Universidad de Mendoza
Facultad de Ingeniería
Licenciatura en Sistemas

“GESTOR DE CONTENIDOS MVC”

CREACIÓN DE UN GESTOR MVC ORIENTADO A COMPONENTES Y MÓDULOS DRAG&DROP CON SISTEMA DE CACHÉ

Autor: Borja Abad López
Legajo: 5544
Profesor titular: Ing. Osvaldo Marianetti
Año: 2013

AGRADECIMIENTOS

La Universidad de Mendoza, fue la tercera universidad a la que accedí en mi vida. En las dos ocasiones anteriores, por circunstancias personales, no pude completar mi formación. Con 25 años decidí una vez más intentarlo, y como dicen, a la tercera va la vencida.

Ahora ya con 30 años y a punto de recibirme, quiero animar a toda la gente que por circunstancias no haya podido completar sus estudios, a que lo hagan, merece la pena el esfuerzo.

Agradecimientos a la Dra. Ing. Cristina Párraga quien me animó a inscribirme, el primer día que accedí a preguntar a la secretaría de la Facultad de Ingeniería.

Agradecimiento obligado a mis padres por todo su apoyo y por transmitirme la importancia de la formación. También a toda la gente que me ha apoyado. A profesores, compañeros y amigos que lo han hecho posible. Gracias.

ÍNDICE DE CONTENIDOS

INTRODUCCIÓN

Todo empezó por una idea, el patrón *MVC*. He conocido decenas de códigos *Open Source*, y siempre me ha llamado mucho la atención la manera en que estructuran sus archivos, como llaman a los métodos, como organizan sus variables, etc. Si algo detestaba siempre era el afrontar un nuevo proyecto de código desconocido, y el árduo camino que era necesario recorrer hasta desentrañar sus misterios y hacerse con el código.

Al conocer el patrón *modelo-vista-controlador,* entendí que cualquier proyecto, por diferente que fuera, siempre que lo implementara, podría seguir siendo fácilmente comprensible, más allá del lenguaje de programación empleado.

La elección de esta temática en mi trabajo final, no es casual, viene de atrás. Hace años que trabajo en una empresa cuyo principal producto es un gestor de contenidos (*CMS*), Editmaker, que desarrollan desde hace 15 años.

Esto me ha ayudado a comprender los problemas y necesidades en la creación de software a largo plazo, que evoluciona, que crece y se complica.

En mi formación universitaria también he aprendido la necesidad de normalizar y regir las comunicaciones que genera el desarrollo de software en equipo. El proceso unificado de la creación de software inspirado en la organización de Microsoft, las *metodologías ágiles* orientadas a la productividad, comunicación y trabajo en equipo, y, la necesidad de usar patrones, técnicas de abstracción y otras arquitecturas que de alguna manera permitan serializar o estandarizar la creación de un software concreto, han sido valores que me han generado la necesidad de integrar, o al menos intentarlo, todos estos conocimientos en este trabajo.

La creación de un CMS, implica aparatosidad. Son aplicaciones pesadas bastante complejas. Clonar decenas de instancias (sitios webs) en un solo servidor, va acompañado de seguro de una gran saturación en el rendimiento del mismo.

La necesidad de crear un *framework* ligero me hizo descartar todos los CMS que existen en el mercado (llevan años evolucionando y son muy completos), y tratar de empezar uno desde abajo, de cero, teniendo en cuenta desde su concepción la importancia de que sea ligero.

Un factor en común entre los existentes en el mercado, es que son *PHP*, usan MVC y en su mayoría son orientados a componentes.

Me ha ayudado mucho seguir los cursos de Jaisiel Delance, un dominicano dedicado a la enseñanza online de tecnologías que tiene un canal de Youtube, cuenta de Twitter, perfil de Facebook y página web propia, dlancedu.com. Experto en el uso de CMS Open Source, ha rescatado lo básico y común a todos, haciendo una base realmente ligera y potente que valoro enormemente y he adoptado como punto de partida.

Entiendo que la creación de un proyecto de este tipo debería contar con el trabajo de un equipo, así como meses de planeación y desarrollo. Todo depende de la ambición de la iniciativa. Si bien soy consciente de este límite, en esta tesis, me gustaría al menos poder sentar las bases de una arquitectura válida para un sistema con los requerimientos planteados, que sea escalable, ligero, preparado para el trabajo en equipo, etc.

RESUMEN

El trabajo final será una bitácora de experiencias, caminos, aprendizajes y conclusiones surgidas en la creación de la arquitectura fundamental de un CMS.

Se trata de desarrollar una aplicación con su *backend* para administrar el *frontend*. La estructura del CMS permitirá el desarrollo en equipos por ser orientado a componentes.

Por un lado, el back, ha de ser robusto y no importa si pesado, confiable, escalable, etc. El tema de orientarlo a componentes, significa que una vez establecido el núcleo de la aplicación cualquier desarrollador pueda agregar componentes si está bien definida la *interfaz*.

Por otro lado, el *front*, ha de ser simple, y sobre todo muy liviano, ya que es el que sufre la mayor carga en el servidor. También será orientado a módulos, los cuales agregados, son los que conforman las páginas.

Se profundizará en el concepto página, se desarrollará un sistema *Drag&Drop* para que desde el back de puedan crear las páginas arrastrando módulos dinámicos, estáticos o propios sobre ellas.

Se hará manejo de los *buffers* del servidor web para implementar un sistema de caché bajo demanda apoyada en el sistema de archivos, con los *outputs* procesados como archivos planos, tratando así de demostrar que un sistema como el descrito puede ser de mayor rendimiento sin aumentar notablemente la complejidad del mismo y ni que la experiencia del usuario se vea afectada.

Metodología

Por trabajar solo y no en equipo, he podido ahorrar mucho tiempo en la comunicación con diagramas e información del sistema, puesta en común, etc. En grandes organizaciones, el uso de herramientas de modelado *UML*, o la producción bajo el concepto *RUP*, se vuelve imprescindible, ya que en la práctica a veces el programador no tiene ni idea de para qué programa.

Se ha hecho uso del lenguaje de programación *PHP5*, apoyado en la *BBDD Mysql* para afrontar las funcionalidades y objetivos planteados en esta tesis. Ésta ha sido mi principal herramienta, y aunque he estado más cerca del *Extreme Programming* que del *RUP*, he incluido varios diagramas de análisis y diseño.

El desarrollo ha sido espejado en *GitHub* como central de versionado.

Avance establecido

Se parte del patrón MVC base de Jaisiel Delance. Se define el proyecto como la adición de dos aplicaciones, una el Front y otra el Back.

Empiezo por el Back, agregando una capa de abstracción al sistema con el recurso Componentes. Se definen algunos básicos y se genera una interfaz para su creación.

En el Front, se introduce en el concepto de página, y por tanto el de módulo. Se usa una librería drag&drop para la creación de páginas en el back. Se adapta al sistema y se le agrega la posibilidad de incluir módulos dinámicos. Por último, y en base al concepto página, se desarrolla un sistema de caché bajo demanda.

CAPÍTULO I - ESCENARIO

PLANTEAMIENTO DEL PROBLEMA

La mayoría de las pequeñas y medianas empresa productoras de software suelen producir sistemas y proyectos de pequeño y mediano alcance.

En el mercado existen soluciones de software para casi todo. En concreto para la gestión de contenidos hay algunos muy extendidos, hasta tal punto que muchos de los sitios web por los que solemos navegar esconden detrás un *CMS*. Existen algunos especializados en el sector editorial, otros en sector audiovisual, también hay *CMS* para *bloggers*, o para anuncios de clasificados, etc.

No hay duda de que los CMS son una tendencia al alza. En el caso de los *open source*, con las nuevas versiones cada vez son más modulares, se pueden agregar o eliminar funcionalidades, se forman comunidades de gente que colabora, agregando fragmentos o tomando otros y sirven de solución a millones de empresas y particulares.

Hoy en día cualquier *PYME* sabe que necesita organizar su información en torno a sistemas informáticos fiables, y además con el crecimiento de la web, también quieren poder gestionar esa información *online*.

Entiendo, que estos CMS existentes en el mercado, son por lejos unos gigantes, en cuanto a volumen de código empleado en su desarrollo, horas/hombre invertidas, comunidad de apoyo y sobre todo número de sitios reales que los usan.

Estos CMS poseen un gran nivel de personalización, se ajustan a casi todo. Además, tiene una interfaz muy bien definida que permite agregar módulos, componentes o *plugins*, programados a medida según las necesidades.

Desde el punto de vista del desarrollador, usar estas herramientas para la construcción de sistemas en la web es sin duda algo muy positivo y ventajoso.

Desde un punto de vista empresarial de los **productores de software**, la perspectiva cambia. Adoptar este tipo de herramientas como la base de sus desarrollos a medio y largo plazo puede convertirse en un lastre, son muchos los riesgos y **problemas** que podrían encontrar:

1. Los CMS han evolucionado tanto que son realmente completos, y por tanto **complejos**. Es posible que para un determinado proyecto web, se ajusten bien, pero para sistemas más grandes posiblemente sea necesario modificar ciertos comportamientos, que con un software de tal magnitud serían muy altos los recursos a invertir necesarios para su modificación.

2. El core de la aplicación que pudieran desarrollar, estaría atado a terceros, y esto en productos de software es una incongruencia muy limitante. Tendrían que actualizar sus sistemas según el calendario de otra organización, y la obsolescencia que pudiera imponer el tercero en su producto, afectaría al que lo consume, el productor de software. En definitiva, su **versionado** sería **esclavo**, o quedarían atados a una versión obsoleta.

3. Son aplicaciones que consumen muchos recursos de los servidores que las alojan, por los tanto la implementación en serie de éstas supondría **altos niveles de**

consumo, aunque éstas no contuvieran un modelo de negocio complejo o con un número alto de requerimientos.

4. **Son focos de ataques informáticos**. Al ser código libre, con estudio y esfuerzo, cualquiera puede conocer sus misterios, es por ello que diferentes individuos u organizaciones con diferentes propósitos consiguen hackear sistemáticamente estos sistemas, obligando la organización productora del software a sacar parches de seguridad y nuevas versiones. No todas las nuevas versiones incluyen mejoras, sino también bug´s resueltos.

Preguntas

¿Qué software debe usar como base un productor de software de sistemas web en sus proyectos?

¿Es correcto hacer un sistema a medida en lugar de tomar uno del mercado?

¿Un único sistema puede ser la base de casi todos los proyectos web de un productor de software?

La respuestas a estas preguntas en ningún caso podrán ser categóricas. Todo dependerá de quien la responda y de cuáles sean sus experiencias y necesidades. En los anexos, se podrán consultar las respuestas de algunos miembros de la comunidad de desarrollo a estas preguntas.

Objetivos

En este trabajo, intentaré probar si un software hecho a medida hecho desde abajo, puede tener ventajas arquitectónicas como las tienen los CMS más destacados del mercado.

Objetivo principal:

Presentar una estructura mayormente válida para la creación de un software propio que sirva como base a una empresa productora de software a modo de gestor de contenidos. Generar una interfaz de carga de páginas visual en el backend diferente a las del mercado, tratándola de hacer más natural e intuitiva. Configurando la página y el contenido que puede alojar, en lugar de que contenido va en qué página.

Objetivos adicionales:

Implementar un sistema sencillo de caché bajo demanda. Y que no sea caché por expiración de tiempo.

Usar componentes en el back, para encapsular funcionalidades.

Usar módulos en el front, para pintar las páginas.

Mantener esquema MVC en la aplicación.

Definir interfaces para la creación de módulos y componentes.

Límites

El número de funcionalidades de la aplicación estará muy limitado, la cantidad de componentes o módulos implementados será pequeña, así como funciones de la aplicación.

El programa informático resultado de este estudio, no será muy complejo por cuestiones de recursos, pero trabajado, con el tiempo, puede llegar a ser una base de software o marco de trabajo para pequeños y medianos productores de software.

Ventajas

Cuando trato de equiparar las ventajas de un software de producción 'local' con la de los grandes del mercado, me refiero a las principales ventajas de su arquitectura:

1. Suelen ser orientados a **componentes**, encapsulando funcionalidades, y con posibilidad de agregarlos o eliminarlos al gusto. Y permite el trabajo en equipos.

2. Cuentan con sistemas de **caché**, todos ellos caché de expiración por tiempo.

3. Son altamente **escalables** y configurables.

4. Implementan el patrón **MVC** en su concepción.

5. Usan gestores de estilos como **themes o plantillas**.

Viabilidad

Creo que si se consiguen reunir las principales ventajas de su arquitectura comunes a los CMS existentes, en un software de producción propia, que pueda servir como base para casi todos los proyectos que pueda tener un pequeño y mediano productor de software, se habrá conseguido una herramienta poderosa y maleable para hacer crecer su organización.

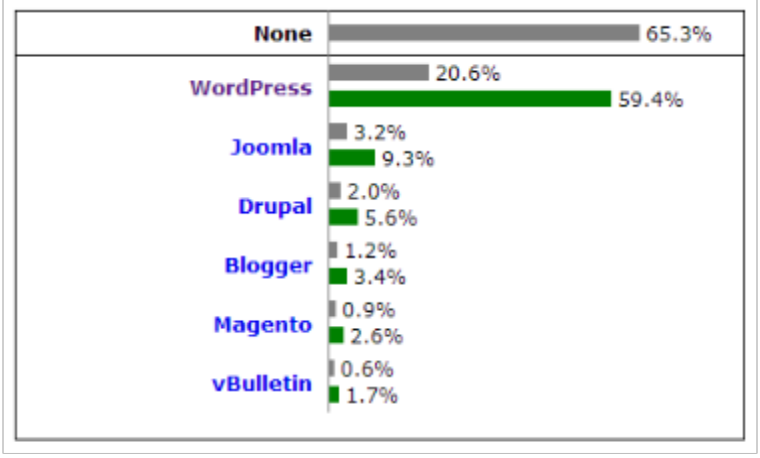
Datos concretos

Penetración de los CMS

Según la *W3techs* [1] el 65.3% de entre los 10 millones de sitios analizados no usa ningún gestor de contenidos.

Por tanto, si decimos que los CMS conforman en torno al 35% de todas las web mundiales, inmediatamente observamos la relevancia de estas herramientas.

En el siguiente gráfico vemos como Wordpress, Joomla y Drupal lideran el mercado, hasta tal punto que la relevancia a nivel mundial solo de Wordpress es apabullante. La medición verde se refiere al total dentro de los gestores de contenidos, pero la medición gris corresponde al total de las webs del mundo, así es pues decir que Wordpress acapara más de un 20% de las webs del mundo es mucho decir.

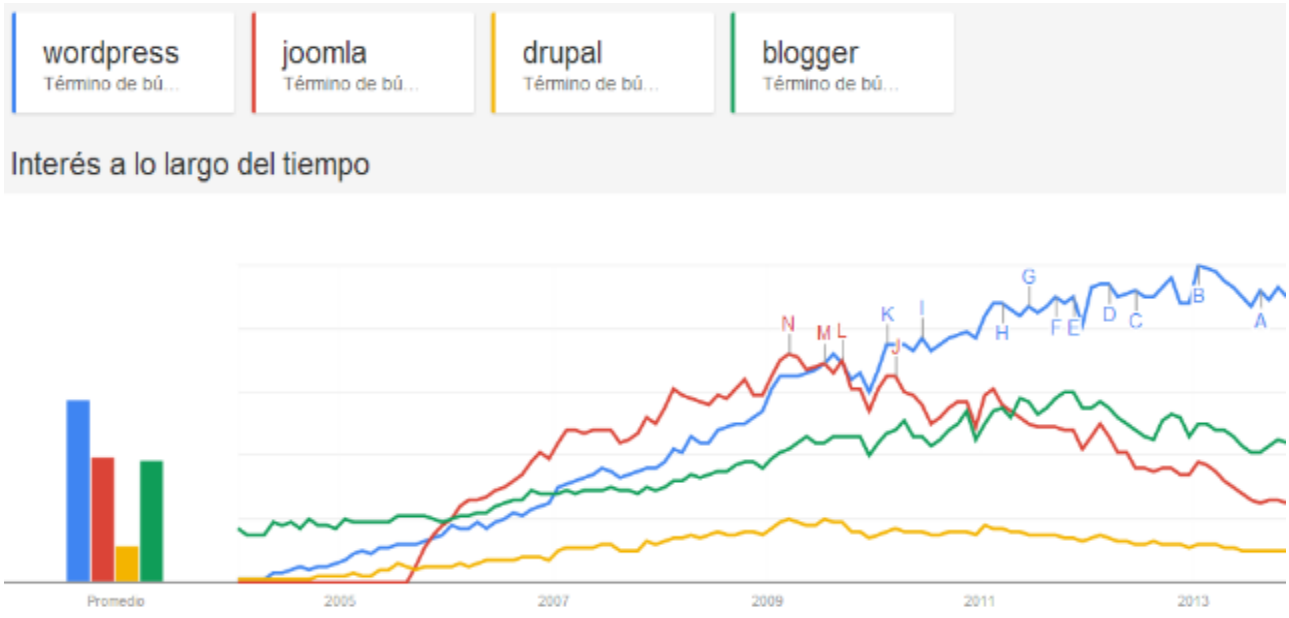


Fuente: http://w3techs.com/technologies/overview/content_management/all

[1] El sitio w3techs.com provee información acerca del uso de varios tipos de tecnología en la web. Aunque es una empresa privada (como casi todas las que mueven internet), promete información veraz y dice no tener ningún tipo de afiliación con los proveedores tecnológicos que aparecen en sus informes. Investigan la tecnología de los sitios web, no páginas concretas. Para hacer el estudio han incluido las 10 millones de páginas más vistas. A diferencia de Alexa (una compañía adquirida por el gigante Amazon, que lidera la medición de tráfico, visitas y estadísticas web a nivel mundial), W3techs no considera, a los subdominios como sitios diferentes, sino que los considera como el mismo sitio.

Evolución del interés en los CMS

Otros datos que pueden ser analizados, demuestran una vez más la magnitud e inclusión de estos programas en la sociedad actual. Son los datos provenientes de la evolución en **términos de búsqueda** de palabras claves relacionadas con el sector, se usará *Google Trend* [2] para llegar a estas conclusiones mostradas en el siguiente gráfico.:



A pesar de que Joomla muestra una tendencia a la baja, vemos que todos tienen valores de crecimiento positivos desde el comienzo de la gráfica en el año 2005. Y es que los CMS son un pilar de la web moderna.

[2] Google Trend es una herramienta de Google Inc. que provee información sobre las diferentes tendencias y su evolución de los diferentes términos de búsqueda, usados en su buscador, gracias al cual obtienen la información. Entendemos a Google como el buscador de referencia para nuestro estudio.

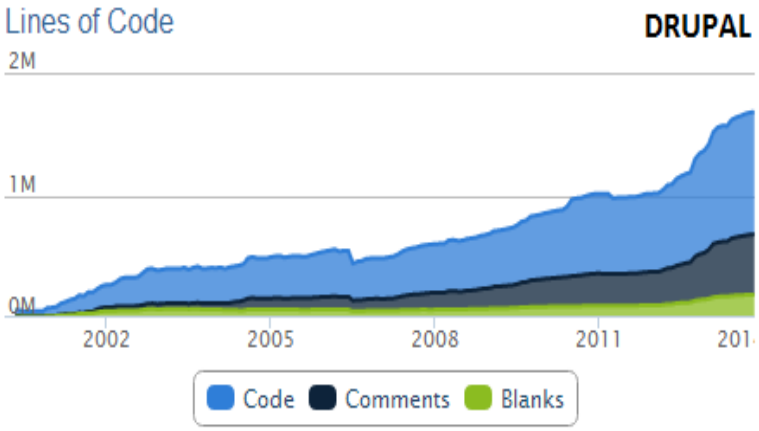
Rendimiento de los CMS

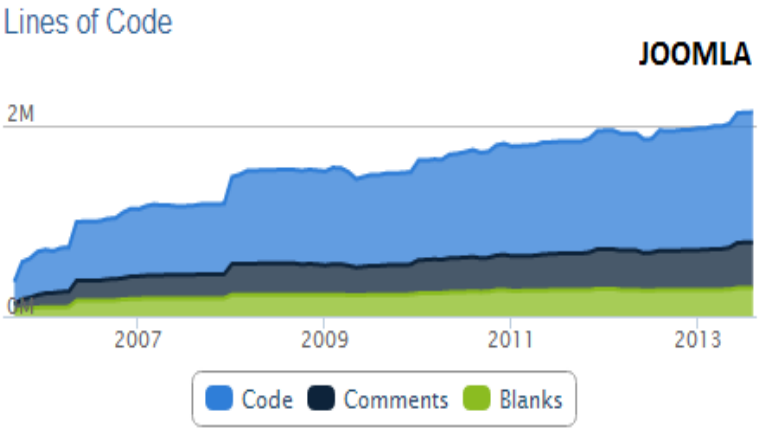
No hay duda de que los CMS han crecido exageradamente en código, a veces pueden volverse monstruosos para según que proyectos.

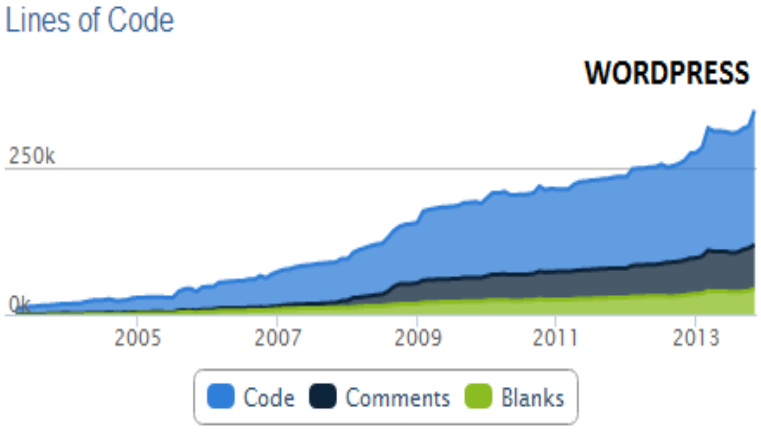
Usando la herramienta provista por Ohloh.net [3] podemos llegar a conclusiones muy interesantes.

Se puede observar como todos los CMS han aumentado sustancialmente el número de líneas en los últimos años, razón más para entender que son aplicaciones cada vez más completas, y por tanto pesadas.

Y esto es sin contar a la comunidad que aporta y colabora, estos gráficos se han realizado sólo teniendo en cuenta a la empresa que produce el core del CMS.







[3] Ohloh.net es un sitio web que permite comparar proyectos Open Source. Usa *trackers* para rastrear los *repositorios* públicos donde éstos se alojan para así obtener información como el número de contribuciones a cada proyecto, líneas de código, etc.

Problemas de seguridad en los CMS

Haciendo una búsqueda por la cadena “hack joomla” en *google.es* se obtienen más de tres millones de resultados. Aparecen cientos de post, de gente que ha encontrado vulnerabilidades de seguridad, y explica como puedes reproducirlas.

Haciendo la búsqueda con la cadena “parche seguridad joomla” se obtienen casi 30.000 resultados. Se pueden ver cientos de enlaces a páginas que ofrecen parches de seguridad para las diferentes versiones.

Caché

El principal problema que he detectado en los CMS del mercado, es que todos usan una caché basada en el tiempo. En sitios de noticias donde rápidamente actualizan la información, y en muchas ocasiones los textos se editan en caliente, la caché puede ser un problema, que provoque que los usuarios que visitan la web vean contenido desactualizado.

Es cierto que los CMS permiten vaciar la caché, pero es un proceso manual que depende del entrenamiento del usuario, y por tanto muchas veces genera problemas.

CAPÍTULO II - MARCO TEÓRICO

ANTECEDENTES

Un sistema de gestión de contenidos (Content Management Systems) es un software que se utilizar para el almacenamiento, edición y publicación de información, normalmente son aplicaciones web, aunque pudiera usarse en una *intranet* igualmente.

Permite que multitud de personas puedan acceder a comunicarse a través de Internet, sin necesidad de que tengan conocimientos técnicos, como lenguajes de programación.

Los CMS aparecen a finales de los 90 con la popularización de Internet, cubriendo las nuevas necesidades que se iban generando. Los grandes editores del papel, empezaban a sacar su versión web, y rápidamente se dieron cuenta que tendrían que desarrollar sistemas que les permitiesen actualizar continuamente sus contenidos.

El boom llegó con *Php Nuke*[4], cada vez más gente tenía conexión a internet, salían los planes de hosting baratos y todos querían comunicar y aparecer en la red.

Que fueran los editores de diarios y revistas los precursores de los primeros CMS, no impide que, su uso se haya extendido a todos los sectores, siendo hoy día estas herramientas un pilar del Internet actual.

Actualmente en el mundo hacen uso de ellos todo tipo de empresas, individuos, universidades, organismos oficiales y ONG´s.

[4] Php Nuke fue el más popular de los primeros CMS, también el primero que generó una gran comunidad. Más tarde llegarían otros como Mambo del cual surgió Joomla.

Logos de CMS´s populares.

Existen cientos de tipos, muchos gratuitos (los más populares), y también muchos privados. Existen CMS específicos para cada sector, CMS estandarizados, CMS a medida. Los hay con diferentes tecnologías, para diferentes usos.

BASES TEÓRICAS

Funcionalidades básicas

Basándonos en el esquema planteado por James Robertson[5] podemos decir que cualquier CMS debería poder subdividirse en estas cuatro funcionalidades básicas:

1. **Creación**: esto resumiría todos los formularios de carga de información y los procesos que los hacen funcionar. Ya sea para la inclusión de una noticia, video, galería, etc.

2. **Gestión**: toda la información creada debe poder gestionarse, es decir, editarse, eliminarse, modificarse. Además tiene que ser posible modificar algunos elementos estructurales, textos o páginas que aparecen en la web.

3. **Publicación:** debe ser posible publicar o despublicar elementos. Algunos CMS lo hacen por fechas, por ediciones, u otros factores. La información que no se publica se archiva y está disponible cuando se requiera.

4. **Presentación**: esto hace referencia a todo el marco y estructura de la web, donde se aloja el contenido. Ej: Puede ser que un CMS concreto tenga una versión para móviles y otra de escritorio.

[5] James Robertson es el fundador y Director Ejecutivo de *Step Two Designs*, una empresa Australiana que tiene como producto un CMS especializado en *intranets*. Se define como especialistas de los gestores de contenido, es fundador miembro de la asociación profesional de gestores de contenidos (*CM Professionals association*) y ha ayudado a muchas organizaciones a elegir el CMS adecuado a sus necesidades. *http://www.steptwo.com.au*

Clasificaciones

Si es por **lenguaje** utilizado, la mayoría están escritos en *Php,* aunque hay muchos en *Asp, Java, .net, Ruby, Perl, python*.

Por su **licencia** podríamos resumirlos en *OpenSource* y *privados.*

Por su **uso** tenemos CMS pensados para *blog´s*, foros, publicaciones, tiendas, redes sociales, etc.

Frameworks y CMS

Un *Framework* proporciona un marco de trabajo sobre el que construir tu aplicación. Tal como una caja de herramientas sirve al mecánico, los *frameworks* ayudan a los programadores en la creación de aplicaciones, otorgándoles en su entorno de desarrollo librerías, funciones y otras soluciones.

Los frameworks te dan cosas hechas, pero no dan una base de la web. Hoy en día, hay *frameworks* que se acercan muchos a algunos CMS, esto es causado por una evolución de los mismos, aunque conceptualmente sean muy distintos.

Un CMS es más que un *framework* y seguramente su construcción está realizada sobre uno a medida. Se podría decir que, un CMS es un framework aplicado.

Existen determinados CMS, que pueden ser usados como frameworks porque tienen interfaces bien definidas que permiten a los programadores realizar sus aplicaciones como extensiones de estos CMS.

Persistencia

Con la evolución de Internet, han surgido diferentes sitios webs que ofrecen *Site Builders*, y de manera muy rápida y sencilla permiten generar webs sobre plantillas con estilos para contenidos estáticos. Podría decirse excepcionalmente que son CMS, aunque no usen BBDD y aunque sea muy simple la gestión que hacen del contenido.

Habitualmente, los CMS siempre han ido asociados a una base de datos para poder persistir la información, además de clasificarla deben poder permitir tener libre acceso a ella. Los motores de BBDD son tantos como ofrece el mercado, y según el tipo de CMS que los emplea.

En determinadas ocasiones para persistir o compartir determinadas configuraciones, pueden hacer uso de archivos de configuración. El manejo de imágenes y otro tipo de recurso visual normalmente se aloja en el directorio web.

Caché

La principal razón de la caché es ahorrar procesamiento al servidor. Hay que entender, que cada vez que una página es llamada, multitud de procesos se desencadenan, la ejecución de los script del servidor, con sus variables, funciones, llamadas, etc; la llamada a los recursos físicos del sistema de archivos; los accesos a la BBDD, alocaciones de memoria, etc.

El principio de la caché es bien sencillo. Toma el resultado que devuelve el conjunto de procesos que hicieron ejecutar ese requerimiento, y lo guarda en un archivo de texto plano en el sistema de archivos. En la próxima llamada que tiene al mismo requerimiento,

reutiliza el resultado devuelto en la llamada anterior, ahorrándose así, todo este procesamiento.

Plantillas o themes

Si algo en común tienen todos los CMS son las plantillas o *themes.*

Una plantilla hace referencia al continente donde se va a materializar el contenido y en conjunto conforman la interfaz o respuesta que se genera al usuario tras su requerimiento.

Una plantilla permite alojar el contenido que pueda generar el CMS.

A las plantillas está asociada la estructura. Así por ejemplo, cambiando una plantilla estaríamos cambiando la organización visual del contenido, así como además los estilos del mismo.

Los *themes* en cambio, son más básicos, no afectan a la estructura sino que sólo a los estilos. En el caso de los *themes*, la organización visual del contenido permanece constante.

CAPÍTULO III - METODOLOGÍA APLICADA

Hipótesis

Sería correcto que un productor de software de sistemas web pudiera elaborar un software a medida, en lugar de tomar uno del mercado. Un único sistema web que fuera la base de casi todos sus proyectos. Un sistema que de una manera visual permitiera organizar su contenido, usando un sistema de caché que no produzca errores de actualización y con un sistema de componentes que permita el trabajo en equipo.

Tipo de estudio

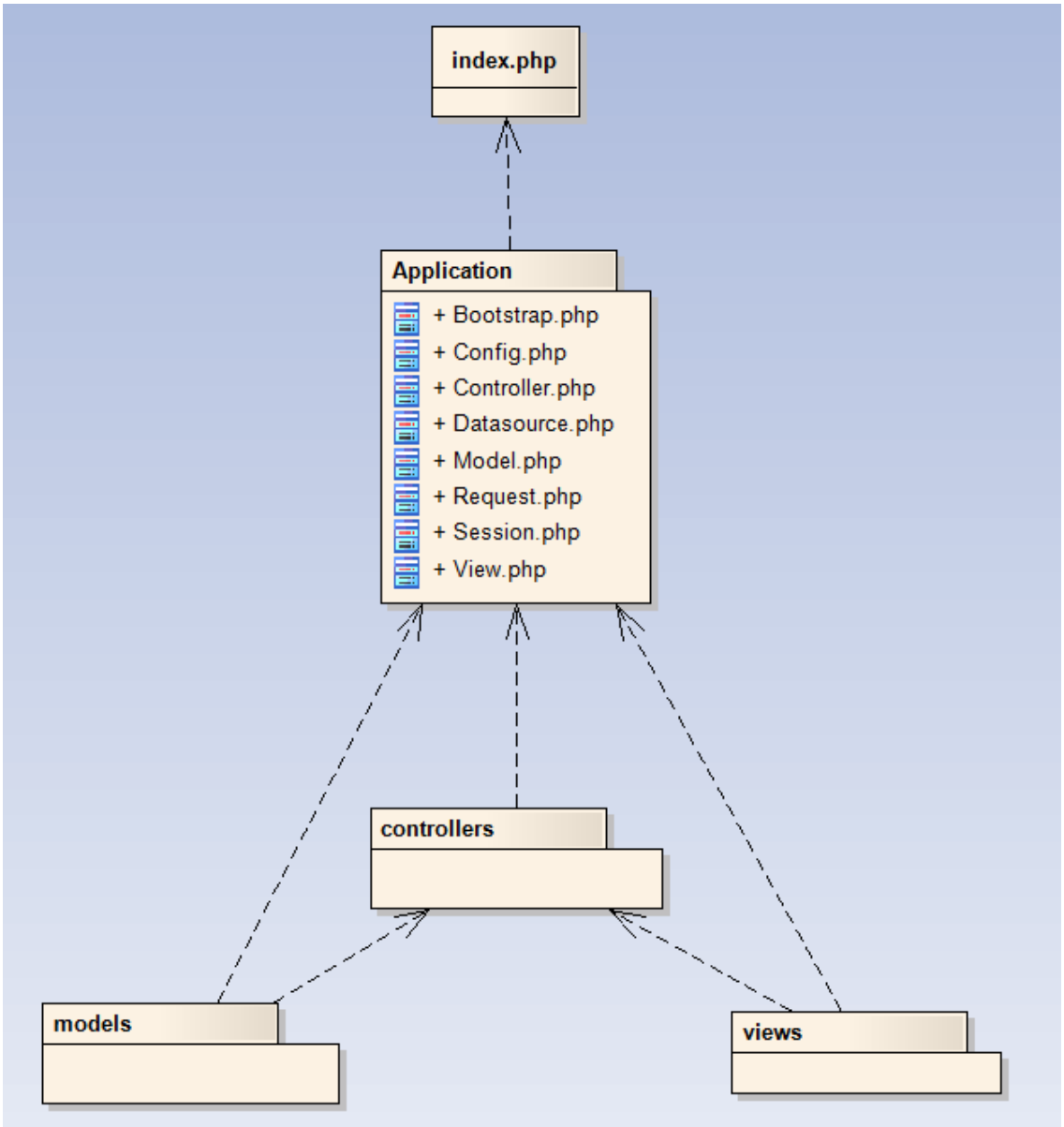
El tipo de estudio es propositivo. Trataré de formular una propuesta concreta al problema planteado, en la realización de una arquitectura básica del CMS.

El estudio se presenta en detalle en el CAPÍTULO IV.

CAPÍTULO IV - CREACIÓN DEL CMS

Base de la app

Como se dijo antes, partiré de la aplicación en lenguaje Php del patrón MVC propuesta por Jaisiel Delance, cuya estructura se resume en el siguiente diagrama:



Se puede observar perfectamente, cómo implementa el patrón modelo, vista, controlador.

Analicemos el index.php del sitio en cuestión:

```
<?php
try{
	define('DS', DIRECTORY_SEPARATOR);
	define('ROOT', realpath(dirname(__FILE__)) . DS);
	define('APP_PATH', ROOT . 'application' . DS);

	require_once APP_PATH . 'Config.php';
	require_once APP_PATH . 'Request.php';
	require_once APP_PATH . 'Bootstrap.php';
	require_once APP_PATH . 'Controller.php';
	require_once APP_PATH . 'Model.php';
	require_once APP_PATH . 'View.php';
	require_once APP_PATH . 'Session.php';

	Bootstrap::run(new Request());
}
catch (Exception $e){
	$e->getMessage();
}

?>
```

Lo primero que hace es *setear* unas constantes, que serán accesibles en toda la *app*. Luego agrega más constantes en el archivo *config.php*. Con los *require_once*, carga la aplicación para que sea accesible en el código.

La magia la hace el *Bootstrap*, que sería como el lanzador de la aplicación. Éste recibe el *Request*, y en base a éste, instancia el controlador indicado con el método requerido y los argumentos enviados.

Formato de url en el patrón MVC base

BASE_URL/controlador/método/arg/s

Si no se pasa ningún método se asigna el método index, si no se asigna ningún controlador se asigna el indexController.

El poder del *Bootstrap* se debe a la siguiente función, cuyos parámetros son provistos por el request:

```
call_user_func_array(array($controller,$metodo), $args);
```

Cuando se hace una petición a una página, el *Request* asigna el controlador, método y argumentos de la misma, y es el *Bootstrap* el que lo instancia.

Luego, y en base al patrón MVC, el controlador accede a los modelos y pasa los datos a las vistas, que conforman el *Response* para el usuario.

Otra peculiaridad que se observa frente a otros sistemas más tradicionales, que es toda la web siempre es el *index.php*, hace de **embudo**, canalizando todo el código de manera jerárquica. Todo pasa por el index.

Todo esto aquí analizado, resulta muy **práctico**. Dos aplicaciones que tengan modelos de negocios completamente diferentes, pueden mantener una estructura similar de archivos, organizaciones y funciones.

Significa, que cualquier programador que entienda el patrón, sabrá cómo está organizada la aplicación y acotará mucho sus tiempo de búsqueda y conocimiento del código, ya que le será muy familiar.

Además, con una base realmente ligera, de apenas 100 líneas de código, se pueden generar sitios web medianamente escalables.

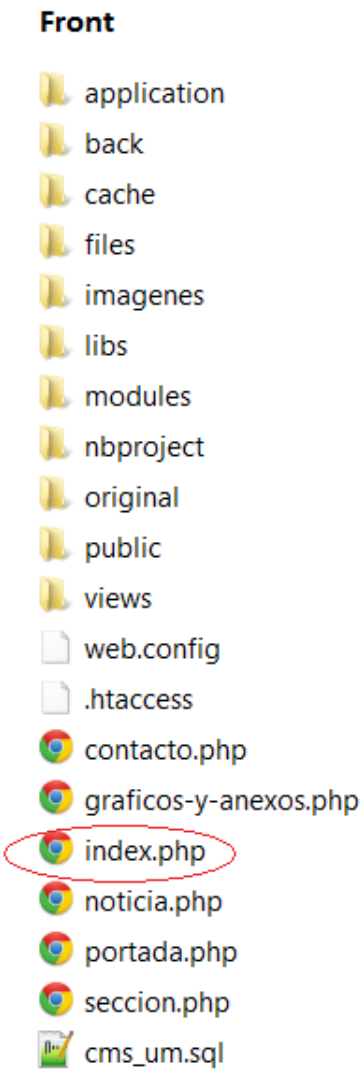
Entendiendo este concepto, como la **esencia** de lo que debiera ser una aplicación web, trataré de escalarlo para lograr los objetivos planteados.

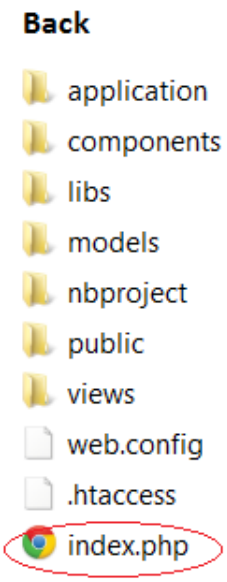
Dos aplicaciones

Como ya se sabe, los CMS son estructuras altamente complejas.

La idea de dividir un gran problema en problemas pequeños, creo que es la base de la informática. La capacidad de abstraer y encapsular diferentes funcionalidades, es lo que nos permite evolucionar los sistemas de las tecnologías de la información, que siempre se apoyan en lo establecido previamente.

Por eso considero una buena idea, aislar el *Front* del *Back*. Si se aplicara el patrón MVC en php propuesto para una aplicación que tuviera el típico *front* público, la web, y su administrador, el *back*, donde poder editar esa información, nos encontraríamos ante un pequeño problema.





Los controladores del front y del back estarían mezclados, también sus modelos y vistas. Al compartir sus modelos por ejemplo, compartirían todos los métodos. Es decir, que si alguien consulta la vista de noticia en el front, se está cargando un modelo que además de tener el método que proporciona la información para esa vista, también estaría cargando otros métodos que sirven para realizar las operaciones conocidas como CRUD o ABM (altas, bajas y modificaciones), o cualquiero otro método que fuera propio del back..

Esto no tiene sentido, bajaría la performance del sitio, y sería más confuso en lo que al código se refiere.

Adaptaré el patrón MVC propuesto en el Back, y otro en el Front. Por tanto se generarán **dos index.php** que cargarán aplicaciones diferentes.

BACK

El Back, los componentes

El back será la parte más robusta del sistema. No importa si carga muchas librerías, archivos, etc. Se sobreentiende que la frecuencia de cambio del contenido es mucho menor que la frecuencia de acceso al mismo (consultar Anexo, 'Relativización de los contenidos dinámicos').

La principal abstracción organizacional que planteo para el back, se define como **Componente**. Un componente tendrá la esencia del patrón MVC, como el primer ejemplo planteado, pero dentro de un marco de trabajo con más niveles. Un componente tendrá sus propias vistas y modelos, también podrá acceder a modelos de otros componentes si así se requiere, a las librerías comunes, a la aplicación cargada, etc.

Lo organización del código en componentes es de gran utilidad en la creación de sistemas complejos y escalables.

A pesar de que aumentar el número de capas de abstracción en nuestro código puede traer ventajas, también puede suponer la creación de más código, archivos y en general 'burocracia' del *framework* para empezar a trabajar. No cabe duda que cuanto más definida está la arquitectura de nuestro software, más reglas e interfaces tendremos que cumplir.

Aún así, sin duda son más las ventajas que inconvenientes en contextos como el mencionado:

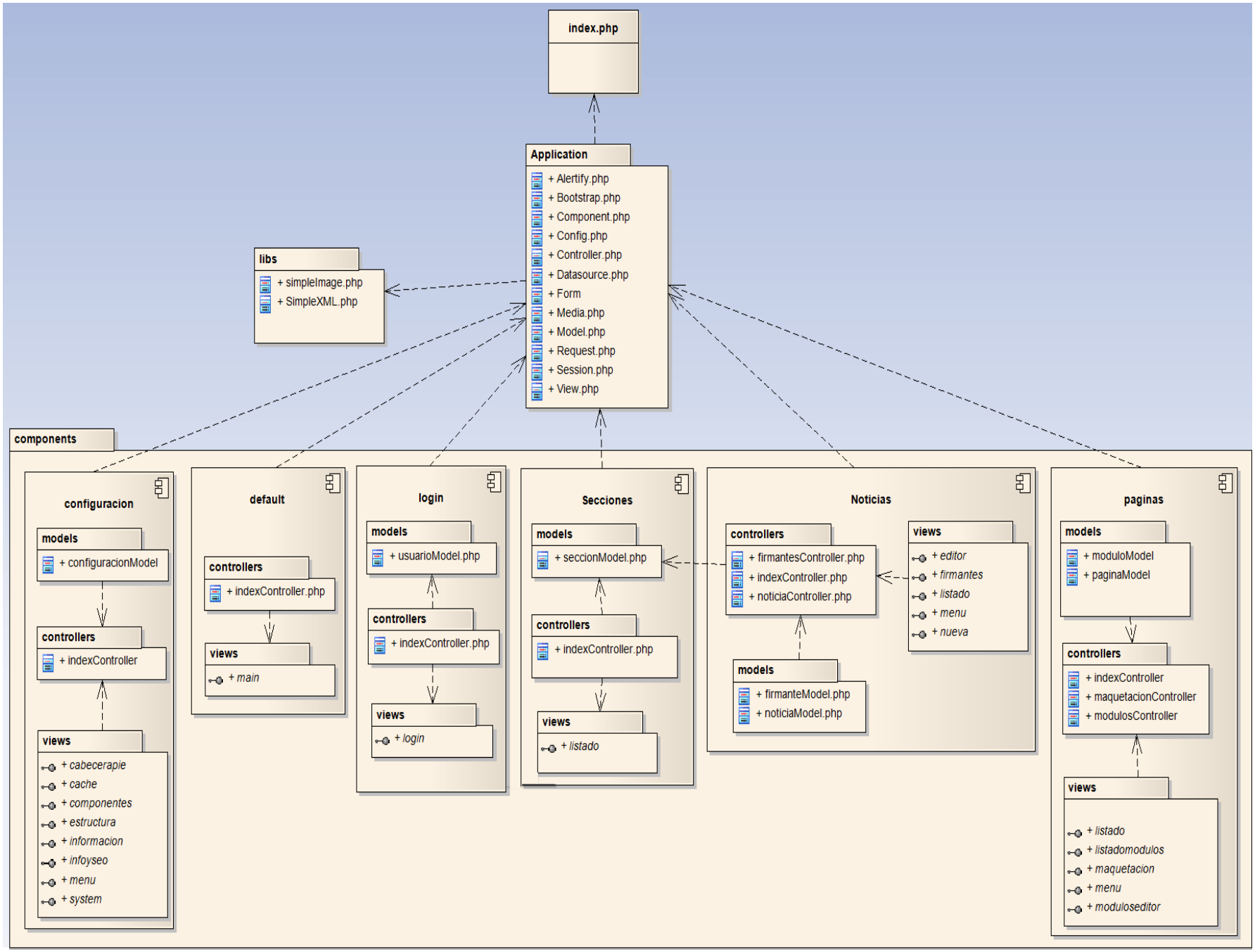
- Cada componente se comporta como una mini aplicación en sí misma, cumpliendo las reglas del entorno sí, pero con su **funcionalidad propia y encapsulable.**

- **Permite el trabajo en equipo**, ya que diferentes miembros pueden trabajar en diferentes componentes de manera independiente, haciendo crecer el sistema en paralelo.

- Mantener **diferente versionado** en cada componente. Se puede actualizar el *front* sin afectar al *back*, o actualizar componentes determinados sin afectar la estabilidad del resto de la aplicación.

- No perder el foco. Si el modelo de negocio resulta ser desarrollar un software en una línea concreta, evitar que las peticiones de los clientes hagan de la aplicación un parche encima de otro, desarrollar componentes a medida y **activar o desactivar su funcionamiento**.

- **Reutilizar componentes.** Se puede generar un sistema a medida activando y desactivando componentes. Aunque las necesidades de dos supuestos clientes pueden parecer muy diferentes, desde el punto de vista de las soluciones tecnológicas y del software, seguro tiene muchas cosas en común, y la propuesta es reutilizar esas funcionalidades análogas que no forman parte de su modelo de negocio.


index.php
Application
+ Alertify.php
+ Bootstrap.php
+ Component.php
+ Config.php
+ Controller.php
+ Datasource.php
+ Form
+ Media.php
+ Model.php
+ Request.php
+ Session.php
+ View.php
libs
+ simpleImage.php
+ SimpleXML.php
components
configuracion
models
+ configuracionModel
controllers
+ indexController
views
+ cabecerapie
+ cache
+ componentes
+ estructura
+ informacion
+ infoyseo
+ menu
+ system
default
controllers
+ indexController.php
views
+ main
login
models
+ usuarioModel.php
controllers
+ indexController.php
views
+ login
Secciones
models
+ seccionModel.php
controllers
+ indexController.php
views
+ listado
Noticias
controllers
+ firmantesController.php
+ indexController.php
+ noticiaController.php
views
+ editor
+ firmantes
+ listado
+ menu
+ nueva
models
+ firmanteModel.php
+ noticiaModel.php
paginas
models
+ moduloModel
+ paginaModel
controllers
+ indexController
+ maquetacionController
+ modulosController
views
+ listado
+ listadomodulos
+ maquetacion
+ menu
+ moduloseditor

Se puede ver en el anterior diagrama, un resumen de la arquitectura propuesta para el back.

En comparación con el gráfico inicial del patrón MVC, vemos una *Application* más cargada, la agregación de **librerías**, y sobre todo los **componentes**, organizados bajo el mismo patrón, pero como mini aplicaciones en sí mismas, que comparten un entorno, y que pueden **interrelacionarse**, como vemos en el caso del componente noticias, que importa el modelo secciones de otro componente. Creo que este planteamiento resulta muy conveniente por las ventajas ya comentadas.

API - Creación de un componente

Definición de las interfaces para la implementación de componentes

Nuevo componente

Debe agregarse a la lista del archivo 'components.xml' que se encuentra en la raíz del directorio componentes:

https://github.com/borjaabad/cms.um

El contenido de este archivo puede ser consultado en cualquier parte del código con la variable global $components.

Constantes definidas por el framework en la clase Component.php cuando se carga un componente y accesibles desde toda la app:

NAME_COM	name definido en components.xml (obligatorio)
ROOT_COM	Ruta física al directorio raiz del componente
DISPLAYNAME_COM	displayName definido en components.xml
MENU_COM	menu definido en components.xml
SQLPREFIX _COM	sqlprefix definido en components.xml
ENABLED_COM	enabled definido en components.xml

Cada componente tendrá por defecto el siguiente conjunto de directorios:

ROOT.DS.'components'.DS.'component'.DS.'controllers'

ROOT.DS.'components'.DS.'component'.DS.'models'

ROOT.DS.'components'.DS.'component'.DS.'views'

ROOT.DS.'components'.DS.'component'.DS.'public'

Para utilizar un componente bajo sesiones:

Session::control();

Utilizar en el constructor de los controladores de los componentes.

Si ho ha iniciado sesión se vuelve a BASE_URL

Componentes y Controladores

La estructura de URL completa y por defecto es la siguiente:

```
BASE_URL/componente/controlador/método/arg/s
```

Y las rutas:

```
/back/components/controllers/controlador.php

/back/components/models/model.php

/back/components/views/view.php
```

Ejemplo de un requerimiento: http://cmsum.com/back/login

```
/back/login/controllers/indexController.php

/back/login/models/usuarioModel.php

/back/login/views/index/login.phhtml
```

Nota: en el ejemplo anterior, se puede apreciar como la vista está anidada un directorio más cuyo nombre coincide con el del controlador que la invoca.
Si no existe el controller del componente busca el método en el indexController:

/back/componente/controlador/metodo/arg/s

Reproduce esto:

/back/login/index/validar/arg/s

Por esto:

/back/login/validar/arg/s

Componentes y Modelos

Para cargar un modelo de nuestro componente desde el controlador:

```
$modelo = $this-> loadModel(‘modelo’);

ROOT_COM.DS.'models' *

(*) ROOT_COM para referirse a la ruta del componente
```

Para cargar un modelo de otro componente desde el controlador:

```
$modelo = $this-> loadModelFromOtherComponent(‘modelo’,’componente’);
```

La carga de la conexión a la BBDD no se hace por defecto, ya que no todos los modelos tienen porqué tener acceso a datos, así que para definir si un modelo tiene acceso pone el siguiente código en el constructor del modelo:

```
public function__construct() {
        parent::__construct();
        $this->cargaBD();
}
```

Pudiera cargarse la BD en cualquier otro método del controlador, no tiene porqué ser en el constructor, todo depende de las necesidades.

Ejemplo de un método usando PDO:

```
public function getSecciones(){
        try {
            $secciones = $this->_db->query("SELECT * FROM secciones")

                return $secciones->fetchAll();
         }
        catch (PDOException $e) {
              echo $e->getMessage();
               return false;
         }
       }
```

Componentes y Vistas

Las vistas de cada componente, tienen acceso a diferentes funciones del framework, como por ejemplo es el sistema de mensajes o la clase media.

Para llamar a una vista desde el controlador de nuestro componente:

```
$this->_view->renderizar('vista');

$this->_view->renderizar('vista',true);
```

La bandera **true** agrega el header y footer por defecto de la aplicación:

Renderización de una vista:
ROOT .DS . views .DS . layout .DS . default .DS . header.php
ROOT .DS . components .DS . component .DS . views .DS . view . DS . vista.phtml
ROOT .DS . views .DS . layout .DS . default .DS . footer.php

Para enviar información a las vistas, las variables se asignan así en el controlador:

```
$this->_view->_variable = 'foo';
```

Para recibir información en las vistas, leemos las variables así:

```
echo $this->_variable;
```

Media

Para la distribución e inclusión de archivos públicos en nuestra vista , se ha creado la clase Media.php cargada en el núcleo de la aplicación.

```
<? php   Media::css();  ?>  // Carga los archivos css básicos del framework

<? php  Media::js();  ?> // Carga los archivos js básicos del framework

<? php  Media::addCss('estilos.css');  ?> // Agrega css específico a la vista

<? php  Media::addJs('main.js');  ?> // Agrega js específico a la vista

 Ruta por defecto:

    * components/component/views/public/css/archivo.css

    * components/component/views/public/js/archivo.js

Los archivos básicos del framework es recomentable cargarlos antes de la etiquet
</head>.
```

Cabe destacar que la inclusión de JS y CSS también se puede realizar por medio del archivo de configuración components.xml.

Sistema de mensajes

Usando la librería alertify.js y alertify.css como medio de envío de mensajes desde el código a nuestras vistas, se ha creado la clase Alertify.php en el directorio application, es decir, el núcleo del framework, para su gestión y manejo desde la vista de cualquier componente.

Para mostrar mensajes en una vista:

```
<?php  Alertify::mostrarMensajes();  ?>
```

Este código se usará en las plantillas de las vistas, solo es condición necesaria que esté antes de la etiqueta de cierre </body>.

Este código ya está implementado en el footer.php por defecto.

Para mostrar mensajes en una vista:

```
<?php  Alertify::add($mensaje,$tipo);  ?>
```

Este código se usará en las plantillas de las vistas, solo es condición necesaria que esté antes de la etiqueta de cierre </body>.

Los posibles tipos de mensajes son:

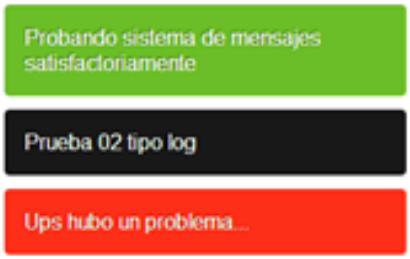
'success' - 'log' - 'error'

```
Alertify::add('Probando sistema de mensajes satisfactoriamente','success');

Alertify::add(('Prueba 02 tipo log','log');

Alertify::add(('Ups hubo un problema...','error');

Alertify::add(('login');
```



Creación del componente noticias

Este nuevo componente se realizará sobre la estructura básica del framework, que ya incluye un login, sistema de mensajes (alertify), gestión de media (js y css), ayuda en formularios, gestor de menú, etc. Le permitirá agregar, editar y eliminar noticias en el sistema.

Lo primero será agregar nuestro componente en el archivo components.xml para que el framework conozca lo necesario acerca del mismo:

```
<component>
        <name>noticias</name>
        <displayName>Noticias</displayName>
        <menu>true</menu>
        <menus>
                <menuItem>
                <name>main</name>
                <path>noticias/views/common/menu_main.phtml</path>
                <default>true</default>
                </menuItem>
         </menus>
        <sqlprefix>not_</sqlprefix>
        <enabled>true</enabled>
</component>
```

El nombre se usará para referirnos a él en el código y como identificador único. La opción menú, sirve para mostrar en el menú principal de la aplicación.

En la opción menuItem, podremos poner los menus que deseemos que use nuestro componente, y luego podrán ser definidos desde el controlador, solo deberemos setear la constante MENU con el nombre del menú que queramos cargar.

Archivos básicos:

Controllers	Models	Views	Tables
index	noticia	listadonoticias	noticias
firmante	seccion (extern)	editor	firmantes
	firmante	listadofirmantes	
		menu_componente	

Analicemos como va quedando el contenido del controlador principal de nuestro componente noticias:

```
<?php

class indexController extends Controller{

        public function __construct(){
        parent::__construct();
        Session::control();
        }

        //noticias/
        public function index(){
        $this->_view->renderizar('listado',true);
        }

        //noticias/nueva
        public function nueva(){
        $firmante = $this->loadModel('firmante');
```

```
        $seccion = $this->loadModelFromOtherComponent('seccion','secciones');

        $this->_view->_secciones = $seccion->getSecciones();
        $this->_view->_firmantes = $firmante->getFirmantes();
        $this->_view->renderizar('nueva',true);
        }
}
?>
```

Vemos cómo ejerce el sesión de control en el constructor. Si lo supera el método por defecto es index(), el cual llama a la vista listado.

El método nueva() ya es más interesante, vemos como carga los firmantes, los cuáles son un modelo del propio componente, y cómo también puede cargar modelos de otros componentes, como es el caso del componente sección. Aquí vemos pues una clara relación entre componentes.

Vemos un ejemplo de firmanteModel.php:

```
class firmanteModel extends Model{
        public $id;
        public $nombre;
        public $email;

        public function __construct(){
                parent::__construct();
                $this->cargaBD();
        }

        public function getFirmantes() {
                try {
                  $firmantes = $this->_db->query("SELECT * FROM firmantes") or  die(mysql_error().mysql_errno());
                        return $firmantes->fetchAll();
                }
                catch (PDOException $e) {
                        echo $e->getMessage();
                        return false;
                }
        }
```

```
}
```

Como resultado se llama a la vista, como por ejemplo nueva:

```
<form method="post" action="<?php echo BASE_URL.'noticias/nueva'; ?>">
<table>
		<tr>
		<td>
		<?php Form::select($this->_firmantes,'id','nombre','Firmantes'); ?>
		</td>
		<td>
		<?php Form::select($this->_secciones,'id','seccion','Secciones'); ?>
		</td>
		<td>Orden
		<input type="text" name="orden"  size="5"  maxlength="9"/>
		</td>
		<td></td>
		<td>
		<?php Form::button('Guardar noticia','positive','add'); ?>
		<?php Form::button('Ampliar noticia','regular','blue'); ?>
		<?php Form::button('Eliminar noticia','negative','delete'); ?>
		</td>
		</tr>
		</table>
		<h2>Título</h2>
		<textarea rows="1"  name="texto"></textarea>
		<h2>Subtítulo</h2>
		<textarea rows="1"  name="subtitulo"></textarea>
		<h2>Entradilla</h2>
		<textarea rows="4"  name="entradilla"></textarea>
		<h2>Texto</h2>
		<textarea rows="15"  name="texto"></textarea>
</form>
```

Vemos alguna otra función del framework, como la que proporciona la clase Form que nos ayuda a pintar elementos de formulario asociados a fuente de datos, como en el caso de los *select*, o no, como en el caso de los botones que simplemente agrega una imagen y estilo a los botones.

APUNTES

WYSIWYG

La idea de poder combinar la potencia de los grandes CMS con la de los grandes editores HTML, es sin duda un gran reto. WYSIWYG por sus siglas en inglés, lo que ves es lo que tienes, *what you see is what you get*, es mucho más que un acrónimo, supone un objetivo de performance para el usuario muy importante. Permite que los usuarios se adapten de manera intuitiva y segura con el sistema.

Hay programas WYSIWYG para hacer páginas web´s, algunos muy conocidos como el *Dreamweaver de Adobe*. También los hay a modo de editores modulares, que sirven de complementos a los desarrolladores para mejorar la entrada de datos del usuario.

Al emplear el término WYSIWYG en este proyecto, trataré de abordarlo desde un punto de vista integral, refiriéndome tanto a las entradas de datos aislados del usuario (editores modulares), como al de la maquetación de los contenido dinámicos, estáticos y estructurales, que mediante una facilidad arrastre y suelte, *drag&drop*, permitirá diagramar las páginas web de una manera rápida e intuitiva.

Desde un punto de vista social, creo que podría acercar un poco más este tipo de herramientas a la gente, si se sigue trabajando en este sentido.

Estandarización

El sistema puede ser implementado en servidores Unix, MacOS y Windows. Puede correr sobre Apache, IIS y cualquier otro servidor web que sea intérprete de php.

Deben tenerse en cuenta las reescrituras de *URL*, ya que para conseguir url´s amigables la funcionalidad la proporciona el servidor web, que atiende el requerimiento. En el caso de Apache, será necesario configurar un *.htaccess* y en el caso de IIS, un *web.config.*

La aplicación se presenta con ambas configuraciones, ya que son compatibles y cohexistentes.

Manejo de los componentes en el back

Los componentes podrán agregarse o deshabilitarse según las necesidades. Si un componente está deshabilitado no se cargará en el sistema.
De esta manera también es posible implementar un sistema ACL de usuarios, y configurar su acceso por componentes. Así por ejemplo el administrador tendrá acceso a todos, pero un periodista solo tendría acceso al componente noticias y secciones.

FRONT

El CMS, se presenta con los **componentes** noticias, secciones, páginas, default, login y configuración instalados en el back.

En el front, se empleará el uso de **módulos** para presentar unidades de información asociadas o no a un componente, y se usarán las **páginas** como contenedores de éstos.

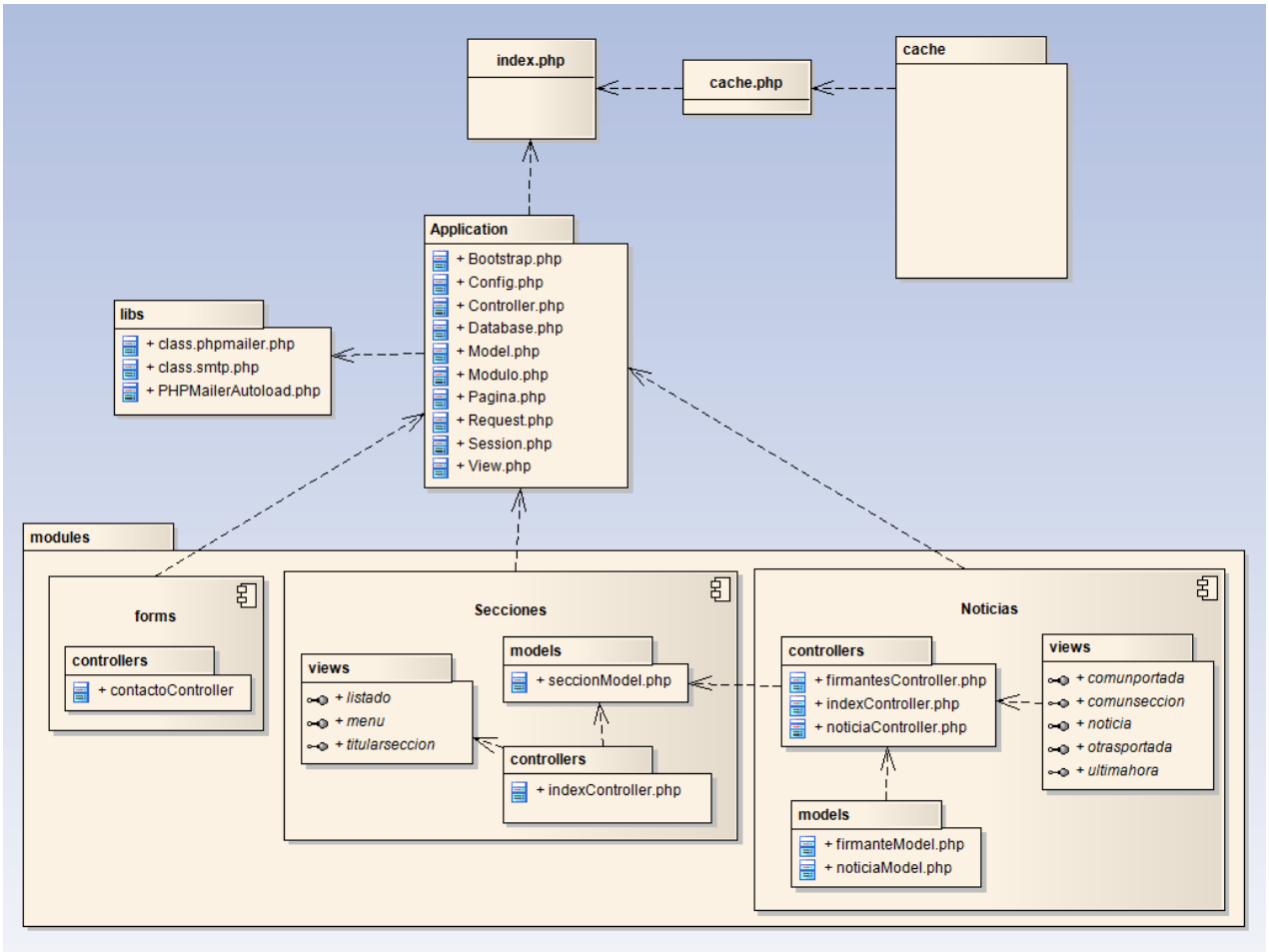
Suponiendo que la utilización del software se realice en un entorno editorial, podemos entender que organizarán su contenido en noticias y secciones. También que, habrá una página por cada sección así como, una página por cada noticia que haya cargada.

No debería de ser necesario pues, tener que configurar una nueva página con la creación de una nueva sección, indicando que módulos irían en dicha página, sino que entenderíamos a la página sección como un contenedor de todas las secciones.

Análogamente, lo mismo sucedería con las noticias, la página noticia aglutinaría a todas.

Además, seguramente sea necesario crear otras páginas que muestren contenido estático, o páginas que muestren módulos de otros componentes, o páginas mixtas, como suele la página de inicio, *index*, que incluye módulos de diferentes componentes.

Arquitectura básica del Front



Concepto de página

La mayoría de los gestores de contenidos del mercado, usan un sistema muy parecido al descrito, usando la inclusión de módulos, también gestión de categorías, y la organización en componentes.

Lo que no deja de ser complicado en todos los sistemas que conozco, es el de insertar un determinado contenido en una determinada posición, algo muy habitual por parte de los administradores del sitio.

En Joomla, por ejemplo, la configuración del contenido del front, se hace a través de los módulos, es decir, se define el límite y alcance de cada módulo, especificando en qué páginas debe mostrarse y en qué posición, así como el nivel de privilegios si es que estuviera definido con algún sistema ACL.

En esta oportunidad que tengo con mi trabajo final, uno de mis propósitos es el de intentar esbozar un sistema, que a diferencia del resto, realice la configuración del contenido del front desde las páginas, y no desde los módulos.

Configurar las páginas eligiendo qué módulos se muestran y cómo, en lugar de elegir donde mostrar cada módulo, es el punto. Puede parecer una nimiedad, pero en mi opinión es un gran paso adelante en la gestión y organización de los contenidos.

Las páginas son los productos o vistas finales de nuestra aplicación, aquellas que conforman el *output* del *front*.

Vemos, para tratar de simplificar, a todo nuestro *front* como un conjunto de páginas. Es por tanto que el primer argumento de nuestra url, será la página: Así, en nuestro supuesto editorial, aquí van algunas páginas de ejemplo:

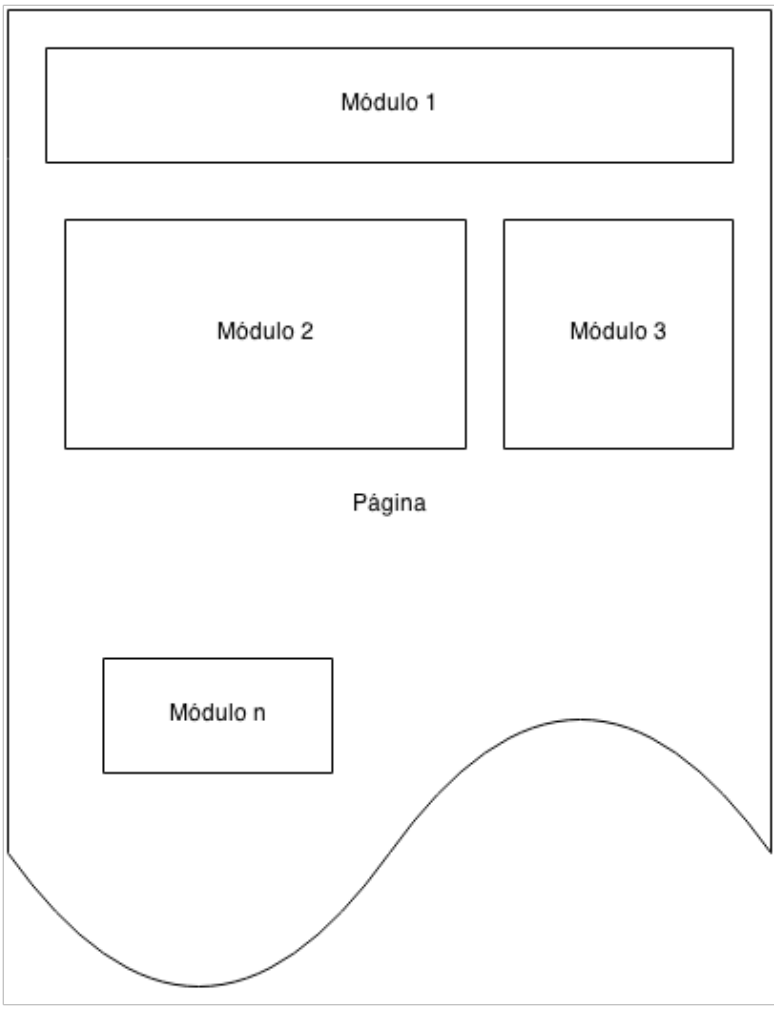
Página	**Url de ejemplo**
Portada	http://cmsum.com
Quiénes somos	http://cmsum.com/quienes-somos
Política de privadidad	http://cmsum.com/politica-de-privacidad
Contacto	http://cmsum.com/contacto
Sección	http://cmsum.com/seccion/3/cultura
Noticias	http://cmsum.com/noticia/123/buena-agogida-exposicion

Resumiendo:

http://cmsum.com/pagina/arg/s/

Concepto de módulo

Son los ladrillos de nuestro sitio. Con éstos, encapsulamos bloques de contenido en nuestra página incrustándolos en diferentes posiciones.



Un componente puede tener muchos módulos, así por ejemplo, el componente noticias tendría una instancia de un módulo para mostrar la vista detalle de una noticia en concreto, otra para la vista de las noticias de última hora, otra para las más leídas, etc.

Al encapsularlos, podemos moverlos líbremente por el sistema, actualizarlos sin comprometer la aplicación, o armar una configuración de páginas concreta.

Lo normal, es que una página siempre tenga un módulo principal, y la descripción, título y otros valores de la página deban ser establecidos por ese módulo, que es la razón propia de la página.

Implementación del front

Para la implementación del *front*, a diferencia del *back*, no vamos a usar componentes, usaremos módulos.

Usará el patrón arquitectónico *MVC* como base de cada módulo.

El archivo index.php del sitio web recibe el GET de la url y llama a la página en cuestión por el primer argumento.

Nuestro index.php será relativamente sencillo, lo que hará será incluir la página en cuestión que se solicita, y es ésta quien tendrá la magia dentro y hará todas las llamadas los módulos que contenga, los cuales son los que tienen propiamente el modelo MVC. Cada módulo tiene el método run heredado, y es usado por la página para su creación.

El sistema analiza la *url* que viene en el *request* de la petición, si el primer argumento es una página que existe, la incluye e interpreta.

PÁGINA → CARGA DE MODULOS → RESPONSE HTML

```
public function procesarPlantilla(){

   $plantilla = file_get_contents($this->rutaPlantilla);
   preg_match_all('/\[\[.*\]\]/i', $plantilla, $this->_modulos);
   $this->_modulos = $this->cargaModulos($this->_modulos[0]);

   if($this->_modulos = $this->procesaModulos($this->_modulos))
      foreach($this->_modulos as $mod)
         $plantilla = str_replace($mod->string,$mod->buffer, $plantilla);

   $this->SetPlantillaProcesada($plantilla);
}
```

En el extracto de código anterior, se puede observar como se buscan los módulos contenidos en la página, con la expresión regular, tratando de obtener cadenas como esta, '[[noticias::index::ultimahora]]'. Una vez procesados cada uno de los módulos, reemplaza esas cadenas por el buffer obtenido en la salida del método *run* de cada uno de estos módulos.

La cuestión de las páginas y el selector de página es muy práctico. Hacemos el código más humano, organizándolo en archivos los cuales incluyen llamadas a los módulos, tal y como lo vemos en la web.

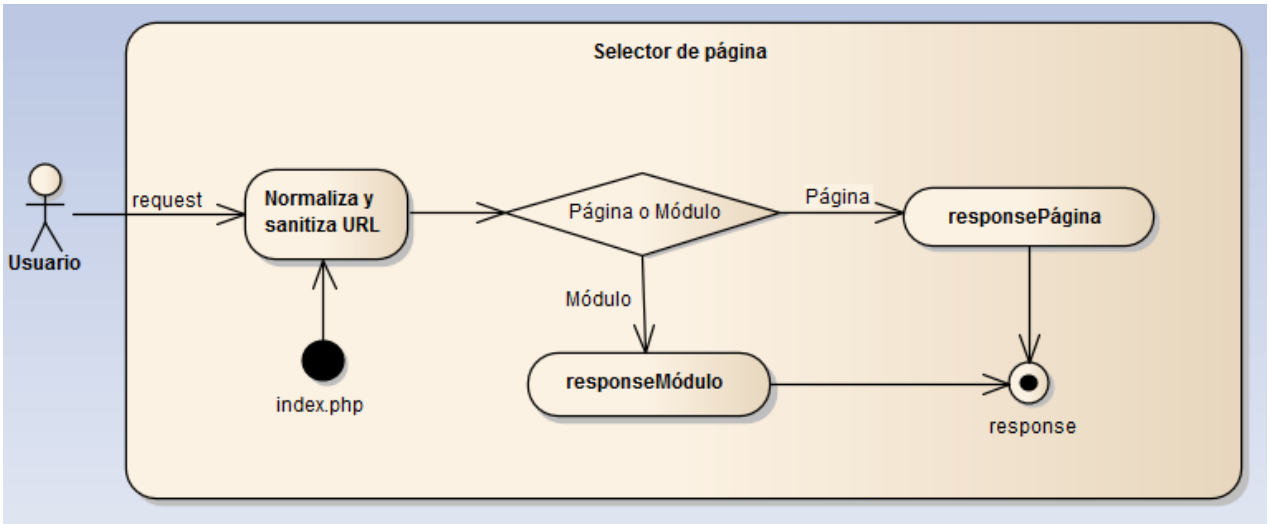
Además, con este nivel de encapsulamiento y organización, poniendo como ejemplo la implementación de un sistema de caché, podría efectuarse en diferentes jerarquías superiores, a nivel de página e incluso de módulo. En este caso, se ha implementado a nivel de página en el propio *index.php*, pero el sistema está preparado para implementarlo a nivel de módulo. Este uso podría ser adecuado para sistemas que no pueden cachear determinadas áreas, como la zona de usuario, etc.

Acceso directo a un módulo

Hasta aquí todo bien con las páginas, pero pudiera darse el caso que necesitemos acceder algunos servicios que ofrece la web y que no tendría sentido adaptarlos en páginas. Por ejemplo un *Web Service*, o un controlador que atienda peticiones ajax, etc.

Por ello, no son sólo las páginas las que pueden llamar a los controladores de los módulos, también debemos poder **acceder desde una URL a nuestros módulos**.

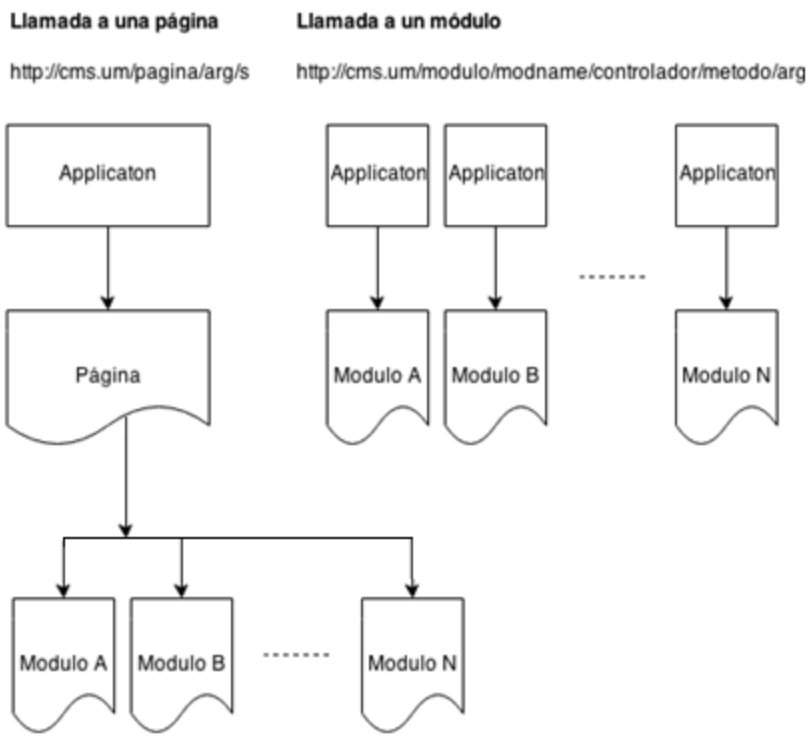
Aquí nace un nuevo artefacto, es el **selector de página**:



Para ello el primer argumento de nuestra petición contendrá el string **'modulo'**.

```
http://cms.um/modulo/noticias/votes/vote/positive/123
```

http://cms.um/modulo/modname/controlador/metodo/argumento/arg/s



Vemos en la ilustración, cómo el sistema está pensado para trabajar con páginas, y se da por hecho que son las que conformarán la web.

Cuando una página se carga, lo hace en un contexto en el cual, todos los módulos que contienen ya disponen de todas las clases de *Application*, así, una sola instancia del sistema gestiona la carga de los módulos de la página.

Aquí vemos un extracto de código de la clase *Request* aplicando lo arriba dicho:

```
//http://cmsum.com/modulo/modname/controlador/metodo/arg/s
if ($this->first == 'modulo') {//Cargo módulo
        $this->_modulo = strtolower(array_shift($url));
        $this->_controlador = strtolower(array_shift($url));
        $this->_metodo = strtolower(array_shift($url));
        $this->_argumentos = $url;
        $modulo = $this->_modulo;
}
```

Metadata de los módulos hacia la página

Al concretar el concepto página, surgen nuevas cuestiones a resolver. La metadata más común que tiene una página, son atributos como el título, la descripción, los keywords, etc. Pero también hay otros que debemos tener en cuenta, como son los archivos JS y CSS.

Es muy posible que sea necesario hacer que el tittle, descripcion o keywords de la página sea definido por algún módulo.

Lo normal al establecer un layout al sitio web, es que todas las páginas tengan unos CSS y JS en común, y también pudiera ser que un módulo concreto tuviera necesidad a un script externo.

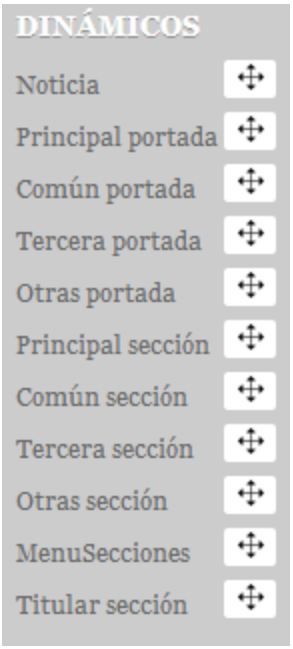
Desde los módulos es posible llamar a métodos de la página que los está instanciando y setear dichos valores. Está semi implementado en la aplicación.

Módulos estáticos

Son módulos que van embebidos en la aplicación, se consideran genéricos y de uso común. Son HTML plano, que se puede editar con un *WYSIWYG* flotante en la interfaz de creación de páginas.

Módulos dinámicos

Son módulos desarrollados a medida para mostrar la información generada por los componentes. Estos módulos son los que se configuran en el archivo *modules.xml* y que implementan el patrón MVC, por tanto son capaces de recibir variables por *get*. Así el módulo noticia esperará que un parámetro.



Es el programador el que los crea y sirve al cliente o usuario. El diseñador podrá acceder a las hojas de estilo y a las vistas. El usuario básico solo podrá usar estos módulos.

Módulos propios

Sugen con la necesidad de personalizar y persistir los módulos estáticos. Con la facilidades de un WYSIWYG permite elaborar contenidos y almacenarlos en el sistema, para su inclusión en las páginas. Van almacenados en la BBDD como etiquetado HTML.

index.php

```
<?php
   define('DS', DIRECTORY_SEPARATOR);
   define('ROOT', realpath(dirname(__FILE__)) . DS);
   define('APP_PATH', ROOT . 'application' . DS);

   require_once APP_PATH . 'Cache.php';
   $cache = new Cache();

   if($cache->isCached()){
         $cache->response($cache->buffer);
      exit;
   }

   try{
   //Aplicación
   require_once APP_PATH . 'Config.php';
   require_once APP_PATH . 'Request.php';
   require_once APP_PATH . 'Bootstrap.php';
   require_once APP_PATH . 'Controller.php';
   require_once APP_PATH . 'Database.php';
   require_once APP_PATH . 'Model.php';
   require_once APP_PATH . 'View.php';
   require_once APP_PATH . 'Pagina.php';
   require_once APP_PATH . 'Modulo.php';

         ob_start();
         $peticion = new Request();

         //Selector de página
         if($peticion->paginaModulo($peticion)=='pagina')
            Bootstrap::cargaPagina(PAGINA);
         else
            Bootstrap::run($peticion);

         $buffer = ob_get_contents();

         $cache->set($buffer);
   }
   catch (Exception $e){
         $e->getMessage();
   }
?>
```

Sistema de caché

Cuando pensé en que sistema se podría implementar para una buena caché, tuve en cuenta que fuera muy simple, de alto rendimiento y que no rompiera la integridad de la información al mostrar contenido no actualizado en la web.

Puede verse en el index.php y en el diagrama de la arquitectura del front, cómo la clase Cache.php se carga antes que la aplicación, de tal manera que si la verificación de la caché resulta certera, hace un *exit* y no carga el sistema.

```
if($cache->isCached()){
        $cache->response($cache->buffer);
    exit;
  }
```

En caso de que no estuviera cacheado el recurso, tiene que procesar la página para elaborar el response, y una vez hecho, aprovecha para guardar este último recurso solicitado en la caché para ahorrar su procesamiento en el siguiente requerimiento.

```
$cache->set($buffer);
```

En caso de que no estuviera cacheado el recurso, tiene que procesar la página para elaborar el response, y aprovecha para guardar este último recurso solicitado.

Caché bajo demanda

La caché siempre tiene que tener un límite, por una cuestión de recursos, y por una cuestión de *performance*. Como ya se ha comentado, muchos sistemas utilizan una caché de demanda basada en el tiempo. Esto significa que el sistema está alerta y

periódicamente según un tiempo predefinido, se recolectan los elementos de caché que hayan expirado. Este es uno de los sistemas más generalizados.

En cambio, pretendo poner el límite en el número de recursos almacenados y no en el tiempo de almacenamiento de éstos.

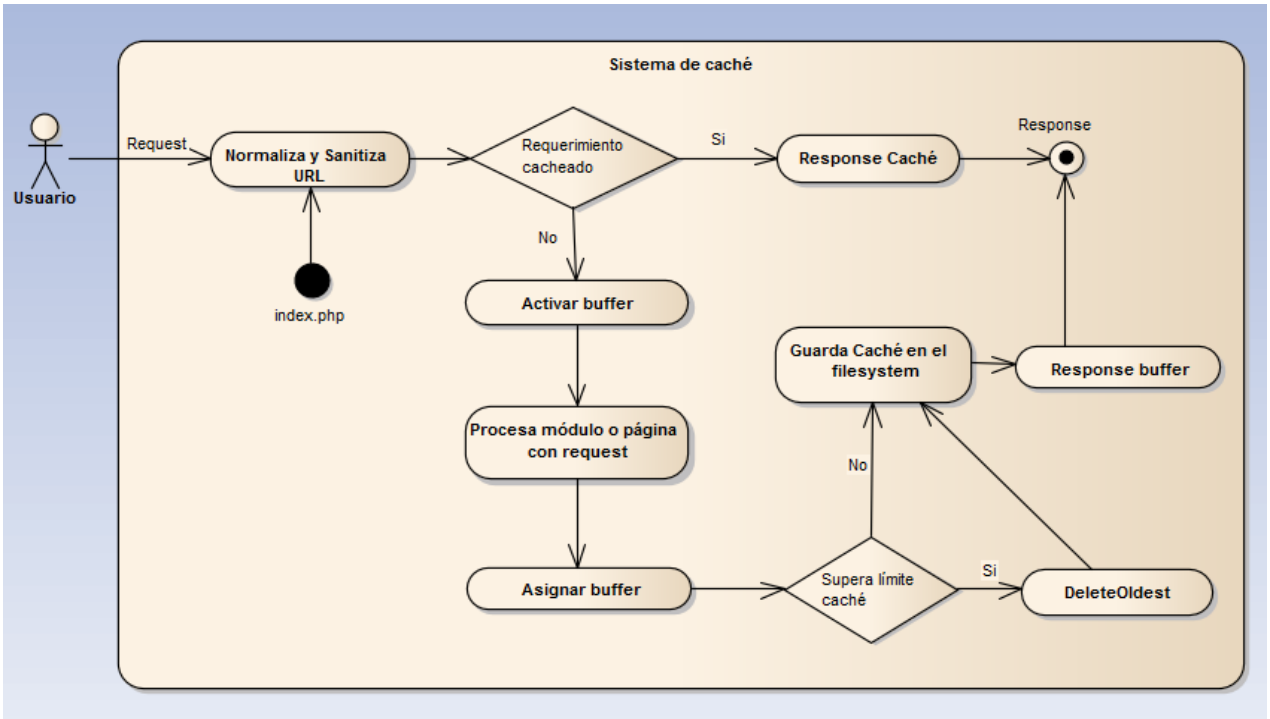
```
$cacheFile = $this->folder.$this->request;
$dir = opendir($this->folder);

 //Si supero el máximo de archivos permitidos, elimino el más viejo
if($this->countFiles() >= $this->limit){
   if($this->deleteOldest())
        if(file_put_contents($this->folder.$this->request, $buffer));
            return true;
}
else{
    if(file_put_contents($this->folder.$this->request, $buffer));
            return true;
}
```

De tal manera si se fija un límite, la próxima vez que se solicite un recurso no cacheado, se guardará la caché de este y para no superar el límite, se eliminará el recurso almacenado más antiguo.

Así el sistema siempre tendrá cacheados los recursos que más se soliciten en ese momento.

A diferencia de los otros sistemas basados en el tiempo, que han de estar alerta (consumo de recursos) al tiempo de expiración de los contenidos, en este la caché se va renovando automáticamente dentro de los límites establecidos.



Para que no se vea alterada la integridad de la información en el proceso de refresco y actualización por culpa de la caché, en el backend se programan métodos genéricos que eliminen la caché cada vez que un contenido es modificado, eliminado, o agregado. Este punto no está implementado, pero es el sentido de la caché.

Una posible implementación podría ser el de la creación de un procedimiento almacenado en la BD, que se dispare con cualquier consulta *insert*, *update* o *delete* (modificación del contenido) y setee un valor *true* en un campo determinado, de tal manera que cada vez que la aplicación corre, chequea este valor para saber si debe consumir la caché existente, o, si ésta quedó obsoleta, y el sistema debe descartarla.

Otra posible implementación sería la de poner un método manual en el código del back, que elimine la caché en cada operación *CRUD* realizada en los modelos.

De esta manera, por seguro el usuario final nunca verá un contenido desactualizado.

CAPÍTULO V - APRENDIZAJE

ANÁLISIS E INTERPRETACIÓN DE LOS RESULTADOS

El resultado bajo mi punto de vista es muy satisfactorio y se han logrado los objetivos planteados.

Rendimiento

No ha sido hasta la conclusión del mismo, que he podido hacer una prueba de rendimiento del CMS UM en comparación a los existentes en el mercado, consultar anexo 'Estudio interno tiempos de respuesta CMS existentes'.

En este estudio se puede ver como Joomla y Wordpress con o sin caché, para mostrar un contenido tipo, son capaces de proporcionar entre 7 y 8 requerimientos por segundo.

Al analizar CMS UM bajo el mismo escenario, se puede observar que proporciona unos 40 requerimientos por segundo. Lo sorprendente resulta al activar la caché, ya que el servidor web es capaz de proporcionar más de 1200 requerimientos por segundo.

No hay que cantar victoria, pero es un buen comienzo. La idea es tener en cuenta este tipo de pruebas durante la construcción del CMS, para ser conscientes de la pérdida de performance a medida que se aumente la complejidad.

Usabilidad

Además de opiniones de gente cercana, no tengo más datos o estudios que puedan cuantificar la usabilidad, solo mi experiencia propia. Entiendo que una vez definidos los

módulos que compondrán nuestro sitio, y los estilos, la creación de página resulta muy rápida y sencilla. Este apartado es algo subjetivo y depende de los gustos del usuario, pero creo se han alcanzado los propósitos planteados.

Puede verse la aplicación del proyecto en http://cmsum.com

CONCLUSIONES

Con trabajo y esfuerzo, creo que el camino iniciado para la creación de un CMS propio que sirva al productor de software como la base de casi todos sus proyectos webs, puede ser útil y ayudar a mejorar la productividad de la organización, así como su competitividad.

Hay que tener en cuenta, que todas las webs, por diferentes que sean tiene muchas cosas en común. Todos los editores de contenidos gustan de recursos cada vez más habituales, como la gestión de artículos, publicidad, SEO, galerías multimedia, videos, foros, tiendas, etc. Estos recursos se pueden encapsular en los componentes, y así, de un cliente a otro reutilizar muchos de ellos, haciendo que la producción de software sea mucho más eficiente, que el cliente tenga un producto mucho más completo, y que los esfuerzos se realicen en el modelo de negocio del cliente y no administración de su sitio.

FALTANTES Y RECOMENDACIONES

Sería necesario aplicar el patrón singleton en la creación de instancias de los módulos. Devolviendo el anterior estado, así se mejoraría el rendimiento cuando haya muchos módulos del mismo componente, solo se instanciaría un modelo de los mismos.
En la elección de librerías Drag&Drop, o editores WYSIWYG, se confía en software de terceros, y son cientos de archivos agregados al proyecto. Puede ser un talón de Aquiles si la implementación o encapsulación de dichas funcionalidades no es correcta. Por tanto,

sería conveniente hacer más énfasis en determinadas métricas que ayuden a valorar la elección de un buenos softwares complementarios.

Así por ejemplo, analizar el HTML producido por los editores WYSIWYG para ver cual ofrece mayor estandarización o limpieza, cual cumple los cometidos de estilos del CMS, etc.

Seguramente, con la creación de muchos módulos, la interfaz visual propuesta no sería escalable, y habría que usar otro sistema de submenús o desplegables.

Respecto a la caché, si el sitio implementa zonas totalmente dinámicas no cacheables, como la zona de usuario, sería necesario implementar una caché a nivel de módulo y no de página.

BIBLIOGRAFÍA

Sistemas de gestión de contenidos en la gestión del conocimiento

Autores

Jesús Salinas, Bárbara de Benito, Victoria Marín, Juan Moreno, María Eugenia Morales

Edita

Universitat Illes Balears

Ingeniería del software

Autor

Roger S. Pressman

Edita

Mc Graw Hill

Patterns For Dummies

Autor

Steve Holzner

Edita

Wiley Publishing, Inc

OTROS RECURSOS

Introducción a los Sistemas de Gestión de Contenidos: CMS

http://www.slideshare.net/santillan/introduccin-a-los-sistemas-de-gestin-de-contenidos-cms

Sistemas de gestión de contenidos

http://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_contenidos

Gestión de contenidos

Título: Artículos sobre los gestores de contenidos

Autor: James Robertson

http://www.steptwo.com.au/category/papers/content-management

Memoria de librerías

Interfaz gráfica Front

Web responsive Bootstrap Twitter CSS

Theme Journalist Bootswatch

Interfaz gráfica BACK

Web responsive basada en KamiDesign KATO

Layoutit como generador de estructura Drag&Drop

CkEdior como Wysiwg para las noticias

Aloha editor como Wysiwg flotante para la maquetación

Imagen y curiosidad

Imagen obtenida de una página que muestra actualidad acerca del mundo de los CMS, Superwebsitebuilders.com en un artículo acerca de unos premiosde *Cmscritic.com* que se celebran para distintos CMS. Premios que van desde el mejor CMS *OpenSource,* hasta el mejor CMS empresarial, *sitebuilders*, foros o los más reducidos

En 2013 la gente eligió como ganador a Wordpress en la categoría de los *OpenSource*.

Premios:

http://www.cmscritic.com/critics-choice-cms-awards/

http://superbwebsitebuilders.com/critics-choice-cms-award/

Imagen:

http://superbwebsitebuilders.com/wp-content/uploads/2013/07/logos-CMS.jpg

Auge de los sitios web basados en CMS sin base de datos

http://www.interdixit.com/esp/noticia.php?pag=auge_los_sitios_web_basados_cms_sin_bases_datos

Metáfora de escritorio

http://es.wikipedia.org/wiki/Met%C3%A1fora_de_escritorio

Proveedores de estadísticas sobre el uso de los CMS

URL del proveedor estadístico:

http://w3techs.com/

URL de los datos sobre los CMS:

http://w3techs.com/technologies/overview/content_management/all

Quiénes son W3techs:

http://w3techs.com/about

GLOSARIO

ACL

Access control list, se refiere a la técnica o sistema de seguridad que permite asignar privilegios y/o roles a las entidades en el sistema.

Traducción: Lista de control de acceso

Backend

Se refiere a la parte de un sitio web que solo es visible por los usuarios administradores del mismo y que normalmente sirve para realizar diferentes funciones de gestión sobre los datos, el modelo de negocio y sobre la estructura de el sitio público.

Abreviatura: back.

Buffers

Son alocaciones (reservaciones) temporales de memoria que sirven de apoyo a los procesos para ir almacenando dinámicamente la información procesada de manera incremental. También pueden ser usados para amortiguar una recepción de información.

Traducción: amortiguador.

Caché

Es el sistema por el cual se almacenan los resultados de un procesamiento a la espera de que se vuelvan a requerir para así ahorrar su procesamiento.

CRUD

Create, read, update or delete, son el conjunto principal de operaciones que se realizan sobre la información contenida en una BBDD.

Traducción: crear, leer, actualizar o eliminar.

CMS

Content Managment System, es un sistema informático que permite a usuarios sin altos conocimientos técnicos tener organizada su información y poder agregarla, eliminarla o editarla, además de presentarla a su gusto. Suelen ser aplicaciones web.

Traducción: gestor de contenidos.

Drag&Drop

Hace referencia a la técnica que permite a los usuarios arrastrar objetos visualmente en destinos contenedores.

Traducción: Arrastrar y soltar.

Extreme Programming

Técnica ágil de desarrollo de software que soluciona los problemas a medida que los encuentra en lugar de preveerlos mediante el análisis exhaustivo.

Traducción: programación extrema

Framework

Es un conjunto de herramientas, archivos, librerías funciones, etc junto con una arquitectura base que sirve a los desarrolladores como puntos de partida sobre el que crear sus aplicaciones.

Traducción: marco de trabajo

Frontend

Hace referencia a todo el conjunto de páginas web públicas de libre acceso que conforman el sitio. Abreviatura: front.

Github

Sistema usado para mantener el sistema de versionado de software, de manera que sea accesible y recuperable. Basado en protocolo GIt.

Insert, update o delete

Son palabras reservadas del lenguaje de programación SQL que permiten insertar, eliminar o modificar información en una BBDD. Junto con la palabra Select conforman el conjunto de palabras que permiten las operaciones CRUD,.

Traducción: insertar, actualizar o eliminar.

Interfaz

Es la definición de las reglas para la comunicación entre sistemas informáticos.

Metodologías ágiles

Son un conjunto de métodos, técnicas y recomendaciones para facilitar las tareas en la creación de software, ejemplos SCRUM, XP, etc.

Output

Hace referencia a las páginas que han sido interpretadas y procesadas. A la salida del proceso.

Traducción:salida

Open Source

Son el conjunto de proyectos de software cuyos fuentes de código están a disposición pública.

Traducción: código abierto

Patrón MVC

El Modelo Vista Controlador (MVC) es un patrón de arquitectura de software que separa los datos y lógica de negocio de una aplicación de la interfaz de usuario. Implementa el artefacto controlador que es el que recibe la petición del usuario, y pasa la información de los modelos a las vistas.

Performance

Indica el desempeño que se experimenta en el uso de un sistema, programa, etc.

Traducción: rendimiento.

PHP

Es el lenguaje de programación más utilizado en creación de páginas webs. Es interpretado por los servidores webs procesando la lógica establecida y devolviendo HTML al usuario.

RUP

Rational unified process es el proceso de elaboración de Software propuesto por microsoft.

Traducción: proceso unificado de Rational.

Request

Hace referencia a la petición del protocolo HTTP. La consecuencia al Request es el Response.

Traducción: requerimiento.

Repositorio

Se refiere a los almacenes de código donde se mantiene una copia y se lleva el versionado de cada proyecto de los desarrolladores. También admiten archivos comunes. Algunos: Git, SVN, y Mercurial.

Response

Hace referencia a la respuesta del protocolo HTTP. Es la respuesta al requerimiento o request inicial.

Traducción: respuesta

Site Builders

Son aplicaciones que permiten crear sitios webs de manera rápida y sencilla. No cuentan con un avanzado sistema de gestión de la información, si que son bien primarios.

Traducción: constructor de sitios.

SEO

Search engine optimization, son el conjunto de técnicas y consideraciones que tienen en cuenta los editores y desarrolladores para que su sitio cumpla los estándares y tenga un buen posicionamiento.

Traducción: optimizador de motores de búsqueda.

Tracker

Un tracker es un rastreador de una determinada información y es creado con esa finalidad. Puede acudir recurrentemente para actualizarse. Su uso en conjunto, permite obtener información de fuentes remotas, públicas y ajenas.

Traducción: rastreador.

True

Es un concepto que en la jerga informática se usa para evaluar una condición como cierta. También es una palabra reservada en muchos lenguajes de programación.

Traducción: verdadero.

UML

Unified Modeling Language es un lenguaje que usa como recursos principalmente diagramas para representar flujos, procesos, estados, descripciones, vistas y arquitecturas de software.

Traducción: lenguaje unificado de modelado.

Web Service

Es una técnica que permite la comunicación de datos entre distintas aplicaciones. Vease el caso de una aplicación móvil (aplicación local del usuario) que consume el WS (datos remotos) para mostrar la información. Abreviatura: WS.

AXENOS

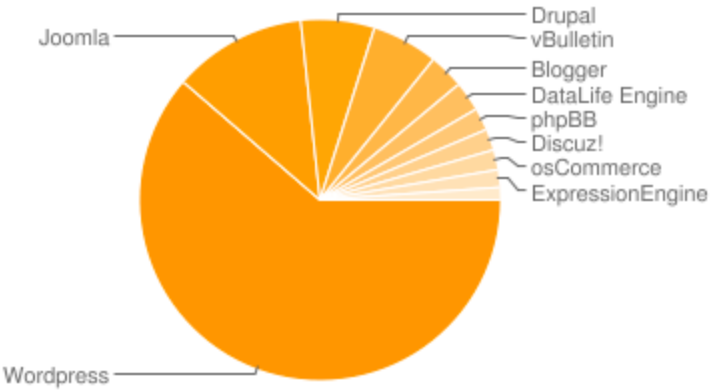
Estudio externo comparativo de Joomla y Wordpress

URL del estudio completo:
http://redgiantdesign.co.za/rg-design-blog/wordpress-vs-joomla-2013-infographic.html

URL de la inforgrafía comparativa:
http://redgiantdesign.co.za/images/easyblog_images/42/WordPress-vs-Joomla.png

Realizado por el cofundador del prestigioso estudio de diseño *Red Giant Design*, ofrece información muy interesante sobre estos dos gigantes, acerca de sus datos, su evolución, etc.

En una serie de artículos va desentrañando los misterios, analizando su estructura y funcionalidades, y realizando diferentes comparativas.



Relativización de los contenidos dinámicos

La frecuencia de cambio del contenido es mucho menor que la frecuencia de acceso al mismo. Para ello cuento con acceso a información privada de un sitio amigo, www.vivetorre.com, el cual ha facilitado acceso a sus registros. Este sitio web, es un típico portal de noticias de una localidad, que usa un CMS tradicional (Editmaker).

En la siguiente tabla, vemos un listado de 20 noticias, su fecha de última modificación, el número de visitas hasta la fecha, y el enlace a la noticia en cuestión:

Enlace a la noticia	Última modificación	Creación	Visitas
http://www.vivetorre.com/noticia/3210/x/x	2012-12-24	2012-12-21	243
http://www.vivetorre.com/noticia/3202/x/x	2012-12-24	2012-12-21	58
http://www.vivetorre.com/noticia/3197/x/x	2012-12-24	2012-12-21	78
http://www.vivetorre.com/noticia/3195/x/x	2012-12-21	2012-12-21	72
http://www.vivetorre.com/noticia/3194/x/x	2012-12-21	2012-12-21	83
http://www.vivetorre.com/noticia/3193/x/x	2012-12-21	2012-12-21	98
http://www.vivetorre.com/noticia/3192/x/x	2012-12-21	2012-12-21	76
http://www.vivetorre.com/noticia/3191/x/x	2012-12-21	2012-12-21	73
http://www.vivetorre.com/noticia/3190/x/x	2012-12-21	2012-12-21	129
http://www.vivetorre.com/noticia/3189/x/x	2012-12-21	2012-12-21	79
http://www.vivetorre.com/noticia/3188/x/x	2012-12-21	2012-12-21	135
http://www.vivetorre.com/noticia/3174/x/x	2012-12-21	2012-12-21	121
http://www.vivetorre.com/noticia/3186/x/x	2012-12-21	2012-12-21	74
http://www.vivetorre.com/noticia/3187/x/x	2012-12-21	2012-12-21	70
http://www.vivetorre.com/noticia/3185/x/x	2012-12-21	2012-12-21	57
http://www.vivetorre.com/noticia/3183/x/x	2012-12-21	2012-12-21	78
http://www.vivetorre.com/noticia/3182/x/x	2012-12-21	2012-12-21	75
http://www.vivetorre.com/noticia/3181/x/x	2012-12-21	2012-12-21	84
http://www.vivetorre.com/noticia/3180/x/x	2012-12-21	2012-12-21	72
http://www.vivetorre.com/noticia/3179/x/x	2012-12-21	2012-12-21	71

Tasa de modificación

Con la información presente, no podemos saber cuántas veces se modificó una noticia, pero si cuáles fueron modificadas. Estimo que las noticias que se actualizaron lo hicieron de media una vez.
Vemos que de 20 noticias, sólo 3 fueron modificadas después de la fecha de creación. Y tiene sentido. En este caso, como en muchos, la actualización de la web va atada a una actualización en papel, por lo que se hace un volcado de noticias periódicamente.

Tasa de modificación ⟶ 3 noticias modificadas / 20 noticias totales = **0.15**

Tasa de acceso

Si ahora analizamos el número de veces que se accedió a cada noticia, y calculamos la media obtenemos la tasa de acceso.

Tasa de acceso ⟶ 1826 visitas totales / 20 = **91.3**

Uno puede llegar a la conclusión de la gran mayoría de las peticiones realizadas al servidor provienen de las solicitudes que los usuarios navegantes generan en el sitio. Dicho de otra manera, observamos que las peticiones al servidor provenientes del BackEnd* , es decir las que modifican los contenidos son mínimas o no mayores al 1%:

Nro. Solicitudes del Front >>> Nro. Solicitudes del Back

En el ejemplo anterior, observamos como la frecuencia con la que se accede a un contenido es mucho mayor que la frecuencia con la que cambia dicho contenido, por tanto

hemos de relativizar el hecho, y es que desde el punto de vista del sistema, la información no es tan dinámica como uno podría pensar inicialmente.

En conclusión:

Tasa de acceso al Front >>> Tasa de modificación en el Back

Nota:

Se podría argumentar que el análisis de un solo sitio web no es reflejo del conjunto poblacional, pero me parece de sentido común lo expuesto y que puede ser generalizable para cualquier gestor de contenidos.

Es un hecho que hay más consumidores de la información, que editores que la produzcan. Unos pocos periodistas pueden escribir para cientos, miles o millones de usuarios. La comunicación es un servicio de masas, sino no tiene sentido. Pudiera darse el caso de algún sitio web de contenidos muy específico, de carácter científico o gubernamental, que tenga muy poco tráfico y sin embargo tenga una alta tasa de actualización, pero no es lo habitual en la red, en lo que a CMS se refiere.

Estudio interno tiempos de respuesta CMS existentes

Para realizar las pruebas se ha usado el mismo hardware, usando una página tipo básica, con una concurrencia de peticiones al servidor de 10, en un total de 1000 requerimientos al servidor.

La misma prueba se ha hecho sobre los siguientes CMS:

Joomla

C:\xampp\apache\bin> ab -n 1000 -c 10 http://localhost/um_joomla/

Tiempo total: 138.29 s

Nro. requerimientos por segundo: **7.23**

Joomla usando sistema integrado de caché

C:\xampp\apache\bin> ab -n 1000 -c 10 http://localhost/um_joomla/

Tiempo total: 122.29 s

Nro. requerimientos por segundo: **8.19**

Wordpress

C:\xampp\apache\bin> ab -n 1000 -c 10 http://localhost/um_wordpress/

Tiempo total: 124.77 s

Nro. requerimientos por segundo: **8.01**

CMSUM

C:\xampp\apache\bin> ab -n 1000 -c 10 http://localhost/cmsum/

Tiempo total: 26.7 s

Nro. requerimientos por segundo: **40.13**

CMSUM usando sistema de caché bajo demanda

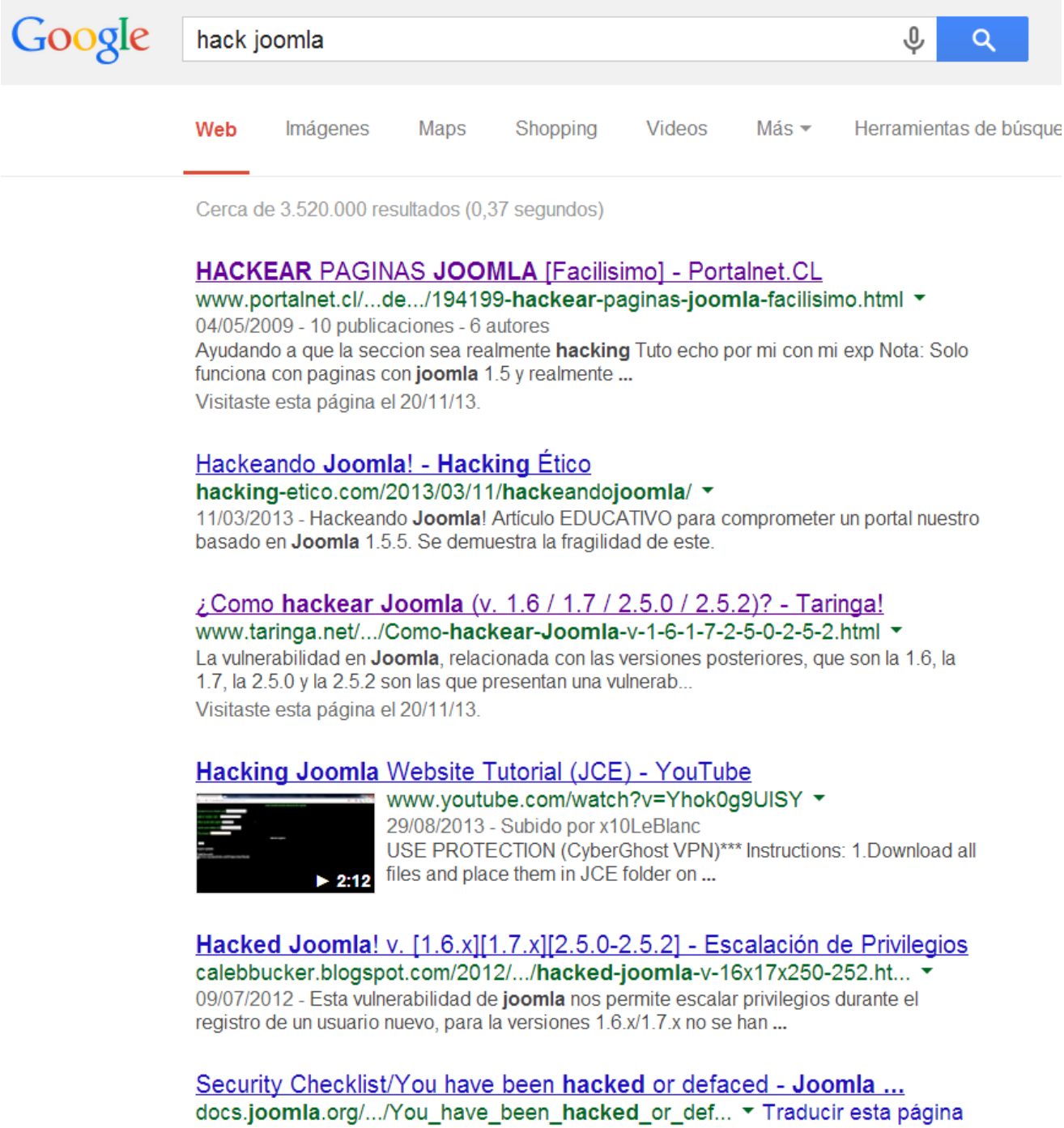
C:\xampp\apache\bin> ab -n 1000 -c 10 http://localhost/cmsum/

Tiempo total: 0.81 s

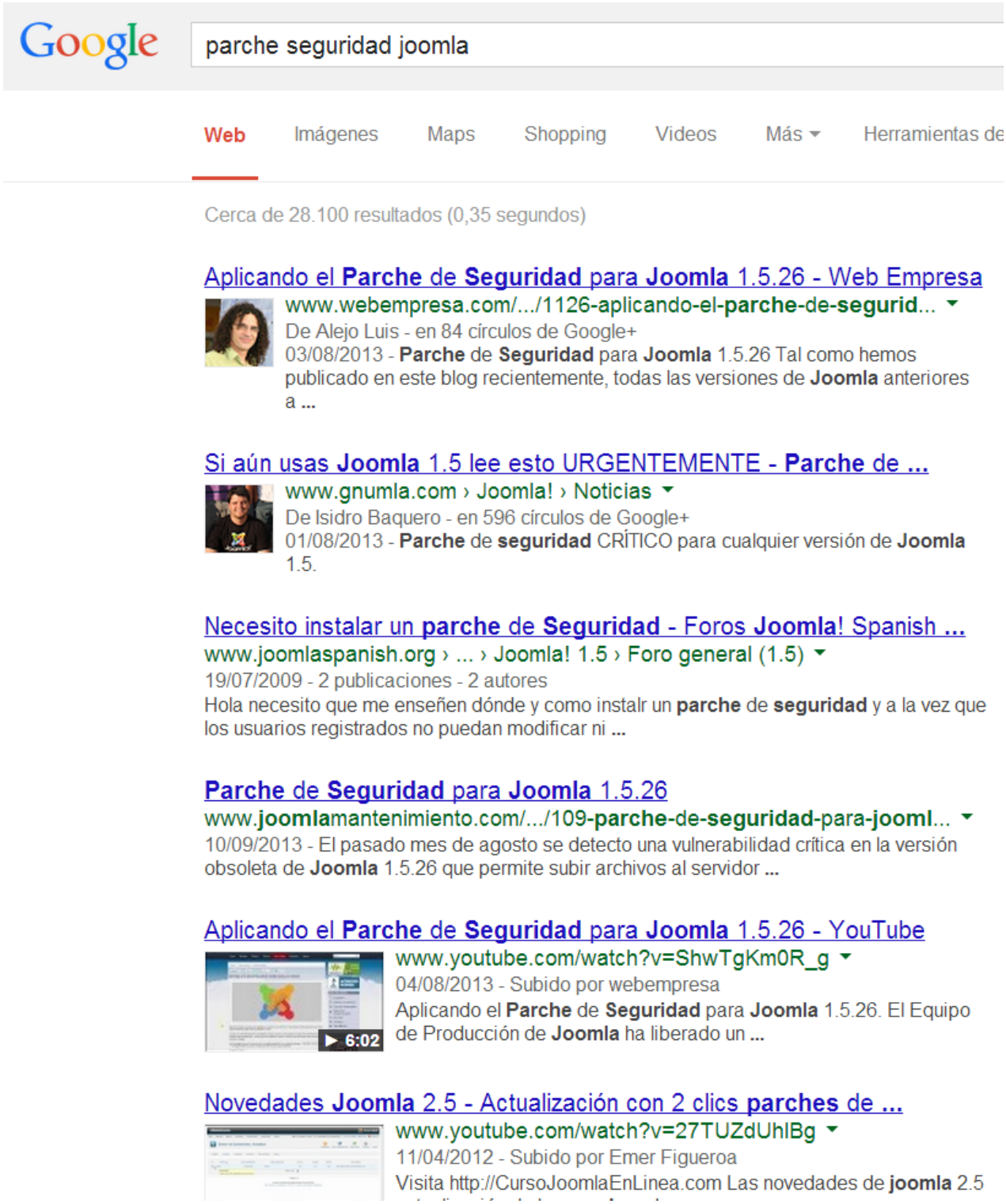
Nro. requerimientos por segundo: **1235.27**

Búsquedas en Internet

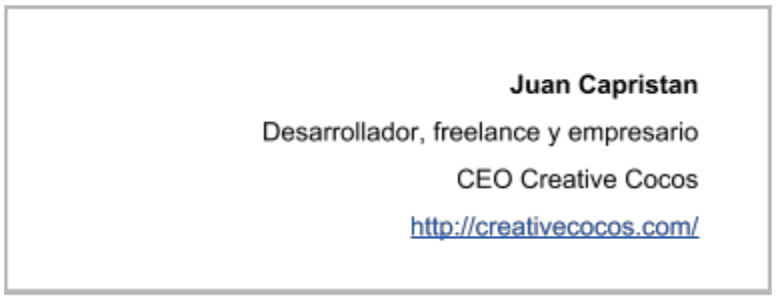
https://www.google.es/search?q=hack+joomla



https://www.google.es/search?q=parche+seguridad+joomla



Cuestionario 1



¿Qué software debe usar como base un productor de software de sistemas web en sus proyectos?

Hay tantas opciones válidas como gustos. ¿Qué tipo de brocha debe usar un pintor? ¿qué juego de herramientas debe usar un mecánico?. Intentando centrar un poco la respuesta en mi caso particular: desarrollo la mayor parte del tiempo en un equipo Windows, aplicaciones Rails, PHP y ASP, así que mis herramientas básicas son:

– IIS, Apache y Webmin como servidores locales
– MySQL, PostgreSQL y Riak como servidores de BD
– Notepad++ y RubyMine como editor de código e IDE.
– Todos los navegadores disponibles, junto con plugins de desarrollo web para cada uno.
– Diferentes validadores, minimizadores de código, consolas, etc.

¿Es correcto hacer un sistema web a medida en lugar de tomar uno del mercado?

Depende del caso, de lo que se pretenda conseguir, del cliente, del presupuesto... ¿Es correcto construir una casa a medida en vez de comprar una prefabricada?. No es una pregunta que tenga una respuesta correcta genérica. En general es más cómodo reutilizar, adaptar, configurar, sin embargo no siempre es posible, ni deseable. Por otro lado, en general creo que es preferible una aplicación a medida bien fabricada que una aplicación genérica bien fabricada. Finalmente, hay distintos niveles de reutilización de código. Que

una aplicación se haya fabricado a medida no significa que hayamos tenido que reinventar la rueda.

¿Un único sistema web puede ser la base de casi todos los proyectos de un productor de software?

Sí. Sobre todo lo digo pensando en cualquiera de los frameworks de desarrollo web existentes, como Ruby on Rails o Yii.

Cuestionario 2

Carlos Peralta
Licenciado en informática
Team Leader Doopress
Administrador de Sistemas
http://doopress.com/

¿Qué software debe usar como base un productor de software de sistemas web en sus proyectos?

Entiendo se le denominan “stack” o pilas de sistemas (software). Sugiero la pila LAMP por su amplio uso y cantidad de sistemas realizadas con la misma: Linux, Apache, MySQL y PHP (la última P también puede ser Phyton o Perl; pero PHP es el más popular)

¿Es correcto hacer un sistema web a medida en lugar de tomar uno del mercado?

Si el software del mercado es de código libre y tiene soporte por la comunidad; sin duda mejor seleccionar algo ya creado. Hay que analizar en este caso la popularidad del sistema para ver hasta cuando o la cantidad de personas que le darán soporte a mediano, largo plazo. Para el caso de los sistemas pagos creo que el factor más relevante es el costo. Por último el sistema a medida puede ser y a veces es la única forma de satisfacer los requerimientos; pero hay que tener en cuenta el mantenimiento de dicho sistema a mediano y largo plazo; por las personas o empresas que desarrollaron el mismo.

¿Un único sistema web puede ser la base de casi todos los proyectos web de un productor de software?

La diferencia entre “sistema Web” y “proyecto Web” no creo sea muy relevante o clara. De todas formas si se refiere a las tecnologías el hecho es que una pila (como LAMP) no siempre es la base de un proyecto Web. Solo puede haber Apache y PHP; y atrás otro SO, otro motor de base de datos, lenguajes compilados a nivel de SO (Ej. C++), etc.

Cuestionario 3

¿Qué software debe usar como base un productor de software de sistemas web en sus proyectos?

Dependiendo de la envergadura del proyecto y las necesidades del mismo, podemos optar por distintos tipos de frameworks para ayudarnos en su desarrollo.

Si nuestro proyecto no requiere algo muy específico podríamos usar diferentes CMS (joomla, drupal…) como base del mismo y de esta manera podríamos ahorrarnos bastante trabajo, ya que a parte de tener parte del código desarrollado, como los gestores de artículos, usuarios ACL… nos permitiría utilizar módulos, componentes y plugins desarrollados por terceros evitando tener que realizar parte de la programación. A demás estos CMS poseen una gran comunidad de desarrolladores y documentación a la cual podemos acudir en cualquier momento.

En cambio, si nuestro proyecto es de mayor envergadura podríamos recurrir a frameworks como Zend o Symfony los cuales nos permiten tener nuestro código estructurado y organizado.

¿Es correcto hacer un sistema a medida en lugar de tomar uno del mercado?

Desarrollar un sistema a medida tiene sus ventajas y desventajas.

En primer lugar el desarrollo íntegro de un sistema a medida puede resultar muy costoso, dependiendo de la envergadura del sistema, además de que se debe de desarrollar con un equipo de programadores experimentados, llevando unas pautas de trabajo muy bien

estructuradas, con una planificación y organización estricta y ofreciendo una escalabilidad del sistema.
La ventaja de desarrollar un sistema a medida, es que se puede optimizar mucho, ya que evitamos pasar por clases, funciones… que se incluyen en los sistemas del mercado orientados para un uso más extendido.

También evitamos ser víctimas de agujeros de seguridad de los sistemas del mercado los cuales una vez detectados pueden ser atacados rápidamente ya que poseen una gran comunidad que los utiliza, pero atención, por este mismo motivo debemos ser más cautelosos con la seguridad de nuestro código ya que no obtenemos beneficio de una comunidad que advierte y corrige estos problemas de seguridad.

Por el contrario no nos beneficiarnos de un código y documentación que a sido desarrollado por una comunidad de expertos y la cual esta en continuo desarrollo.

¿Un único sistema web puede ser la base de casi todos los proyectos web de un productor de software?
Esto depende mucho de los proyectos del productor de software. Generalmente podría decirse que sí, ya que normalmente un productor de software utilizará un sistema como base para sus proyectos ya que este le ahorrará parte del desarrollo de los mismo, y podrá ofrecer precios más competitivos a sus clientes.
Existirán proyectos específicos en los cuales no utilizara un sistema base, ya sea porque este no cumpla con las necesidades del proyecto o por exigencias de sus propios clientes.
En todo caso un productor de software acumulara a lo largo del tiempo “scripts” de código que reutilizará en sus futuros proyectos.