() {
12         if(isset($_SESSION['USER'])) {
13             return 'logout';
14         }
```

```
15          return 'login';
16      }
17
18      public function getUser(){
19          if(isset($_SESSION['USER'])){
20              return $_SESSION['USER'];
21          }
22          return NULL;
23      }
24  }
25  ?>
```

Código 31: Helper del bloque 'Login'.

Login, sirve como punto de entrada al bloque y contiene porciones de código que hacen uso del Helper y también de la plantilla "logintpl".

```
1   <?php
2   ...
3
4   require_once $GLOBALS['JHA_BLOCK_PATH'] . 'helper.php';
5
6   $helper = new JhaLoginHelper($params);
7   $task = $helper->getTask();
8   $user = $helper->getUser();
9
10  require_once $GLOBALS['JHA_BLOCK_PATH'] . 'html' . DS . 'logintpl.php';
11  ?>
```

Código 32: Punto de entrada al bloque.

Params, Este archivo es unicamente utilizado al momento de la creación o edición de un bloque de tipo "login" y su finalidad es la configuración de los parametros extra que se utilizarán por el renderizador al renderizar el bloque.

```
1   <?php
2   ...
3
4   require_once $this->blocktypepath.DS.'helper.php';
5   $helper = new JhaLoginHelper(array('showtitle' => '0', 'suffix' => ''));
6
7   $params = array();
8   $params['showtitle'] = array('', 'checked="checked"');
9   $params['suffix'] = '';
10
11  if($blockparams != NULL){
12      $params['showtitle'] = array(($blockparams['showtitle'] == '0' ? 'checked="checked"' : ''),
            ($blockparams['showtitle'] == '1' ? 'checked="checked"' : ''));
13      $params['suffix'] = $blockparams['suffix'];
14  }
15  ?>
16  <fieldset>
17      <legend>Parametros del Bloque</legend>
18      <table cellspacing="1" width="100%" class="">
19          <tr>
20              <td width="40%" class="paramlist_key">
21                  <label>Mostrar t&iacute;tulo</label>
22              </td>
23              <td class="paramlist_value">
24                  <input type="radio" <?php echo $params['showtitle'][0] ?> value="0" id="params_showtitle0"
                        name="params_showtitle">
25                  <label for="params_showtitle0">No</label>
```

```
26              <input type="radio" <?php echo $params['showtitle'][1] ?> value="1" id="params_showtitle1"
                    name="params_showtitle">
27              <label for="params_showtitle1">S&iacute;</label>
28          </td>
29      </tr>
30      <tr>
31          <td width="40%" class="paramlist_key">
32              <label>Sufijo de la clase CSS</label>
33          </td>
34          <td class="paramlist_value">
35              <input type="text" value="<?php echo $params['suffix']; ?>" id="params_suffix"
                    name="params_suffix">
36          </td>
37      </tr>
38  </table>
39  </fieldset>
```

Código 33: Parametros de configuración.

Info.xml, Proporciona información sobre el bloque tales como: nombre del bloque, renderizador, descripción del bloque, autor y fecha de creación.

```
1  <?xml version="1.0" encoding="ISO-8859-1" ?>
2  <block>
3      <info>
4          <name>Login</name>
5          <renderer>block_login</renderer>
6          <description>Bloque de Inicio de Sesion</description>
7          <author>Richard Jaldin L.</author>
8          <date>Lunes, 04 de Abril de 2011</date>
9      </info>
10 </block>
```

Código 34: Información del bloque.

Logintpl, Es un archivo que contiene una estructura HTML con porciones de codigo PHP para insertar la información obtenida a través del Helper, su nombre sigue el formato: «nombre_bloque»tpl.

```
1  <?php
2  defined( '_JHAEXEC' ) or die( 'Access Denied' );
3  ?>
4  <div>
5      <form action="index.php" method="post">
6          <?php if($task == 'logout') { ?>
7              <div>Bienvenido, <a href="index.php?elem=mod_user&task=edituser&id=<?php echo $user->id;
                    ?>"><?php echo $user->nombre; ?></a></div>
8              <div><center><input name="submit" type="submit" value="Cerrar Sesion" /></center></div>
9          <?php } else { ?>
10             <div>
11                 <table border="0" cellpadding="0" cellspacing="0">
12                     <tr><td><label for="username">Nombre de Usuario: </label></td></tr>
13                     <tr><td><input name="username" id="username" type="text" /></tr>
14                     <tr><td><label for="passwd">Contrase&ntilde;a: </label></td></tr>
15                     <tr><td><input name="passwd" id="passwd" type="password" /></tr>
16                 </table>
17             </div>
18             <div><center><input name="submit" type="submit" value="Iniciar Sesion" /></center></div>
19         <?php } ?>
20         <input name="elem" type="hidden" value="mod_user" />
21         <input name="task" type="hidden" value="<?php echo $task; ?>" />
22     </form>
23 </div>
```

Código 35: Plantilla del bloque.

5.4. Relación entre módulos y bloques

Los bloques y módulos no guardan relación a nivel de código, cada uno tiene una finalidad específica.
Los módulos procesan las interacciones del usuario con el sistema, es decir, que cada solicitud que el usuario hace al sistema es procesada a través de algún módulo. Como resultado de este procesamiento, se genera contenido HTML, el cual se ubica en una región particular de la plantilla a través de los renderizadores.
Los bloques por otro lado, solo se encargan de mostrar contenido, y lo hacen en regiones distintas a las del módulo.

5.5. Módulos

A continuación vemos los distintos módulos que forman parte de *i*CMS.

5.5.1. Módulo Base

El módulo base contiene funciones básicas para el funcionamiento del CMS, tales como: la personalización de la plantilla, agregado de nuevas plantillas, gestionado de bloques de contenido, configurado del CMS, entre otras.

5.5.2. Configurado del CMS

La configuración del CMS se hace a través del archivo *JhaConfiguration*, gestiona los datos del servidor de base de datos así como el prefijo de las tablas, también se puede gestionar la vigencia de las noticias y también los meta-datos para lo que se conoce como *alta en buscadores*.

5.5.3. Gestión de plantillas

La gestión de plantillas comprende dos elementos importantes.
El primero de estos elementos, permite agregar nuevas plantillas e intercambiar la apariencia de la página a través de las plantillas.
El segundo elemento permite personalizar la plantilla que se designa por defecto para mostrarse en el sitio. Para este efecto el módulo provee mecanismos para ver las regiones definidas en la plantilla, asimismo tiene mecanismos para dividir las regiones ya sea en columnas o filas. También las regiones pueden moverse para reacomodar las regiones.

5.5.4. Gestión de bloques

Los bloques son los contenidos secundarios, todos los bloques trabajan de forma muy similar, todas ellas tienen un renderizador específico, en síntesis cada bloque tiene su propio renderizador. A través de este gestor de bloques se definen elementos como la región donde se mostrará el bloque, el orden entre bloques, y la relación entre los bloques y los menús, todo esto a nivel de

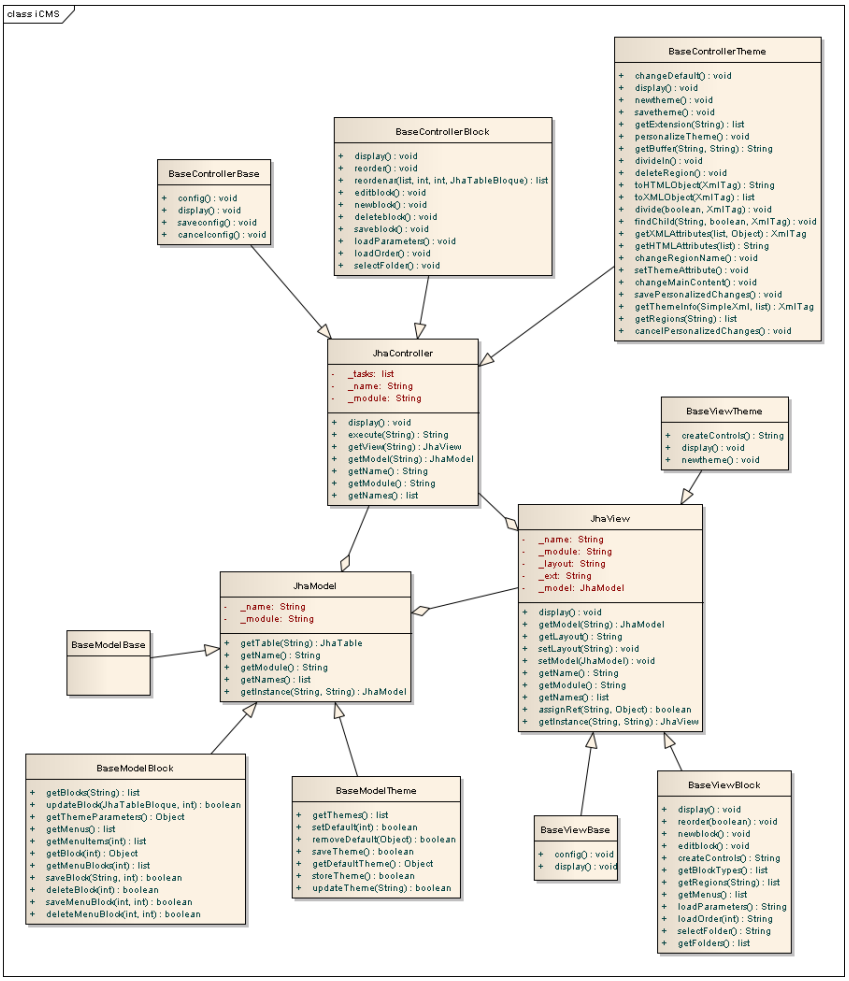

Figura 5.2: Modelo de clase: Módulo Base. [Elaboración propia].

administración avanzada del *i*CMS, cuando hablamos de una administración estándar, hacemos referencia a los cambios que se pueden hacer directamente manipulando el contenido sobre la plantilla, o como decimos vulgarmente, "En caliente".

5.5.5. Módulo Content

El módulo de contenido contiene funciones para el gestionamiento de las publicaciones del CMS, tales como: la gestión de artículos, gestión de secciones y gestión de noticias.

5.5.6. Gestión de artículos

A través de este medio se pueden crear, editar, eliminar y listar los artículos, los artículos pertenecen a una sección, además cada artículo puede estar o no relacionado a un item de menú. También se cuenta con el editor WYSIWG para agregar el contenido de los artículos.

5.5.7. Gestión de secciones

Las secciones muestran un conjunto de artículos al mismo tiempo, por el momento solo maneja una vista estándar de dos columnas, donde en la primera se muestra el artículo mas reciente, y

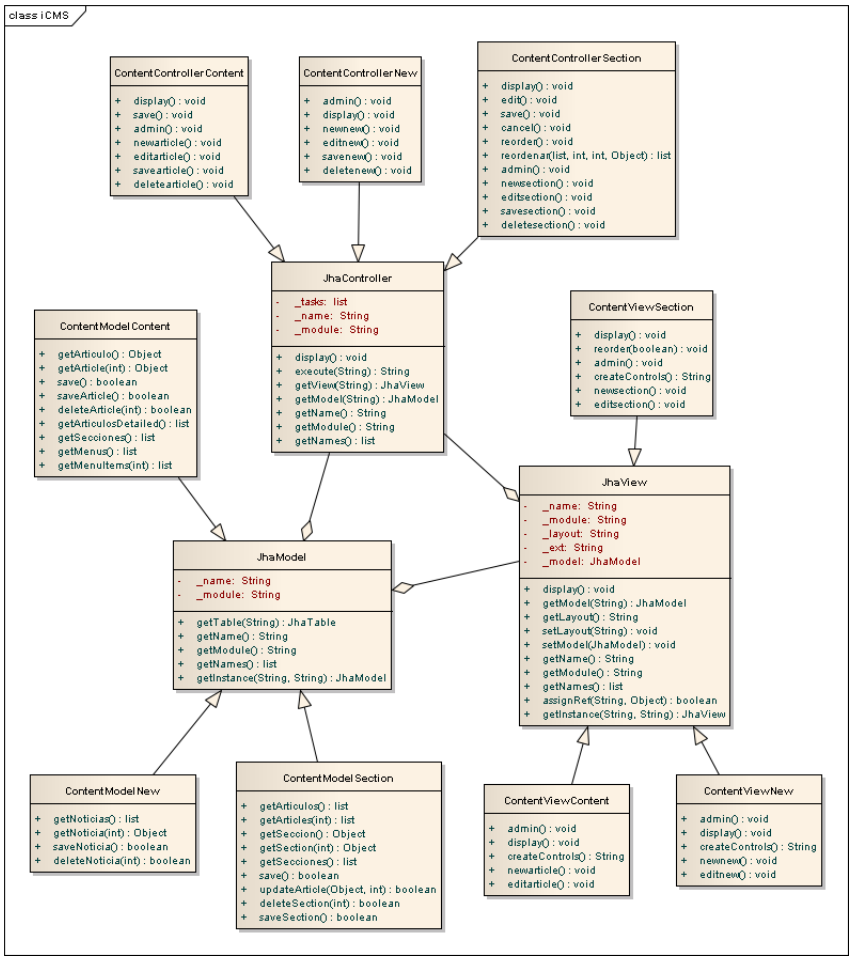

Figura 5.3: Modelo de clase: Módulo Content. [Elaboración propia].

en la segunda se muestran los artículos anteriores.
Las secciones al igual que los artículos, también estan relacionados a un enlace de menú.

5.5.8. Gestión de noticias

Las noticias solo son párrafos muy breves, avisos muy cortos que se sirven del bloque de noticias para mostrarse.
Al igual que la gestión de secciones y artículos se pueden crear, editar, listar y eliminar las noticias.

5.5.9. Módulo Menú

El módulo de menús contiene funciones para el gestionamiento de los menús del CMS.

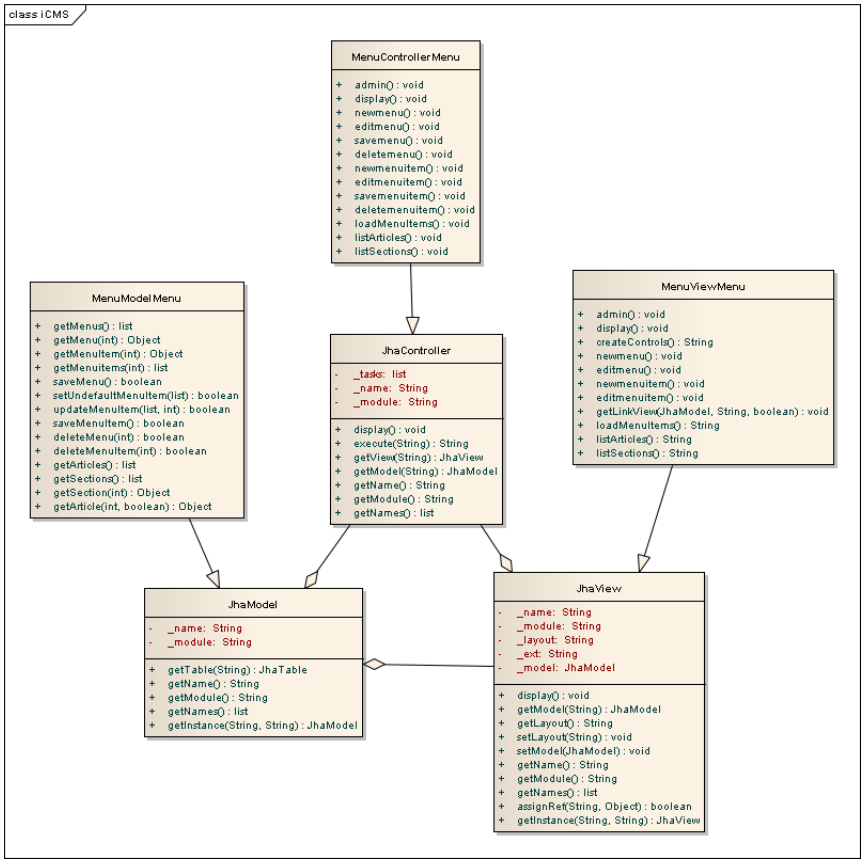

Figura 5.4: Modelo de clase: Módulo Menú. [Elaboración propia].

5.5.10. Gestión de menús

La gestión de menús contiene dos elementos: Los menús propiamente dichos y los ítems de menú.
Los menús solo agrupan un conjunto de ítems de menú.
Los ítems de menú contiene un enlace a artículos, secciones o a páginas externas al sitio, cada ítem de menú tiene un número de orden con respecto a los otros ítems del menú, también pueden tener un ícono asociado.

5.5.11. Módulo User

El módulo de usuario contiene funciones para el gestionamiento de los usuarios del CMS.

5.5.12. Gestión de usuarios

A través de este gestor se pueden crear, modificar, eliminar y listar los usuarios registrados, los usuarios tienen roles definidos (Super Administrador, Editor) y el usuario visitante que no

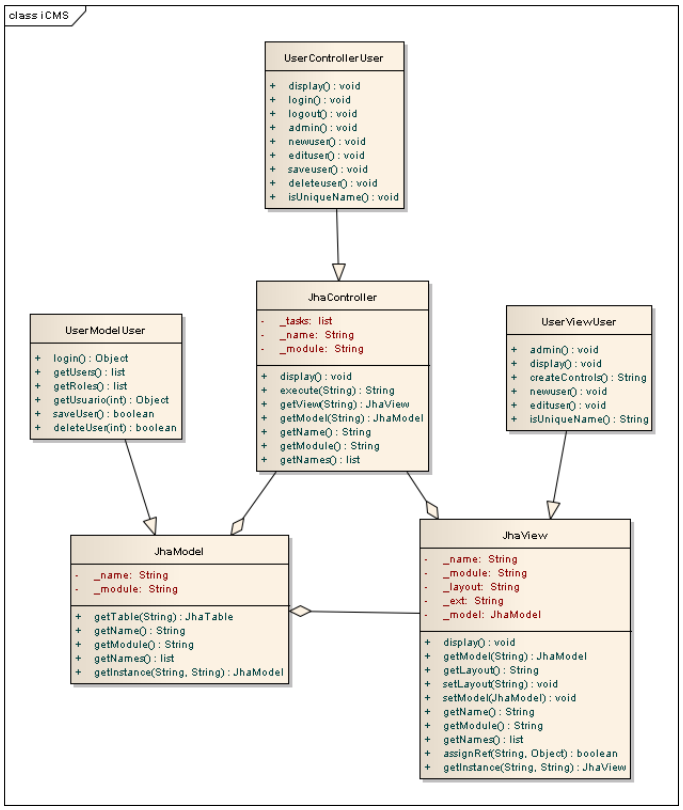

Figura 5.5: Modelo de clase: Módulo User. [Elaboración propia].

necesita registrarse. Dado que este CMS es un prototipo, no se contempla muchos de los datos de los usuarios.

5.6. Bloques

Como ya vimos en el capítulo VI los bloques tienen renderizadores específicos, en este capítulo veremos la especificación de cada uno de estos renderizadores de bloques.

5.6.1. Bloque Banner

Este renderizador sirve para mostrar banners de imágenes, estos banners dependiendo de los parametros del bloque pueden estar animados o no, los efectos son por defecto los que estan presentes en la librería *effect* del *framework jhaley* (sobreponer, entrar/salir vertical, entrar/salir horizontal).

5.6.2. Bloque Breadcrumb

El renderizador de la barra de navegación es bastante útil, este renderizador crea los enlaces de la barra de navegación en función del menú, y va creaciendo con los enlaces de los artículos.

5.6.3. Bloque Custom

Quizás este renderizador sea el más útil, dado que la finalidad que tiene es de crear HTML personalizado. El HTML puede ser de cualquier tipo (imágenes, flash, etc.). Al momento de crear los bloques con este renderizador también se crea el contenido HTML.

5.6.4. Bloque Login

Este renderizador tiene la finalidad de crear la ventana de acceso a las cuentas de usuario, el bloque de login trabaja en conjunto con el módulo de usuario para validar los datos del usuario o también para cerrar la sesión del usuario

5.6.5. Bloque Menú

En capítulos anteriores vimos el módulo de menú, en el cual se gestiona los menús e ítems de menú, pues bien, este renderizador muestra los ítems de menú ordenados de acuerdo a un número de orden, cada bloque de menú puede acomodarse en menús verticales, horizontales, vertical en listas, también es posible hacer visibles o no los iconos asociados a los ítems de menú.

5.6.6. Bloque New

Este renderizador trabaja en conjunto con el módulo de contenido, concretamente con el gestor de noticias, este bloque solo se encarga de mostrar las noticias, las noticias pueden tener efectos animados dado que también utilizan los efectos de la librería *effect* del *framework jhaley*.

5.7. Temas

Los temas o *templates* son simplemente elementos que le dan al CMS la apariencia en función de los CSS e imágenes.
Por el momento se tiene dos temas creados especificamente para el CMS:

- **Tema Default** Este tema está creado como apariencia por defecto del CMS
- **Tema Enikma** Este tema es una alternativa de apariencia para el CMS.

*i*CMS tiene la capacidad para subir nuevas plantillas, estas plantillas deben tener una estructura concreta. Se deben tener mínimamente los siguientes archivos en un compreso ya sea tar.gz, tar.bz:

- index.php
- info.xml
- install.sql

... Adicionalmente se pueden tener un directorio para imágenes y otro directorio para los css. Este motor de gestión de las plantillas provee una gama de opciones para que el usuario pueda personalizar aún más su plantilla.

A continuación vemos en detalle cada uno de los elementos que componen la plantilla:

Index.php, contiene una definición de las regiones de las plantillas donde irán los distintos tipos de contenido.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es-es" lang="es-es" >
    <head>
        <jhadoc:include type="head" />
        <link rel="stylesheet" type="text/css" href="themes/default/css/template.css" />
    </head>
    <body>
        <div id="art-page-background-simple-gradient"></div>
        <div id="art-page-background-glare">
            <div id="art-page-background-glare-image"></div>
        </div>
        <div id="art-main">
            <jhadoc:include type="admin-menu" />
            <div class="art-Sheet">
                <div class="art-Sheet-tl"></div>
                <div class="art-Sheet-tr"></div>
                <div class="art-Sheet-bl"></div>
                <div class="art-Sheet-br"></div>
                <div class="art-Sheet-tc"></div>
                <div class="art-Sheet-bc"></div>
                <div class="art-Sheet-cl"></div>
                <div class="art-Sheet-cr"></div>
                <div class="art-Sheet-cc"></div>
                <div class="art-Sheet-body">
                    <div class="art-Header">
                        <div class="art-Header-png"></div>
                        <div class="art-Header-jpeg"></div>
                    </div>
                    <jhadoc:include type="maincontent" />
                    <div class="cleared"></div>
                </div>
            </div>
            <div class="cleared"></div>
        </div>
    </body>
</html>
```

Código 36: Index de la plantilla.

Algo que destacar acerca de este archivo son la forma de definir las regiones, las cuales tienen el siguiente formato para la mayoría de los elementos: <jhadoc:include type=“type” />
Para las regiones definidas como bloques se usa el siguiente formato: <jhadoc:include type=“block” region=“region” />

Info.xml, Al igual que con los bloques, este elemento sirve para definir las propiedades de la plantilla, como el nombre, autor, además de las regiones de los bloques.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<template>
  <info>
    <name>Segunda plantilla por defecto</name>
    <description>Plantilla por defecto creado para el iCMS</description>
    <author>Richard Jaldin L.</author>
    <date>Viernes, 03 de Junio de 2011</date>
  </info>
  <regions>
    <region>left</region>
    <region>breadcrumb</region>
    <region>news</region>
    <region>reggion1</region>
    <region>footer</region>
  </regions>
```

```
16 </template>
```

Código 37: Propiedades de la plantilla.

Install.sql, Index.php define el marco principal de la plantilla, pero la definición de las regiones son guardadas en la Base de datos, este archivo contiene una sentencia SQL para guardar dicha información.

```
1 update #__plantilla set html = '<div class="art-contentLayout"><div class="art-sidebar1"><jhadoc:include
      type="block" region="left" /></div><div class="art-content"><div class="art-Post"><jhadoc:include
      type="block" region="breadcrumb" /></div><div class="art-Post"><jhadoc:include type="block"
      region="news" /></div><div class="art-Post"><jhadoc:include type="module" /></div></div></div><div
      class="art-Footer"><jhadoc:include type="block" region="footer" /></div>', xml = '<?xml version="1.0"
      encoding="ISO-8859-1" ?><content><elem class="art-contentLayout"><elem class="art-sidebar1"><jhadoc
      type="block" region="left"></jhadoc></elem><elem class="art-content"><elem class="art-Post"><jhadoc
      type="block" region="breadcrumb"></jhadoc></elem><elem class="art-Post"><jhadoc type="block"
      region="news"></jhadoc></elem><elem class="art-Post"><jhadoc
      type="module"></jhadoc></elem></elem></elem><elem class="art-Footer"><jhadoc type="block"
      region="footer"></jhadoc></elem></content>'
```

Código 38: Fragmento de plantilla almacenable en la base de datos.

Capítulo 6

FRAMEWORK JHALEY

6.1. Introducción

Como todo CMS tiene un conjunto de librerías definidas para su funcionamiento, *i*CMS no es la excepción, en el presente capítulo se presenta un vistazo general de los elementos del "Framework Jhaley" creados para desarrollar este CMS, diseñado para hacer de la reutilización de código algo innato en el desarrollo del CMS y principalmente para el soporte de nuevas extensiones y en los estándares necesarios para el CMS.
El framework contempla elementos para ciertas acciones concretas como:

- Gestión de variables REQUEST.
- Gestión de archivos compresos (*.tar, *.tgz, *.tbz).
- Conexiones y manipulación de datos (Base de Datos).
- Gestión de archivos y directorios.
- Renderizadores.
- Librerías Javascript.
- Patrón MVC
- Librerías para gestión de elementos web.
- Gestión de XML.
- Librerías para editor WYSIWYG ("What You See Is What You Get").

... Entre otras.

6.2. Arquitectura del Framework

El framework *Jhaley* para el motor de renderizado utiliza un patrón arquitectónico de Modelo-Vista-Controlador, gracias a este patrón todas las extensiones estan obligadas a seguirlo también, la principal ventaja de esta "regla" es que todas las extensiones estan muy bien estructuradas facilitando el mantenimineto de las mismas.

6.3. Librería JhaFactory

La librería JhaFactory, como su nombre indica sigue el patrón de diseño "*Factory*", el cual se encarga de tener únicamente funciones dispuestas para la creación de instancias de otros objetos. En nuestro caso en concreto se encarga de crear instancias como:

- **JhaDBO**, Objeto de Conexion a la base de datos.
- **JhaConfiguration**, Objeto de configuración general del CMS para un sitio en concreto.
- **Renderers**, Objetos de Renderización (JhaRenderer, JhaModuleRenderer, JhaBlockRenderer, etc).
- **JhaEditor**, Instancia del editor WYSIWYG.

```php
<?php
...

class JhaFactory extends JhaObject {
    static function &getRenderer($type = null){
        static $renderer;
        if($type == null){
            if(JhaRequest::getVar('elem') == 'mod_base' && JhaRequest::getVar('controller') == 'theme' &&
                 (JhaRequest::getVar('task') == 'personalizeTheme' || JhaRequest::getVar('task') ==
                 'savePersonalizedChanges')){
                jhaimport('jhaley.html.theme');
                $renderer = new JhaThemeRenderer();
            }
            else {
                jhaimport('jhaley.html.renderer');
                $renderer = new JhaRenderer();
            }
        }
        elseif($type == 'block'){
            jhaimport('jhaley.html.block');
            $renderer = new JhaRendererBlock();
        }
        elseif($type == 'module'){
            jhaimport('jhaley.html.module');
            $renderer = new JhaRendererModule();
        }
        elseif($type == 'head'){
            jhaimport('jhaley.html.head');
            $renderer = new JhaRendererHead();
        }
        elseif($type == 'admin-menu'){
            jhaimport('jhaley.html.adminmenu');
            $renderer = new JhaRendererAdminMenu();
        }
        elseif($type == 'ajax'){
            jhaimport('jhaley.html.ajax');
            $renderer = new JhaAJAXRenderer();
        }
        return $renderer;
    }

    static function &getDBO(){
        static $dbo;
        if (!is_object($dbo)) {
            jhaimport('jhaley.db.dbo');
            $dbo = new JhaDBO(JhaFactory::getConfig());
        }
        return $dbo;
    }

    static function &getConfig(){
        static $config;
        if (!is_object($config)) {
            require_once JHA_CONFIGURATION_PATH . DS . 'config.php';
            $config = new JhaConfiguration();
        }
        return $config;
    }

    static function &getEditor(){
        static $editor;
        if (!is_object($editor)) {
            jhaimport('jhaley.editor.ckeditor');
            jhaimport('jhaley.filefinder.ckfinder');
            $editor = new CKEditor();
            $finder = new CKFinder();
            $finder->BasePath = 'libraries/jhaley/filefinder/';
```

```
66              $finder->SetupCKEditorObject($editor);
67          }
68          return $editor;
69      }
70  }
71  ?>
```

Código 39: Clase JhaFactory

También cabe destacar que JhaFactory al igual que otras clases es global con respecto del sistema, esto significa que puede ser utilizado desde cualquier punto dentro de la aplicación.

6.4. Librería JhaRequest

La librería JhaRequest, es una clase global que provee funciones estáticas para la manipulación principalmente de las variables del HTMLRequest (REQUEST, POST, GET), pero su uso también está en las variables SERVER, SESSION.
Entre las funciones que cumple está el que tiene la capacidad de filtrar los valores que se obtienen del HTMLRequest, bajo tres criterios bien marcados:

- **NOTRIM** Este filtro le permite considerar como válido el valor obtenido no importando si dicho valor lleva espacios al inicio o final.
- **ALLOWRAW** Este filtro le permite considerar como válido el valor obtenido sin ninguna restricción.
- **ALLOWHTML** Este filtro le permite considerar como válido el valor obtenido solo si es HTML.

También tiene una directiva que le permite obtener un conjunto de valores (Lista), siempre y cuando los nombres con los que llegaron dichos valores tienen un prefijo en común.

6.5. Librería de Utilidades (JhaUtility)

Está librería ofrece funciones estándar de ayuda para realizar distintas tareas tales como: Buscar un elemento en funcion de su id, ver si un elemento pertenece a una lista, remover un elemento de una lista, insertar un elemento en una posición dada de una lista, verificar si los usuarios son Editores o Administradores.

```
1   <?php
2   ...
3
4   class JhaUtility {
5       function parseAttributes($string) {
6           $attr = array();
7           $res  = array();
8           preg_match_all( '/([\w:-]+)[\s]?=[\s]?"([^"]*)"/i', $string, $attr );
9           if (is_array($attr)) {
10              $numPairs = count($attr[1]);
11              for($i = 0; $i < $numPairs; $i++ ) {
12                  $res[$attr[1][$i]] = $attr[2][$i];
13              }
14          }
15          return $res;
16      }
17
18      function search($id, $array) {
```

```
19          $res = false;
20          if(!($id && $array)) return $res;
21          for ($i = 0; $i < count($array); $i++){
22              if($array[$i]->id == $id){
23                  $res = $array[$i];
24                  break;
25              }
26          }
27          return $res;
28      }
29
30      function inArray($elem, $array){
31          $res = -1;
32          if(!($elem && $array)) return $res;
33          $i = 0;
34          foreach ($array as $key => $value) {
35              if($array[$key] == $elem){
36                  $res = $i;
37                  break;
38              }
39              $i++;
40          }
41          return $res;
42      }
43
44      function remove($elem, $array) {
45          $res = array();
46          if(!($elem && $array)) return $res;
47          for ($i = 0; $i < count($array); $i++){
48              if($array[$i] != $elem){
49                  $res[] = $array[$i];
50              }
51          }
52          return $res;
53      }
54
55      function insert($elem, $pos, $array) {
56          $res = array();
57          if(!($elem && $array)) return false;
58          for ($i = 0; $i < count($array); $i++){
59              if($i == $pos){
60                  $res[] = $elem;
61              }
62              $res[] = $array[$i];
63          }
64          if($pos >= count($array)) $res[] = $elem;
65          return $res;
66      }
67
68      function userCanEdit(){
69          $user = (isset($_SESSION['USER']) ? $_SESSION['USER'] : NULL);
70          if($user){
71              return $user->rol == 'Super Administrador' || $user->rol == 'Editor';
72          }
73          return false;
74      }
75
76      function userCanAdministrate() {
77          $user = (isset($_SESSION['USER']) && $_SESSION['USER']->rol == 'Super Administrador' ?
                $_SESSION['USER'] : NULL);
78          if($user){
79              return $user->rol == 'Super Administrador';
80          }
81          return false;
82      }
83  }
84  ?>
```

Código 40: Clase JhaUtility

6.6. Librería Import

Está librería no contiene una clase sino mas bien una función, cuyo único trabajo es importar las librerías del framework en cualquier parte de la aplicación. Recibe una cadena con el formato 'jhaley.mvc.model', donde los puntos representan la sucesión de los paquetes, en este caso el paquete mvc está dentro del paquete jhaley, el último elemento, vale decir model, es la librería como tal.
Obviamente que para poder acceder al archivo de la librería de manera interna se construye la ruta absoluta.

6.7. Librerías del paquete **base**

6.7.1. JhaObject

Esta librería provee un objeto base estándar con ciertas características que son comunes a todos las clases del sistema.
Las propiedades que define son:

- getProperties($isPublic:boolean):array, cada objeto es capaz de devolver la lista de sus atributos, si $isPublic es true devuelve solo los atributos públicos.
- redirect($url:String):void, cada objeto es capaz de redireccionar a una dirección distinta por medio del parámetro $url.

6.8. Librerías del paquete **compress**

Este conjunto de librerías fueron creadas para manipular archivos comprimidos (.ZIP, .TAR, .TAR.GZ o .TGZ, .TAR.BZ o .TBZ).

6.8.1. Archive

Superclase para los archivos comprimidos, dado que cada tipo de compreso lleva características especiales, a través de está se definen funciones y atributos comunes para la manipulación de los compresos.
Todas las clases que heredan de está son capaces de crear archivos compresos.

6.8.2. ZIP

Está clase provee una estructura para manipular y crear archivos .ZIP, está librería no posee mecanismos para la descompresión de los compresos .ZIP.

6.8.3. TAR

Está clase se convierte en la base tanto para la manipulación de los .TGZ o .TAR.GZ y también para los .TBZ o .TAR.BZ, por si solo también es capaz de empaquetar y desempaquetar los archivos .TAR.

6.8.4. GZIP

Uno de los tipos de archivos compresos más comunes dentro de la comunidad del software libre. Está clase especializa la clase TAR, para proveer un mecanismo de compresión alternativo al ZIP tradicional. AL igual que las otras clases es capaz de crear compresos y descomprimirlos.

6.8.5. BZIP

Está clase, al igual que GZIP, también hereda de la clase TAR convirtiendose en otra alternativa al ZIP tradicional o al GZIP, también posee la capacidad de crear compresos y descomprimirlos.

6.9. Librerías del paquete **css**

Este conjunto de librerías esta compuesto enteramente de archivos .CSS, que proveen estilos estándar para el CMS independientemente de la plantilla.

6.9.1. base.css

Durante la administración muchos de los elementos son capaces de moverse entre las regiones definidas de la plantilla, base provee estilos para resaltar estos elementos móviles.

6.9.2. menu.css

Solo es utilizable durante la administración, provee estilos para el menú de administración.

6.10. Librerías del paquete **db**

Este conjunto de librerías esta creado para gestionar todo lo relacionado a las bases de datos, incluidos el objetos de conexion, las clase que proveen la estructura de las tablas.

6.10.1. JhaDBO

Esta librería se encarga de crear la conexion entre la aplicación y la base de datos, asimismo provee de funciones para ejecutar consultas, funciones de inserción, funciones de eliminación entre otras.

- open():boolean, utiliza una instancia de la clase JhaConfiguration para obtener los datos como el servidor de base de datos, nombre de usuario y nombre de la base de datos, y con ellos abrir una conexion con el servidor.
- isConnected():boolean, verifica si existe una conexion activa.
- close():void, cierra la conexion activa.
- setQuery($query:String, $offset:int, $limit:int):void, establece la consulta para su futura ejecución, asimismo recibe los parametros $offset y $limit para determinar el nivel de paginación, en caso de no haberlos se establecen valores por defecto.
- executeQuery($query:String):boolean, ejecuta la consulta pasada a través del parametro $query y devuelve su resultado.

- loadObject():Object, Devuelve un objeto estándar con los resultados de la consulta, generalmente utilizado cuando la consulta solicita un registro en concreto.
- loadObjectList():array, Devuelve una lista de objetos estándar con los resultados de la consulta, generalmente utilizado cuando la consulta solicita varios registros de la base de datos.
- insertObject($table:String, &$object:Object, $keyName:String):boolean, inserta un registro en la tabla $table de la base de datos, con los datos contenidos en el objeto $object y utilizando a $keyName como elemento auxiliar para verificar si se registro correctamente.
- updateObject($table:String, &$object:Object, $keyName:String, $updateNulls:boolean):boolean, actualiza un registro en la tabla $table de la base de datos, con los datos contenidos en el objeto $object y utilizando a $keyName como parte de la condición para verificar que es el registro correcto, la bandera $updateNulls especifica si los campos que ahora son NULL deben actualizarse también o no.
- deleteObject($table:String, &$object:String, $keyName:String):boolean, funciona de manera similar a los dos anteriores, con la diferencia de que esta función está diseñada para eliminar registros de la base de datos.

6.10.2. JhaTable

Está superclase define propiedades y comportamientos similares para las tablas de la base de datos, facilitando principalmente la inserción, actualización y eliminación de los registros de la base de datos.
Entre los atributos de esta clase están: $table_key, $table y $dbo.

- load($objid:int):void, está función es la encargada de cargar la información desde la base de datos en funcion del $table_key como campo y $objid como valor.
- bind($elem:mixed):boolean, Establece los atributos del objeto tabla en función a $elem que puede ser un array u objeto.
- store():mixed, es una función que guarda la información contenida en el objeto tabla a la base de datos, esta función es capaz de diferenciar si lo que se guarda es un nuevo registro o si se actualiza uno existente, devuelve false si no logra guardar y devuelve un entero con la clave del registro si se logra guardar.
- delete():boolean, es una función que elimina un registro de la base de datos en función de la información contenida en el objeto tabla.
- getInstance($name:String):JhaTable, la clase JhaTable es capaz de construir instancias en base al nombre de la tabla.

6.10.3. Librerías del paquete **tables**

Este paquete contiene las clases para la creación de estructuras que representan a las tablas de la base de datos, cada clase hereda de la clase JhaTable para especializar y definir los atributos correspondientes.

6.11. Librerías del paquete editor

Este es un conjunto de librerías creado por terceros, concretamente, el editor lleva el nombre de ckeditor, siendo uno de los más potentes editores WYSIWYG, permite la manipulación del contenido desde el HTML o de manera gráfica, facilita la inserción de imágenes, animaciones en flash, tipo de letra, tamaño de letra y varias otras funciones más.

6.12. Librerías del paquete file

Las librerías de este paquete están diseñadas para manipular los archivos y directorios.

6.12.1. JhaDirectory

Esta librería provee una estructura para manipular los archivos de un directorio.

- read():void, lee un directorio en función del atributo $_pathfile, que es propiedad de la clase JhaDirectory, y almacena el resultado en una lista $_files que también es un atributo de JhaDirectory.
- getFolders():array, filtra y devuelve solo los directorios de la lista $_files.
- getFiles($filetypes:array):array, filtra y devuelve solo los archivos de la lista $_files, siempre y cuando la extensión de estos sea alguno de los que vienen en $filetypes.

6.12.2. JhaFile

JhaFile es una librería que facilita la lectura y escritura de archivos.

- setContent($content:String):void, se establece el valor del atributo $_content con el valor de $content.
- getContent():String, devuelve el valor contenido en $_content.
- write():void, escribe el valor contenido en $_content en el archivo descrito en el atributo $_pathfile.
- read():void, lee el contenido del archivo descrito en $_pathfile y lo guarda en el atributo $_content.

6.13. Librerías del paquete filefinder

Este es un conjunto de librerías creado por terceros, concretamente, el editor lleva el nombre de ckfinder, de los mismos creadores del ckeditor, estas librerías tienen como finalidad proveer el soporte para manipular las imagenes en el servidor, y hacerlas visibles en el cliente para su uso a través del ckeditor. Es decir, si se quiere agregar nuevas imágenes al contenido del CMS, esta librería se encargará de subir al servidor las nuevas imágenes y también direccionarlas correctamente en el contenido.

6.14. Librerías del paquete **html**

Este es un conjunto de librerías son en realidad los motores de renderización de la aplicación, cuenta con un renderizador de nivel superior que hace uso de los otros renderizadores, generando como resultado HTML que se muestra en el navegador del cliente.

6.14.1. JhaRenderer

Como se ha mencionado antes, este es el renderizador principal, cuya única tarea es renderizar la plantilla por defecto y acomodar el contenido sobre dicha plantilla, para ello obtiene la estructura de la plantilla y utiliza a los renderizadores secundarios para obtener el contenido de la página.
Las funciones presentes en esta librería son:

- render():void, está función es el punto de inicio del renderizado de la plantilla y su contenido.
- loadTemplate($row:mixed):String, obtiene el HTML de la plantilla que se irá a mostrar, este HTML contiene también una descripción de las regiones donde irá el contenido de la página.
- renderTemplate($template:String):String, esta función se encarga de incluir el contenido para cada region.
- getBuffer($type:String, $region:String):String, esta función determina el renderizador secundario a utilizar para obtener el contenido almacenado en la base de datos.

6.14.2. JhaModuleRenderer

Este renderizador solo obtiene el contenido que se procesa por las extensiones conocidas como *módulos* y lo devuelve al renderizador principal para ser acomodado sobre la plantilla.

6.14.3. JhaBlockRenderer

Este renderizador obtiene el contenido que procesan los bloques, estos bloques solo obtienen información de la base de datos en función de la region para la que se esta obteniendo el contenido.

6.14.4. JhaHeadRenderer

JhaHeadRenderer se encarga de renderizar los elementos del tag HTML <head></head>, tales como las referencias a CSS, o a Javascript.

6.14.5. JhaAJAXRenderer

Este renderizador se utiliza cuando se hacen llamadas via AJAX al servidor, ya sea para obtener contenido a través de los módulos o bloques. Retorna solo el contenido generado por los renderizadores de módulos o bloques y no asi el de toda la página.

6.14.6. JhaThemeRenderer

Este renderizador solo se utiliza a la hora de personalizar las plantillas, la razón es que no se muestra nada de contenido, solo estructura.

6.14.7. JhaRendererAdminMenu

Este renderizador es utilizado para renderizar el menú de administración, este menú solo aparece para el administrador, se trabaja en conjunto con la librería *xml*. Cuenta con la capacidad de crear submenus a distintos niveles, es decir, submenus de submenus.

6.14.8. JhaHTML

Esta librería renderiza HTML arbitrario, tales como checkboxes, controles (botones guardar, eliminar, nuevo, editar, cancelar, etc), tags *link*, tags *script* y cualquier elemento HTML.

6.15. Librerías del paquete **mvc**

Este es un conjunto de librerías proveen superclases que definen el patrón Modelo-Vista-Controlador, generalizando las características comunes a cada elemento del patrón.

6.15.1. JhaModel

Superclase que abstrae los modelos para todos los módulos. Cuenta con *$_name* y *$_module* como atributos.
Las funciones que provee esta clase son:

- getTable($name:String):JhaTable, esta función devuelve un objeto JhaTable específico de acuerdo al parametro $name, si este parametro es nulo, se utiliza el atributo $_name de la clase.
- getNames():mixed, obtiene tanto el nombre del módulo como el nombre del modelo en concreto de acuerdo al controlador que lo requiere.
- getInstance($name:String, $preffix:String):JhaModel, esta función se encarga de construir una instancia del modelo de acuerdo a los parametros recibidos en $name y $preffix.

6.15.2. JhaController

Superclase que abstrae los controladores para todos los módulos. Cuenta con *$_tasks*, *$_name* y *$_module* como atributos.
Las funciones que provee esta clase son:

- display():void, esta función genera el HTML por defecto a través de la vista asociada al controlador.
- execute($task):mixed, solo se encarga de llamar a la función definida en el parametro $task, si este parametro es nulo, se llama a la funcion *display()*.
- getNames():mixed, obtiene tanto el nombre del módulo como el nombre del modelo en concreto de acuerdo al controlador que lo requiere.
- getView($view):JhaView, construye y devuelve un objeto JhaView de acuerdo al módulo y al parametro $view, si $view es nulo, se toma el atributo $_name como parametro.
- getModel($model):JhaModel, construye y devuelve un objeto JhaModel de acuerdo al módulo y al parametro $model, si $model es nulo, se toma el atributo $_name como parametro.

6.15.3. JhaView

Superclase que abstrae las vistas para todos los módulos. Cuenta con *$_name*, *$_module*, *$_layout*, *$_ext* y *$_model* como atributos.
Las funciones que provee esta clase son:

- display():void, esta función genera el HTML que el módulo devolvera al renderizador, utiliza a $_layout como elemento para decidir que html mostrar.
- getNames():mixed, obtiene tanto el nombre del módulo como el nombre del modelo en concreto de acuerdo al controlador que lo requiere.
- getModel($model):JhaModel, construye y devuelve un objeto JhaModel de acuerdo al módulo y al parametro $model, si $model es nulo, se toma el atributo $_name como parametro.
- setLayout($layout:String):void, establece el valor del atributo $_layout.
- getLayout():String, devuelve el atributo $_layout que sirve para definir el HTML a mostrar.

6.16. Librerías del paquete **web**

Este conjunto de librerías provee un medio para representar el HTML como la estructura DOM, asimismo provee un medio para representar el XML. Esta librería esta basada en el Goaamb Framework.

6.16.1. Tag

Provee una estructura general para representar cualquier elemento, ya sea HTML, XML, JSON o PHP.

6.16.2. JSON

Hereda las propiedades y funciones de la clase Tag, también provee funciones concretas para la representación de elementos JSON.

6.16.3. XmlTag

Hereda las propiedades y funciones de la clase Tag, y provee funciones concretas para la representación de elementos XML.

6.16.4. HtmlTag

Hereda las propiedades y funciones de la clase Tag, y provee funciones concretas para la representación de elementos HTML.

6.16.5. PhpTag

Hereda las propiedades y funciones de la clase Tag, y provee funciones concretas para la representación de elementos PHP.

6.17. Librerías del paquete xml

Este paquete solo contiene un archivo *adminmenu.xml*, que describe la estructura del menú de administración, esta librería trabaja en conjunto con el JhaRendererAdminMenu.

6.18. Librerías del paquete js

Este paquete provee una serie de archivos Javascript, que ofrecen funciones para trabajar con: ajax, drag and drop (arrastrar y soltar), efectos (sobreponer, entrar y salir), elementos DOM, efectos de menú, menús contextuales y ventanas emergentes.

6.18.1. jha.js

Contiene un objeto JSON que provee funciones de utilidades para obtener información del navegador, utilidades para manipulación de objetos DOM, utilidades para manejo de listas y utilidades para manejo de HTML (validaciones).

```
1
2  var Jha = {
3      nav : {
4          isIE : false,
5          isNS : false,
6          isOP : false,
7          isSA : false,
8          isCH : false,
9          version : null,
10         init : function() {
11             ...
12         }
13     },
14     dom : {
15         $ : function(id) {
16             ...
17         },
18         $$ : function(name, position) {
19             ...
20         },
21         $$$ : function(tagname, position, doc) {
22             ...
23         },
24         scan : function (obj){
25             ...
26         },
27         create : function (tagname){
28             ...
29         },
30         position : function (obj){
31             ...
32         },
33         getStyle : function (el, style) {
34             ...
35         },
36         toCamelCase : function (s) {
37             ...
38         },
39         parseObject : function(obj, destine) {
40             ...
41         },
42         posFromEvent : function (event){
43             ...
44         }
45     },
```

```
46      util : {
47          inArray : function (elem, array){
48              ...
49          },
50          removeFromArray : function (index, array) {
51              ...
52          },
53          insertIn : function (pos, elem, array){
54              ...
55          }
56      },
57      html : {
58          checkbox : {
59              checkAll : function (obj, idname){
60                  ...
61              },
62              isChecked : function (obj){
63                  ...
64              },
65              validate : function (){
66                  ...
67              }
68          },
69          input : {
70              validate : function (valor, type){
71                  ...
72              }
73          },
74          select : {
75              validate : function (valor, defecto){
76                  ...
77              }
78          }
79      }
80  };
81
82  Jha.nav.init();
```

Código 41: Objeto JSON Jha.

A continuación vemos en detalle las distintas funciones presentes en este script:

- nav, este objeto interno se encarga de recuperar la información acerca del navegador del cliente.
- dom, se encarga de la manipulación de los elementos DOM, la busqueda de elementos, la creación de nuevos elementos, así como también de la gestión de los eventos.
- util, provee mecanismo para la manipulacion de objetos "array".
- html, este objeto provee mecanismos para la verificación de los campos de los formularios.

6.18.2. ajax.js

Contiene un objeto JSON que provee funciones netamente para manipular las conexiones vía AJAX.

```
1
2  Jha.ajax = function(object){
3      this.idUpdate = "";
4      this.action = "";
5      this.responseText = "";
6      this.responseXML = "";
7      this.url = "";
```

```
8       this.post = null;
9       this.json = false;
10      if (object) {
11          Jha.dom.parseObject(object, this);
12      }
13      var xmlhttp = false;
14      if (typeof ActiveXObject != 'undefined') {
15          try { xmlhttp = new ActiveXObject("Msxml2.XMLHTTP"); }
16          catch (e) {
17              try { xmlhttp = new ActiveXObject("Microsoft.XMLHTTP"); }
18              catch (E) { xmlhttp = false; }
19          }
20      }
21      if (!xmlhttp && typeof XMLHttpRequest != 'undefined') {
22          xmlhttp = new XMLHttpRequest();
23      }
24
25      this.verificarDireccion = function(){
26          ...
27      };
28
29      this.enviarpeticion = function(){
30          ...
31      }
32      this.conectado = false;
33      this.error = 0;
34      this.enviado = false;
35      var elobjeto = this;
36      this.responseJSON = "";
37      xmlhttp.onreadystatechange = function(){
38          ...
39      }
40  };
41
42  function ajaxupdate(direccion, idcapa, action){
43      var objajax = new Jha.ajax();
44      objajax.url = direccion;
45      objajax.idUpdate = idcapa;
46      objajax.action = action;
47      objajax.enviarpeticion();
48  }
```

Código 42: Objeto JSON Jha.ajax.

- jha.ajax, se encarga de realizar una llamada ajax estándar, es decir que una vez recibida la respuesta se ejecuta la acción por defecto.
- ajaxupdate, hace una llamada estándar pero con la diferencia de que una vez recibida la respuesta se ejecuta una acción definida por el usuario en vez de la acción estándar.

6.18.3. drag.js

Contiene un objeto JSON que brinda funciones para el efecto de arrastrar y soltar, que tienen los bloques de contenido, además provee funciones adicionales para poder guardar el estado y el orden de los bloques en la base de datos.

6.18.4. effect.js

Contiene un objeto JSON que tiene funciones para generar efectos visuales como: sobreponer y efectos verticales, utilizado principalmente por el bloque de noticias.

6.18.5. pmenu.js

Contiene un objeto JSON que provee funciones para crear los menus contextuales.

6.18.6. popup.js

Contiene un objeto JSON que tiene funciones para gestionar las ventanas emergentes.

Parte III

CONCLUSIONES

Capítulo 7

CONCLUSIONES Y RECOMENDACIONES

7.1. Introducción

Cada vez que se desarrolla un proyecto, surgen varias conclusiones, sugerencias y/o recomendaciones, que dan muestra de la experiencia obtenida del trabajo realizado, as'i como también las bases para la continuación del trabajo de investigación.
En las siguientes secciones se presentan las conclusiones y recomendaciones que son fruto del presente proyecto.

7.2. Conclusiones

7.2.1. Sobre la dinamicidad y flexibilidad del sistema

Uno de los objetivos primordiales del presente proyecto fue darle dinamicidad y flexibilidad a *i*CMS esto se ha conseguido desde dos enfoques principales: Desde la arquitectura del sistema y desde la interfaz gráfica de usuario.

El hecho de utilizar algún patrón arquitectónico, siempre es una mejora significativa para cualquier proyecto de software y su elección depende principalmente de las ventajas que proveen dichos patrones y como se utilizarán en el proyecto.

En el presente proyecto se optó por el patrón Modelo-Vista-Controlador (MVC), principalmente por la estructura que proveerían para los módulos del *i*CMS.
Otra de las razones para elegir el patrón MVC fue porque la aplicación resultante es fácil de entender, en términos de programación, incluso por quienes no tienen mucho conocimiento técnico de los conceptos de programación. Haciendo que el mantenimiento futuro del sistema sea relativamente sencillo.

Trás la finalización del proyecto *i*CMS se pudo apreciar que si bien se ganó un buen nivel de separación de las distintas partes que comprenden las aplicaciones web, existe cierta mezcla entre las capas del controlador y la vista, algo que podría solucionarse haciendo uso estricto del patrón Modelo-Vista-Presentador.

En cuanto a la Interfaz gráfica de usuario se ha logrado un avance significativo, muchos de los CMS actuales tienen una sección para el usuario y otra para la administración, *i*CMS combina ambos elementos en una sola pantalla, además incorpora elementos como AJAX para hacer que la administración sea mas directa, y por consiguiente también se consigue que los resultados sean visibles al momento.

La personalización del contenido sucede de dos formas: A través de la administración que mencionamos en el parrafo anterior, o a través de la personalización de la plantilla. *i*CMS tiene la cualidad de poder manipular las regiones donde va el contenido dividiendolas, de esta forma es posible cambiar el aspecto general del contenido.

Algo que cabe destacar respecto a la personalización de la plantilla, es que si bien *i*CMS cuenta con los medios para cambiar la apariencia general de las regiones de contenido, quizás en un futuro sea necesario pensar en una nueva forma de manipular la información dado que la actual puede llegar a ser ineficiente con el paso de los años.

7.2.2. Sobre las tecnologías usadas

De las tecnologías web utilizadas (PHP, JavaScript, XML y AJAX) se ha visto que si bien existen librerias de terceros, la mayoría de las veces estas proveen tantas funcionalidades que en su mayoría no son utilizadas, y lo único que hacen es que la aplicación final sea muy pesada. En el presente proyecto más del 90 % de las librerías son de creación propia, del restante 10 %, 5 % están basados en librerías de terceros (Goaamb Framework) y el otro 5 % es en su totalidad librerías de terceros (CKEditor and CKFinder).

Esta relaciónde porcentajes muestra el grado de independencia que *i*CMS tiene con respecto a librerías de terceros, con lo que se consiguió que la aplicación sea liviana.

Cada vez que se considere el uso de alguna librería de terceros, es necesario tomar en cuenta sus implicaciones, es decir ¿Cuánto de esas librería va a ser utilizado?, si el porcentaje es mínimo, es preferible crear una librería liviana que responda a las necesidades elementales.

7.2.3. Sobre las herramientas usadas durante el desarrollo

Una de las experiencias más enriquecedoras logradas durante el desarrollo de *i*CMS, fue gracias a las distintas herramientas utilizadas durante el desarrollo del mismo.

Entre ellas tenemos el Entorno Integrado de Desarrollo (IDE) *Zend Studio for Eclipse*, está herramienta está basada en el IDE Eclipse donde Zend la personalizó hasta tener una herramienta exclusivamente para trabajar con el lenguaje PHP. Cuenta con un poderoso depurador que puede ser utilizado mientras la aplicación se ejecuta, lo cual facilitó enormemente la detección de errores que solo suceden en tiempo de ejecución, consiguiendo que la aplicación sea lo más robusta posible.

Otra de las herramientas fue el diagramador "Enterprise Architect", utilizado en el diseño de los distintos modelos de clases, siendo que no solo cuenta con el diagramador de clases sino que también da soporte a una infinidad de modelos tanto UML como otros; estás características la convierten en una poderosa alternativa a la hora de escoger una herramienta de este tipo.

7.3. Recomendaciones

7.3.1. Sobre las herramientas usadas durante el desarrollo

Cuando se seleccionan las herramientas para el desarrollo de cualquier proyecto de software siempre hay que considerar, que la herramienta este diseñada para el lenguaje que uno haya elegido, también considerar las facilidades que provee, Ejm. depurador, manipulación de componentes gráficos, etc.

7.3.2. Sobre los posibles trabajos futuros a *i*CMS

Entre los posibles trabajos a realizarse tomando como base el presente proyecto podrían ser:

- Desarrollar el módulo para la instalación/desinstalación de extensions (módulos y bloques), para las plantillas ya fue implementada, aunque como en todo sistema de software siempre puede ser mejorado.
- Mejorar el módulo base para la personalización de plantillas, si bien en este momento cumple con su cometido, quizás sea bueno pensar en una arquitectura mejor definida para este fin.
- Plantear una mejora a las extensiones, para utilizar el patrón Modelo-Vista-Presentador en vez del patrón Modelo-Vista-Controlador.
- Desarrollar extensiones para la gestión de boletines informativos, galerías de imágenes.

Referencias

Corporation Oracle. (2012). *MySQL.* http://www.mysql.com/about/. ([En Linea; accedido Septiembre-2012])

Deitel, H. M., y Deitel, P. J. (2004). *JAVA How to Program, Introduction to classes and objects.* Prentice-Hall.

Echevarría Alvaro Martínez. (1995). *Manual práctico de HTML.* http://www-app.etsit.upm.es/~alvaro/manual/manual.html. ([En Linea; accedido Septiembre-2012])

Foundation The Apache Software. (2011). *Server HTTP Apache.* http://httpd.apache.org/ABOUT_APACHE.html. ([En Linea; accedido Septiembre-2012])

GangOfFour) Erich Gamma Richard Helm Ralph Johnson John Vlissides (the. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley.

Gerner Yann le Scouarnec Elizabeth Naramore Gary Mailer Jeremy Stoz Jason. (2004). *Fundamentos de Desarrollo WEB con PHP, Apache y MySQL.* Editorial Anaya.

Lie Hakon Wium, y Bos Bert. (1999). *Cascading Style Sheets, Designing for the WEB.* Addison Wesley.

Pérez Javier Eguíluz. (2011a). *Introducción a AJAX.* http://www.librosweb.es/ajax. ([En Linea; accedido Septiembre-2012])

Pérez Javier Eguíluz. (2011b). *Introducción a JavasCript.* http://www.librosweb.es/javascript. ([En Linea; accedido Septiembre-2012])

Robertson, J. (2003). *So, what is a content management system?* http://www.steptwo.com.au/papers/kmc_what/index.html. ([En Linea; accedido Septiembre-2012])

Sommerville, I. (2002). *Ingenieria de software.* Naucalpan de Jurares, México.

W3C. (2011). *Extensible Markup Language.* http://www.w3.org/XML/. ([En Linea; accedido Septiembre-2012])