CMS introducción

Daniel Julià Lundgren

PID_00168316

UOC
Universitat Oberta de Catalunya

Índice

Introducción

Uno de los aspectos que ha influido más en la evolución de Internet en los últimos años ha sido la aparición de múltiples **herramientas de gestión de contenidos** o **CMS** (del inglés *content management system*), que han permitido a una gran parte de la población mundial publicar contenidos fácilmente y hacerlos visibles en la red.

Uno de los momentos más importantes de esta evolución fue la expansión de los blogs o bitácoras personales que se dio hace unos años, debido en gran parte a la aparición de multitud de **herramientas de código abierto** y **gratuitas** que permiten a cualquier persona crear su propio blog sin tener conocimientos de HTML, de programación ni de diseño web.

El mundo de la publicación web, que hasta entonces había estado restringido a gente con conocimientos técnicos y que se daba normalmente en los ámbitos universitarios, se extendió a gran parte del público.

En esta asignatura hablaremos de las herramientas que han permitido esta evolución, los CMS. Uno de estos CMS, WordPress, que trataremos aquí, está específicamente pensado para la creación y mantenimiento de blogs y es, probablemente, el CMS más utilizado en la actualidad, no solo para blogs, sino para casi todo tipo de sitios web.

La aparición de CMS de código abierto corresponde con la evolución natural de Internet.

Los contenidos que existen en la web han ido evolucionando desde los inicios, cuando eran en su mayoría estáticos, de manera que, a medida que la cantidad de información creció, necesitaron ser dinámicos.

Un contenido **estático** es aquel que se crea "manualmente", escribiendo el contenido junto con el código HTML con un editor web como Dreamweaver. Una página estática siempre se verá igual. En cambio, en un sitio web **dinámico** las páginas HTML se construyen automáticamente extrayendo el contenido de una base de datos. Para ello, es necesario algún lenguaje en el servidor que permita leer de la base de datos (hacer consultas SQL) y construir el código de la página HTML. Según los parámetros que se pasen a la página (en la URL, utilizando "?" y "&"), el contenido de la página puede variar. Aquí aparece el concepto de **plantilla** o **tema** (*theme*, en inglés). La misma página debe estar pensada para mostrar diferentes contenidos, de diferente extensión.

Tradicionalmente (especialmente desde la expansión de las páginas web en Internet, aproximadamente en 1994 y hasta aproximadamente el año 2000), las páginas web se habían construido estáticamente o dinámicamente, mediante programación en el servidor y la base de datos. Los sitios web con contenidos dinámicos tenían un panel de administración, pero normalmente muy sencillo y limitado solo a opciones básicas.

El siguiente paso en esta evolución natural es hacia los gestores de contenido de código abierto. Este paso se dió a principios de la primera década del 2000. A medida que las empresas utilizaban más y más la técnica de la creación dinámica de páginas web, se hacía necesario poder reaprovechar el código utilizado en otras páginas web y diversas empresas empezaron a ofrecer comercialmente soluciones de gestión de contenidos propietarias, tras lo cual solo fue cuestión de tiempo la aparición de herramientas de código abierto también potentes.

El fenómeno de la Web 2.0. y el hecho de que cada vez más los propios usuarios son generadores de contenido, se han visto favorecidos por esta filosofía basada en herramientas que permiten a cualquier persona, sin conocimientos informáticos, publicar contenidos en la web.

Hoy en día resulta difícil pensar en la gestión de una web sin un buen gestor de contenidos, ya que la inmensa mayoria de contenidos de Internet procede de CMS propietarios o de código abierto.

1. Definición de CMS

Un **gestor de contenidos** o **CMS** (del inglés *content management system*) es una aplicación web usada para crear, editar, gestionar y publicar contenido digital.

La gestión de los contenidos se hace normalmente usando un navegador web convencional y a través de un **panel de administración**, al cual tiene acceso el administrador o los editores de la publicación.

Una característica común a todos los CMS es que **no son necesarios conocimientos informáticos avanzados** para poder gestionar el contenido del sitio web. Un perfil de editor de contenidos necesitará saber cómo se maneja la herramienta, pero en principio no necesita tener conocimientos de HTML, CSS o lenguajes de servidor. Esta es la ventaja fundamental de cualquier CMS.

En resumen y citando a la Wikipedia:

> "Un sistema de gestión de contenidos (en inglés *content management system*, abreviado CMS) es un programa que permite crear una estructura de soporte (o *framework*) para la creación y administración de contenidos, principalmente en páginas web, por parte de los participantes.
>
> Consiste en una interfaz que controla una o varias bases de datos donde se aloja el contenido del sitio. El sistema permite manejar de manera independiente el contenido y el diseño. Así, es posible manejar el contenido y darle en cualquier momento un diseño distinto al sitio sin tener que darle formato al contenido de nuevo, además de permitir la fácil y controlada publicación en el sitio a varios editores. Un ejemplo clásico es el de editores que cargan el contenido al sistema y otro de nivel superior (directorio) que permite que estos contenidos sean visibles a todo el público (los aprueba)."

Página web

En la Wikipedia podemos encontrar una definición completa de CMS:
http://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_contenidos.

El uso de un CMS tiene multitud de ventajas que detallaremos a continuación. La más importante de dichas ventajas es, como decíamos, que no es necesario un perfil alto de conocimientos en programación para instalar y configurar una aplicación y tener un sitio web dinámico funcionando desde cero, con lo que apenas existe "barrera de entrada".

La eclosión de un gran número de CMS ha coincidido, no casualmente, con el aumento en general de las plataformas de **código abierto**. Además de las ventajas propias de cualquier CMS, también existen otras intrínsecas a las opciones de código abierto en comparación con las opciones propietarias.

1.1. Ventajas y desventajas de las opciones de código abierto respecto a las propietarias

Las **ventajas** de las opciones de código abierto son:

- Coste nulo o bajo.

- Coste de alojamiento más bajo que soluciones propietarias, ya que las aplicaciones del servidor también se basan en plataformas de código abierto, como LAMP (Linux, Apache, MySQL, PHP).

- Acceso al código fuente para hacer cualquier modificación que creamos conveniente.

- Posiblemente la más importante: en muchos casos existe una **comunidad de desarrolladores** que se encargan de mantener y de actualizar el código permanentemente.

- La comunidad generalmente también crea y mantiene **extensiones** (***plugins***) de estos CMS que nos permiten aumentar las funcionalidades

- Solución de dudas a través de esta comunidad.

Naturalmente también existen algunas **desventajas**:

- Si nuestro proyecto no se adapta a la estructura de contenidos o el *workflow* del CMS, los cambios pueden ser muy difíciles de realizar.

- Dependemos de la comunidad para la actualización de la programación.

- La configuración del sistema a menudo es muy laboriosa y requiere bastante tiempo.

La cantidad de CMS disponibles es enorme: existen cientos, posiblemente miles, de CMS diferentes, la mayoría de ellos descargables directamente desde Internet. Para facilitar un poco la búsqueda, han aparecido algunos directorios donde se pueden consultar o buscar información sobre CMS. Uno de ellos es CMS Matrix (http://www.cmsmatrix.org/), un completo servicio de comparativa de características de gestores de contenidos de código abierto.

1.2. Ventajas y desventajas empresariales del empleo de un CMS de código abierto respecto a otras alternativas

A nivel de empresa, el empleo de un CMS tiene varias **ventajas**:

- Un CMS permite a la organización o empresa concentrarse en la creación de contenido o en la función específica buscada para el sitio web, en lugar del diseño o el desarrollo de la herramienta, que normalmente llevaría gran parte del tiempo.

- Perfiles no necesariamente técnicos de la organización pueden editar y publicar contenidos. Normalmente perfiles con conocimientos de usuario pueden ser formados sin demasiada dificultad en la mayoría de los CMS.

- Todos los contenidos quedan alojados en un solo lugar centralizado, en la base de datos, controlada por la propia empresa. Esto, además, tiene un potencial en el momento de ampliar o interconectar esta información con otros sistemas de la organización (intranet, etc.).

- Se pueden hacer búsquedas sobre el contenido, filtrando por fecha, tipo de contenido, autor, o cualquier otro campo.

- Normalmente, los contenidos pueden ser programados para ser publicados en una hora y fecha determinadas.

- Varios autores o traductores pueden trabajar simultáneamente sin dificultades. Para cada perfil de usuario, se pueden definir unos permisos específicos, por ejemplo, publicando el contenido solo en el momento en que un perfil determinado (por ejemplo, el perfil "editor") lo aprueba.

- Se pueden ampliar funcionalidades como calendarios, foros, galerías de fotos, etc. Se trata de funcionalidades que muchas veces están incluidas en los CMS, directamente o a través ampliaciones (*plugins*, extensiones o módulos según la denominación específica de cada uno de los CMS).

- Un CMS se puede integrar con otros sistemas de la organización, como por ejemplo *newsletters* de clientes u otros.

- Dado que el contenido y la presentación están separados, cambiar la apariencia (o *look-and-feel*) del sitio web es mucho más fácil que usando un sistema hecho a medida. Esta es una de las características más importantes que debe cumplir cualquier CMS y que, de hecho, depende en gran parte de la recomendable separación entre HTML y CSS.

Existen también algunas **desventajas** que debemos valorar:

- Aunque código abierto a menudo se equipara a gratuito, pensar esto es un error. Probablemente deberá hacerse una inversión inicial apreciable, tanto en dinero como en tiempo. Aunque podemos afrontar un proyecto con apenas desarrollo, cualquier CMS a menudo requiere mucho tiempo en configuración, adaptación e integración de contenidos.

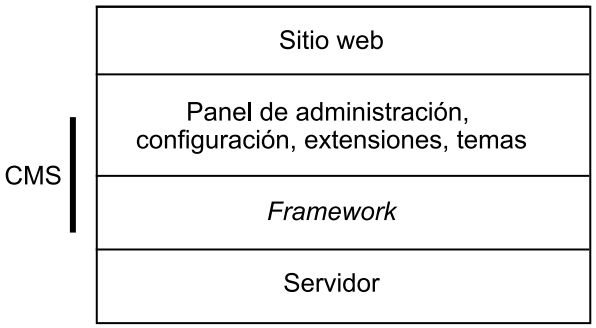
- Existe una curva de aprendizaje inicial en el equipo que se encarga de actualizar los contenidos.

- Usualmente, una persona (o más) de la organización debe convertirse en el *webmaster* del CMS para tomar la responsabilidad de su mantenimiento.

- Un CMS no convierte malos contenidos en buenos, ni convierte un mal escritor/editor en uno bueno. Los contenidos deben estar correctamente escritos. El CMS solo se encarga de gestionar los contenidos, pero no puede corregirlos.

- Cambiar un CMS de una plataforma a otra puede ser complejo y costoso en tiempo y dinero. Hay que tratar siempre de anticiparse a nuevas necesidades para poder hacer las transiciones de una manera gradual.

El hecho de que un CMS satisfaga un gran número de funcionalidades automáticamente también es la causa de que sea difícil en algunas ocasiones cumplir exactamente las funcionalidades del proyecto que queremos implementar. El ajuste de funcionalidades y de la presentación se consigue a través de personalización mediante la configuración en el panel de administración a primer nivel, o mediante programación a segundo nivel. En el caso de que esto no sea posible o sea dificultoso, deberíamos contemplar otras posibilidades, como el uso de un ***framework*** de desarrollo de aplicaciones web. Un *framework* es un paso intermedio entre un sistema desarrollado totalmente desde cero y un CMS completo. En cierta manera, un CMS es una evolución de un *framework* añadiendo una capa de administración y configuración.



CMS visto como *framework* + panel de administración.

2. *Frameworks* frente a CMS

Un ***framework*** es un conjunto de herramientas y librerías de programación bajo una arquitectura determinada (normalmente usan el paradigma modelo/vista/controlador, MVC), que facilitan el desarrollo de aplicaciones web complejas.

Veamos un extracto de la definición de *framework* en la Wikipedia:

> "En el desarrollo de software, un *framework* es una estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros programas para ayudar a desarrollar y unir los diferentes componentes de un proyecto."

Página web

En la Wikipedia podemos encontrar una definición completa de *framework*:
http://es.wikipedia.org/wiki/Framework.

En cierta manera, un *framework* se puede considerar la primera capa de cualquier CMS. De hecho, algunos CMS se pueden usar como *frameworks* de desarrollo. Desde otro punto de vista, podemos pensar que un *framework* está destinado al desarrollador y, en cambio, un CMS al editor de contenidos.

Así como es perfectamente posible crear un sitio web con un CMS sin tener que escribir una sola línea de código, con un *framework* sucede lo contrario: **cualquier funcionalidad implica un desarrollo de programación**, ya que no hay apenas herramientas visuales.

Página web

Para saber más encontraréis una lista de *frameworks* en la Wikipedia: http://en.wikipedia.org/wiki/Content_management_framework.

2.1. El paradigma modelo/vista/controlador

En un proyecto informático complejo es muy importante separar en módulos las diferentes partes del problema. Una separación muy frecuente es el denominado **paradigma modelo/vista/controlador** o **MVC**.

En la mayoría de *frameworks* se usa este sistema para separar tres vertientes de complejidad de una aplicación:

- El **modelo** corresponde a los **datos**. La conexión con la base de datos, el almacenamiento, la recuperación y también la lógica de negocio, etc.

- La **vista** se encarga de la **presentación**, es decir, la interfaz de usuario (correspondería con la plantilla o el tema de un CMS). Se basa en HTML y CSS pero con instrucciones del lenguaje de servidor para insertar los contenidos dinámicos.

- El **controlador** se corresponde con la **interacción con los usuarios** a base de eventos. Decide a partir de parámetros enviados a la página web qué modelo y vista debe llamar.

A efectos prácticos, en un *framework* basado en MVC los archivos del programa correspondientes a cada nivel están físicamente separados en carpetas diferentes. De esta manera, se consiguen notables ventajas:

- Mejora la claridad del código.
- Se puede reutilizar y ampliar más fácilmente.
- Se puede cambiar la presentación de manera independiente sin afectar al resto (solo es necesario modificar la vista).

Página web

En la Wikipedia podemos encontrar una definición más completa de MVC: http://es.wikipedia.org/wiki/Modelo_Vista_Controlador.

Ejemplos típicos de *frameworks* basados en MVC son:

- Ruby on Rails (basado en el lenguaje Ruby).
- Code Igniter (basado en PHP).
- Yii Framework (PHP).
- Zend Framework (PHP).
- CakePHP (PHP).
- Django (basado en el lenguaje Python).

Página web

En http://www.phpframeworks.com/ se puede encontrar una comparativa de *frameworks* basados en PHP.

La **ventaja de usar un *framework*** respecto a desarrollar una aplicación desde cero está en que **muchas funcionalidades** que probablemente necesitamos **ya están implementadas**.

Un ejemplo típico sería la gestión de usuarios (autentificación, registro, etc.): un *framework* que tenga la gestión de usuarios implementada nos permitirá ahorrarnos ese trabajo.

Ahora bien, en el caso de que queramos implementar alguna de estas funcionalidades en un *framework*, siempre lo deberemos hacer a través del **lenguaje de programación** de la plataforma, mediante una librería, es decir, programando a mano esta funcionalidad o editando archivos de configuración (a diferencia de como se haría en un CMS)

2.2. Diferencias entre un *framework* y un CMS completo

Un ***framework*** está enfocado a la programación. El desarrollador usa una serie de **librerías** de código que le permiten crear la aplicación y que requieren programación adicional. En cambio, en un **CMS** la opción de desarrollar código es opcional y, en principio, no necesaria, puesto que se configura todo visualmente a través del panel de administración. Para aumentar las funcionalidades de un CMS, normalmente añadimos **extensiones** o **módulos** que se activan visualmente (sin editar código).

Un **CMS** tiene funcionalidades de alto nivel (de la capa de presentación) que normalmente no tiene un *framework*. Algunas de estas funcionalidades pueden ser:

- Panel de administración para todas las opciones del CMS.
- Activación de temas.
- Gestión de los menús de la aplicación.
- Gestión de los bloques de contenidos en las pantallas.

En un CMS todas estas características se pueden activar sin necesidad de programar.

> Imaginemos un caso concreto: Si hemos creado una aplicación a partir de un *framework* y en un momento posterior queremos añadir una funcionalidad, como la posibilidad de autenticarse (o "loginarse") mediante una cuenta de Facebook, solo la podemos incorporar a base de programarla (bien a partir de una librería ya existente para el *framework*, si tenemos suerte, bien desde cero, en caso contrario). En un CMS, sin embargo, para añadir esta funcionalidad probablemente bastará con activar un módulo o *plugin* y configurarlo adecuadamente, o en caso de que no exista, desarrollarlo nosotros (e idealmente dejarlo a la comunidad para que lo reutilice y mantenga posteriormente).

En cierta manera, podemos considerar un *framework* como una capa inferior a un CMS completo. De hecho, algunos CMS integran, implícita o explícitamente, un *framework* de base sobre el cual funcionan el resto de funcionalidades.

De algún modo podríamos hacer la siguiente definición:

> *Framework* + Funcionalidades de alto nivel = CMS

Un ***framework*** es una herramienta muy útil para un desarrollador que deba crear una aplicación que, por las características que tiene, no se adapta perfectamente a ningún CMS concreto, o en el caso de que se prefiera la flexibilidad total que permite implementar una aplicación usando librerías de programación en lugar del panel de administración visual que se usa en un CMS.

Entre las características genéricas que puede tener un *framework* podemos destacar las siguientes:

- **Modelo/vista/controlador**. Estructuración a nivel de programación de la aplicación en tres capas que corresponden con los datos, la presentación y la interacción con el usuario.

- **Seguridad**. Disponer de mecanismos de protección contra ataques de *hackers*, *spam*, etc.

- **Mapeo de URL limpias** (*URL mapping*). Por ejemplo, mostrar en la URL "listado_productos", en lugar de "index.php?id=234124&op=234". Esto es importante a nivel de accesibilidad y posicionamiento en buscadores.

- **Acceso a la base de datos**. Herramientas para facilitar el acceso a la base de datos y hacerlo independiente del servidor concreto de bases de datos (por ejemplo, MySQL, PostgreSQL, Oracle,etc.).

- ***Templates***. **Plantillas** (o temas) para facilitar la separación de contenido y presentación.

- **Cacheo** (*caching*) **de las páginas para aumentar el rendimiento**. Esta opción permite guardar una versión creada por el lenguaje de servidor y la base de datos de la página en la base de datos, de manera que al recuperarla el acceso es mucho más rápido.

- **AJAX**. Petición y visualización de datos del servidor sin tener que recargar la página mediante JavaScript.

- **Autentificación de usuarios**. Implementación de *login*, registro, datos de perfil, roles y permisos de usuarios.

Dado que, como hemos dicho, un CMS se puede definir como un *framework* más una serie de capas adicionales, estas características también son las principales que encontraremos en cualquier CMS.

En resumen, utilizaremos un ***framework*** cuando necesitemos flexibilidad total o cuando nuestro proyecto no se ajuste bien a las funcionalidades predefinidas en un CMS. En caso contrario, la mejor opción probablemente será usar un **CMS**.

2.3. Ventajas de usar un CMS respecto a otras alternativas

Un CMS nos permitirá centrarnos en la solución concreta de las necesidades de nuestro proyecto y olvidarnos, al menos parcialmente, del desarrollo básico de la aplicación. Entre las **ventajas** más importantes tenemos las siguientes:

- Posibilidad de instalar y configurar la aplicación sin conocimientos de programación en el servidor, HTML o CSS (aunque estos conocimientos muchas veces son los que posibilitan ajustar perfectamente nuestra solución a los requisitos y personalizar las funcionalidades en detalle).

- Tiempo de desarrollo más bajo que implementando la solución desde cero o mediante un *framework* (siempre que se ajuste a las funcionalidades del proyecto).

- Libertad total para definir el diseño visual, ya que el CMS debe separar la presentación del contenido. Normalmente, la presentación se define a partir de temas (*themes*) Una personalización profesional requerirá crear o

modificar plantillas mediante HTML, CSS y, probablemente, PHP (u otro lenguaje de servidor).

- Seguridad garantizada al tener una comunidad pendiente de ofrecer actualizaciones no solo para nuestro proyecto, sino para todos los existentes basados en la misma plataforma. Si nos tuviéramos que encargar nosotros, este sería un proceso laborioso, costoso y continuo en el tiempo. Probablemente es una de las ventajas más importantes.

- Posibilidad de añadir extensiones (según el CMS se denominan *plugins* o *módulos*) que aumentan las funcionalidades.

- Estructura de navegación coherente implementada por defecto. Al mantener la misma plantilla para todos los contenidos, se favorece la coherencia en la navegación.

- Validación HTML+CSS (al menos, la debería garantizar el propio CMS).

- Crear diferentes roles para el flujo de trabajo en la publicación (por ejemplo, administrador, editor, miembro de la comunidad, etc.).

- Disminuir el tiempo de desarrollo. Muchas funcionalidades ya están implementadas.

- Disminuir el coste de desarrollo.

- Facilitar la generación y edición de contenidos.

- Mantenimiento. El equipo de desarrollo y la comunidad de usuarios del proyecto de código abierto se encargan de mantener el código de la aplicación y ofrecer actualizaciones.

De este modo, nos puede interesar usar un **CMS** en el caso de que las especificaciones de este se adapten bien a los requerimientos (funcionalidades) de nuestro proyecto.

> Por ejemplo, en el caso de que queramos desarrollar un blog, una buena elección probablemente será WordPress, ya que es un CMS diseñado específicamente para esta necesidad. O en el caso de que nuestro objetivo sea crear un foro, podríamos usar phpBB, un foro basado en PHP.

En el momento en que las funcionalidades sean más diversas y no tan específicas, quizás nos interese un CMS más flexible como Joomla o Drupal.

Pero si las necesidades que tenemos no se ajustan realmente a ningún CMS del mercado, probablemente utilizar un ***framework*** y desarrollar una aplicación a medida será la mejor solución, aunque en este último caso será necesario desarrollar partes del código de la aplicación.

Uno de los principales **inconvenientes** de los **CMS** es que, debido a que están pensados para implementar un gran número de funcionalidades por defecto, si nuestras funcionalidades no se ajustan perfectamente a las que proporciona el sistema y tampoco la configuración nos permite ajustarlas, el esfuerzo que deberemos realizar (mediante desarrollo a medida) puede ser superior al que deberíamos hacer partiendo de una herramienta más sencilla como un *framework* o incluso partiendo desde cero. Por esta razón, el primer paso debe ser siempre evaluar a partir de los requerimientos de la aplicación si estos se pueden satisfacer con un CMS determinado.

Observación

Es importante tener en cuenta que todos estos CMS son proyectos "vivos", en continua evolución, y que por tanto hay que estar pendiente de actualizaciones y también de la aparición de nuevas alternativas

3. Uso de los CMS

Con cualquier CMS, los pasos que deberíamos seguir para ver si es un candidato a solucionar nuestro problema son los siguientes:

- ¿Se adapta a las necesidades y funcionalidades que queremos implementar?
- ¿Tenemos disponibilidad en nuestro servidor de la plataforma tecnológica en la que está basado?
- ¿Podremos conseguir a través del panel de administración la estructura de contenidos y el flujo de navegación que se requiere?

Para poder contestar a estas preguntas es necesario, por supuesto, conocer bien y tener experiencia sobre las posibilidades de los CMS en cuestión.

En un buen CMS, el contenido y la presentación deben estar totalmente separados, de manera que, en principio, podemos diseñar cualquier presentación usando HTML/CSS e implementarla dentro del sistema. El inconveniente suele estar más en la estructura de contenidos, en el flujo de trabajo, funcionalidades específicas, etc., que en la presentación.

3.1. Tipos de CMS y sus características

Cada CMS tiene unos requerimientos técnicos (plataforma tecnológica), está especializado en una función específica y tiene una licencia de uso determinada. Según estos puntos de vista podemos clasificarlos:

- Según la **plataforma** (incluyendo el lenguaje de programación empleado).
- Según el tipo de **licencia** (propiedad del código: propietario o de código abierto).
- Según el tipo de **aplicación que queremos desarrollar** (foro, blog, portal, etc.).

3.1.1. Según la plataforma

En un servidor web tenemos normalmente tres componentes:

- El **servidor de páginas web** (por ejemplo, Apache).
- El **lenguaje de programación** del servidor (por ejemplo, PHP).
- El **servidor de bases de datos** (por ejemplo, MySQL).

El componente más restrictivo para un CMS es el **lenguaje de programación** del servidor, ya que el código del CMS está escrito en ese lenguaje y, por tanto, es una condición necesaria para su funcionamiento. El servidor web y el de la base de datos son, en muchos casos, intercambiables (debemos consultar los requisitos mínimos de la plataforma para asegurarnos de ello); sin embargo, el lenguaje de servidor no lo es nunca.

La mayoría de servidores de bases de datos relacionales funcionan en SQL, un lenguaje de base de datos bastante estándar, de manera que a menudo es posible cambiar de servidor de bases de datos sin grandes cambios en el código del CMS. Del mismo modo, la tarea principal del servidor web, servir páginas HTML, es independiente del resto de componentes, de manera que muchas veces podemos usar diferentes servidores web (como Apache o MS IIS) sin problemas.

Algunos lenguajes de servidor existentes sobre los cuales existen múltiples CMS:

- ASP.NET (propietario Microsoft).
- Java.
- PHP.
- Ruby On Rails.
- Python.

Observación

En el mismo servidor podemos tener disponibles varios lenguajes simultáneamente, por ejemplo PHP y Java.

Observación

Normalmente, los CMS de código abierto funcionan en plataformas de código abierto (como PHP), pero no tiene que ser así en todos los casos, ya que perfectamente podemos tener un CMS de código abierto sobre ASP (sistema propietario) o un CMS propietario sobre PHP.

3.1.2. Según la propiedad del código

Según esta aproximación podemos dividir las aplicaciones de CMS en dos grandes grupos:

- **De código abierto**. Son las que permiten el acceso al código fuente de manera que, si es necesario, se puede modificar cualquier fragmento o incluso crear nuevas aplicaciones (nuevos CMS) a partir de la versión original.

- **De código privativo**. Solo su creador o empresa puede desarrollar o modificar el código base de la aplicación.

La mayoría de los CMS que vamos a ver aquí son de código abierto. Todos ellos se rigen por una licencia que especifica los términos y condiciones sobre las cuales se pueden usar y modificar. Existen multitud de licencias, aunque algunas están mucho más extendidas que otras, como por ejemplo la licencia GNU GPL (*general public license*), la más extendida entre los CMS que vamos a analizar.

Página web

Para saber más sobre licencias de código abierto, podéis ver la lista de licencias por nombre de la Open Source Iniciative:

http://www.opensource.org/licenses/alphabetical.

3.1.3. Según el tipo de aplicación a desarrollar

En el momento de escoger un CMS, la primera decisión viene determinada por el tipo de aplicación que queremos desarrollar. Partir de un CMS que se ajuste a la funcionalidad principal de nuestro proyecto es crucial para el éxito de su implementación. Así, por ejemplo, existen CMS pensados para distintas necesidades como:

- Creación y administración de blogs.
- Portales de empresas.
- Entornos educativos.
- Redes sociales.

Más adelante veremos en detalle algunos de estos tipos de proyectos más comunes y sus requerimientos, conjuntamente con una lista de CMS disponibles actualmente adaptados a cada uno de ellos.

4. Componentes del servidor necesarios para instalar CMS

Los tres componentes básicos en cualquier servidor de aplicaciones web (**servidor de páginas web**, **lenguaje del servidor** y **servidor de base de datos**) funcionan bajo un determinado **sistema operativo**. Todas estas opciones se pueden combinar entre ellas y existen, además, varias versiones, por lo que es muy importante asegurarnos siempre de los requerimientos mínimos del CMS que vamos a usar.

Algunas de las diferentes opciones que tenemos para cada uno de los componentes son:

- **Sistema operativo**: Linux, MacOS, Windows, etc.
- **Servidor de páginas web**: Apache (el más utilizado), IIS (Microsoft), lighttpd, etc.
- **Lenguaje de servidor** (o servidor de aplicaciones): PHP, ASP, Java, Ruby, Python, etc.
- **Servidor de base de datos**: MySQL, PostgresSQL, Oracle, etc.

En realidad, nuestro CMS podría estar funcionando prácticamente en cualquier combinación de los anteriores cuatro componentes.

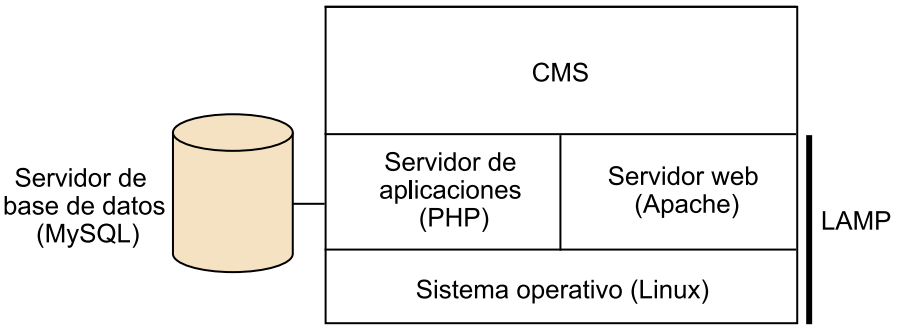
4.1. LAMP

La mayoría de los CMS que vamos a ver están basados en la **plataforma LAMP**, que corresponde a los componentes del servidor web: **Linux+ Apache+MySQL+PHP**.

Todos estos componentes son de **código abierto**, lo que ha facilitado su difusión y su empleo en gran número de aplicaciones.

- **Linux**: Sistema operativo de código abierto con el que funcionan actualmente una buena parte de los servidores web en Internet.
- **Apache**: Servidor web de código abierto, también de los más usados actualmente, aunque existen muchos otros.
- **MySQL**: Servidor de base de datos SQL de código abierto, actualmente propiedad de la empresa Oracle.

- **PHP**: Extensión del servidor web para poder interpretar el lenguage.



Componentes del servidor en una configuración LAMP.

En general, es posible instalar estos CMS en otras plataformas, como Windows o Mac OS, y en otros servidores web (diferentes a Apache). Lo que sí suele ser un requisito importante en los CMS que vamos a ver es **PHP+MySQL**.

En el caso de un sitio web construido a base de páginas estáticas, solo necesitaríamos el servidor de páginas web (no PHP ni MySQL ni otros lenguajes de servidor y bases de datos).

4.2. Servidores web

Podemos referirnos al servidor web como servidor http (por el protocolo) o como http daemon (abreviado httpd), es decir, demonio de http, porque funciona en modo *background* sin presencia visual.

El servidor web más utilizado actualmente es **Apache**, una aplicación de código abierto que forma parte de un grupo de proyectos más amplio de la Fundación Apache y que, como hemos dicho, es el componente de servidor web de la llamada plataforma LAMP.

Usando solo el servidor web podemos tener páginas en formato HTML, texto plano, imágenes o, en general, cualquier archivo que no necesite ser interpretado. Por tanto, es el único componente que necesitamos en el servidor para tener un sitio web con **páginas estáticas**. Sin embargo, estos servidores pueden ser ampliados mediante la activación de determinados módulos, como son el que permite **páginas dinámicas** en PHP (mod_php) o conexiones seguras mediante SSL (mod_ssl), entre muchos otros.

Páginas web

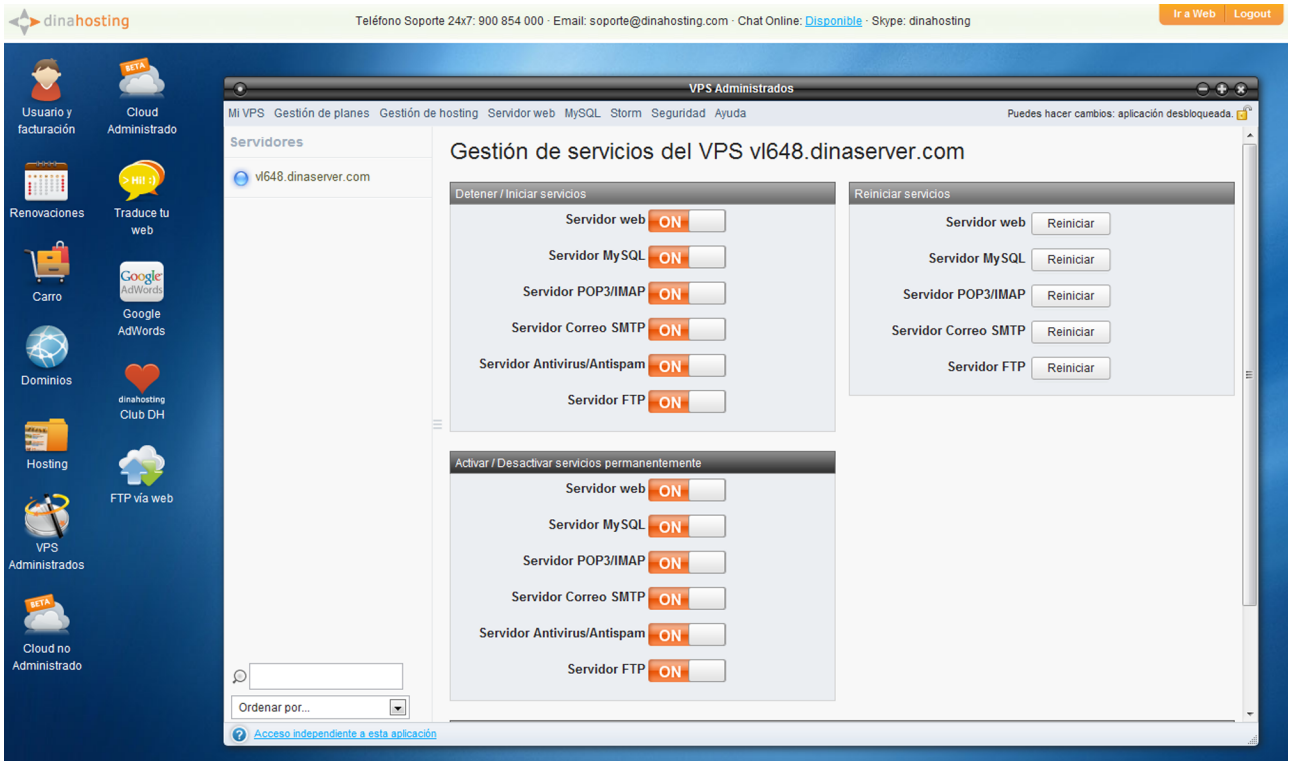
Sobre Apache:
http://httpd.apache.org/.
Otros servidores web en la Wikipedia:
http://es.wikipedia.org/wiki/Servidor_HTTP_Apache.
http://en.wikipedia.org/wiki/Comparison_of_web_server_software.

4.3. Panel del alojamiento web

Para acceder a nuestro servidor normalmente lo haremos usando un **panel de administración** del propio servicio de alojamiento. En el panel del alojamiento tendremos acceso a información importante como:

- Tipo de plataforma.
- Bases de datos (y cuentas asociadas).

- Configuración del servidor.
- Gestión de cuentas de correo y FTP.



Ejemplo de panel de control de un servicio de alojamiento web (*hosting*).

Las opciones y la apariencia del panel del control variarán según la empresa de alojamiento, y pueden ser muy sencillas (o no existir) o muy completas, como las de la imagen anterior.

En general las funciones que necesitaremos para poder instalar un CMS son:

- Acceso por FTP al servidor (cuenta asociada).

- Acceso a la base de datos (por ejemplo, mediante "phpMyAdmin" si usamos PHP+MySQL). Necesitaremos el nombre y la cuenta asociada a la base de datos.

- La dirección pública desde la cual acceder al contenido de la web (para poderlo probar). Por ejemplo, "misitioweb.com" puede corresponder a la dirección del servidor FTP o encontrarse este en otra URL; debemos asegurarnos de este punto.

4.4. Acceso por FTP

Para acceder al servidor remoto normalmente usaremos un *programa de FTP*. *FTP* significa *file transfer protocol* o *protocolo de transferencia de archivos*; como su nombre indica, no es más que un sistema para copiar archivos de una máquina a otra, normalmente entre nuestro PC de trabajo y el servidor web.

Recomendación

Recomendamos usar FileZilla (http://filezilla-project.org/), un programa de FTP de código abierto y disponible en todas las plataformas.

Mediante el protocolo FTP tenemos acceso a todos los archivos (y directorios) del servidor remoto, pudiendo incluso cambiar los permisos de los archivos y directorios.

En algunos casos es necesario tener acceso total al servidor, no solo a sus archivos, sino también a ejecutar programas; para este propósito podemos usar algún programa que permita el acceso mediante telnet/ssh. Gracias al **protocolo telnet** disponemos de un terminal con el que interactuamos con la máquina remota como si estuviéramos ahí mismo, es decir, con acceso a todos los recursos (siempre que tengamos los permisos correspondientes). Eso sí, este tipo de acceso es mucho más avanzado y requiere de conocimientos en el sistema operativo Linux o Unix.

Observación

En muchos CMS algunos directorios necesitan permisos de escritura para que la aplicación pueda funcionar y esto es una causa importante de posibles problemas de funcionamiento.

4.5. Gestores de bases de datos

Aunque existen CMS que no requieren base de datos (usan archivos de texto o XML como sistema para almacenar la información), la mayoría de ellos sí lo requieren.

En la base de datos del CMS no solo se guarda el contenido del sitio web, sino también la configuración y cualquier dato que tenga que ser almacenado para su uso posterior. Por tanto, si queremos trasladar nuestro sitio web desde el servidor de pruebas al de producción, debemos tener en cuenta qué valores de la configuración se guardan en la base de datos para prever problemas.

El gestor de bases de datos más utilizado en CMS de código abierto es **MySQL**, una **base de datos relacional** basada en **SQL**, propiedad de Sun Microsystems desde el año 2008 y de Oracle desde el 2009 (cuando Oracle adquirió Sun) y que está instalado en millones de servidores en todo el mundo. La popularidad de MySQL está muy ligada a PHP, aunque es un servidor totalmente independiente de PHP y puede ser usado por prácticamente cualquier lenguaje de programación.

MySQL está basado en SQL, un lenguaje estándar de acceso a bases de datos relacionales. Las **bases de datos relacionales** están formadas por tablas que contienen registros (fichas de información) que, a su vez, contienen campos. Se llaman *bases de datos relacionales* porque los campos de las diferentes tablas se pueden relacionar.

Por ejemplo, podemos tener un campo autor apuntando a una tabla de autores.

SQL es un lenguaje de alto nivel donde las instrucciones permiten crear, leer o modificar los registros de la base de datos, mediante instrucciones como SELECT, INSERT, etc.

En principio, es posible acceder a diferentes servidores de bases de datos con la misma sintaxis de SQL, aunque existen variantes que pueden limitar la compatibilidad entre sistemas. SQL es un lenguaje basado en "línea de comandos" (instrucciones de texto que se ejecutan en un terminal), pero existen utilidades, como PhpMyAdmin (basado en PHP), que permiten un acceso mucho más visual e intuitivo al servidor MySQL. Otros gestores de bases de datos basados en SQL son SQLServer (Microsoft) o PostgreSQL.

Sqlite (http://www.sqlite.org/) es un caso especial, ya que es una librería de programación que permite hacer consultas SQL estándar sin servidor de base de datos, todos los datos se guardan en un único archivo. Esto tiene la ventaja de que se puede usar, por ejemplo, en aplicaciones de móvil para Android o iOS.

4.6. Lenguajes de programación

Además de usar una base de datos, todos los CMS están implementados en algún lenguaje de programación web. El **lenguaje utilizado condiciona el servidor** donde tendremos finalmente ejecutando nuestra aplicación cuando esté en explotación. En la fase de definición técnica deberemos elegir siempre un CMS que se adapte a los requerimientos del servidor donde lo queremos instalar.

Los lenguajes de programación web más usados actualmente son los siguientes:

- **Active Server Pages** (ASP, ASP.net). Solución propietaria de Microsoft.
- **Java**. De Sun Microsystems (actualmente Oracle).
- **PHP**. Lenguaje diseñado inicialmente para la creación de páginas dinámicas, el más usado en los CMS actuales.
- **Ruby**. Uno de los más recientes (apareció en 1995). A menudo se usa Ruby en un *framework* de programación llamado *Rails* (Ruby on Rails)
- **Python**. Un lenguaje de sintaxis simple y sencilla

Curiosidades

El lenguage Ruby (rubí) está inspirado en contraposición al nombre de otro lenguaje, Perl (perla). Por otro lado, el nombre del lenguaje de programación Python está, efectivamente, inspirado en el grupo humorístico británico Monty Python.

Observación

Por su naturaleza, la inmensa mayoría de lenguajes de programación web son interpretados, lo que garantiza que las aplicaciones puedan funcionar en múltiples plataformas. Además, muchos de ellos también son "orientados a objetos" y de código libre.

4.7. Alojamiento web (*hosting-housing*)

Una cuestión importante en cualquier aplicación web es la del alojamiento. Durante el desarrollo, normalmente, y en su fase de explotación pública, el proyecto estará funcionando en un servidor externo contratado a una empresa externa. Aunque técnicamente es posible tener el servidor web en nuestra empresa, hay diversos motivos por los que es mucho mejor contratar un servicio externo:

- Los *data centers* (como se llaman las instalaciones donde se alojan los servidores de las empresas de alojamiento) están preparados para mantener una temperatura óptima para el funcionamiento de los servidores y para hacer frente a posibles cortes del suministro eléctrico.

- Disponen de personal preparado en caso de incidencias.

- Disponen de buenos canales de comunicación con gran ancho de banda.

- Optimizan recursos al compartirlos entre diversos clientes.

Estas razones explican por qué existen muy pocas empresas que alojen ellas mismas su página web.

Un alojamiento puede ser básicamente de tres tipos, en este caso, organizados de menos a más costoso:

- **Servidor compartido**. Varios sitios web son alojados en el mismo servidor físico. Es una opción menos costosa pero que no tiene el rendimiento necesario en proyectos con un gran número de visitas. Además, no tenemos acceso a la configuración básica de la máquina y, por tanto, no podemos instalar aplicaciones que no vengan por defecto.

- **Servidor virtual privado**. Se trata de una máquina virtual que funciona como un servidor privado completo y que se ejecuta como tal. Tiene las ventajas de un servidor dedicado pero con menor coste; el rendimiento no es tan alto como en el caso del servidor dedicado.

- **Servidor dedicado**. Consiste en alquilar una máquina física completa, de modo que se puede configurar a medida. Se trata de la opción más cara, solo utilizada en sitios con un gran número de visitas o necesidades muy especiales.

Normalmente, tenemos la opción de empezar con un servidor compartido y, a medida que el número de visitas aumenta (y por tanto, el tráfico), pasar luego a un servidor dedicado. Si el número de visitas sigue aumentando, deberemos pasar de uno a varios servidores dedicados. Aquí es donde nuestro CMS debe tener la posibilidad de poder "escalar" bien en un entorno de este tipo (con varios servidores simultáneos).

Por supuesto, sitios web con gran número de visitas, como flickr.com, facebook.com, etc., no solo tienen varios servidores dedicados sino que incluso disponen de uno o varios *data centers*.

Facebook, por ejemplo, dispone de más de 60.000 servidores para su funcionamiento (datos de 2012).

4.8. Mantenimiento, *backups*

Un problema importante en un sistema informático es realizar correctamente el mantenimiento del mismo. Periódicamente hay que comprobar si hay **actualizaciones** del sistema o de extensiones o temas que se están usando. Si hay que actualizar archivos, es importante hacer una copia de seguridad para poder recuperar los datos en el caso de que algo falle.

Normalmente, el propio CMS dispone de mecanismos de aviso cuando algún módulo o componente necesita una actualización. En este caso, deberemos seguir las instrucciones.

En algunos casos, la actualización es casi automática (como en WordPress), pero en otros deberemos descargar nosotros los archivos, subirlos en la ruta correcta y seguir algún paso documentado en el CMS concreto que estemos utilizando.

Es muy importante actualizar nuestro sistema, ya que en caso contrario posiblemente dejaremos abiertos agujeros de seguridad a través de los cuales pueden entrar *hackers* con malas intenciones, o simplemente para demostrar que lo pueden hacer, o programas indeseados que inyecten *spam*.

En general, aparte de las actualizaciones, siempre debemos hacer **copias de seguridad** (o ***backups***) periódicas, especialmente de la base de datos, ya que ahí es donde se guardan tanto los datos de los contenidos como la configuración y personalización que hayamos hecho en el sistema.

Lo más práctico es disponer de alguna herramienta de *backups* automáticos. Algunos servicios de alojamiento (*hosting*) disponen de estas herramientas, que permiten indicar fácilmente con qué periodicidad queremos hacer una copia de seguridad de la base de datos, para poder recuperarla en caso de problemas.

4.9. Servidor de pruebas local

A menudo es interesante usar un servidor local de pruebas para el desarrollo. Una vez hayamos completado el proyecto en local, subiremos los archivos al servidor definitivo (copiando los archivos, la base de datos y ajustando la configuración).

Observación

Es lo que se denomina en argot informático un *deploy* o *deployment*: pasar del sistema temporal o de pruebas al operacional o de explotación.

El hecho de usar un servidor local durante el desarrollo tiene varias ventajas:

- Es mucho más eficiente y rápido, podemos editar los archivos en local (no necesitamos acceder a Internet), por lo que podremos probar cualquier cambio con más rapidez.

- Nuestro proyecto no será público (accesible desde Internet) mientras lo estamos desarrollando y, por tanto, será más seguro si queremos mantener la privacidad del proyecto.

- Podremos visualizar el sitio web desde la red local de nuestra organización si lo deseamos. Si el servidor forma parte de la red local, podremos acceder desde cualquier ordenador de la red usando direcciones ip locales (como por ejemplo, "192.168.1.10").

A efectos prácticos, no hay ninguna diferencia entre un servidor local (en nuestra máquina) y un servidor remoto aparte de los puntos anteriores.

Si usamos una plataforma LAMP, existen varios paquetes que integran el servidor web Apache junto con PHP y MySQL para poder usarlos en modo local; algunos de ellos son:

- **Xampp** (http://www.apachefriends.org/es/xampp.html) (válido tanto para Mac como Windows y Linux).

- **EasyPHP** (http://www.easyphp.org/) (solo Windows).

- **Wamp** (http://www.wampserver.com/en/) (solo Windows).

- **Mamp** (http://www.mamp.info/en/index.html) (solo Mac).

En algunos sistemas operativos, como Mac OS o Linux, estos servidores ya están instalados por defecto (y por tanto, basta con seguir las instrucciones del sistema operativo en cuestión).

Cuando trabajamos en local en lugar de con una dirección "pública", nuestro servidor tendrá una dirección "privada" solo accesible desde nuestro ordenador, normalmente con la dirección "http://localhost" o "http://127.0.0.1".

Observación

Para probar si tenemos algún servidor local funcionando en nuestro sistema, hay que entrar en un navegador web la dirección "localhost" y ver qué aparece. Si aparece un "error 404", no tenemos en este momento ningún servidor local funcionando. En caso contrario, observad de qué servidor se trata.

El servidor web siempre utiliza un directorio de nuestro sistema de archivos (normalmente llamado "www" o "htdocs") donde deben existir los archivos para poder acceder a través del navegador.

Así, por ejemplo, al copiar un archivo "test.html" dentro de una carpeta "test" dentro de "www" o "htdocs", podremos acceder usando el navegador en la dirección "http://localhost/test/test.html".

Además, tendremos acceso a otras utilidades como el gestor de la base de datos (normalmente "phpMyAdmin", si utilizamos MySQL). Dependiendo del sistema escogido, probablemente esta opción aparecerá en el menú de la herramienta.

Usando un servidor local podremos instalar y probar cualquier gestor de contenidos rápidamente.

Ejercicio

Descargar uno de estos servidores locales (por ejemplo Xampp, o usar el que existe instalado por defecto, si es el caso), instalarlo, activarlo y acceder mediante el navegador a la dirección "localhost" para comprobar que funciona correctamente. Buscar el acceso a "phpMyAdmin" y ver si ya existen bases de datos de prueba definidas.

5. Características de un buen CMS

Existen una serie de características que cualquier buen gestor de contenidos o CMS debe cumplir. La principal ventaja de un CMS es que estas características **se activan automáticamente**, por el simple hecho de usarlo, sin esfuerzo adicional por parte del editor, administrador o desarrollador de la aplicación.

5.1. Separación de contenido y presentación

La **separación de contenido y presentación** es posiblemente la característica más importante. El CMS debe encargarse de la gestión de los contenidos y la estructura, pero el nivel de presentación debe estar totalmente separado, normalmente en forma de **temas** (*themes*).

De la misma manera que usando HTML y CSS separamos el contenido de la presentación, en un CMS cumplimos el mismo objetivo utilizando temas.

Un **tema** está construido por unas **plantillas** en algún lenguaje de programación (como PHP) y uno o varios archivos CSS vinculados.

El hecho de utilizar temas tiene múltiples ventajas:

- Asegurar la correcta validación del HTML y CSS, aunque modifiquemos el contenido, ya que la validación dependerá solo de la validación del tema que estemos usando.
- Poder visualizar correctamente el contenido en diferentes plataformas, como las móviles (solo deberemos activar un tema diferente en este caso).
- Asegurarnos de que el editor del contenido no modificará el código HTML o CSS, imposibilitando romper la validación de HTML y CSS.
- Poder cambiar el diseño de nuestro sitio en cualquier momento y por cualquier causa, sin tener que modificar para nada los contenidos.

Observación

Estas ventajas existen solo en el caso de que el editor integrado en el CMS no permita la introducción de código HTML que podría alterar la validación y presentación de las páginas.

En un CMS siempre tenemos la posibilidad de descargar y activar temas desarrollados por la comunidad, pero también la de crear los nuestros según las necesidades que tengamos.

5.2. Facilidad de uso del panel de administración

Un aspecto crucial en cualquier CMS es la facilidad de uso del **panel de administración**. Aunque el sistema tenga muchas funcionalidades avanzadas, no nos servirá de nada si estas no son fáciles de utilizar y comprender.

Debemos tener en cuenta que el panel de administración normalmente tiene, al menos, **dos tipos de usuarios**, cada uno de los cuales entrará con su cuenta especial al panel y tendrá permisos para realizar algunas tareas, pero no para realizar otras:

- **Administrador**. Las opciones enfocadas al administrador del sitio tienen que ser configuradas en el momento de desarrollar, pero que luego normalmente no se van a modificar. Este sería el caso de opciones avanzadas como la configuración de la *caché*, sobre si se publican por defecto los contenidos en la portada o no, bloques en la pantalla, construcción del sitio en general, etc.

- **Editor**. Las opciones que aparecen para este perfil son las referentes al contenido y las que se usan mientras se están actualizando y manteniendo el contenido durante la vida útil del sitio, sin afectar al funcionamiento ni a la estructura del sitio.

En los CMS más avanzados es posible definir tantos perfiles como queramos, cada uno con permisos específicos.

> De esta manera, podríamos definir, por ejemplo, un tipo de usuario "traductor", que tiene permiso para traducir contenidos pero no para publicarlos.

Desde este punto de vista de la usabilidad, para estos usuarios un panel de administración debe:

- Ser **intuitivo**. Las opciones deben ser autodescriptivas. Idealmente, no deberíamos necesitar un manual para poder empezar a usarlo (aunque siempre habrá una curva de aprendizaje).

- Proporcionar un **editor WYSIWYG** (*what you see is what you get*), de manera que no se necesiten conocimientos básicos de HTML y, más importante, evitar errores por el incorrecto cierre de etiquetas HTML, por ejemplo, o la inclusión de elementos que podrían modificar el diseño. En estos editores es conveniente poder activar el uso de determinadas etiquetas (como cabeceras, listas, *strong* o *em*), pero no otras que podrían afectar la presentación del HTML.

- Permitir la **previsualización**. Opción de poder previsualizar el contenido antes de publicarlo definitivamente, en la plantilla (tema) que tengamos activada, de modo que podamos visualizar el contenido tal como lo verá el

usuario final. En algunos CMS el panel de administración tiene un aspecto diferente que el sitio final (es el caso de WordPress), por lo que esto es especialmente importante.

5.3. Seguridad

En cualquier sistema informático complejo (como es el caso del que estamos tratando) existe la posibilidad de que haya **agujeros de seguridad** o **vulnerabilidades** que sean aprovechados por gente y programas maliciosos.

Cualquier agujero de seguridad que exista podría permitir que personas o programas automáticos (*bots*) puedan, entre otras cosas:

- Extraer información privada de las cuentas de los usuarios del sitio.
- Publicar contenidos directamente sin permiso (*spam*).
- Usar el alojamiento web para otros propósitos no deseados.

Normalmente, cuando se detecta un agujero de seguridad la comunidad de desarrolladores del CMS soluciona rápidamente el problema ofreciendo una actualización del software. Por eso es importante actualizar los archivos de nuestro CMS periódicamente para evitar en lo posible que estos problemas de seguridad afecten a nuestro sitio.

Hay que destacar que, aunque pueden existir estos problemas, los CMS son normalmente mucho más seguros que un sistema hecho a medida, ya que en este caso no existen tantos recursos para poder revisar continuamente el código y detectar vulnerabilidades. Por otro lado, también es cierto que son menos probables los ataques porque las vulnerabilidades son menos conocidas. Es decir, la mayor seguridad de un CMS va acompañada de una difusión más grande en el caso de que exista un error de seguridad. Así, cuando existe una vulnerabilidad en un CMS ampliamente utilizado, esta vulnerabilidad se conoce rápidamente, tanto por parte de la gente con buenas intenciones como por la gente con posibles malos propósitos. Motivo de más para estar alerta y actualizar siempre cuando aparezcan las actualizaciones.

5.3.1. Algunas amenazas a la seguridad

Existen algunas técnicas maliciosas que permiten entrar en sistemas informáticos y respecto a los cuales los CMS deben estar protegidos. Algunas de las más conocidas son:

- ***Cross side scripting*** (o **XSS**). Se trata de ejecutar instrucciones en la parte del cliente del navegador (mediante JavaScript o Flash, por ejemplo) desde sitios remotos simulando que se ejecutan en el mismo dominio, con el

objetivo de tener acceso a información de nuestro sitio o modificar información, falsificar *cookies* para simular sesiones de usuarios ficticios, etc.

- **SQL injection**. Esta técnica consiste en "inyectar" instrucciones de SQL en las variables de la URL con el objetivo de tener acceso a la base de datos del sitio. Por ejemplo, pasar en la URL como variable alguna cadena del estilo de "`?q=delete from usuarios...`" podría tener efectos directos sobre la base de datos si el código no está protegido contra ello.

En ambos casos la solución está en el mantenimiento por parte de los desarrolladores y la actualización por parte de los usuarios del código de la aplicación con el objetivo de protegerse de estos posibles ataques. Se trata de un proceso continuo en el tiempo, ya que siempre es posible que se detecten nuevos problemas que no se habían detectado anteriormente. A la protección de este tipo de vulnerabilidades corresponden la mayoría de las actualizaciones que se deben hacer en un CMS periódicamente.

Otra vulnerabilidad distinta es el ***spam***. Se denomina *spam* a cualquier mensaje o comentario no solicitado, normalmente con intenciones publicitarias. Estos comentarios son enviados por programas automáticos que rastrean Internet en busca de sitios donde poder publicar.

Es recomendable usar algún sistema de protección contra el *spam*, como por ejemplo los ***captchas***. Un *captcha* es un sistema desafío-respuesta que tiene como objetivo determinar si quien publica el contenido o comentario es una persona o un robot (programa informático preparado para ello). Normalmente el desafío se implementa en forma de lectura de una imagen con palabras distorsionadas (para que un OCR no lo pueda interpretar) o de una operación matemática. De esta manera los *bots* no pueden publicar información ya que no son capaces de descifrar el contenido de la imagen.

Normalmente la opción de añadir *captchas* en un CMS se proporciona a través de extensiones (*plugins* o módulos, según la denominación de cada CMS).

5.4. Extensibilidad (*plugins* o módulos)

Definimos la **extensibilidad** como la capacidad de aumentar las funcionalidades del sistema con el menor coste de desarrollo posible. En la mayoría de CMS esto se consigue mediante la instalación de **extensiones**, **módulos** o ***plugins*** (la nomenclatura varía según el CMS pero el concepto es el mismo).

En los CMS más utilizados (como WordPress o Drupal), el número de extensiones existentes se cuenta por miles y permite ampliar enormemente el número de funcionalidades de los mismos. Estas extensiones suelen ser desarrolladas y mantenidas por individuos o equipos de personas que normalmente no pertenecen al equipo de desarrollo del CMS principal.

En cualquier caso, antes de usar una extensión hay que asegurarse siempre de que la versión implementada es **estable** (sin errores importante detectados). Una buena manera de saber si una extensión es suficientemente estable es que la base de usuarios sea importante.

Estas ampliaciones se suelen programar a partir de una plataforma propia del mismo CMS (API o *framework*), de manera que se garantiza una capa de separación del núcleo del sistema con el de la funcionalidad que queremos proporcionar (también para evitar problemas de seguridad). Usando API, existe la posibilidad de crear nuestras propias ampliaciones en el caso de que no encontremos ninguna que se adapte a nuestras necesidades.

Cuando instalamos una extensión, debemos asegurarnos siempre de que pertenece a la misma versión que el núcleo del CMS y que está suficientemente probada por la comunidad (en caso contrario, nos podemos encontrar con problemas). También es aconsejable hacer siempre una copia de seguridad de la base de datos antes de instalarla.

Observación

Desde el punto de vista del desarrollador: siempre que queramos crear una nueva funcionalidad que no existe en alguna extensión de la comunidad, deberemos desarrollarla, nunca modificar el código base del CMS (lo que se denomina en argot *hackear* el código).

5.5. Escalabilidad

La **escalabilidad** de un sistema informático se define como la "característica de mantener la capacidad del servicio a medida que la carga sobre el sistema aumenta".

La **carga sobre el sistema** se mide en número de visitas simultáneas. Las visitas se pueden medir de varias maneras:

- Número de peticiones (*hits*). Es decir, el número de archivos que el servidor ha tenido que proporcionar (incluyendo HTML, CSS, imágenes, etc.).
- Número de páginas vistas.
- Número de visitas únicas. Este es uno de los parámetros más utilizados para medir la audiencia de un sitio web.

Todos estos parámetros se miden por unidad de tiempo: por ejemplo, número de visitas por día o páginas vistas por mes. La escalabilidad, como decíamos, es la capacidad que tiene el sistema para asumir un número de visitantes, es decir, de tráfico y volumen de datos, cada vez mayor sin sufrir problemas si el sistema está bien diseñado, tendrá mecanismos que aseguren su funcionamiento a medida que el número de visitantes vaya subiendo.

Un buen CMS debe permitir adaptarse al crecimiento de visitas sin ofrecer problemas, aunque en realidad, en gran parte, la escabilidad se debe implementar en el servidor (aumentando el número de servidores dedicados, replicando la base de datos, etc.).

El CMS debe aportar mecanismos que ayuden a la escalabilidad, como la ***caché*** o el ***throttling***.

- ***Caché***. Algunos CMS almacenan las páginas en la base de datos en lugar de generarlas cada vez mediante PHP y acceso a la base de datos. De esta manera puede incrementarse enormemente el rendimiento y asumir un número de visitas mucho más grande. No es lo mismo tener que hacer 20 peticiones a la base de datos que solo una (recuperar la página ya construida de la base de datos o de la memoria).

- ***Throttle***. Es la capacidad de algunos CMS de desactivar algunas funcionalidades (no básicas) en el momento en que se detecta un aumento importante de peticiones al sitio web, y de esta manera garantizar que el resto del sistema continúa funcionando.

La escalabilidad es una especialidad de la gestión de servidores compleja en sí misma, que debe ser llevada a cabo por especialistas. A medida que las visitas aumentan considerablemente (en cifras de varios miles de visitantes al día), hay que replicar servidores, bases de datos, optimizar la caché, etc. Estos pasos serán independientes del uso de un CMS u otro si están bien implementados.

Una garantía de que nuestra elección permite una buena escalabilidad es observar si existen sitios con gran número de visitas que usen nuestro CMS. Es el caso de WordPress, Drupal, openCms y otros de los más utilizados.

5.6. Estándares

Al igual que cualquier otro sitio web, una aplicación desarrollada con un CMS debería cumplir los **estándares** actuales respecto HTML y CSS. Cumplir estos estándares proporciona diversas ventajas:

- El código HTML podrá ser más ordenado y semántico.

- Mejoraremos el mantenimiento futuro. Será más fácilmente actualizable y probablemente funcionará mejor en navegadores que todavía no han aparecido.

- Facilitamos que las páginas se puedan ver bien en cualquier navegador.

- Es una condición necesaria para hacer el contenido accesible.

- Mejora el posicionamiento en buscadores.

La manera de saber si un sitio web cumple con los estándares web es pasar por un **proceso de validación**.

Observación

El W3C ofrece un servicio de validación para HTML en: http://validator.w3.org/ y de CSS en http://jigsaw.w3.org/css-validator/. Para comprobar que nuestro CMS valida correctamente, solo hace falta verificarlo con estas herramientas.

Debido a que la presentación y el contenido están separados, la validación depende en buena medida de la capa de presentación, es decir, del tema activo en ese momento. Es responsabilidad del creador del tema que el código HTML/CSS sea válido.

Para garantizar que nuestro proyecto cumple los estándares web debemos asegurarnos de los siguientes puntos:

- El código HTML/CSS de la plantilla que hemos escogido o desarrollado es estándar.

- El editor no permite introducir código HTML no válido (lo que rompería la validación).

- Formación en estándares web del personal que se encarga de la actualización. Tema muy importante ya que, de lo contrario, en cualquier momento el sitio web podría dejar de ser válido.

5.7. Sindicación de contenidos (RSS y Atom)

Además de la manera tradicional de ver contenidos en Internet, esto es, visitando páginas web o utilizando un buscador para encontrar contenidos relevantes, existen sistemas mediante los cuales nos podemos suscribir a las fuentes de información. De esta manera, la información viene hacia nosotros en lugar de tener que dedicar tiempo explorando y navegando para saber si existen nuevos contenidos en las páginas que visitamos habitualmente.

Así, al suscribirnos a una fuente de información mediante un **lector de fuentes de información** (también llamados lector o agregador de *feeds* o RSS), la aplicación nos avisa cuando existe algún contenido nuevo que todavía no hemos visitado, pudiendo incluso leer el contenido de esa fuente directamente en la aplicación sin tener que visitar el sitio original.

Algunos de estos lectores de *feeds* funcionan en línea, como Google Reader.

Para poder suscribirnos a estas fuentes de información necesitamos que estas publiquen (**sindiquen**), además de las páginas convencionales HTML, unos archivos en un formato estándar. Estos formatos estándar son los denominados **RSS** y **Atom**. Se trata de archivos de texto en formato XML con etiquetas especiales específicas, que contienen todo el contenido más reciente que se ha publicado en nuestro gestor de contenidos y que sirven para delimitar el contenido de todos los artículos del sitio. Tanto los formatos RSS 1.0 y 2.0 como Atom cumplen la misma función.

Dado que la **sindicación de contenidos** es algo necesario hoy en día, debería de ser una de las características obligatorias en cualquier CMS.

Si un CMS está bien diseñado, al publicar contenidos en el panel de administración estos deberían ser publicados automáticamente también mediante RSS sin tener que preocuparnos de ninguna acción específica para ello.

La sindicación RSS no solo es útil en las plataformas de blogs (como WordPress) sino que también lo es en cualquier sitio web que tenga contenidos que se van actualizando periódicamente.

El hecho de publicar automáticamente en RSS o Atom tiene múltiples ventajas:

- Los usuarios se pueden suscribir a los contenidos usando un lector de *feeds*, herramientas que, como acabamos de decir, permiten visualizar automáticamente el contenido nuevo que se va generando en las fuentes de información a las cuales estás suscrito.

- Mejora la indexación en los buscadores (especialmente en los buscadores específicos de blogs como Google Blog Search).

- Permite que la gente pueda agregar la información de nuestro blog en otros sitios web. Por ejemplo, creando una página personalizada en servicios como Netvibes o similares, o en su propio blog o página web.

Por supuesto que la sindicación de contenidos debería ser opcional y configurable, pudiendo decidir, por ejemplo, sobre qué tipos de contenidos se realiza y sobre cuáles no, o incluso qué campos del contenido se publican (fotografías u otros datos). Lo habitual es que tengamos la opción de sindicar el contenido en formato resumido (obligando al lector a visitar la fuente original) o publicar todo el texto en el RSS.

5.8. Accesibilidad

Otra de las buenas características que debería tener cualquier CMS es un grado alto de accesibilidad. Para que nuestro sitio web pueda ser usado por cualquier persona en cualquier momento, existen **mecanismos reconocidos y estandarizados internacionalmente que permiten que el contenido sea accesible** mediante lectores especializados.

De la misma manera que en la validación HTML/CSS la accesibilidad depende en gran medida de la implementación de un tema en concreto que estemos usando en nuestro CMS. Para que un tema sea accesible es un requisito necesario, aunque no suficiente, que valide correctamente.

También es importante que todos los contenidos estén bien etiquetados para que puedan ser entendidos por lectores de pantalla u otros dispositivos. Un documento web bien marcado semánticamente resultará siempre más fácil de navegar y de interpretar por cualquier dispositivo.

En general, una buena manera de pensar en la accesibilidad es no dar por garantizado que los usuarios navegarán usando un ratón, ni que podrán ver las imágenes. Siempre es buena idea ofrecer contenidos alternativos si estamos usando tecnologías como Flash, etc.

Toda la información sobre los criterios estándar de accesibilidad se pueden encontrar en la página del W3C; encontraremos las recomendaciones WAI (*web accessibility initiative*), aunque estas directrices son de 1999 y han quedado un poco obsoletas. Las WCAG 1.0. se basan en 14 directrices que permiten dar una clasificación de conformidad (A, si se cumplen las de prioridad 1; AA, las de nivel 2, y AAA, la clasificación más elevada). Las WCAG 2.0 (http://www.w3.org/TR/WCAG20/), más evolucionadas, no dependen tanto de la tecnología, por lo que es difícil encontrar un sistema automático de comprobarlo.

Uno de los componentes que pueden causar problemas de accesibilidad en nuestro CMS son las extensiones o módulos “contribuidos” (es decir, desarrollados por gente externa al equipo que mantiene el núcleo del CMS) si no están desarrollados teniendo en cuenta la accesibilidad. La única solución, además de leer bien la documentación, es probarlos y ver si cumplen los estándares.

Desgraciadamente, no existen métodos automáticos para comprobar íntegramente que nuestro sitio web es accesible, aunque se pueden hacer tests parciales mediante programas y el resto se debe comprobar manualmente haciendo pruebas con usuarios reales.

Al igual que en el caso de los estándares web, solo podremos garantizar la accesibilidad mediante tres factores:

- El CMS y el tema que estemos utilizando deberá ser accesible dentro de lo posible.

- El editor debe permitir introducir contenido sin perjudicar la accesibilidad de este.

- Para asegurarnos de que el sitio continúa siendo accesible en el tiempo, habrá que prever un plan de formación sobre accesibilidad del personal que mantiene el sitio web.

Página web

Existen algunas herramientas de comprobación automática de la accesibilidad:

http://www.rnib.org.uk/professionals/webaccessibility/testingtips/Pages/automated_testing.aspx.

Plone, uno de los CMS que veremos más adelante, es uno de los que mejor cumple los criterios de accesibilidad, aunque el resto también suele tener en cuenta estos criterios y a menudo se pueden completar con la instalación de módulos adicionales.

5.9. Estabilidad

En informática se suele definir como un **sistema estable** aquel que es tolerante a fallos o errores, es decir, que no generará ni producirá errores si se hace un uso correcto del mismo.

Para asegurarnos de que nuestro CMS es estable debemos fijarnos en los siguientes puntos:

- ¿El CMS que hemos elegido tiene una base de usuarios y desarrolladores suficientemente grande? En caso contrario es difícil que se haya testeado todavía a fondo en diversas situaciones.

- ¿Estamos usando una versión etiquetada como *estable*? No una versión beta o alfa que todavía no está comprobada totalmente.

- ¿El servidor que estamos usando cumple los requisitos mínimos de instalación? Si, por ejemplo, la versión del lenguaje de servidor que estemos usando no cumple con los requisitos, es posible que el sistema funcione pero con errores.

5.10. Facilidad de instalación y mantenimiento

Un factor esencial para que un CMS sea ampliamente utilizado por la comunidad es la facilidad de instalación. Es por ello por lo que muchos CMS son extremadamente sencillos de instalar en un servidor (siempre que sepamos los datos esenciales). Además, estas facilidades se deben extender también al mantenimiento posterior. Las **actualizaciones** del programa o **extensiones** asociadas deben ser fáciles de realizar. Cualquier CMS va actualizando su código a lo largo del tiempo (por ejemplo, para solucionar errores o agujeros de

seguridad), por lo que es necesario también ofrecer algún sistema de actualización que idealmente debería ser lo más automático posible. Lo mismo se aplica a las posibles extensiones que tenga el sistema.

Uno de los mejores ejemplos en este sentido es WordPress, donde la instalación de extensiones y la actualización del sistema se hacen directamente desde el panel de administrador sin tener que acceder en ningún momento al sistema de archivos o a buscar por Internet las actualizaciones, descargar o descomprimir archivos, etc. En este CMS se ha hecho un gran esfuerzo por facilitar al máximo el trabajo del administrador del sitio, cosa que ha repercutido sin duda en el hecho de que sea una de las plataformas más utilizadas en la actualidad.

Los creadores de CMS saben que un sistema complejo de instalar es una barrera de entrada a usuarios con pocos conocimientos informáticos y, además, puede ser una fuente de errores y problemas.

Idealmente, un CMS debería poderse instalar sin tener que editar a mano ningún archivo de configuración, aunque siempre es importante tener acceso a todas las opciones de configuración y, en caso de desearlo, poder editarlas a mano.

5.11. Facilidad de adaptación y personalización

La adaptación y personalización de los CMS se puede hacer, casi siempre, en dos de sus aspectos básicos:

- **Funcionalidades**. A través de la incorporación de **extensiones**, que permiten modificarlas o aumentarlas

- **Aspecto**. Mediante **temas** que permiten la personalización de la presentación

Un buen CMS no solo debe tener la posibilidad de añadir extensiones y temas, sino que también debe disponer de recursos para que cualquier desarrollador pueda a su vez crearlos, idealmente dejándolos después a disposición de la comunidad. Además, un buen CMS debería disponer de **API** de programación que permitan el acceso a las funciones principales del CMS para poder modificarlas si es preciso. Se trata de poder sacar provecho a las funcionalidades del CMS y al mismo de tiempo poder crear funcionalidades nuevas no previstas.

Este es el caso del Codex (http://codex.wordpress.org/) de Wordpress o de la documentación de la API de Drupal. En estas páginas están documentadas todas las funciones que usan estos dos sistemas.

Cuando se realiza una personalización, es muy importante siempre modificar solo los archivos del módulo o tema en concreto, nunca modificar los archivos base del CMS, ya que si se realiza una actualización se perderían los cambios y además podríamos afectar a la seguridad e integridad del sistema.

5.12. Interconectividad

La **interconectividad** se define como la capacidad de un CMS de conectarse a otros sistemas de la organización (por ejemplo, a bases de datos existentes). Es una característica muy importante en sistemas empresariales (igual que en otros sistemas informáticos) ya que permite:

- Poder compartir recursos.
- Tener acceso instantáneo a bases de datos compartidas.
- Administración centralizada de la red.
- Reducción de presupuestos (tiempo, dinero).
- Optimización de recursos humanos.

Un caso típico sería **LDAP**, un sistema estándar de autentificación de recursos usado por muchas organizaciones. Integrar LDAP en nuestro CMS permite que los usuarios se autentifiquen en LDAP, sin necesidad de crear cuentas nuevas y validando los mismos grupos que puedan existir en la organización.

Otro caso sería la exportación o importación de datos desde bases de datos de la organización al CMS o a la inversa.

> Por ejemplo, imaginemos una empresa que ya dispone de un *stock* de productos con referencias y precios en una base de datos interna; si esta empresa quiere abrir una tienda en Internet, lo lógico sería que la página web utilizase esta misma información en lugar que tener que duplicarla.

Las aplicaciones web que requieren interconectividad son complejas, ya que no dependemos solo del CMS, sino también de sistemas externos que pueden ser muy variados, con tecnologías diferentes y formatos que no siempre son fáciles de integrar en nuestro CMS. Muchas veces será necesario programación a medida para poder conseguir el objetivo.

5.13. Posibles aplicaciones integradas en un CMS

En los CMS más generalistas es importante ver qué **aplicaciones** (**funcionalidades**) se pueden implementar de base o añadiendo módulos.

Algunas de las aplicaciones que pueden implementar estos CMS son:

- Blogs.
- Chats.
- Calendarios de eventos.
- FAQ (*frequently asked questions*).
- *Newsletters*. Sistema automático para enviar correos electrónicos a subscriptores.
- Galerías de fotos.
- Buscador. Con indexación de todo el contenido del sitio.
- *Sitemap*. Mapas del sitio en formato HTML o XML.

- RSS. Sindicación de contenidos.
- *Wiki*. Páginas editables con historial.
- *E-commerce*. Comercio electrónico.

6. Otras características de los CMS

El objetivo principal de un gestor de contenidos es crear páginas estándar en HTML y CSS a partir de los contenidos de la base de datos.

La versión más utilizada de HTML actualmente es la 4.01 (publicada en 1999) y la de XHTML es la 1.0 (publicada en el 2000). En cuanto al CSS, la versión más utilizada es la 2.1 (2005). Sin embargo, estos estándares están en evolución y es importante tenerlo en cuenta en el gestor de contenidos. Además, existen otras tecnologías que acompañan a las páginas HTML:

- **HTML5/CSS3**. Los navegadores modernos ya aceptan los nuevos formatos HTML5 y CSS3. Para poder obtener un buen aprovechamiento de ellos, nuestro CMS debe ser capaz de adaptarse.

- **JavaScript**. La gran mayoría de sitios web hoy en día tienen algún tipo de programación en JavaScript, desde validación de formularios hasta refrescar dinámicamente la información de la página (AJAX). En HTML5 el papel de JavaScript se ve reforzado con más funcionalidades. A menudo se utiliza JavaScript a través de la libreria jQuery.

- **Marcado semántico**. Existen diversos formatos aceptados para la marcación semántica del contenido en páginas web. En el futuro esto cobrará cada vez más importancia. Esto se puede realizar a través de **RDF** o **microformatos**.

- **Optimización para buscadores**. Es un factor muy importante; no se trata solo de presentar adecuadamente el contenido en formatos estándar, sino también que estos contenidos sean fácilmente indexados por los buscadores, de manera que podamos atraer visitantes que están interesados en la información que publicamos.

A continuación vamos a analizar cada una de estas características y cómo debemos tenerlas en cuenta en nuestro CMS.

6.1. HTML5/CSS3

Las especificaciones de HTML y CSS siguen avanzando. El organismo internacional encargado de estas especificaciones, el W3C, sigue publicando nuevas versiones, las últimas de las cuales se llaman HTML5 y CSS3. Lo previsible es que a corto plazo estas versiones sean las habituales en la mayoría de los navegadores web, aunque en este momento todavía no lo son del todo.

Dado que estos estándares son los que aplican a la presentación de una página web, en el gestor de contenidos son competencia de las **plantillas** o **temas**. Cualquier CMS puede ofrecer contenidos en HTML5/CSS3 siempre que exista una plantilla para ello o la creemos nosotros mismos a medida. El problema, como siempre, será mantener la compatibilidad con las versiones anteriores de los navegadores que todavía habrá que tener en cuenta durante unos años.

6.2. JavaScript

JavaScript es un **lenguaje de programación interpretado que se puede incrustar en el código HTML**. Todos los navegadores web tienen un intérprete de JavaScript (a no ser que esté desactivado). JavaScript se está usando cada vez más, sobre todo como alternativa a Flash u otras opciones, y especialmente en los nuevos formatos web como HTML5. Incluso existe JavaScript en versión servidor (interpretado en el servidor en lugar del cliente).

Un buen gestor de contenidos debería contemplar como opción la posibilidad de incrustar código en JavaScript. Aunque no sea directamente, hay multitud de ***plugins*** o **módulos** en los gestores de contenidos que usan JavaScript:

- En los editores de texto incrustados en la página.
- En visualizadores de fotos tipo Lightbox.
- En cualquier efecto interactivo que no esté usando Flash.
- Una de las posibilidades más interesantes de JavaScript es solicitar información al servidor sin tener que recargar la página de nuevo (**AJAX**). Esto permite mejorar el flujo de envío de información por parte de los usuarios de la web.

Existen muchas **librerías** de JavaScript que sirven para multitud de tareas, pero la librería más importante sin duda es **jQuery**.

6.2.1. jQuery

jQuery es una **librería de JavaScript** que permite simplificar mucho el desarrollo en este lenguaje. Actualmente, un porcentaje importante de páginas web y de gestores de contenidos usan jQuery (entre ellos, por ejemplo, Wordpress y Drupal). Empresas importantes, como Microsoft o Nokia, también la integran en sus productos.

Básicamente, jQuery permite:

- Abstraer el comportamiento de los diferentes navegadores. Uno de los principales problemas es que el JavaScript de un navegador puede ser diferente que el de otro, o variar entre las implementaciones de un navegador para diferentes sistemas operativos. jQuery intenta separar estas diferencias debajo de una capa común.

- Simplifica enormemente las instrucciones que podemos usar para lograr funcionalidades o efectos en nuestro sitio web.

- Es ideal para desarrollar aplicaciones usando AJAX/JSON.

- Tiene *plugins* (normalmente desarrollados por la comunidad) que permiten ampliar las funcionalidades.

jQuery simplifica la manera de interactuar con los documentos HTML, manipular el árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción con AJAX. La primera versión es del 2006 y desde entonces se ha establecido prácticamente como uno de los principales estándares para programar JavaScript. jQuery es software libre y de código abierto, con doble licenciamiento bajo las licencias MIT y GNU General Public License, Versión 2.

jQuery, al igual que otras bibliotecas, ofrece una serie de funcionalidades basadas en JavaScript que de otra manera requerirían de mucho más código. Es decir, con las funciones propias de esta biblioteca se logran grandes resultados en menos tiempo y espacio.

Algunas **funcionalidades básicas** de jQuery son:

- Selección de elementos DOM.

  Por ejemplo,

```
$('p')
```

  selecciona todos los elementos p del documento.

- Interactividad y modificaciones del árbol DOM, incluyendo soporte para CSS 1-3 y un *plugin* básico de Xpath.

```
//Añade un elemento p después de todos los h2
$('h2').append("<p class='more'>More</p>");
```

- Eventos.

```
 $('#info').click=function( console.log("elemento clicado");
```

- Manipulación dinámica de la hoja de estilos CSS.

```
// eliminar el estilo "activo" con removeClass()
y aplicar uno nuevo con addClass()
$(".activo").removeClass("activo").addClass("inactivo");
```

- Efectos y animaciones.

```
//esconde el elemento con id="aviso" lentamente
$("#aviso").hide("slow");
```

- AJAX.

```
//ejemplo de llamada a un archivo JSON
$.getJSON('ajax/test.json', function(data) {
```

- Utilidades varias, como obtener información del navegador, operar con objetos y *arrays*, función trim (elimina los espacios en blanco del principio y final de una cadena de caracteres), etc.

Además, las funcionalidades de jQuery se pueden ampliar mediante *plugins*.

Página web

Listado de *plugins* de Jquery: http://plugins.jquery.com/project/Plugins.

6.2.2. Ajax/JSON

AJAX (*Asynchronous JavaScript And XML*, JavaScript asíncrono y XML) es una **técnica de programación que permite refrescar parte del contenido de la página actual sin tener que recargarla**. Esto se consigue a través de una petición asíncrona de JavaScript al servidor, y una vez obtenidos los datos también se usa JavaScript para actualizar el contenido de la página. En algunos casos, esto mejora mucho la usabilidad de las aplicaciones, permitiendo agilizar tareas y dando *feedback* a los usuarios de manera mucho más rápida.

JSON es un formato de archivo que almacena contenidos en un árbol, de manera similar a XML pero formateado como un *array* JavaScript, de manera que se puede leer más fácilmente. Este es el motivo por el cual la combinación AJAX/JSON está siendo muy utilizada.

Un caso típico del uso de ambas tecnologías es Google Maps, la aplicación de mapas de Google. Por otro lado, gran parte de las API y servicios de Internet (como Facebook, Twitter, etc.) utilizan JSON como mecanismo de respuesta a las llamadas.

Esta tecnología también es muy utilizada en CMS; por ejemplo, en muchos casos en el panel de administración cuando tenemos la opción de hacer *drag & drop* (arrastrar y soltar) de elementos de manera visual.

Observación

A menudo la programación con la técnica AJAX y JSON se hace a través de la librería jQuery.

6.3. Soporte para plataformas móviles

Nuestro sitio web se debe ver correctamente en diversos tipos de dispositivos. La variable que afecta más en este sentido es el tamaño de la pantalla, aunque también hay otras como la resolución o el hecho de que la pantalla sea táctil.

Teniendo en cuenta estas variables podemos considerar que hay tres tipos principales de dispositivos:

- Ordenadores de escritorio.
- Tabletas (iPad, Android, Windows 8, etc.).
- Móviles (*smartphones*).

La responsabilidad de que el sitio web se vea correctamente en todos estos dispositivos recae sobre todo en el tema. Existen varias técnicas (que pueden funcionar simultáneamente) que debe aplicar el tema para abordar el problema:

- Aplicar técnicas de diseño *responsive*. Mediante *media queries*, en el CSS podemos variar la composición y las propiedades de los elementos según el tamaño de la pantalla actual.

- Detectar el dispositivo que está accediendo al sitio y activar un tema especialmente diseñado para cada caso (por ejemplo, un tema para móviles y otro para tabletas y ordenadores de escritorio).

- Aplicar técnicas de detección del tipo de dispositivo en el servidor de manera que las plantillas del tema estén adaptadas a los diferentes tamaños de pantalla. Esta técnica tiene la ventaja respecto la opción *responsive* de poder variar realmente el tamaño de las imágenes o de no servir parte de los contenidos si lo consideramos necesario en algún caso (cosa no posible en *responsive*, porque es una técnica que se aplica en el cliente). Esto es posible gracias al *user-agent* que envía el navegador en las peticiones. Existen bases de datos y librerías que nos permiten detectar el tipo de dispositivo a partir de este *user-agent*, como wurfl (http://wurfl.sourceforge.net/), y ex-

tensiones y módulos en los CMS que nos permiten modificar las plantillas del tema en función del tipo de dispositivo que está accediendo.

Todas las alternativas tienen sus propias limitaciones (si elegimos servir varios temas, también elegimos mantener una diversidad de temas cada vez que modifiquemos algún aspecto, por ejemplo). Es nuestra responsabilidad evaluar las diferentes opciones, con sus puntos fuertes y débiles, y tomar una decisión informada.

6.4. Optimización para SEO

El **SEO** (*Search Engine Optimization*) o **posicionamiento en buscadores** tiene gran relevancia en los gestores de contenido. Normalmente, uno de los principales objetivos es hacer llegar la información generada con estos CMS a los usuarios. Hoy en día esto se consigue en gran medida apareciendo en las páginas de resultados para determinadas búsquedas.

El posicionamiento en buscadores es una materia amplia y este apartado no es más que un breve resumen. Pero ¿cómo podemos optimizar, a grandes rasgos, el posicionamiento en buscadores web con un gestor de contenidos? Algunas de las condiciones necesarias son:

- Todos los contenidos deben tener una página única con un título descriptivo. Así, un portal de noticias con 1.000 noticias debe tener al menos 1.000 páginas únicas con título único.

- Emplear código estándar, válido según el W3C. De esta manera, aseguramos al menos en parte que el contenido podrá ser extraído fácilmente por los motores de búsqueda.

- Es necesario que las plantillas del CMS validen y que las etiquetas CSS sean semánticas. El volumen de contenidos no debería afectar a esta validación, ya que la presentación debe estar siempre separada del contenido, y las plantillas deberían ser siempre las mismas.

- Usar adecuadamente los encabezamientos (h1, h2, h3...).

- Añadir siempre la etiqueta “alt” a las imágenes. Especialmente importante para la búsqueda de imágenes.

- Crear un mapa del sitio. Muchos gestores de contenidos permiten crear un mapa del sitio (de base o mediante extensiones).

- La existencia de un mapa de navegación facilita que los buscadores puedan rastrear el contenido.

- URL con nombres "amigables" y semánticos. Todos los contenidos deberían tener una página única con un título que los defina y, a ser posible, con una URL que también los describa mínimamente.

  Por ejemplo, es mucho mejor "/zapato-camper-rojo" como URL y "Zapato Camper Rojo" como título de la página, que "index.php?=456" como URL y "Zapatería Muñoz" como título.

Algunas características no tan importantes pero también a tener en cuenta son:

- Limitar el número de subdirectorios. Los buscadores darán más importancia a los contenidos que están a un nivel superior.

- Evitar enlaces inexistentes. Si los usuarios generan contenidos, hay que asegurarse de que las URL que pueda haber existan realmente; en caso contrario, puede ser motivo de penalización de posicionamiento por los buscadores.

- Evitar URL duplicadas.

- Textos descriptivos en enlaces, en lugar de "clicar aquí", "informe contable 2011".

6.5. Microformatos y RDF

6.5.1. Microformatos

Una de las cosas que parecen claras es que en el futuro la web será cada vez más **semántica**, o lo que a veces se denomina **Web 3.0**. En este sentido, existen varias propuestas para añadir el componente semántico al contenido html.

Una de ellas son los **microformatos**, una propuesta para estructurar la información y dar nombres usando clases predefinidas, manteniendo el mismo html, que **sirven para identificar y dar valor semántico al contenido de la página html**.

Imaginemos un ejemplo. En la ficha de un equipamiento municipal queremos almacenar la posición geográfica. Utilizando microformatos podríamos hacerlo de la siguiente manera:

```
<span class='geo'>
   <span class='latitude'>49.205486</span>
   <span class='longitude'>-122.892036</span>
</span>
```

Página web

Para saber más sobre microformatos:
http://microformats.org.

Esta información no necesariamente ha de ser visible por el usuario de la web, ya que está pensada para ser interpretada por los buscadores o para ser utilizada por extensiones del navegador o por otros servicios web para localizar en un mapa el equipamiento, almacenarlo en una agenda personal, etc.

6.5.2. RDF

Una propuesta mucho más avanzada sobre la capa semántica es **RDF** (*resource description framework*).

Páginas web

Para saber más sobre RDF:
http://en.wikipedia.org/wiki/Resource_Description_ Framework.
http://www.w3.org/RDF/.

En este caso se trata de añadir la **información semántica** siguiendo un estándar definido por el W3C, que incluye, entre otras cosas, la definición de etiquetas especiales en el HTML.

Es deseable que a medida que estas posibilidades semánticas se vayan materializando, los CMS hagan uso de ellas. Es el caso por ejemplo de Drupal 7, que ya implementa RDF de origen.

7. Tipos de CMS

A continuación vamos a ver un listado de CMS especializados en implementar aplicaciones de diferentes tipos. Empezaremos definiendo estos tipos de aplicaciones y seguidamente veremos algunos CMS pensados para desarrollar proyectos de esa tipología. Para cada uno de los CMS vamos a ver sus funcionalidades principales, junto con una lista de ventajas y desventajas.

Los tipos de aplicaciones que hemos definido son los siguientes:

- **Genéricos**.
- **Foros**.
- **Blogs**.
- ***Wikis***.
- ***E-learning***.
- **Redes sociales**.
- **Comercio electrónico** (*e-commerce*).

A veces los límites entre un tipo u otro de aplicación son un poco difusos. Además, hay que tener en cuenta que cuanto más flexible sea un CMS, más se podrá adaptar a diferentes tipos de aplicaciones. En cambio, existen otros CMS tan especializados (como por ejemplo algunos pensados para foros) que realmente resulta difícil aplicarlos en otros campos.

Dado que un CMS permite bastante flexibilidad en el diseño y la composición, es casi imposible a primera vista descubrir sobre qué CMS se ha construido un sitio. Sin embargo, observando el código fuente podemos obtener pistas que nos permitan deducir el sistema utilizado.

> Por ejemplo, en el caso de WordPress rutas del tipo "/wp-content/plugins" o "/wp-content/themes", o, en el caso de Drupal, "sites/all/themes".

También podemos utilizar extensiones de Firefox o Chrome para este propósito (como por ejemplo Wappalyzer).

7.1. Genéricos

Una **aplicación genérica** es aquella que no tiene una sola funcionalidad específica sino que pretende cubrir diversas funcionalidades.

Aparte de poder gestionar contenidos dinámicos y temas, debe cumplir al menos alguno de los siguientes requisitos:

- Gestión de usuarios, roles y permisos.

- Gestión de la estructura de contenidos (tipos de contenidos, campos, taxonomías).

- Posibilidad de añadir módulos para cubrir aplicaciones más específicas (como tiendas *on-line*, etc.).

Es decir, los CMS de este tipo están pensados para adaptarse a cualquier funcionalidad. Normalmente, integran varias de estas funcionalidades y la posibilidad de crear estructuras de contenidos más o menos complejas, por lo que las funcionalidades que no se pueden implementar mediante las opciones básicas, se satisfacen normalmente instalando módulos adicionales.

La desventaja que tienen es que al ser más completos, el panel de administración y la configuración también suele ser más complicada.

Casos típicos de este tipo de CMS son OpenCms, Joomla, Plone o Drupal. A continuación vamos a ver más en detalle alguno de ellos.

7.1.1. OpenCms

OpenCms (http://www.opencms.org/en/) es un gestor de contenidos de código abierto desarrollado por la compañía Alkacon Software que está implementado sobre la plataforma Java (J2EE) / XML. Java es un lenguaje semiinterpretado orientado a objetos de Sun Microsystems (hoy dia, parte de Oracle).

Como en la mayoría de CMS, la administración es vía web, y dispone además de un editor de páginas WYSIWYG. Usa la metáfora del "explorador de archivos", de manera que todas las páginas del sitio se pueden ver desde el administrador como archivos editables. Cada uno de ellos tiene campos dinámicos que se pueden modificar.

Las **funcionalidades básicas** son:

- Panel de administración basado en un navegador web.

- Gestión de activos, por ejemplo, imágenes, documentos PDF, enlaces externos, etc.

- Editor de texto WYSIWYG.

- Control de versiones de contenido con *roll-back* (posibilidad de recuperar versiones anteriores).

- Vista previa de páginas.

- Apoyo a los contenidos estructurados y no estructurados.
- Gestión integrada de usuario y permisos del sistema.
- Plantillas basadas en JSP (*Java server pages*).
- API de Java.
- Apoyo a la internacionalización.
- Publicación de contenidos en modo estático y dinámico.
- Posibilidad de añadir módulos.
- Función de búsqueda.

Las principales **ventajas** son:

- Pensado para funcionar sobre muchos tipos de servidores web y de bases de datos diferentes.
- Posibilidad de tener páginas estáticas y dinámicas.
- API en Java que permite personalización de la aplicación.
- Uso de JSP para incluir elementos dinámicos en la página.
- Extensión mediante módulos opcionales que pueden ser reutilizados.
- Facilidad de instalación (solo los datos imprescindibles de ubicación, base de datos y contraseña).

Y las **desventajas**:

- Es un poco complejo.
- Falta de una buena documentación y de una buena comunidad de desarrolladores. El hecho de que esté desarrollado por una empresa y no una comunidad es la causa de este problema.
- Al escribir una plantilla de JSP, los desarrolladores deben utilizar el editor de textos básicos que se encuentran dentro de OpenCms. A diferencia de las herramientas convencionales IDE como Eclipse, el editor nativo de OpenCms no ofrece ninguna comprobación de sintaxis ni ayudas sobre la API. Tampoco hay depurador integrado para ayudar en la solución de problemas complejos.

Página web

Listado de módulos y extensiones para OpenCms:
http://www.opencms-wiki.org/wiki/Available_Modules.

- OpenCms solo proporciona una funcionalidad limitada de flujo de trabajo en torno a contenido.

- Como en la mayoría de CMS, las funcionalidades se pueden ampliar mediante la instalación de módulos o extensiones; sin embargo, su número no es tan elevado como en otros CMS.

Algunos ejemplos de sitios web que usan este CMS:

UOC. Universitat Oberta de Catalunya (http://www.uoc.edu/).

Universidad Pompeu Fabra (http://www.upf.edu/).

7.1.2. Joomla

Joomla (http://www.joomla.org/) es un CMS de código abierto con licencia GPL (*GNU general public license*), basado en PHP/MySQL. Se trata de un CMS bastante flexible y adaptable a diferentes necesidades.

Curiosidad

El nombre Joomla se deriva de la palabra swahili *Jumla,* que significa 'todos juntos'.

Joomla es una bifurcación (o *branch*) de otro CMS anterior, Mambo. Este tipo de bifurcaciones es bastante común en el campo de las aplicaciones de código abierto; sin ir más lejos, Firefox es una bifurcación del código original del navegador Netscape.

Las **funcionalidades** de la versión básica de Joomla (sin instalar extensiones adicionales) son:

- Cacheo de páginas.
- Pensado para optimizar la indexación en buscadores.
- Genera *feeds* RSS.
- Versiones para imprimir de todas las páginas.
- Generación de blogs.
- Forums.
- Encuestas.
- Calendarios.
- Buscador integrado.
- Internacionalización.

Además de estas funcionalidades que existen por defecto, hay cientos de **extensiones** y una gran comunidad de usuarios y desarrolladores. Las extensiones pueden ser de diversos tipos:

- Componentes.
- Módulos.
- Plantillas.
- *Plugins*.
- Lenguajes.

Además, Joomla dispone de una API de programación y se sostiene en un *framework* MVC. Para desarrollar componentes se recomienda usar esta filosofía MVC.

Ejemplos de sitios desarrollados sobre esta plataforma:

Museo Guggenheim (http://www.guggenheim.org/).

Oklahoma State University (http://osu.okstate.edu/welcome/).

7.1.3. Plone

Plone (http://www.plone.org/) es un sistema de gestión de contenidos web también bajo la licencia GPL. Su interfaz cumple los requerimientos de accesibilidad WCAG-AAA y U.S. Section 508.

Plone es un CMS basado en Zope y programado en Python.

Zope y Python

Zope (http://zope.org/)es un servidor de aplicaciones web de código abierto que tiene miles de desarrolladores en todo el mundo y que está escrito en el lenguaje de programación Python. Python, por su parte, es un lenguaje de programación interpretado creado en el año 1991. Se compara habitualmente con otros lenguajes como Perl, Java y Ruby. En la actualidad, Python se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation (http://www.python.org/psf/).

Es un desarrollo basado en código abierto. Plone puede utilizarse para construir diversos tipos de proyectos como portales, sitios webs corporativos, sitios de noticias, extranets o intranets, sistemas de publicación, repositorios de documentos, herramientas colaborativas (*groupware*), o incluso para *e-commerce*.

Otras de las posibles aplicaciones de Plone es como CRM (*customer relationship management*) o como sistema de gestión del conocimiento (KMS, del inglés *knowledge management system*).

El proyecto Plone comenzó en 1999. En el 2004 se creó la Fundación Plone para proteger y promover el uso de Plone.

Curiosidad

El estilo visual "Monobook" de la Wikipedia está basado parcialmente en el estilo de las páginas de Plone.

Las **funcionalidades básicas** son:

- Edición *inline*.
- Control de versiones.
- Revisión de la integridad de los enlaces.
- Editor HTML visual.
- Capacidades de *workflow*.
- Capa de autentificación.
- Indexación de documentos Word y PDF.
- Buscador basado en el protocolo *sitemap*.
- Soporte para wikis.
- Navegación adelante/atrás automática.
- Generación automática de tablas para el contenido.

- Soporte para contenidos multilingües.
- Publicación diferida.
- URL limpias.
- Editor gráfico.
- Compresión de recursos.
- Cacheo con integración de *proxy*.
- Reordenación de contenidos con *drag & drop*.
- Exportación de la configuración del sitio en XML.
- Formateo automático para impresión.
- XHTML y CSS estándar.
- Cumple criterios de accesibilidad.
- Exportación automática de *feeds* RSS.
- Escalado automático de imágenes y generación de *thumbnails*.
- Multiplataforma.
- Soporte de microformatos.
- Posibilidad de añadir comentarios en cualquier contenido.
- Soporte para FTP y WebDAV.
- Edición *in-context*.
- Soporte para *backups*.
- Operación de *cut/copy/paste* en los contenidos.

Como vemos, la lista de funcionalidades es muy extensa.

Las principales **ventajas** son:

- Basado en ZODB (una base de datos orientada a objetos, no relacional).
- Python suele ser más fácil de programar que otros lenguajes, como PHP.
- Usa un buen sistema de plantillas llamado TAL (*template attribute language*)
- Flexible.

Y las **desventajas**:

- La documentación no está muy actualizada.

- La importación de datos no está muy bien resuelta.

- Las tecnologías sobre las que se sustenta no son muy conocidas todavía por la comunidad de desarrolladores.

Ejemplos de sitios web que usan este CMS:

Lufthansa (http://www.lufthansa.com/).

Oxfam America (http://www.oxfamamerica.org/).

7.1.4. Drupal

Drupal (http://www.drupal.org) es un CMS de código abierto con licencia GPL basado en PHP/MySQL. La comunidad que está detrás de Drupal es muy amplia, lo que permite que exista mucha documentación, soporte para posibles problemas y un gran número de módulos adicionales

Las **funcionalidades básicas** son las que dan los módulos del núcleo (*core*), los que se instalan automáticamente sin necesidad de añadir módulos:

- Acceso a estadísticas.
- Búsqueda avanzada en el sitio.
- Publicar en modo blog, "Libro", fórums y encuestas.
- Gestión de bloques y regiones de contenido.
- Cacheo y *throttling*.
- URL limpias.
- Sistema de menús multinivel.
- Soporte para múltiples sitios con la misma instalación.
- Soporte para múltiples usuarios.
- Integración con OpenID.
- Publicación en RSS.
- Posibilidad de agregar *feeds*.
- Notificaciones de actualizaciones de seguridad.
- Perfiles de usuario.
- Roles de usuario.
- Herramientas de *workflow*.
- Multiidioma.

Las principales **ventajas** son:

- Plataforma altamente testeada.
- Una misma instalación para varios sitios.
- Se adapta bien a aplicaciones no solo de gestión de contenidos, sino también comunidades.
- Potente sistema de plantillas. Cualquier XHTML o CSS plantilla puede ser fácilmente convertido a Drupal.
- Fácil de usar con las opciones básicas.
- API.

Y las **desventajas**:

- Si no cumple perfectamente los requerimientos de nuestro proyecto, puede ser difícil la adaptación (probablemente requerirá la programación de módulos a medida).

- El panel de administración no es demasiado intuitivo.

- Es necesario buscar y probar en la lista de miles de módulos disponibles.

- Hay que tener cuidado con el gran número de módulos disponibles, ya que muchos de ellos no son estables.

Ejemplos de páginas que usan Drupal:

The White House (http://www.whitehouse.gov/).

Ubuntu (http://www.ubuntu.com/).

7.1.5. Google Sites

Google Sites (http://sites.google.com/) es un caso especial ya que, a diferencia del resto de CMS de los que hablamos, es un **servicio en línea** y **no es de código abierto**. Puede ser una buena solución para sitios no demasiado complejos.

Google Sites está incluido en el paquete "Google apps" que incluye soluciones para las empresas como Gmail, Calendar, Docs o Gtalk.

Google Sites permite crear un sitio web mediante un editor sencillo e incluso utilizar un dominio propio y la integración con otros servicios ofrecidos por Google. Google Sites se puede considerar un CMS, ya que cumple dos de los requisitos principales que debe cumplir este tipo de aplicaciones:

- Cualquier persona puede muy fácilmente crear una página web con contenidos actualizables.

- Se pueden usar temas intercambiables (presentación).

En un solo paso se puede crear un sitio en http://sites.google.com/.

Entre las opciones predeterminadas (plantillas) tenemos, por ejemplo, el sitio de los alumnos de una clase, una wiki para un proyecto o una página familiar.

Al no tener acceso al código fuente del programa, no podemos modificarlo ni integrarlo con otros sistemas. Es una solución limitada que, sin embargo, puede ser útil, por su sencillez, en casos puntuales. Su funcionamiento está basado en la creación de páginas y su edición posterior.

Las **funcionalidades básicas** son:

- Exportación de *feeds* RSS.
- Suscripción por e-mail a los cambios de la página.
- Integrado con todos los servicios de Google, incluytendo AdSense, Google Docs, Calendar, Maps, Picasa, Analytics.
- Control de revisiones de los contenidos.
- Diferentes *layouts* posibles.
- Posibilidad de añadir *gadgets*.
- No se muestran anuncios (a diferencia de otros servicios de Google).

Las principales **ventajas** son:

- Es una solución muy sencilla; el panel de administración, por ejemplo, no tiene ninguna complejidad.
- Podemos elegir entre un gran número de plantillas que, en muchos casos, están implementadas por otras personas.
- Si implementas un proyecto usando Google Sites puedes "liberar" el proyecto para que otras personas lo puedan reutilizar, pasando desde este momento a ser otra plantilla de las disponibles. Este componente social hace posible que existen gran cantidad de plantillas disponibles.

Y las **desventajas**:

- Falta de privacidad y, sobre todo, que los datos están almacenados en los servidores de Google y no en nuestra organización.
- Está limitado a 100Mb de almacenamiento en la versión gratuita.
- No se puede añadir CSS en los temas y el HTML está limitado.
- No se pueden añadir comentarios anónimos.

Ejemplo:

T and M Photography (http://www.tandmphoto.co.uk/).

7.2. Foros

Un **foro** es un sitio de Internet donde la gente intercambia mensajes en forma de discusión, haciendo preguntas y respondiéndolas.

Estas conversaciones suelen agruparse por temas. Es una de las formas de creación dinámica de contenidos que surgió más tempranamente en la red.

Los foros son los descendientes de sistemas más antiguos como los BBS (*bulletin board system*) de la década de los ochenta y noventa y, en cierta manera, son los precursores de las redes sociales actuales.

Las personas que participan en un foro de Internet se agrupan en grupos de interés a partir de debates. Cada usuario puede dejar su aportación en modo de respuesta a un mensaje de otro usuario, o como nueva aportación, pero nunca editando las entradas de otros usuarios (tal como sucede en una wiki).

Un foro está controlado por uno o varios moderadores (*mods*), que son los encargados de que los usuarios sigan las reglas del sitio. Las aportaciones de los usuarios se agrupan en temas (*topics* o *threads*).

7.2.1. phpBB

phpBB (*PHP bulletin board*) (http://www.phpbb.com/) es uno de los CMS para foros más usados. La primera versión surgió en el año 2000 y desde entonces se ha ido actualizando manteniendo su propósito principal, la gestión de un foro completo.

phpBB es un sistema de foros gratuito basado en un conjunto de paquetes de código programados en el popular lenguaje de programación web PHP y publicado bajo la licencia pública general de GNU. Su intención es la de proporcionar fácilmente, y con amplia posibilidad de personalización, una herramienta para crear comunidades.

Las **funcionalidades básicas** son:

- Gratuito y de código abierto.
- Soporte para diversos servidores de bases de datos (MySQL, SQL Server, Oracle, etc.).
- Creación ilimitada de foros y subforos.
- Mejora en el rendimiento desde su versión anterior.

- Uso de caché para todos los archivos del foro.
- Registro de usuarios y personalización de cada campo.
- Mensajes privados a múltiples usuarios y carpetas de mensajes.
- Búsqueda de temas y usuarios.
- Panel de administrador y de moderador por separado.
- Creación de encuestas con múltiples opciones.
- Perfil para cada usuario con sus datos personales.
- Aplicar "Ban" por tiempo definido o indefinido.
- Los usuarios pueden tener "Amigos" o "Ignorados", los mensajes de "Ignorados" son ocultados automáticamente.
- Personalización de BBCode.
- Creación de grupos de usuarios, moderadores o administradores.
- Advertencia y reportes por usuarios a moderadores ante *posts* indebidos.
- Posibilidad de crear nuevos campos para el perfil de usuario.
- Posibilidad de editar, desde el panel de administración, los archivos del tema usado.
- Múltiples archivos adjuntos.
- Modificaciones y estilos gratuitos desarrollados por la comunidad.
- Fácil creación y asignación de rangos por *posts* o por grupos.
- *Log* de acciones de usuarios, moderadores y administradores.

Las principales **ventajas** son:

- Es una herramienta muy completa para su propósito, difícilmente encontraremos un CMS que tenga más funcionalidades en este sentido.
- La comunidad aporta MOD, modificaciones que extienden las funcionalidades o que cambian la presentación.

Y las **desventajas**:

- Si nuestro proyecto se desvía del propósito principal de un foro, no será la solución óptima.
- Existen multitud de CMS orientados a la gestión de foros en Internet, phpBB es solo uno de ellos.

Página web

Comparativa de CMS para foros basados en PHP:
http://en.wikipedia.org/wiki/Comparison_of_Internet_forum_software_ %28PHP%29.

7.3. Blogs

Un **blog** (o bitácora) es una publicación electrónica que recopila artículos publicados por uno o más autores ordenados cronológicamente de más a menos reciente.

Normalmente, los visitantes del blog pueden dejar comentarios. La mayoría de blogs publican automáticamente sus contenidos a través del formato RSS o Atom, de manera que la gente (entre otras cosas) se puede suscribir a la publicación.

El gran crecimiento de la blogosfera observado los últimos años (desde mediados de los noventa) ha sido principalmente propiciada por la eclosión de diversos CMS que han facilitado enormemente la gestión de estos blogs. Actualmente existen millones de blogs de diversos tipos, tanto personales como corporativos que se usan como herramienta de comunicación entre las empresas y los clientes.

Otras funcionalidades que debe incluir cualquier blog son:

- Sindicación en RSS/Atom de las entradas y los comentarios.
- *Ping*. Mecanismo por el cual el blog avisa a determinados servidores de que se ha publicado un nuevo contenido.
- *Blogroll*. Posibilidad de mantener una lista de enlaces a otros blogs que el autor considera interesantes.

Los ***podcasts*** y los ***vodcasts*** son variantes de blogs en los cuales el medio principal no es el texto, sino el audio (*podcast*) o el vídeo (*vodcasts*). A efectos prácticos, funcionan exactamente igual que un blog de texto. Los usuarios se pueden suscribir a los contenidos usando agregadores, aunque en este caso pueden ser agregadores específicos de *podcasts*, como por ejemplo iTunes de Apple, o reproductores de vídeo en Internet como Miro.

7.3.1. WordPress

WordPress (http://www.wordpress.org/) es una de las plataformas CMS más utilizadas en blogs y también en proyectos web más generales. Desde el principio se planteó como un sistema para que los autores de blogs pudieran administrar el contenido de una manera sencilla, pero ha evolucionado tanto que actualmente este CMS se usa en sitios web de propósito más genérico.

Las **funcionalidades básicas** son:

- Sistema de *widgets* que permite añadir funcionalidades mediante *drag & drop*.
- Activación sencilla de temas (*themes*).
- *Plugins* que añaden funcionalidades.
- Instalación y actualización de temas y *plugins* directamente desde el panel de administración (sin tener que descargar archivos).
- Múltiples categorías para los artículos.
- Definición de diferentes tipos de contenido.
- *Trackback* y *pingbacks*.
- Aplicaciones nativas para actualizar Wordpress desde Android y iPhone.
- Actualización remota del blog.

Las principales **ventajas** son:

- Enfocado a la sencillez de uso. El panel de administrador es muy intuitivo.
- Optimizado para buscadores (SEO).
- Las páginas de WordPress pueden tener "padres" de manera que se puede crear una jerarquía (secciones y subsecciones).
- Existen infinidad de *plugins* que añaden funcionalidades.

Y las **desventajas**:

- Aunque es bastante flexible, si las necesidades del sitio van más allá de las propias de WordPress puede ser problemático; no es, por ejemplo, la mejor opción para un sitio de comercio electrónico.

- No es la mejor opción para un sitio con miles de páginas (por su rendimiento).

Ejemplo:

Radio Associació de Catalunya. RAC1 (http://www.rac1.org/).

7.3.2. PivotX

PivotX (http://pivotx.net/) es otro ejemplo de CMS orientado a la creación de blogs, también con licencia GPL. Una de las particularidades que tiene es que **no precisa un motor de base de datos** (puede almacenar la información en un archivo), factor que facilita la instalación en entornos donde no tengamos disponible ningún sistema de base de datos.

El sistema de plantillas (o temas) que usa está basado en Smarty, sistema de plantillas web basado en PHP. También usa otros proyectos como jQuery y tinyMCE (editor HTML).

Las **funcionalidades básicas** son:

- Sistema de plantillas basado en Smarty.
- Múltiples blogs con una sola instalación.
- Múltiples autores.
- Posibilidad de incrustar imágenes e incluso usar la extensión "picnik" que permite editar las imágenes.
- Extensiones.
- Posibilidad de usar un archivo para almacenar los datos o una base de datos MySQL.
- Sistema *anti-spam*.
- Optimizado para buscadores (SEO).
- Basado en un *framework* que permite crear extensiones.

Las principales **ventajas** son:

- Es sencillo.
- Es una buena opción si no disponemos de base de datos.

Y las **desventajas**:

- La comunidad de PivotX es todavía muy incipiente.
- Pocas extensiones y temas disponibles.

Observación

Este CMS todavía no está suficientemente extendido para proporcionar buenos ejemplos.

7.4. Wikis

Una wiki[1] es un sitio web en el que los visitantes pueden editar directa y fácilmente cualquiera de sus páginas y añadir secciones nuevas.

(1)Del hawaiano wiki: "hacer las cosas de forma sencilla y rápida".

Muchas wikis permiten editar el contenido sin estar ni siquiera registrado. En cualquier caso, se guarda el historial de todos los cambios y el autor (o, si es anónimo, su ip), para poder recuperar en cualquier momento un estado anterior de cualquier página.

La wiki más conocida es la **Wikipedia**, que está implementada en la CMS wiki de código abierto MediaWiki.

En la mayoría de wikis el texto se edita en texto llano (en lugar de utilizar HTML) para mejorar la facilidad de uso. Algunas convenciones especiales del texto sirven para añadir caracteres especiales al texto.

Por ejemplo, es el caso del CamelCase, una convención que permite definir enlaces entre páginas escribiendo varias palabras sin espacios y marcando en mayúsculas la primera letra de cada palabra:

EstoEsUnEjemplo

De esta manera, la aplicación de wikis sabe que es un enlace a otra página y lo traduce como un enlace de HTML.

7.4.1. MediaWiki

MediaWiki (http://www.mediawiki.org/wiki/MediaWiki) funciona bajo la plataforma PHP y está licenciada en GPL.

Se puede escribir en MediaWiki en un formato especial llamado *wikitext*, que permite editar fácilmente y dar formato al texto sin instrucciones de HTML o CSS. El desarrollo de MediaWiki está patrocinado por la fundación WikiMedia.

Las **funcionalidades básicas** son:

- Se pueden definir tipos de páginas.
- Cada página de la wiki tiene una página de discusión propia.

- Soporte de TeX, para visualizar fórmulas matemáticas.
- Sistema de *plugins* que permite extender fácilmente las funcionalidades.
- Capacidad de bloquear temporalmente usuarios o páginas.
- Soporte de plantillas personalizadas con parámetros.
- Creación de líneas de tiempos a través de código wiki.
- Sistema de categorías jerárquico, que permite crear listados de artículos o de *thumbnails* de imágenes.
- Admite varios niveles de usuario.
- Soporte para almacenamiento de memoria virtual o caché, también conocidos como *memcached* y el sistema de *caché* Squid.
- *Skins* personalizables por cada usuario.

Las principales **ventajas** son:

- A diferencia de las wikis clásicas, los nombres de las páginas no tienen por qué estar en CamelCase, lo que permite tener nombres más naturales.
- Es un sistema altamente probado ya que es la base de la Wikipedia.

Y las **desventajas**:

- Una wiki es una buena herramienta de trabajo colaborativo; sin embargo, no es la mejor opción para personal con conocimientos a nivel de usuario debido a que el formateo de texto requiere un tiempo de aprendizaje.
- Hay que tener en cuenta que la estructura de una wiki se adapta solo a cierto tipo de proyectos y que requiere de conocimientos avanzados de edición (al menos más altos que en CMS donde se actualiza el texto en campos sin ningún tipo de formateo).

  Ejemplo:

  Wikipedia (http://es.wikipedia.org/).

7.4.2. Tiki Wiki

Tiki Wiki (http://info.tikiwiki.org/) es otro CMS enfocado a wikis de código abierto, con licencia GPL y basado en PHP. Para diferenciarse del resto, pretende cubrir otras áreas de uso como webs genéricos, portales, intranets o extranet y, más en particular, el área del trabajo colaborativo.

Las **funcionalidades básicas** son:

- Posibilidad de crear diferentes tipos de contenido como artículos, foros, boletines, blogs, galería de imágenes y, por supuesto, wikis.
- Posibilidad de crear dibujos mediante un *applet* de Java.
- Rastreadores.
- Sondeos, encuestas y cuestionarios.
- FAQ.
- Chat.
- Sistema de gestión de *banners*.
- Integración con sistemas de correo.
- Calendario.
- Mapas y gráficos.
- Versión para móviles: Tiki móvil.
- *Feeds* RSS.
- Sistema de categorías.
- Activación de temas.
- Flujo de trabajo.
- Shoutbox.
- ACL (sistema estándar de permisos sobre objetos en una red).

Las principales **ventajas** son:

- No solo es un CMS para wikis, sino que aporta varias herramientas útiles para el trabajo colaborativo; posiblemente, es uno de los CMS más completos en este sentido.

- Puede funcionar con un gran número de servidores de bases de datos.

Y las **desventajas**:

- El rendimiento y escalabilidad necesitan mejoras.

- *Featuritis*. Tiki Wiki ofrece tantas funcionalidades que es difícil que todas ellas se puedan ajustar bien a nuestros requerimientos.

- La gran cantidad de funcionalidades que implementa hace difícil la traducción de todas ellas a los diferentes idiomas.

  Ejemplo:

  Mozilla Firefox Support (http://support.mozilla.com/).

7.5. *E-learning*

En el ámbito de la educación y el *e-learning* también podemos encontrar varios CMS especializados. Las funcionalidades muy específicas que se requieren en este tipo de entornos hace realmente necesario que los CMS estén adaptados a cubrir estas necesidades. El flujo de trabajo está muy determinado por el tipo de contenidos (de tipo didáctico en forma de lecciones, unidades, pruebas, etc.) y los perfiles (profesores, alumnos).

7.5.1. Moodle

Uno de los CMS para *e-learning* más usados y con una gran comunidad de soporte es **Moodle** (http://www.moodle.org/), también basado en PHP/MySQL y con licencia GPL. La comunidad de desarrolladores de Moodle ha permitido que este CMS esté traducido a infinidad de idiomas (más de 80) y que se hayan desarrollado un gran número de extensiones para cubrir necesidades específicas.

Las **funcionalidades básicas** son:

- Vienen dadas por los módulos.
- Tareas y calificaciones para los alumnos.
- Consultas. Funcionan como una votación.
- Foro. Exclusivos para profesores, para alumnos o para todos.
- Diario. Información privada entre profesor y alumno.
- Cuestionario. Definir preguntas en cuestionarios.
- Recursos. Contenidos digitales en formato Word, PowerPoint, vídeo, etc.
- Encuestas.

- Wiki. Para trabajo colaborativo en documentos.

Las **ventajas principales** son:

- Hay una gran comunidad detrás de Moodle, lo que hace que existan gran cantidad de extensiones y que muchos problemas se puedan solucionar buscando información en Internet.

- Muy configurable gracias a la existencia de muchos *plugins*.

Y las **desventajas**:

- No existe un sistema de avisos general para todos los cursos, es necesario ir uno por uno para verlos.

- El control de grupos de estudiantes es a nivel de curso, no general.

- La instalación podría simplificarse.

7.6. Redes sociales

Las **redes sociales** son un tipo de aplicación cada vez más común en Internet; Facebook, Twitter o MySpace son solo algunos ejemplos. Este hecho ha generado la necesidad de crear aplicaciones de este tipo a medida, lo que podríamos llamar aplicaciones de redes sociales,

> Una **red social** es una aplicación web donde todos los usuarios quedan identificados mediante perfiles (datos básicos, foto, etc.) y pueden generar contenidos y establecer relaciones entre ellos (red de contactos, mensajes internos, etc.).

Aunque los CMS genéricos (como Drupal o incluso Wordpress) pueden utilizarse como base para crear una red social de este tipo (mediante la instalación de algunos módulos especiales), existen CMS que de partida ya están pensados para este propósito. Hay alternativas de aplicaciones *on-line* como Ning o SocialEngine, aunque se trata de soluciones que son propietarias y de pago.

Algunos de estos CMS de código abierto son:

- **Pligg** (http://www.pligg.com/). Implementa una red social basada en un sistema de recomendación (tipo “digg.com” o “meneame.com”). Es un CMS completo con posibilidad de añadir módulos.

- **Meneame.com** (http://www.meneame.com/). Basado en el mismo principio, también es una herramienta de código abierto y de hecho ya existen clones de esta aplicación en otros idiomas o con otras temáticas.

- **Elgg** (http://elgg.org/). Es una herramienta más completa, ya que no solo es una red de recomendaciones, sino que su objetivo es poder funcionar como base de una red dentro de una organización (dentro de una universidad, por ejemplo, o simplemente un grupo de gente con los mismos intereses).