Curso Desarrolladores de Tiendas en Magento

Cámara Zaragoza

Índice de contenido

Curso para creadores de tiendas Magento

Durante el presente curso vamos a estudiar los aspectos fundamentales, y los pasos básicos, que nos permitirán crear una tienda on-line con el software Magento. A la finalización del mismo seremos capaces de desarrollar sólidas soluciones de comercio electrónico, robustas y con gran amplitud de prestaciones.

Para ir entrando en materia me gustaría que viéramos un poquito sobre las funcionalidades y puntos fuertes de Magento. A los que hayáis trabajado con otros sistemas de comercio electrónico quizá os suenen nombres como:

- Oscommerce
- OpenCart
- ZenCart
- Virtuemart (extensión de Joomla!)
- Ubercart (extensión de Drupal)

La mayoría de estos sistemas ofrecen grandes prestaciones y, en general, son buenas soluciones para desarrollar webs de comercio electrónico. Pero, si hay algo en lo que Magento destaca, a primera vista por supuesto, es en lo cuidado de su aspecto gráfico. Sin desmerecer a las otras soluciones, Magento tiene una estética de panel de gestión que impresionará a la mayoría de clientes por su profesionalidad.

Como programadores php nos beneficiaremos de que esté desarrollado sobre el magnífico Zend Framework, con lo que los programadores que hayan trabajado con este entorno ya tienen alguna ventaja.

Algunos de los puntos fuertes de Magento, recién instalado y sin que tengamos que hacer nada son:

- **Gran amplitud de métodos de pago** -> no solo dispondremos del conocido PayPal, sino Google Checkout y, visitando Magento Connect (como veremos más adelante) podremos descargarnos extensiones de terceros para métodos de pago de bancos españoles.
- **Seguridad en el método de pago** -> protegido por SSL
- **Estadísticas** -> nada más entrar al panel de gestión veremos un resumen de las ventas que ha tenido nuestra tienda, absolutamente práctico.
- **Opiniones sobre los productos y valoraciones** -> algo básico, pero no deja de ser útil.
- **Mejorado para SEO** -> por defecto Magento ya hace un gran trabajo con el SEO de nuestra web, incluso es capaz de generar Site Maps para Google.
- **Cupones y descuentos**-> herramientas esenciales de marketing para nuestro sitio.

Todas estas opciones, de por si, conferirían una herramienta muy interesante, pero hay muchas más que iremos viendo poco a poco.

Instalación

Instalar Magento es una tarea realmente sencilla, que puede ser realizada en pocos pasos, sin embargo existen varios requisitos que debe satisfacer nuestro entorno de trabajo. Bien sea nuestro entorno de pruebas, o nuestro entorno de producción.

Técnicamente hablando ambos entornos, idealmente, deberían de ser similares y cubrir los

requisitos de Magento. Muy resumidamente podemos decir que necesitaremos:

- Apache 1.3+
- PHP 5.2+
- mySQl 4.1.2+

Aunque realmente nos estaríamos quedando algo cortos con la especificación, pues realmente vamos a necesitar:

- Linux x86, x86-64
- Extensiones de php:
  - PDO_MySQL
  - simplexml
  - mcrypt
  - hash
  - GD
  - DOM
  - iconv
  - curl
- Safe_mode off
- Memory_limit de al menos 256Mb aunque más es preferible (512 es sugerido)
- Posibilidad de ejecutar tareas Cron
- Posibilidad de modificar los ficheros .htaccess

La lista de requisitos es bastante amplia, y, como hemos comentado antes, tanto el entorno de pruebas como el de producción deberían cumplirla.

Entorno de producción

Aunque durante el curso no vamos a trabajar en un entorno de producción real, como sería instalar nuestra web en un proveedor de hosting. Me gustaría hacer hincapié en algunos puntos importantes a la hora de valorar que proveedor contratar:

1) El limite de memoria, 256Mb hace referencia a la cantidad de memoria que el servidor cede para la ejecución de los scripts php, **no a la memoria del servidor**. Magento es un software más exigente que otras soluciones, y por lo tanto los requisitos son bastante elevados. No todos los proveedores de hosting ofrecen esta cantidad de memoria (yo he visto casos de ofrecer 16 o 32Mb). Como resultado nuestro sitio puede **o bien no funcionar en absoluto, o hacerlo muy lentamente**, estropeando la experiencia a nuestros visitantes, y potenciales clientes.
2) Posibilidad de modificar los ficheros .htaccess, ya sea para utilizar la extensión mod_rewrite de Apache u otras tareas. **No todos los proveedores de hosting permiten esto**, sobre todo en los hostings compartidos donde la configuración es global para todos los clientes.
3) Posibilidad de ejecutar tareas Cron. De nuevo nos encontraremos con que la mayoría de hostings compartidos no nos ofrezcan esta posibilidad, ya que el grado de control sobre el

servidor es bastante amplio.

Como podemos ver, son unos requisitos bastante específicos, que no todos los proveedores cumplen. De esta manera la mejor opción es enviarles la lista de requisitos antes de contratar nada, y una vez contratado comprobar que el servicio cumple con todo lo necesario.

Una manera muy sencilla de comprobar esto es descargandonos un pequeño script desde esta página:

http://www.magentocommerce.com/knowledge-base/entry/how-do-i-know-if-my-server-is-compatible-with-magento

Este script comprobará que nuestro servidor cumple con todo lo necesario, evitándonos dudas y futuros problemas.

Entorno de pruebas

O de desarrollo. Será en este entorno donde desarrollemos nuestro sitio, para posteriormente migrarlo al entorno de producción. A partir de ahí todos los cambios, pruebas y modificaciones, siempre serán realizadas, y probadas, en este entorno previamente a implantarlas en el entorno de producción.

Esto nos garantiza que lo que tengamos en producción ha sido probado previamente, y funciona correctamente.

Como entorno de pruebas nos sirve prácticamente cualquier equipo, solo necesitaremos instalar un servidor apache, php y mySQL. Afortunadamente existen paquetes que instalan todos estos componentes automáticamente. Como puede ser el caso de XAMPP y WAMP. Durante el curso vamos a utilizar el software WAMP, lo podemos descargar desde esta página:

http://www.wampserver.com/en/

El proceso de instalación es bastante sencillo, ya que consiste en aceptar las opciones que nos vienen por defecto.

Al finalizar dicho proceso se habrá creado una carpeta **wamp** en nuestra unidad **c:** (en la configuración por defecto).

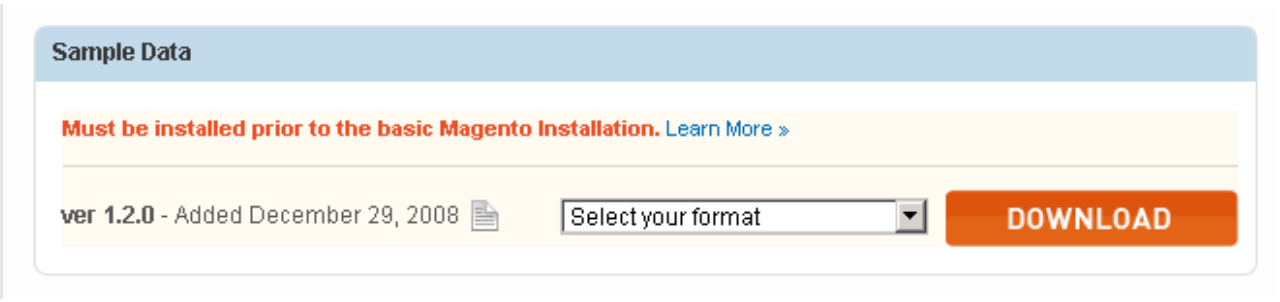
Dentro de esta carpeta **wamp**, encontraremos otra llamada **www**, que vendría a ser la carpeta raíz de nuestro servidor. Será aquí dentro donde instalaremos nuestra copia de Magento. Pero no lo vamos a instalar directamente en www, sino en una nueva carpeta que crearemos, llamada **proyecto-mg**, por ejemplo. De esta manera, la ruta final hasta nuestra instalación de Magento será c:/wamp/www/proyecto-mg.

Bien, nuestro siguiente paso será descargar una copia de Magento, esto lo podemos hacer desde la siguiente página:

http://www.magentocommerce.com/

Desde ahí podremos hacer clic en el menú download, y, posteriormente en Download Magento. Luego se nos mostrará una ventana donde deberemos elegir la versión que queremos descargar. Para el presente curso vamos a utilizar la **magento-1.4.1.1**, que es la última versión estable disponible a fecha de hoy.

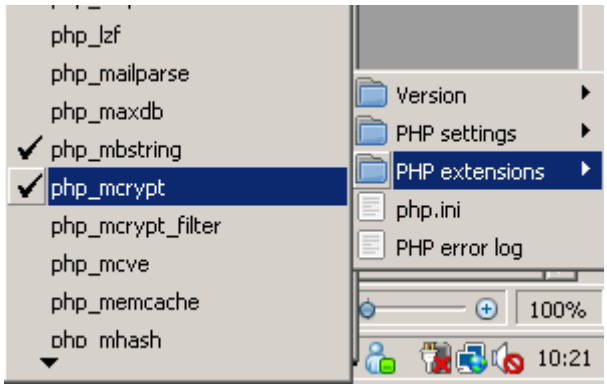
En este momento también nos es posible descargarnos los datos de ejemplo, y es interesante hacerlo, pues nos permitirá contar con una web aparentemente funcional, permitiéndonos hacer pruebas y navegar por ella de una forma mas realista.



Antes de continuar es importante asegurarnos de que nuestro servidor local está en marcha, en caso afirmativo deberemos ver un pequeño símbolo en la zona del reloj, como este:

Si no lo vemos, significará que el servidor está apagado. Encenderlo es tan trivial como hacer clic en el icono de escritorio que nos habrá generado la instalación del mismo:

No debemos tampoco olvidar el habilitar todos las extensiones de php que hemos visto en el apartado de requerimientos. Lo haremos desde el icono de wamp, el que está al lado del reloj, haciendo clic en el. Navegaremos al menú php, luego al menú extensiones, e iremos haciendo clic en las opciones necesarias, para ir activándolas, en caso de que no estén.



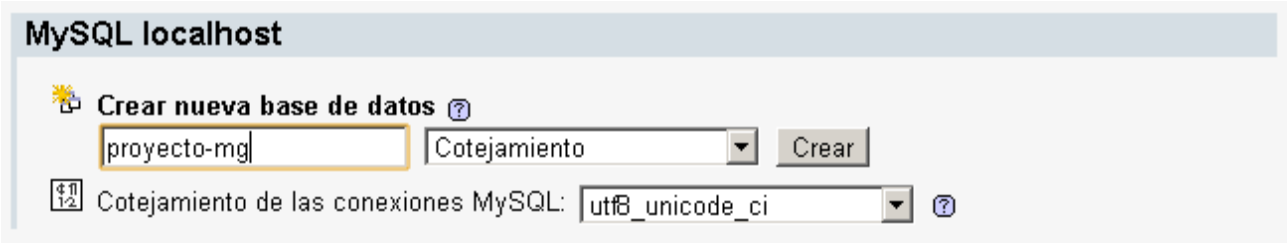
Como podemos ver es muy simple activar las extensiones desde la interfaz de wamp. Repasaremos todas para que no se nos olvide ninguna. Aunque de ser así, Magento, durante el proceso de instalación se encargaría de recordárnoslo.

Comenzando con la instalación

Antes de poder comenzar con la instalación, deberemos de crear también una base de datos, afortunadamente wamp instala una copia de phpmyadmin, que podremos lanzar visitando la siguiente url:

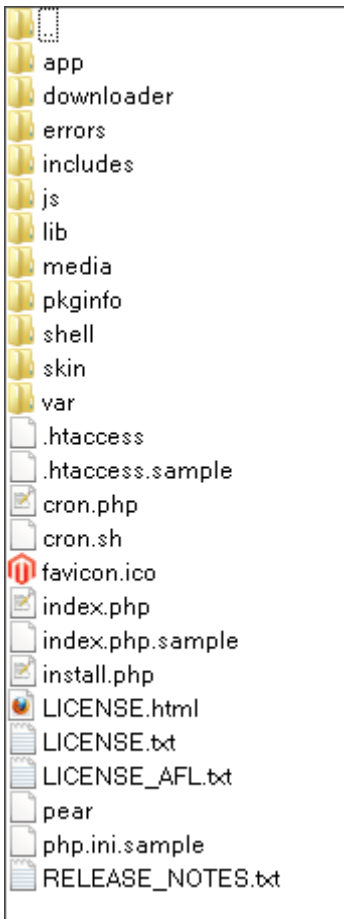
http://localhost/phpmyadmin/

Desde esa pantalla daremos nombre, y crearemos, nuestra base de datos, para este ejemplo la llamaremos proyecto-mg, para que coincida con la carpeta del servidor, aunque realmente cualquier nombre sería igual de correcto.



Ahora descomprimiremos el zip de Magento que hemos descargado, en mi caso magento-1.4.1.1.zip, y moveremos todo su contenido a la carpeta c:/wamp/www/proyecto-mg

El aspecto de la carpeta, al final de la copia de archivos, será más o menos el siguiente:



Pero no vamos a entrar todavía en detalles sobre esta estructura. Si recordamos, también hemos descargado otro fichero con los datos de ejemplo para la instalación, es el momento de que utilicemos dicho fichero. En mi caso el fichero se trata de magento-sample-data-1.2.0.zip, lo descomprimiremos. Dentro encontraremos dos cosas:

- Una carpeta "media" → que contiene las imágenes de los productos de ejemplo.
- Un fichero SQL "magento_sample_data_for_1.2.0.sql" que generará las tablas de base de datos con los contenidos de los datos de ejemplo.

La carpeta "**media**" simplemente la copiaremos y pegaremos en nuestra carpeta **proyecto-mg**. Para instalar el fichero SQL volveremos a nuestra pantalla de phpmyadmin:

http://localhost/phpmyadmin/

Desde el menú importar, seleccionaremos este fichero, y aceptaremos. Al final del proceso nuestra base de datos contendrá 229 nuevas tablas, así como los datos de ejemplo necesarios. Pero no nos preocupemos, no será necesario que indaguemos en esta tablas. Haberlas creado será suficiente.

Después de esto, si navegamos a http://localhost/proyecto-mg veremos una pantalla similar a la siguiente, la pantalla inicial de instalación de Magento:

En este momento es muy tentado aceptar los términos y continuar con la instalación. **Pero hay un punto importante que me gustaría que observáramos primero**. Como vemos estamos instalando nuestro sitio de ejemplo en http://localhost/proyecto-mg pero esto tiene un problema.

Una vez hayamos terminado la instalación, e intentemos acceder al panel de gestión, veremos que aún introduciendo datos correctos, no podremos. Se debe a que Magento utiliza cookies, pero, por defecto, muchos navegadores no crean cookies para webs que no contengan un punto. Es decir, para

localhost

No se crearían cookies, pero para

www.localhost.com

si que se crearían. Al no haber cookies, no podremos acceder al panel de administrador de Magento. ¿Que posibles soluciones tenemos para este problema? En realidad existen varias, por ejemplo, utilizar la ruta http://127.0.0.1/proyecto-mg/ en lugar de http://localhost/proyecto-mg . Esta es un solución bastante sencilla, que no requiere de ninguna modificación o trabajo por nuestra parte.

Por supuesto existen formas más elegantes de solucionar este problema, por ejemplo, añadiendo una entrada al fichero **hosts** de windows. Este fichero se encuentra en C:\Windows\System32\drivers\etc

y la entrada que debemos añadir es esta:

127.0.0.1 www.localhost.com

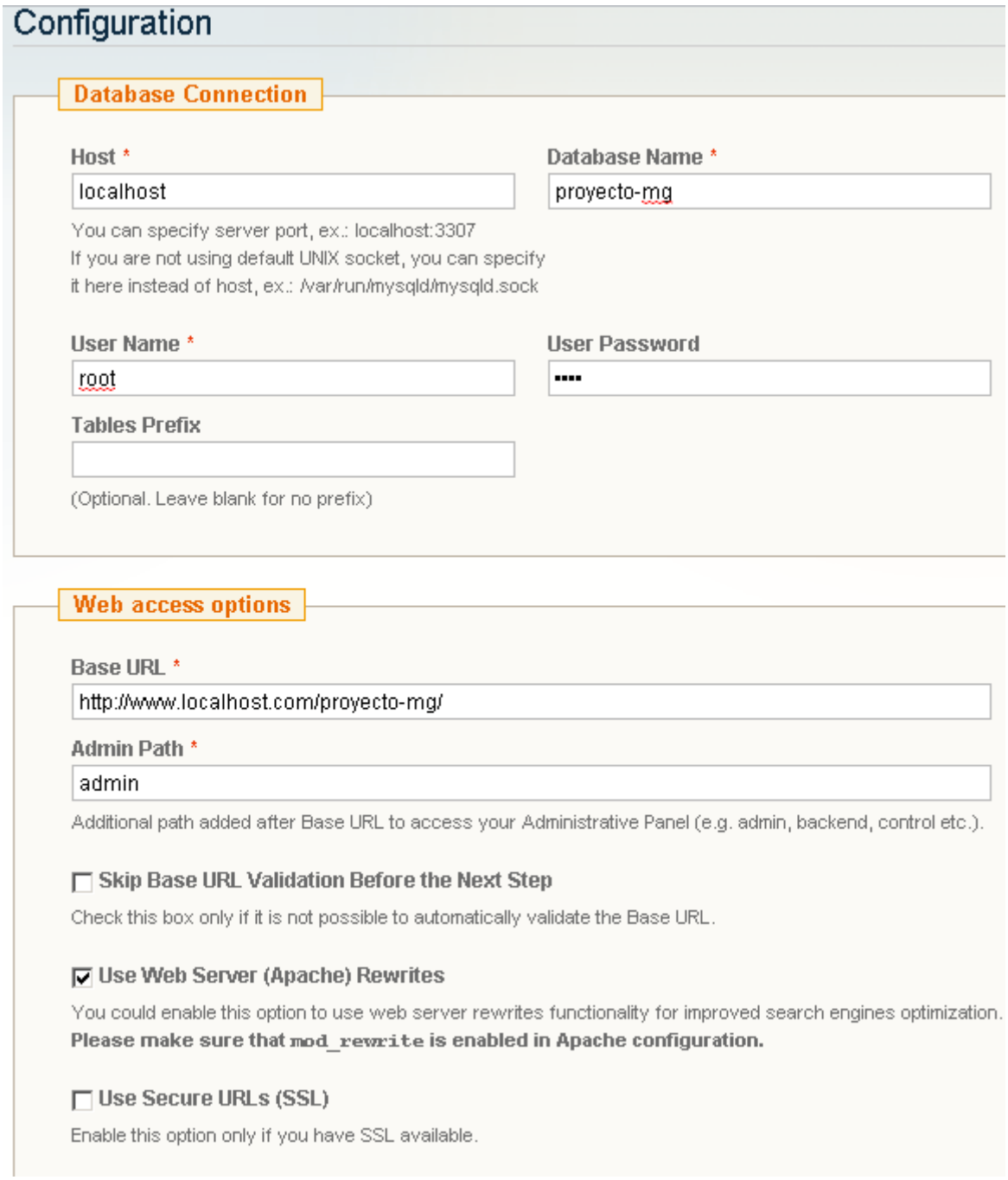
De manera que cuando en nuestra navegador insertemos http://www.localhost.com/proyecto-mg se resuelva nuestro dominio. Con este pequeño cambio habremos solucionado nuestro problema y podremos, ahora sí, continuar con la instalación.

El proceso de instalación

A partir de ahora nos encontramos con un proceso de instalación común, donde, entre otras configuraciones, tendremos que seleccionar la moneda, zona horaria etc

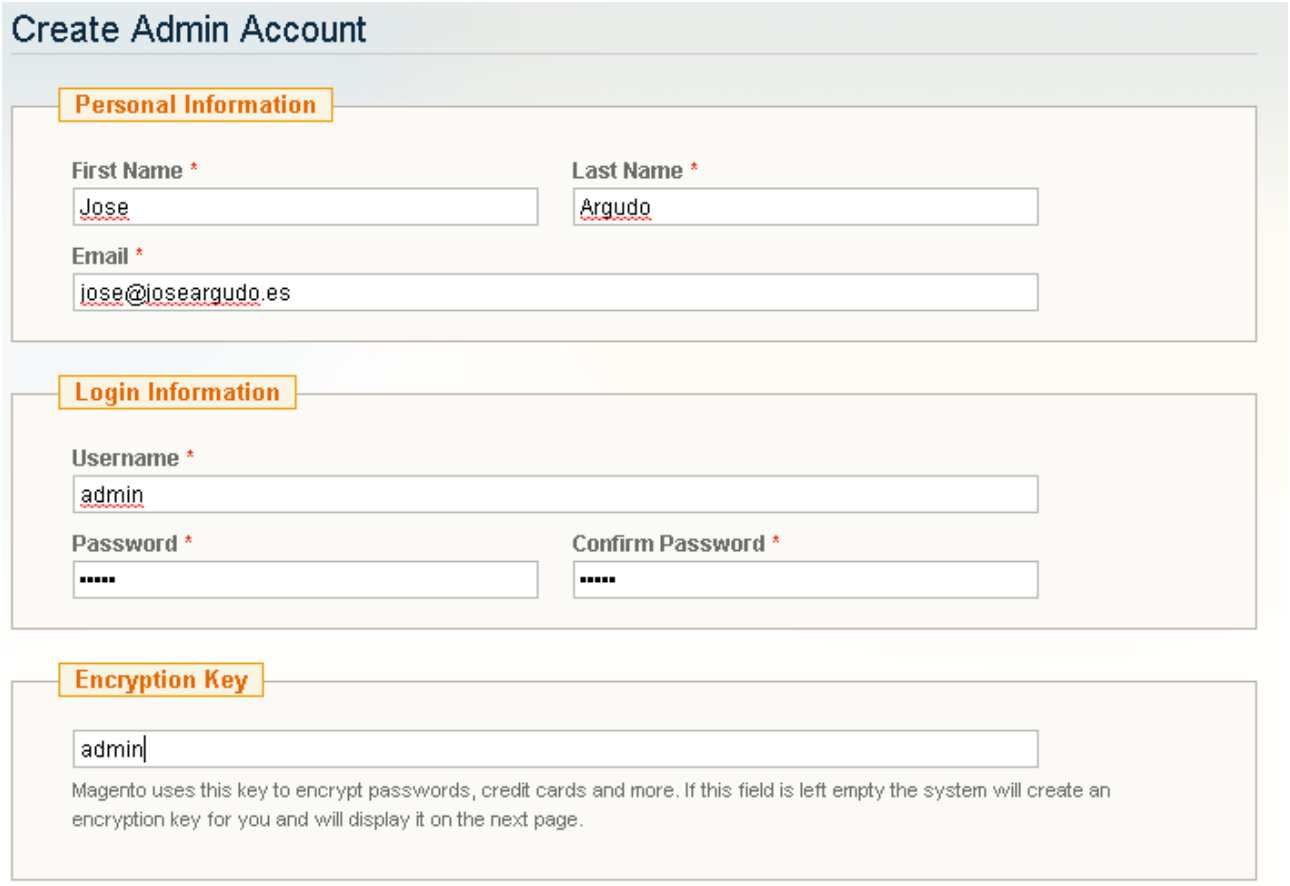
Además, como se puede apreciar en la siguiente imagen, tendremos que seleccionar:

- Nuestro host → en este caso localhost
- El nombre de la base de datos → en nuestro caso proyecto-mg
- El usuario de la base de datos
- Y su contraseña
- La url base → en nuestro caso http://www.localhost.com/proyecto-mg/
- La ruta hacia el panel de administración → dejaremos la opción por defecto, “admin”
- Habilitaremos el uso de Apache rewrites, para generar URLs SEO



Una vez aceptemos esta pantalla, Magento generará la base de datos y nos presentará la última pantalla de instalación. En ella elegiremos las siguientes opciones:

- Nombre y apellido
- Cuenta de email
- Usuario → en nuestro caso "**admin**"
- Password para la zona de administración → en nuestro caso "**administrador1**"
- Y la llave de encriptación → en este caso "**admin**" también.



Y con esto estaría finalizada la instalación. Rápida y sencilla. Pero, antes de continuar me gustaría remarcar un par de aspectos.

Configuración de los permisos en servidores linux

Hasta ahora hemos trabajado en el entorno de pruebas instalado por el WAMP bajo windows, sin embargo, en caso de que lo instalemos en un servidor linux, necesitaremos conceder permisos de escritura a ciertos ficheros y directorios. Pasaremos ahora a enumerarlos:

- fichero -> var/.htaccess
- directorio -> app/etc
- directorio -> var
- directorio -> media

No son necesarias más modificaciones para instalar Magento.

Actualizando nuestra versión de Magento

Aunque existen varios métodos para actualizar nuestra versión de Magento, una de las formas más sencillas de realizarlo es siguiendo los siguientes pasos:

- Lo primero sería realizar una copia de seguridad de la base de datos, desde nuestro panel de

phpmyadmin.

- Hay que tener en cuenta, para el punto anterior, que, al ser la base de datos tan grande, puede fallar la exportación debido al limite de tiempo de ejecución de los scripts php. Podemos cambiar esta configuración en el fichero **php.ini -> max_execution_time**
- Después haremos una copia de todos nuestros ficheros, poniendo especial énfasis en los siguientes:
  - Los que hayamos modificado
  - La carpeta **media**
  - Copiaremos el fichero app/etc/local.xml
- A partir de ahí descargaremos un nuevo fichero de instalación, conteniendo la nueva versión.
- Extraeremos esos ficheros y sobrescribiremos los actuales.
- Borraremos el contenido de los directorios:
  - var/cache
  - var/session
- Cargaremos, en nuestro navegador, cualquier página de nuestra instalación de Magento, lo que finalizará con el proceso de instalación.

Migrando nuestra instalación de Magento

Una vez hayamos finalizado de trabajar en nuestra instalación local, necesitaremos trasladar a un servidor online, u otro servidor de producción. Para conseguirlos seguiremos los siguientes pasos:

- Primero accederemos a nuestro panel de **admin**, después accederemos al menú **System** -> **configuration** -> **web**. Tanto en el panel **Secure**, como en el **Unsecure**, cambiaremos la información de **Base Url** a la que corresponda en nuestro nuevo servidor, o bien, por **{{base_url}}**
- Copiaremos y exportaremos la base de datos, importándola en el nuevo servidor.
- Copiaremos todos los ficheros al nuevo servidor.
- Borraremos el contenido de los directorios:
  - var/cache
  - var/session

Esta información, sobre la URL base de nuestra web, también la podemos encontrar en la tabla:

- core_config_data

En caso de que la configuración de conexión con la base de datos sea diferente, la podremos modificar en el fichero **app/etc/local.xml**, donde encontraremos las siguientes lineas:

```
<connection>
    <host><![CDATA[localhost]]></host>
    <username><![CDATA[root]]></username>
    <password><![CDATA[]]></password>
    <dbname><![CDATA[proyecto-mg]]></dbname>
    <active>1</active>
</connection>
```

Es en estas líneas donde configuraremos los nuevos parámetros de conexión con la base de datos. Con esto hemos visto los aspectos más importantes de la migración. Recalcar que puede ser necesario configurar los permisos de las carpetas, para que todo funcione correctamente.

¿Problemas con los datos de ejemplo?

Es posible que, incluso habiendo instalado correctamente los datos de ejemplo, estos no aparezcan en la página, o nos devuelvan mensajes de error 404. Solucionaremos este problema accediendo al panel de **admin** menú **system** -> **index management**, ahí haremos clic en los links que aparecen en la columna Action (reindex data), tras lo cual todos los productos de demostración funcionarán correctamente.

Deshabilitar la cache

Ahora que estamos en el panel de control, vamos a aprovechar para deshabilitar la cache de Magento. Si no lo hiciésemos, cuando empezásemos a trabajar, y realizar modificaciones, no las veríamos, pues Magento estaría utilizando la cache. Esto puede dar lugar a mucha confusión, e intentar solucionar problemas que realmente no existen.

¿Como hacer esto? Desde el panel de **admin**, iremos a **System** -> **cache management**, seleccionaremos todos los checkbox, y en el desplegable **Actions**, seleccionaremos **disable** y haremos clic en el botón **submit**.

Temas

El siguiente paso en nuestro pequeño viaje por Magento son los temas. Aunque en un principio puede parecer más complejo que con otros sistemas, como Joomla! o Wordpress, en un momento veremos como realmente no es tan complejo como parece.

Los temas, a veces también llamados plantillas, definen el aspecto de nuestro sitio, y nos permiten cambiar su aspecto sin necesidad de afectar a los datos que contiene. Esto nos ofrece una gran flexibilidad y la habilidad de cambiar la imagen gráfica de nuestro sitio dependiendo de nuestras necesidades.

Instalación de un tema en Magento

Básicamente tenemos dos formas de instalar un tema de Magento, vía Magento Connect (el sistema de instalación de extensiones de Magento) o, de manera más tradicional, copiándolas directamente en nuestra instalación de Magento.

Ambos métodos son igualmente sencillos, y normalmente la mayor dificultad residirá en seleccionar que plantilla queremos utilizar. Vamos a empezar viendo el método clásico de instalación de plantillas, es decir, copiar los ficheros directamente sobre nuestra instalación.

Instalación manual

Como hemos dicho, esta es la forma tradicional de instalar una plantilla, para poder algunas pruebas vamos a descargarnos una plantilla gratuita. Por ejemplo de esta página:

http://pelfusion.com/freebies/15-free-high-quality-magento-templates/

En este caso vamos a utilizar la plantilla "Free Magento Classic Theme". Descargaremos el fichero f002.classic.zip y, al descomprimirlo tendremos la siguiente estructura:

- graphic source -> aquí encontraremos los ficheros PSD del diseño, por si acaso necesitamos realizar alguna modificación al diseño.
- installation -> con información sobre como instalar la plantilla.
- template source 1.3.2 -> la plantilla para la versión 1.3.2 de Magento
- template source 1.4.0.1 -> la plantilla para la versión 1.4.0.1 de Magento
- template source 1.4.1.0 -> la plantilla para la versión 1.4.1.0 de Magento

Como podemos ver el tema viene bastante completo, con una amplia variedad de extras, así mismo es capaz de servir para diferentes versiones de Magento.

Aunque podemos pensar que una misma plantilla debería de servir para todas las versiones de Magento, esto no suele ser así.

Habitualmente las plantillas utilizan funciones bien sea para cargar datos de la base de datos, menús etc Es por eso que debemos de comprobar para que versión fue hecha cada determinada plantilla.

En nuestro caso vamos a utilizar la versión 1.4.1.0 del tema, que, aunque difiere un poco de nuestra versión de Magento, es bastante probable que funcione correctamente.

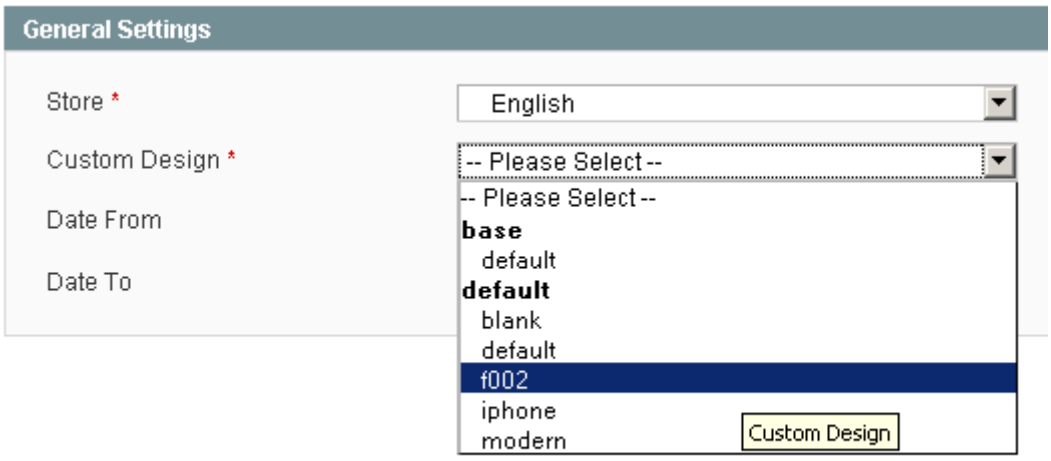
Abrimos la carpeta **template source 1.4.1.0** y dentro podemos ver otras dos carpetas:

- **app** -> aquí encontraremos los ficheros de layout (estructura de la web), locale (traducciones) y template (trocitos de la plantilla)
- **skin** -> aquí encontraremos los ficheros css, js e imágenes necesarios para nuestra plantilla.

Copiaremos estas dos carpetas en nuestra instalación de Magento, en nuestra carpeta **proyecto-mg**. Una vez hemos copiado estas dos carpetas, tenemos el 50% de la instalación finalizada. Ahora solo nos queda habilitar la plantilla para poder ver como queda.

Esto también es muy sencillo de realizar, desde nuestro panel de **admin** nos dirigiremos a **system** -> **design** y haremos clic en el botón **Add Design Change**.

Esto nos llevará a una pantalla con el siguiente aspecto:



El concepto de tienda lo veremos más adelante, de momento haciendo clic en el desplegable **Custom Design** seleccionamos **f002**, que es el que hemos instalado, y hacemos clic en el botón **save**. Con esto nuestra plantilla será utilizada, y nuestro sitio web presentará el siguiente aspecto una vez que accedamos a la url http://www.localhost.com/proyecto-mg/

Poco a poco iremos entrando en detalles sobre el funcionamiento y estructura de las plantillas, pero, de momento, ya hemos podido comprobar lo sencillo y rápido que es instalar una nueva plantilla.

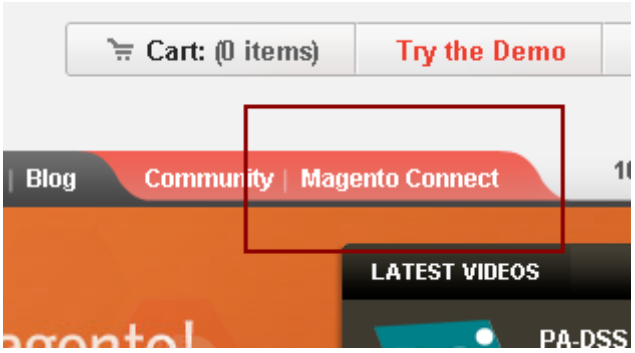
Nota

Podemos seleccionar una plantilla para que tenga efecto durante un determinado periodo de tiempo. Esto es muy práctico para automatizar el cambio de aspecto de nuestra tienda durante las celebraciones (navidad, san valentín etc)

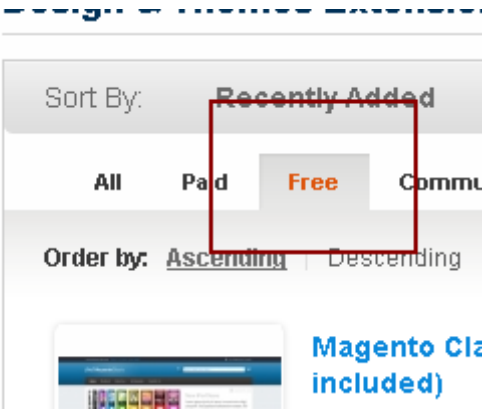
Instalación vía Magento Connect

Instalar temas usando Magento Connect es igual de sencillo, aunque requiere unos pasos bastante diferentes. El primer paso es cargar la página de Magento, http://www.magentocommerce.com/

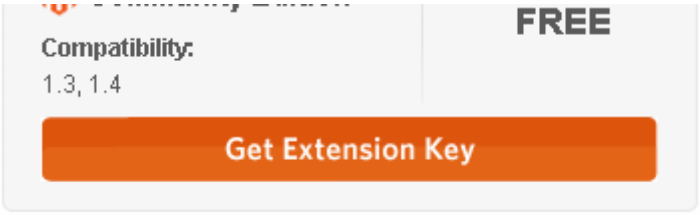
Ahí iremos al link "Magento Connect":



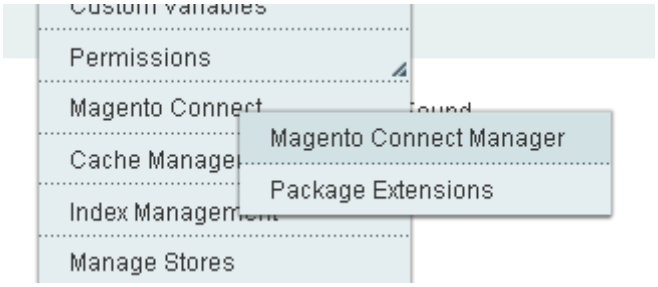
Eso nos llevará a una página con todo el catálogo de extensiones de Magento, tanto de pago como gratuitas, en el menú de la derecha podremos ver un link indicando "Design & Themes". El siguiente paso será elegir las gratuitas (en caso de que sea lo que nos interesa), tal y como podemos ver en la siguiente imagen:



De ahí podemos seleccionar el tema que queramos, siempre fijándonos en que la versión coincida con nuestra instalación de Magento, o al menos que se aproxime lo más posible. Haciendo clic en el template que nos interese accederemos a los detalles de la misma. En esa pantalla de detalle haremos clic en el botón "Get Extension Key":



Esto requiere que estemos registrados en la web de Magento, tras de lo cual obtendremos un código, el cual podremos utilizar desde nuestro panel de administración. Copiaremos el código y nos dirigiremos al admin de Magento. Iremos al menú **system** -> **magento connect** -> **magento connect manager**:



Al hacer clic en dicho menú seremos redirigidos a la página de Magento Connect Manager, donde se nos volverán a pedir nuestros datos de acceso al panel de administración. La próxima, y definitiva pantalla, nos mostrará un campo donde poder introducir el código de extensión que hemos copiado:

Una vez instalado, podremos utilizar el template de la misma forma que en el caso anterior. Como podemos ver, ambos métodos son equivalentes, así que el uso de uno u otro dependerá en gran medida de nuestros gustos.

Conceptos básicos de los temas de Magento

Antes de continuar, y ver como podemos modificar los demás, para acomodarlos a nuestras necesidades, vamos a ver algunos puntos clave que conforman los templates de Magento. El primero de ellos son los Layouts.

Layouts

Básicamente, los ficheros de layout son los que definen la estructura de las páginas de Magento, definiendo las posiciones de los elementos. Así tendremos ficheros de layout para la mayoría de páginas de Magento. Por ejemplo, vamos a observar uno de los ficheros de layout del tema que hemos instalado hace un momento. Iremos a la carpeta **app** -> **design** -> **frontend** -> **default** -> **f002** -> **layout** -> **page.xml**, si lo abrimos veremos algo similar a:

```
<block type="core/text_list" name="left" as="left" translate="label">
    <label>Left Column</label>
</block>

<block type="core/messages" name="global_messages" as="global_messages"/>
<block type="core/messages" name="messages" as="messages"/>

<block type="core/text_list" name="content" as="content" translate="label">
    <label>Main Content Area</label>
</block>

<block type="core/text_list" name="top_slider" as="top_slider"/>

<block type="core/text_list" name="right" as="right" translate="label">
    <label>Right Column</label>
</block>
```

Aquí podemos ver la estructura de bloques, que conforman el layout, sin código html, simplemente definiendo la estructura que tendrá la página en cuestión. Cada uno de estos bloques presentará una parte del código, cargándola, de manera que un layout en concreto estará formado por un conjunto de bloques.

Por ejemplo si vemos este bloque:

```
<block type="page/html" name="root" output="toHtml" template="page/3columns.phtml">
```

Podemos ver que definimos el typo de bloque, su nombre, el tipo de output (el tipo de salida del bloque, en este caso html) y el template que cargaremos (page/3columns.phtml).

Pero no nos preocupemos demasiado por estos conceptos, ya que los iremos viendo poco a poco durante el curso. De momento, y antes de continuar, me gustaría aclarar la diferencia entre los dos tipos de bloques que podemos tener, para ello veamos el siguiente código:

```
<block type="page/html_header" name="header" as="header">
  <block type="page/template_links" name="top.links" as="topLinks"/>
  <block type="page/switch" name="store_language" as="store_language"
template="page/switch/languages.phtml"/>
  <block type="core/text_list" name="top.menu" as="topMenu"/>
  <block type="page/html_wrapper" name="top.container" as="topContainer" translate="label">
    <label>Page Header</label>
    <action method="setElementClass"><value>top-container</value></action>
  </block>
</block>
```

- Por un lado podemos ver bloques **estructurales**, que nos ayudan a definir las diversas zonas de la web. Tienen una función más bien organizativa, muy al estilo de las secciones en HTML5, podemos ver por ejemplo este código:

<block type="page/html_header" name="header" as="header">
Este bloque sería de tipo estructural, y serviría como contenedor para otros bloques, que irían dentro de el, probablemente bloques de contenido.

- Bloques de **contenido**, son bloques similares al siguiente ejemplo:
<block type="page/switch" name="store_language" as="store_language" template="page/switch/languages.phtml"/>
Estos bloques cargan un archivo phtml, generando el contenido, propiamente dicho, del bloque.

Templates (ficheros phtml)

Es en estos ficheros donde encontraremos el html de nuestro sitio, en ellos veremos código html común y corriente, mezclado con funciones propias de Magento, como podemos ver en la siguiente imagen:

```
<div class="main-container col3-layout">
        <?php echo $this->getChildHtml('top_slider') ?>
    <?php echo $this->getChildHtml('breadcrumbs') ?>
    <div class="main">
        <div class="col-wrapper">
            <div class="col-main">
                <?php echo $this->getChildHtml('global_messages') ?>
                <?php echo $this->getChildHtml('content') ?>
            </div>
```

El extracto pertenece al fichero que podemos encontrar en **app** -> **design** -> **frontend** -> **default** -> **f002** -> **template** -> **page** -> **3columns.phtml** , en el podemos ver el uso de funciones que llaman a otros bloques de html, conformando la página final.

Skins

Es en esta categoría/carpeta donde se guardan el resto de ficheros que conformaran nuestra plantilla. Aquí están los ficheros CSS, JS, imágenes, ficheros flash etc

Si navegamos nuestro sitio, siguiendo esta ruta **skin** -> **frontend** -> **default** podremos ver varias carpetas nombradas así:

- f002
- f002_grey
- f002_green
- etc

Cada una de estas carpetas corresponde a un skin diferente, de hecho si las fuéramos intercambiando el aspecto de nuestro sitio (en cuanto a colores, no estructura) iría cambiando.

Interfaces

De momento, y para no liarnos demasiado con conceptos complejos, resumiremos un interfaz como

un conjunto de temas que definirán como se ve nuestro sitio. Por ejemplo, si examinamos nuestra instalación veremos esta estructura de directorios:

- app/design/frontend/default
- app/design/frontend/base

Tanto la carpeta **default** como **base** tienen en su interior un conjunto de carpetas, cada una de ellas será un tema. Por lo tanto ambas carpetas (**default** y **base**) son un conjunto de temas, un interface.

Resumiendo

Para intentar resumir un poco todos los conceptos que hemos visto, vamos a destacar la estructura general de un template:

- app/design/frontend/default/f002 -> aquí dentro tendremos los ficheros de layout y los phtml (templates)
- skin/frontend/default/f002 -> aquí estarán los ficheros css, js etc

Creación de temas

El siguiente paso en nuestro viaje por Magento es la creación de un tema o plantilla propio. En si misma no es una tarea excesivamente compleja, pero son necesarios un número de pasos determinados que ahora iremos viendo. Uno de los primeros pasos que debemos dar es la creación del html de nuestro tema. Esto nos servirá principalmente para ver de antemano como tiene que quedar, y nos ayudará a realizar las divisiones necesarias.

Estableciendo las bases, el html

Esta parte es bastante sencilla, ya que simplemente se trata de crear el html básico, por ejemplo podríamos tener la siguiente estructura:

```
<!DOCTYPE html>
<html lang="es-ES">
<head>
<title>Nuestra primera plantilla de Magento</title>
<link rel="stylesheet" href="css/estilos.css" />
</head>

<body>
<div id="wrapper" class="border">

<div id="header">
<div id="logo"><img src="images/logo.gif" /></div>
<div id="hud">
```

```
<h3>Bienvenido, admin</h3>
<ul class="links">
        <li><a href="#" title="Mi cuenta">Mi cuenta</a></li>
        <li ><a href="#" title="Mi lista de deseos">Mi lista de deseos</a></li>
        <li ><a href="#" title="Mi carrito">Mi carrito</a></li>
        <li ><a href="#" title="Finalizar">Finalizar</a></li>
        <li><a href="#" title="Salir" >Salir</a></li>
   </ul>
</div>
</div>

<div id="utilities">
<div id="breadcrumbs">Inicio >> categoría 1</div>
<div id="header-search"><input type="text" class="border" value="Buscar en la tienda" /></div>
</div>

<div id="content" class="product">
<h1>Lorem ipsum dolor sit amet</h1>, consectetur adipiscing elit. Nulla lorem dolor, congue mollis posuere id, facilisis et urna. Cras lectus elit, gravida a ullamcorper gravida, consectetur eget sem. Fusce ut odio ut lacus mollis elementum. Suspendisse vitae purus dui. Ut malesuada ullamcorper suscipit. Vestibulum consequat magna ac odio ultricies tempus. Nulla molestie faucibus egestas. Phasellus ac turpis dignissim lectus rutrum porttitor. Cras nisl purus, rhoncus at mattis ac, porttitor at mi. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae;
</div>

<div id="footer" class="border">
© 2010 Magento Template
</div>

</div>
</body>
</html>
```

Este simple código generará una página que tendrá más o menos el siguiente aspecto:

Bienvenido, admin

- Mi cuenta
- Mi lista de deseos
- Mi carrito
- Finalizar
- Salir

Inicio >> categoría 1

Buscar en la tienda

Lorem ipsum dolor sit amet

, consectetur adipiscing elit. Nulla lorem dolor, congue mollis posuere id, facilisis et urna. Cras lectus elit, gravida a ullamcorper gravida, consectetur eget sem. Fusce ut odio ut lacus mollis elementum. Suspendisse vitae purus dui. Ut malesuada ullamcorper suscipit. Vestibulum consequat magna ac odio ultricies tempus. Nulla molestie faucibus egestas. Phasellus ac turpis dignissim lectus rutrum porttitor. Cras nisl purus, rhoncus at mattis ac, porttitor at mi. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae;
© 2010 Magento Template

No es mucho de momento, así que lo completaremos con un poco de CSS. Como podemos ver hemos preparado la página para enlazar una hoja de estilos llamada **estilos.css**:

```
<link rel="stylesheet" href="css/estilos.css" />
```

Así que crearemos una carpeta llamada css, y dentro un fichero **estilos.css**, que contendrá el siguiente código:

```
* {
	margin: 0;
	padding: 0;
	border: none;
	outline: none;
	color: #333;
	font-family: "Lucida Sans Unicode", "Lucida Grande", sans-serif;
	font-size: 14px;
	list-style: none;
	line-height: 1.3em;
}

a {
	color : #7db000;
}

h1, h2, h3, h4 {
	font-weight: normal;
}
```

```
h1 {
	font-size: 32px;
	margin-bottom: 10px;
}

h2 {
	font-size: 24px;
	margin: 10px 0 12px 0;
}

h3 {
	font-size: 20px;
	margin-bottom: 5px;
}

h4 {
	font-size: 20px;
}

.border {
	border: 1px solid #666;
}

/* Base Elements */

#wrapper {
	width: 920px;
	margin: 10px auto;
	padding: 20px;
}

#header {
	margin: 0 0 20px 0;
	overflow: auto;
}

#content {
	margin: 20px 0;
	overflow: auto;
}
```

```
#footer {
	padding: 10px;
	border: 1px solid #E1E1E1;
	background: #F3F3F3;
	text-align: center;
}

/* Header content */

#logo {
	float: left;
}

#hud {
	float: right;
	width: 320px;
	height: 50px;
	padding: 10px;
	border: 1px solid #E1E1E1;
	background: #F3F3F3;
}

#hud li a {
	float: left;
	font-size: 12px;
	margin: 0 10px 0 0;
}

#utilities {
	clear: both;
	margin: 20px 0;
	overflow: auto;
	padding: 7px 10px;
	border: 1px solid #E1E1E1;
	background: #F3F3F3;
}

#breadcrumbs {
	float: left;
}
```

```
#header-search {
        float: right;
}
```

No es mucho, pero ayudará a que nuestro página tenga un aspecto bastante mejor:

Con esto habremos terminado con las bases de nuestro diseño, y como podemos ver, la dificultad de esta tarea está en proporción a la dificultad del diseño, de momento no hemos entrado en la parte de Magento, eso será el siguiente paso.

Page.xml

La creación del fichero page.xml es realmente el primer paso para convertir nuestro simple diseño html en un tema de Magento. Tal y como hemos comentado antes, sabemos que Magento utiliza ficheros XML para controlar la estructura de las páginas, así como definir que elementos se mostrarán.

Cada pantalla tendrá su propio fichero XML de layout, pero en general tenemos un fichero principal, **page.xml**, que define la estructura global. Y ese es el fichero en el que vamos a trabajar ahora mismo:

```
<?xml version="1.0"?>
<layout version="0.1.0">
   <default translate="label" module="page">
      <label>All Pages</label>
      <block type="page/html" name="root" output="toHtml" template="page/1column.phtml">
                  <block type="page/html_head" name="head" as="head">
         <action method="addCss"><stylesheet>css/estilos.css</stylesheet></action>
        </block>
```

```
        <block type="page/html_header" name="header" as="header">
                                        <block type="page/template_links" name="top.links" as="topLinks"/>
                                        <block type="page/html_breadcrumbs" name="breadcrumbs" as="breadcrumbs"/>
          <block type="core/template" name="top.search" as="topSearch"/>
        </block>

        <block type="core/text_list" name="content" as="content"/>

        <block type="page/html_footer" name="footer" as="footer" template="page/html/footer.phtml"/>
      </block>
    </default>
</layout>
```

Ahora que tenemos el fichero creado, vamos a ir por partes examinando el propósito de cada una de las líneas. Las primeras declaraciones de XML no son realmente importantes, hasta que llegamos a:

```
<block type="page/html" name="root" output="toHtml" template="page/1column.phtml">
          ...
</block>
```

Este es nuestro bloque principal, y, si miramos detenidamente, contiene al resto de bloques. Podríamos pensar en este bloque como en el div **id="wrapper"** que creamos en el fichero **index.html**

Además estamos indicando que utilice el fichero **1column.phtml** como template para nuestra página (este fichero lo crearemos más tarde).

De esta manera tenemos un bloque principal, que indica que fichero phtml utilizar, y que contiene al resto de bloques.

Siguiente paso:

```
<block type="page/html_head" name="head" as="head">
    <action method="addCss"><stylesheet>css/estilos.css</stylesheet></action>
</block>
```

Este bloque define que elementos se cargarán en el **head** de nuestra página. Podrían ser ficheros JS, CSS etc, en nuestro caso, indicaremos que es preciso cargar nuestro fichero **estilos.css**, y nada más. Después tenemos los bloques que se cargarán en el header de nuestra página:

```
<block type="page/html_header" name="header" as="header">
       <block type="page/template_links" name="top.links" as="topLinks"/>
      <block type="page/html_breadcrumbs" name="breadcrumbs" as="breadcrumbs"/>
```

```
        <block type="core/template" name="top.search" as="topSearch"/>
 </block>
```

Podemos ver un bloque principal **html_header**, y, dentro del mismo, otros bloques para cargar los links de la parte superior, los links de la miga de pan y el buscador. Más tarde prepararemos todo lo necesario para estos bloques, pero, aquí, es donde se realiza la llamada a los mismos.

¿Que es lo próximo que cargamos? El contenido principal de nuestro sitio:

```
<block type="core/text_list" name="content" as="content"/>
```

Y por último el pie de nuestro sitio:

```
<block type="page/html_footer" name="footer" as="footer" template="page/html/footer.phtml"/>
```

Aquí no solo indicamos que bloque es necesario cargar, si no el fichero **phtml** del mismo.

Importante recordar

Dos pequeños conceptos que pueden ser muy importantes:

- Diferenciar entre **html_head** y **html_header**, el primero carga los bloques que irán entre las etiquetas **<head></head>** de nuestro sitio web. El segundo cargará los bloques dentro de lo que nosotros definamos como header, por ejemplo ** **
- Hay que notar que no siempre especificamos el fichero .phtml a utilizar, por ejemplo en este caso:
  `<block type="page/html_footer" name="footer" as="footer" template="page/html/footer.phtml"/>`
  Estamos especificando que debe usarse footer.phtml, sin embargo en este caso:
  `<block type="page/template_links" name="top.links" as="topLinks"/>`
  No lo especificamos. Este es un concepto que podemos resumir en que, cuando el tipo (**type**) coincide con los especificados, por defecto, en Magento, se utilizan los bloques por defecto. Esto es muy cómodo para nosotros, pues no es necesario que creemos todos los ficheros **.phtml**, ya que en caso de que los existentes por defecto sirvan, se usaran esos.

Creando los ficheros .phtml

Este es nuestro siguiente paso, por un lado ya tenemos los ficheros de layout, que se encargan de indicar que ficheros .phtml deben ser cargados. Ahora tenemos que crear dichos ficheros, empezando por el **1column.phtml**

1column.phtml

Este fichero, ya que es nuestro fichero principal, simplemente cargará las diversas partes de nuestro sitio, sin entrar mucho en detalle, veamoslo:

```
<!DOCTYPE html>
<html lang="en-GB">
   <head>
   <?php echo $this->getChildHtml('head') ?>
   </head>

<body>
   <div id="wrapper" class="border">

   <?php echo $this->getChildHtml('header') ?>

   <div id="content" class="product">
   <?php echo $this->getChildHtml('content') ?>
   </div>

   <?php echo $this->getChildHtml('footer') ?>

   </div>
</body>
</html>
```

Muy simple este fichero, solo podemos remarcar las llamadas **getChildHtml**, que posicionaran los bloques que hayamos cargado en el layout.

head.phtml

Cuando en el fichero 1column.phtml vemos la llamada a **getChildHtml('head')**, estamos llamando al fichero head.phtml, ¿Que podemos encontrar dentro de dicho fichero? Veamoslo:

```
<title><?php echo $this->getTitle() ?></title>
<link rel="icon" href="<?php echo $this->getSkinUrl('favicon.ico') ?>" type="image/x-icon" />
<link rel="shortcut icon" href="<?php echo $this->getSkinUrl('favicon.ico') ?>" type="image/x-icon" />
<?php echo $this->getCssJsHtml() ?>
<?php echo $this->getChildHtml() ?>
<?php echo $this->getIncludes() ?>
```

Aquí podemos apreciar varias funciones de Magento, especialmente dos de ellas:

- getTitle -> que obtendrá automáticamente el título de nuestra página
- getCssJsHtml -> que cargará los ficheros CSS y JS

header.phtml

Nuestro siguiente fichero es header.phtml, con una estructura también sencilla:

```
<div id="header">
<div id="logo">
    <a href="<?php echo $this->getUrl('') ?>" title="<?php echo $this->getLogoAlt() ?>" class="logo"><img src="<?
php echo $this->getLogoSrc() ?>" alt="<?php echo $this->getLogoAlt() ?>" /></a>
</div>
<div id="hud">
    <h3>Bienvenido</h3>
    <?php echo $this->getChildHtml('topLinks') ?>
</div>
</div>

<div id="utilities">
    <?php echo $this->getChildHtml('breadcrumbs') ?>
    <?php echo $this->getChildHtml('topSearch') ?>
</div>
```

La estructura refleja la que creamos en index.html, con la diferencia del uso de las funciones **getChildHtml** de Magento.

footer.phtml

Este es el último de los bloques principales, e introduce algunas novedades, veamoslo:

```
<div id="footer" class="border">
<?php echo $this->__('Help Us to Keep Magento Healthy') ?> - <a href="http://www.magentocommerce.com/bug-
tracking"
onclick="this.target='_blank'"><strong><?php echo $this->__('Report All Bugs') ?></strong></a>
<?php echo $this->__('(ver. %s)', Mage::getVersion()) ?> <?php echo $this->getCopyright() ?></address>
</div>
```

La novedad es la llamada a las traducciones de las cadenas de texto, que se realiza con la función __()

links.phtml

Este va a ser el primero de los bloques secundarios que preparemos, y se encargará de mostrar los links de nuestra zona de menú. En nuestra página estática teníamos la siguiente estructura:

```
<ul class="links">
        <li><a href="#" title="Mi cuenta">Mi cuenta</a></li>
        <li ><a href="#" title="Mi lista de deseos">Mi lista de deseos</a></li>
        <li ><a href="#" title="Mi carrito">Mi carrito</a></li>
        <li ><a href="#" title="Finalizar">Finalizar</a></li>
        <li><a href="#" title="Salir" >Salir</a></li>
    </ul>
```

Este trocito de código se convertirá en el siguiente:

```
<?php $_links = $this->getLinks(); ?>
<?php if(count($_links)>0): ?>
<ul class="links"<?php if($this->getName()): ?> id="<?php echo $this->getName() ?>"<?php endif;?>>
  <?php foreach($_links as $_link): ?>
    <li<?php if($_link->getIsFirst()||$_link->getIsLast()): ?> class="<?php if($_link->getIsFirst()): ?>first<?php endif; ?><?php if($_link->getIsLast()): ?> last<?php endif; ?>"<?php endif; ?> <?php echo $_link->getLiParams() ?>><?php echo $_link->getBeforeText() ?>
      <a href="<?php echo $_link->getUrl() ?>" title="<?php echo $_link->getTitle() ?>" <?php echo $_link->getAParams() ?>><?php echo $_link->getLabel() ?></a> <?php echo $_link->getAfterText() ?>
    </li>
  <?php endforeach; ?>
</ul>
<?php endif; ?>
```

Este trocito de código da mucho juego, y es bastante diferente a lo que hemos visto hasta ahora. Primero llamamos al método **getLinks**, que devolverá un array que guardaremos en la variable **$_links**.

Preparamos un bucle que recorrerá todo el array, mostrando los enlaces, además, podemos ver funciones para averiguar si el link es el primero, o el último, y en cada caso añadir los estilos necesarios.

breadcrumbs.phtml

Este fichero va a ser muy similar al anterior, pero trabajando en la miga de pan, en lugar de los links del menú:

```
<?php if($crumbs && is_array($crumbs)): ?>
<div id="breadcrumbs">
    <?php foreach($crumbs as $_crumbName=>$_crumbInfo): ?>
      <?php if($_crumbInfo['link']): ?>
        <a href="<?php echo $_crumbInfo['link'] ?>" title="<?php echo $this->htmlEscape($_crumbInfo['title']) ?>"><?php echo $this->htmlEscape($_crumbInfo['label']) ?></a>
      <?php elseif($_crumbInfo['last']): ?>
```

```
            <strong><?php echo $this->htmlEscape($_crumbInfo['label']) ?></strong>
        <?php else: ?>
            <?php echo $this->htmlEscape($_crumbInfo['label']) ?>
        <?php endif; ?>
        <?php if(!$_crumbInfo['last']): ?>
            <span>» </span>
        <?php endif; ?>
    <?php endforeach; ?>
</div>
<?php endif; ?>
```

form.mini.phtml

Este va a ser el último bloque que construyamos, y es también de los más cortos:

```
<div id="header-search">
                    <form id="search_mini_form" action="<?php echo $this->helper('catalogsearch')->getResultUrl() ?>"
method="get">
    <input id="search" type="text" name="<?php echo $this->helper('catalogsearch')->getQueryParamName() ?>"
value="<?php echo $this->helper('catalogsearch')->getEscapedQueryText() ?>" class="input-text border" />

</form>
</div>
```

Recordar

Cuando llamamos a la función **getChildHtml('topSearch')**, para poder mostrar el bloque que cargamos en el layout, hay que tener en cuenta que el parámetro que pasamos en el método, **topSearch** en este caso, es el que encontraremos en el atributo as. En este caso lo veríamos así:

```
<block type="core/template" name="top.search" as="topSearch"/>
```

Finalizando nuestro tema, juntando las partes

Una vez que tenemos todos los ficheros creados, vamos a crear el tema, y a utilizarlo en nuestra instalación de Magento. Vamos a llamar a nuestro tema **cursomg**, así que crearemos las siguientes carpetas:

- proyecto-mg\skin\frontend\default\cursomg
  - proyecto-mg\skin\frontend\default\cursomg\css
- proyecto-mg\app\design\frontend\default\cursomg

- proyecto-mg\app\design\frontend\default\cursomg\layout
- proyecto-mg\app\design\frontend\default\cursomg\template

Iremos colocando los ficheros que hemos creado en esas carpetas, y nos quedará la siguiente estructura:

- proyecto-mg\skin\frontend\default\cursomg
  - proyecto-mg\skin\frontend\default\cursomg\css**estilos.css**
- proyecto-mg\app\design\frontend\default\cursomg
  - proyecto-mg\app\design\frontend\default\cursomg\layout**page.xml**
  - proyecto-mg\app\design\frontend\default\cursomg\template\page**1column.phtml**
  - proyecto-mg\app\design\frontend\default\cursomg\template\page\html**breadcrumbs.phtml**
  - proyecto-mg\app\design\frontend\default\cursomg\template\page\html**footer.phtml**
  - proyecto-mg\app\design\frontend\default\cursomg\template\catalogsearch**form.mini.phtml**
  - proyecto-mg\app\design\frontend\default\cursomg\template\page\html**head.phtml**
  - proyecto-mg\app\design\frontend\default\cursomg\template\page\html**header.phtml**
  - proyecto-mg\app\design\frontend\default\cursomg\template\page\template**links.phtml**

Es importante que situemos cada fichero en su lugar correcto, o de lo contrario Magento cargaría las versiones por defecto, lo que puede dar lugar a problemas y confusiones.

Con esto en marcha ya tenemos lo suficiente como para echar un vistazo a nuestra plantilla. Recordemos como se hacia esto:

- Accedemos al panel de gestión de nuestra instalación de Magento.
- Menu System -> Design
- Hacemos clic en la opción por defecto

- Una vez dentro desplegamos "Custom Design" y seleccionamos **cursomg**
- Aceptamos los cambios

Hay otra cosa que debemos tener en cuenta, si miramos nuestro page.xml veremos la siguiente línea:

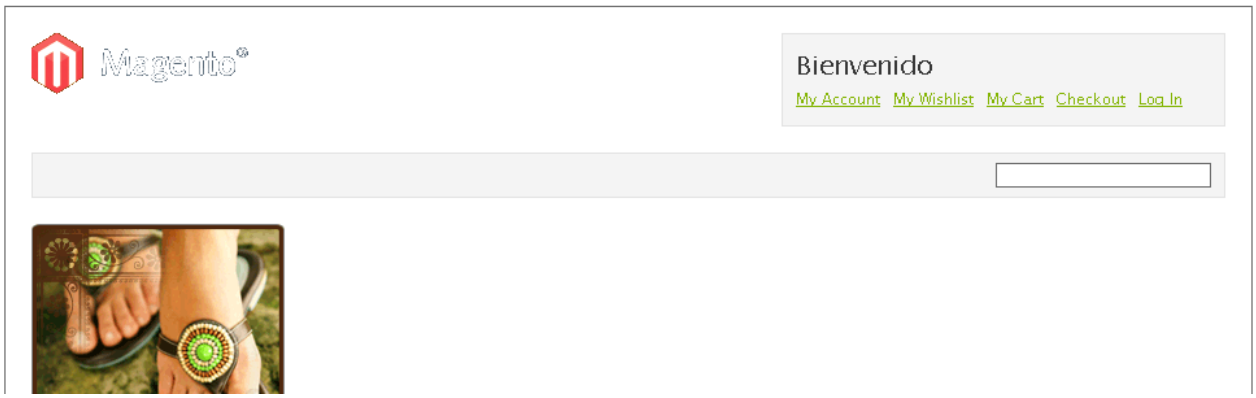
```
<block type="page/html" name="root" output="toHtml" template="page/1column.phtml">
```

En ella indicamos que se haga uso del fichero 1column.phtml, bien, ahora vayamos al menú **CMS -> Pages**, allí veremos una pantalla similar a la siguiente:

About Us	about-magento-demo-store	1 column
Customer Service	customer-service	3 columns
Enable Cookies	enable-cookies	1 column
Home page	home	2 columns with right bar
Home page	home	1 column
404 Not Found 1	no-route	2 columns with right bar

Esta página es un listado de las páginas estáticas de nuestro sitio web, con distinta información sobre la empresa etc

Aunque vemos dos páginas "**Home page**", solo la primera está habilitada, y tiene marcada la opción de "**2 columns with right bar**". Para la plantilla que hemos realizado, el fichero **phtml**, solo existe para la versión de una columna. Así que deberemos hacer clic en esa fila "**Home page**", una vez en la página de opciones ir a **Design**, y en el desplegable seleccionar **1 column**. Guardamos las opciones, y, si refrescamos la parte pública de nuestro sitio, tendrá el siguiente aspecto:



No es gran cosa de momento, pero es una buena base desde la que podemos ir trabajando e ir ampliando nuestro sitio web.

Siguiente paso, la vista de producto

De todas maneras, si os habéis fijado en el nombre de layout que hemos creado, habréis notado que se denomina **page.xml**, y, si en nuestro sitio web de prueba pinchamos en alguno de los productos, veremos como la página de deshace y no mantiene el estilo que habíamos creado.

Esto se debe a que, dependiendo del tipo de página se carga un layout u otro. Para la mayoría de páginas estáticas se utilizara el layout **page.xml**, pero para las páginas de producto se utilizará el layout **catalog.xml**, y este es el fichero que vamos a ir creando ahora.

proyecto-mg\app\design\frontend\default\cursomg\layout\catalog.xml

```
<?xml version="1.0"?>
<layout version="0.1.0">
    <catalog_product_view translate="label">
        <label>Catalog Product View (Any)</label>
        <!-- Mage_Catalog -->
        <reference name="root">
            <action method="setTemplate"><template>page/1column.phtml</template></action>
        </reference>
                        <reference name="head">
            <action method="addCss"><stylesheet>css/producto.css</stylesheet></action>
        </reference>
        <reference name="content">
            <block type="catalog/product_view" name="product.info" template="catalog/product/view.phtml">
                <block type="catalog/product_view_media" name="product.info.media" as="media"
template="catalog/product/view/media.phtml"/>
                <block type="catalog/product_view_description" name="product.description" as="description"
template="catalog/product/view/description.phtml"/>
                <block type="catalog/product_view_type_simple" name="product.info.simple" as="product_type_data"
template="catalog/product/view/type/simple.phtml"/>
                <block type="catalog/product_view" name="product.info.addtocart" as="addtocart"
template="catalog/product/view/addtocart.phtml"/>
            </block>
        </reference>
    </catalog_product_view>
</layout>
```

Vamos a estudiar las diferentes partes de este código. La primera línea importante que nos encontramos es la siguiente:

```
<catalog_product_view translate="label">
...
</catalog_product_view>
```

Este es el bloque principal, que contiene al resto de bloques y código. El fichero **catalog.xml** puede contener la estructura para diversos tipos de página, y con esta declaración de bloque estamos indicando que lo que hay dentro, es la especificación de la página de **vista de producto**.

Sigamos:

```
<reference name="root">
        <action method="setTemplate"><template>page/1column.phtml</template></action>
</reference>
```

Aquí estamos definiendo que template utilizar para el nivel de root, en este caso volvemos a utilizar el template **1column.phtml** que ya utilizamos anteriormente. Si no especificásemos nada se utilizaría el template que hubiésemos indicado en el fichero **page.xml** Aquí es redundante, pues, especificar el uso del template, pero lo indicamos para que posteriormente sea más sencillo de localizar, y por lo tanto de modificar.

```
<reference name="head">
   <action method="addCss"><stylesheet>css/producto.css</stylesheet></action>
</reference>
```

El siguiente paso es añadir los ficheros css necesarios para la hoja de producto. Y después:

```
<block type="catalog/product_view" name="product.info" template="catalog/product/view.phtml">
```

Especificamos el template a utilizar en la zona principal del detalle de producto. Y finalmente:

```
 <block type="catalog/product_view_media" name="product.info.media" as="media"
template="catalog/product/view/media.phtml"/>
 <block type="catalog/product_view_description" name="product.description" as="description"
template="catalog/product/view/description.phtml"/>
 <block type="catalog/product_view_type_simple" name="product.info.simple" as="product_type_data"
template="catalog/product/view/type/simple.phtml"/>
 <block type="catalog/product_view" name="product.info.addtocart" as="addtocart"
template="catalog/product/view/addtocart.phtml"/>
```

El resto de templates a utilizar. Con esto ya tenemos generado nuestro catalog.xml, el siguiente paso será, como ya vimos anteriormente, crear los templates necesarios para cada bloque.

Creando el template principal

Ahora vamos a ver el contenido que tendrá nuestro template principal, es decir el template que a su vez contendrá al resto de templates.

proyecto-mg\app\design\frontend\default\cursomg\template\catalog\product\view.phtml

```
<?php
   $_helper = $this->helper('catalog/output');
   $_product = $this->getProduct();
?>

<form action="<?php echo $this->getAddToCartUrl($_product) ?>" method="post" id="product_addtocart_form"<?
php if($_product->getOptions()): ?> enctype="multipart/form-data"<?php endif; ?>>
      <div class="no-display">
         <input type="hidden" name="product" value="<?php echo $_product->getId() ?>" />
         <input type="hidden" name="related_product" id="related-products-field" value="" />
<div id="main-product-image"><?php echo $this->getChildHtml('media') ?></div>

<div id="product-details">
<?php echo $this->getChildHtml('product_type_data') ?>
<?php echo $this->getChildHtml('addtocart') ?>
</div>

<h1><?php echo $_helper->productAttribute($_product, $_product->getName(), 'name') ?></h1>

<?php if ($_product->getShortDescription()):?>
               <h2><?php echo $this->__('Quick Overview') ?></h2>
               <p class="quick-overview"><?php echo $_helper->productAttribute($_product, nl2br($_product-
>getShortDescription()), 'short_description') ?></p>
<?php endif;?>

<?php echo $this->getChildHtml('description') ?>
</form>
</div>
```

Este código es muy similar al que vimos anteriormente, podemos ver como se van mostrando los distintos bloques haciendo uso del código **getChildHtml**. Es importante recordar que para poder utilizar el método getChildHtml previamente tendremos que haber cargado dichos bloques en el fichero de layout, en caso contrario no estarán disponibles. La primera parte importante que nos encontramos en el código es la siguiente:

```
<?php
```

```
    $_helper = $this->helper('catalog/output');
    $_product = $this->getProduct();
?>
```

Aquí básicamente lo que hacemos es seleccionar la información del producto a mostrar. Otro paso importante es el siguiente:

```
<form action="<?php echo $this->getAddToCartUrl($_product) ?>" method="post" id="product_addtocart_form"<?
php if($_product->getOptions()): ?> enctype="multipart/form-data"<?php endif; ?>>
```

Vemos como abrimos un nuevo formulario, y, utilizando el API de Magento, generamos la url que permitirá añadir el producto al carrito de la compra.

```
<h1><?php echo $_helper->productAttribute($_product, $_product->getName(), 'name') ?></h1>
```

Y seguimos utilizando las funciones de Magento, esta vez para recoger el nombre del producto, y mostrarlo.

```
<?php if ($_product->getShortDescription()):?>
                <h2><?php echo $this->__('Quick Overview') ?></h2>
                <p class="quick-overview"><?php echo $_helper->productAttribute($_product, nl2br($_product-
>getShortDescription()), 'short_description') ?></p>
<?php endif;?>
```

De manera similar recogemos la descripción corta del producto, y con esto finalizamos el trabajo con este template, veamos los siguientes.

Template Media

Que será utilizado para mostrar las imágenes del producto. Lo crearemos en:

proyecto-mg\app\design\frontend\default\cursomg\template\catalog\product\view\media.phtml

```
<?php
    $_product = $this->getProduct();
    $_helper = $this->helper('catalog/output');
?>

    <?php
        $_img = '<img id="image" src="'.$this->helper('catalog/image')->init($_product, 'image').'" alt="'.$this-
>htmlEscape($this->getImageLabel()).'" title="'.$this->htmlEscape($this->getImageLabel()).'" />';
```

```
    echo $_helper->productAttribute($_product, $_img, 'image');
?>
```

Nuevamente recogemos el producto y generamos la imagen a mostrar.

Disponibilidad y precio

Este nuevo template lo vamos a crear en el siguiente fichero:

proyecto-mg\app\design\frontend\default\cursomg\template\catalog\product\view\type\simple.phtml

```
<?php $_product = $this->getProduct() ?>
<div id="product-availability">Availability
<?php if($_product->isSaleable()): ?>
   <span class="available"><?php echo $this->__('In stock') ?></span>
<?php else: ?>
   <span class="unavailable"><?php echo $this->__('Out of stock') ?></span>
<?php endif; ?>
</div>
<div id="product-price">Precio <span><?php echo $this->getPriceHtml($_product) ?></span></div>
```

Este bloque se cargará con la llamada **getChildHtml('product_type_data').**

Añadir al carrito

Este es un bloque esencial, y que es cargado llamando al siguiente método getChildHtml('addtocart'), crearemos el siguiente fichero:

proyecto-mg\app\design\frontend\default\cursomg\template\catalog\product\view\addtocart.phtml

```
<?php $_product = $this->getProduct() ?>

<?php if($_product->isSaleable()): ?>
    <button title="<?php echo $this->__('Add to Cart') ?>" class="button btn-cart"><?php echo $this->__('Add to Cart') ?></button>
<?php endif; ?>
```

Solo nos queda un template que crear, el de la descripción detallada del producto.

Descripción del producto

Como hemos dicho es nuestro último template, vamos pues a crearlo:

proyecto-mg\app\design\frontend\default\cursomg\template\catalog\product\view\description.phtml

```
<?php $_description = $this->getProduct()->getDescription(); ?>
<?php if ($_description): ?>
    <h2>Product Description</h2>
        <div class="product-description"><?php echo $this->helper('catalog/output')->productAttribute($this->getProduct(), nl2br($_description), 'description') ?></div>
<?php endif; ?>
```

Fichero producto.css

Ahora que tenemos todos los bloques en su sitio, necesitamos crear nuestro fichero css, vamos a ello:

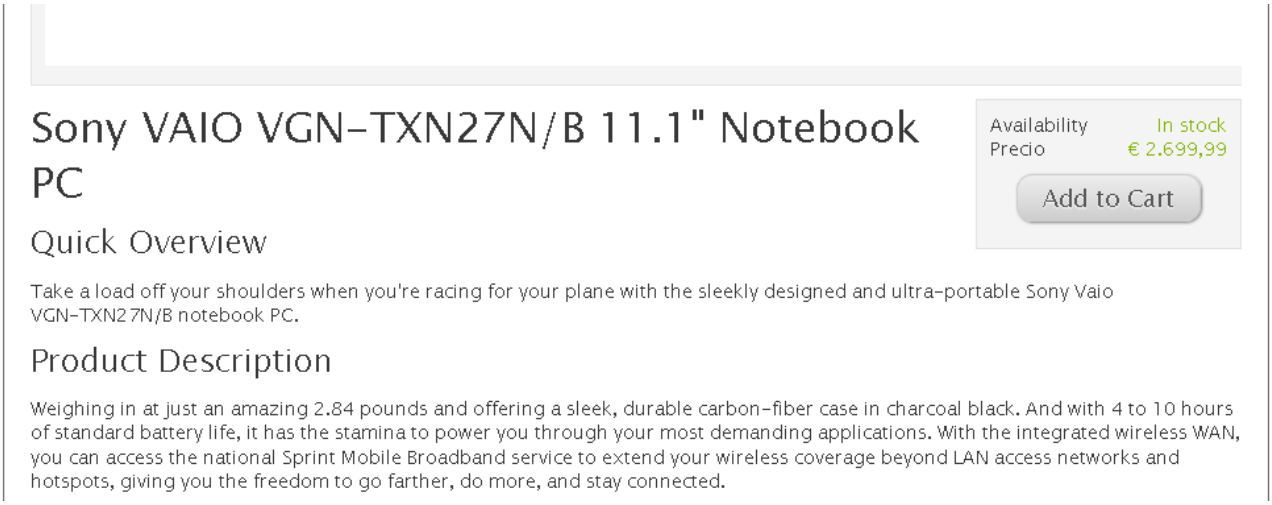
proyecto-mg\skin\frontend\default\cursomg\css\producto.css

```
/* Product page */

#main-product-image {
    margin: 0 20px 10px 0;
    padding: 10px;
    float: left;
    border: 1px solid #E1E1E1;
    background: #F3F3F3;
}

#product-details {
    width: 180px;
    padding: 10px;
    float: right;
    border: 1px solid #E1E1E1;
    background: #F3F3F3;
    margin: 0 0 0 20px;
}

#product-availability span.available, #product-price span {
    color: #7db000;
```

```
        float: right;
}

.button {
        margin: 10px auto;
        display: block;
        width: 140px;
        padding: 5px 10px;
        text-align: center;
        text-decoration: none;
        color: #555;
        border: 1px solid #ccc;
        font-size: 18px;
        background: #ddd;
        border-radius: 12px;
        -webkit-border-radius: 12px;
        -moz-border-radius: 12px;
        box-shadow: 1px 1px 2px rgba(0,0,0,.5);
        -webkit-box-shadow: 1px 1px 2px rgba(0,0,0,.5);
        -moz-box-shadow: 1px 1px 2px rgba(0,0,0,.5);
        text-shadow: #fff 0px 1px 1px;
        background: -webkit-gradient(linear, left top, left bottom, from(#eeeeee), to(#cccccc));
        background: -moz-linear-gradient(top,  #eeeeee,  #cccccc);

        }

.button:hover {
        background: #014464;
        background: -webkit-gradient(linear, left top, left bottom, from(#cccccc), to(#999999));
        background: -moz-linear-gradient(top,  #cccccc,  #999999);
        color: #000;
        }

.button:active {
        -moz-box-shadow: 0 2px 6px black;
        -webkit-box-shadow: 0 2px 6px black;
        }
```

Finalizando

Con todo esto ya tenemos listo todo lo necesario para que nuestras páginas de detalle de producto se muestren tal y como nosotros queremos. Vamos pues a ver como quedaría, más o menos como se pueden ver en la siguiente imagen:



Por supuesto solo hemos visto la punta del iceberg, ya que para generar todo el tema de Magento serán necesarios muchos más layouts y templates. La ventaja que tenemos es que, en caso de no existir, se usarán los layouts y templates por defecto.

Otra manera de crear nuestros temas - template blank

Como hemos podido comprobar, crear un template desde cero puede ser bastante laborioso, necesitando la creación de gran cantidad de ficheros, tanto de layout como bloques de template. Otra opción, se nos puede ocurrir, sería utilizar un tema ya existente y modificarlo acorde a nuestras necesidades.

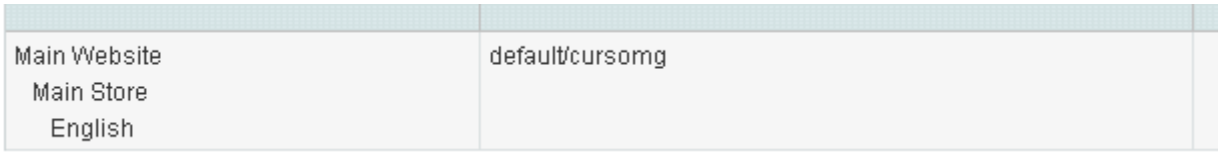
La mayoría de las veces esto sería un muy buen comienzo, ya que tendríamos toda la base creada. Pero ¿Que pasa si no encontramos el template adecuado? ¿Conviene realizar grandes modificaciones a un tema ya existente, buscando adaptarlo?

Es cierto por lo tanto que a veces la adaptación puede suponer un trabajo tan, o más, laborioso que crear un template desde cero.

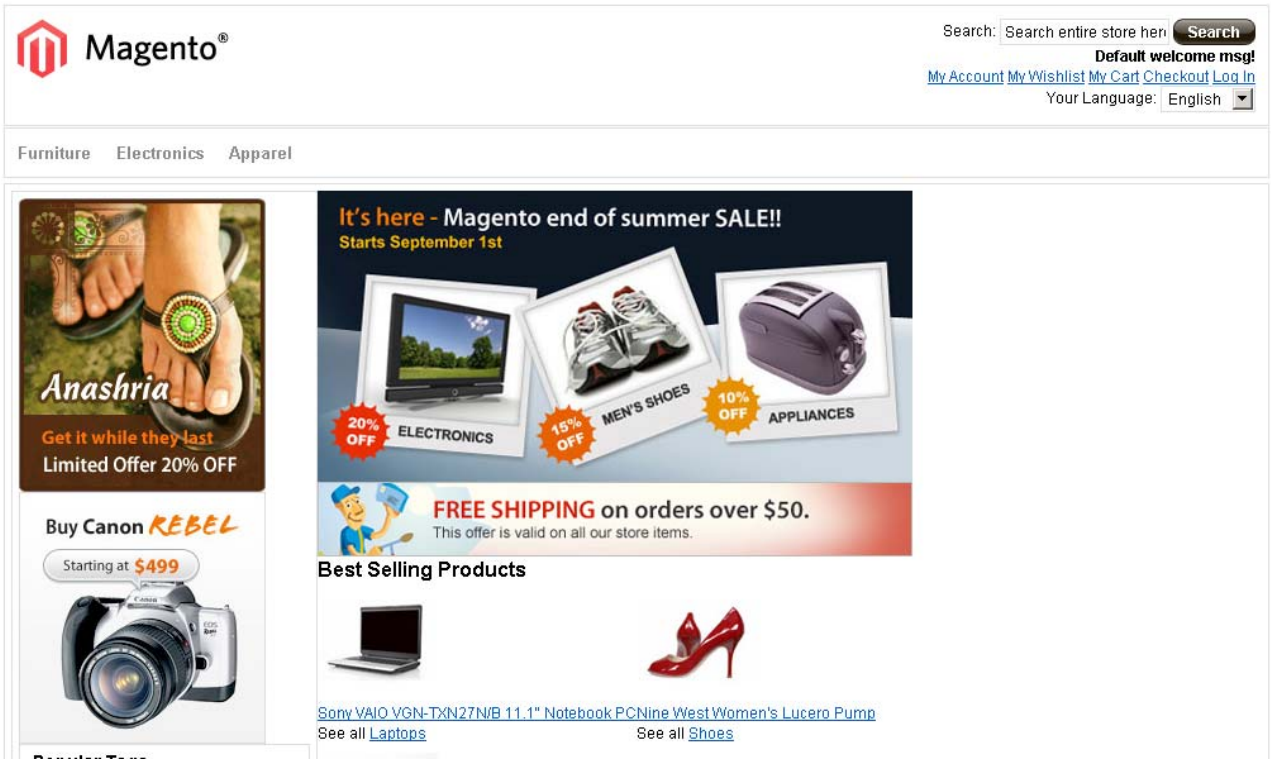
Pero tenemos una opción intermedia, se trata del tema **blank**, un tema creado para poder ser la base de cualquier tema que queramos hacer. Este tema lo podremos descargar fácilmente de Internet, ya que es muy popular (extension key **magento-community/Yoast_Blank_Seo_Theme)**.

Puede ser que la instalación de Magento de la que disponemos ya tenga un tema blank, ese lo mantendremos, pero el que descarguemos lo renombraremos a **blank2**, u otra cosa que queramos. Para instalarlo solo tendremos que copiar sus ficheros en nuestra instalación de Magento.

Una vez hecho lo cual iremos al **panel de administración** de nuestro Magento, menú **System** -> **Design**, seleccionamos la opción existente:



Una vez dentro cambiaremos el actual tema **cursomg** a **blank2** y guardaremos las modificaciones. Podremos ver que nuestro frontend ahora tiene el siguiente aspecto:



A partir de ahí tendremos una base que ir modificando y adaptando a nuestras necesidades, y es una base que no interfiere demasiado con lo que queramos hacer.

Jerarquía de temas

Durante las últimas versiones (1.3 y 1.4) se han efectuado cambios en Magento para facilitar la creación y mantenimiento de los temas. Esto es gracias a la Jerarquía de temas, vamos ahora a pasar a ver que es, y como funciona.

Pack Base

Esta es una de las características añadidas más interesantes. Se trata de un tema de Magento que podemos encontrar en:

- proyecto-mg\app\design\frontend\base\default
- proyecto-mg\skin\frontend\base\default

Este tema incluye todos los ficheros necesarios para el correcto funcionamiento de Magento. Es decir todos los layouts, templates, ficheros CSS básicos etc

No es recomendable modificar estos ficheros, ni crear nuestros temas dentro de la carpeta base.

¿Debemos pues copiar este tema base para crear los nuestros? No será necesario.

Prioridad en la carga de ficheros

Pensemos en nuestro tema, el que estábamos creando de ejemplo, está situado en:

- proyecto-mg\app\design\frontend\default\cursomg

Al navegar por nuestro sitio, Magento buscará, para cada página que necesite crear, los ficheros de layout, phtml ... Que necesite para crear la página. En caso de que no los encontrara los buscaría en:

- proyecto-mg\app\design\frontend\default\default

Y, en caso de que no esté en ninguna de esas carpetas lo buscaría en:

- proyecto-mg\app\design\frontend\base\default

Esto hace que crear templates sea muy sencillo, ya que realmente solo tenemos que crear los ficheros que vayan a ser diferentes a los existentes.

Por ejemplo, podemos imaginar que no necesitamos modificar nuestro fichero de layout **page.xml**, su estructura nos sirve y podemos adaptarla utilizando ficheros CSS. En este caso no es necesario que en nuestro tema incluyamos el fichero page.xml. Cuando Magento lo necesite lo buscará en **nuestro tema**, al no encontrarlo irá a **default**, y al no encontrarlo tampoco allí, lo buscará en **base**. Donde definitivamente lo encontrará.

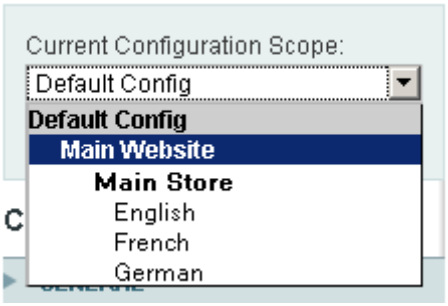
Con esto podemos resumir que:

- No necesitamos crear todos los ficheros, solo los que realmente tengamos que modificar.
- No debemos modificar los ficheros dentro del tema base, ya que pueden ser utilizados por otros temas.
- Tanto **base/default** como **default/default** son sobre-escritos durante las actualizaciones, por eso es mejor no trabajar en esas carpetas.

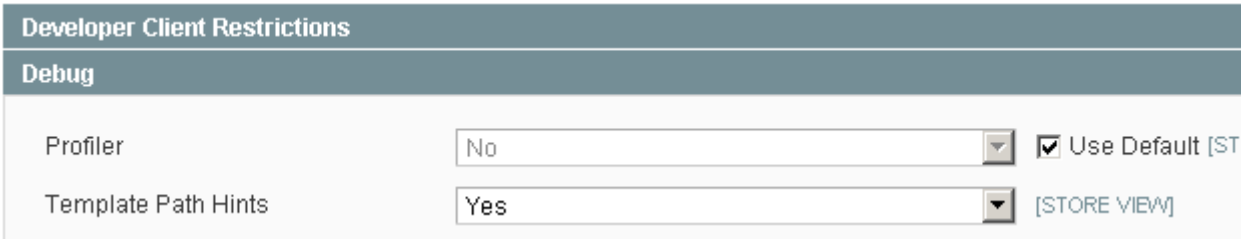
Ayudas a lo hora de crear los temas

Como hemos podido ver, crear un tema, aunque no es algo excesivamente complejo, si que involucra un gran número de ficheros, y en muchos casos es difícil saber que fichero buscamos o necesitamos modificar. Afortunadamente existen algunas ayudas para facilitarnos esta tarea.

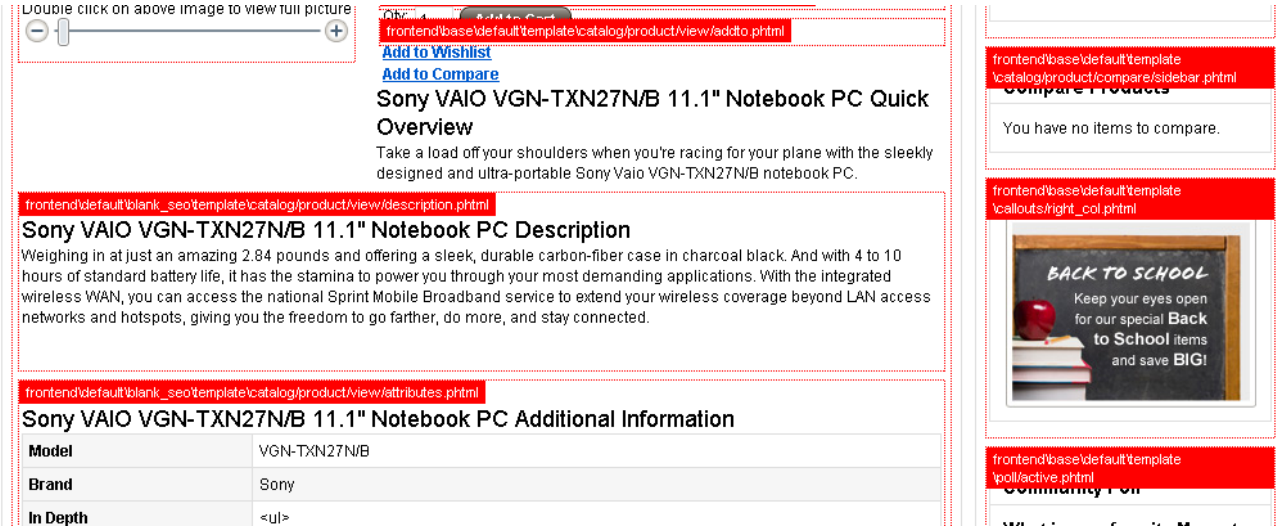
Veamos la más interesante de ellas, para ello debemos acceder a nuestro panel de administración. Allí iremos al menú **System** -> **Configuration**. El siguiente paso es seleccionar el ámbito de la modificación que vamos a efectuar. Por defecto estará seleccionado **default config**, lo que no nos sirve. Seleccionaremos **Main Website**:



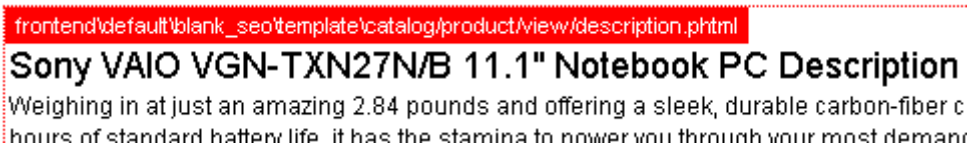
Una vez seleccionada esta opción iremos a la etiqueta **Advanced** -> **Developer** y abriremos la pestaña **Debug**, ahí habilitaremos la opción **Template Path Hints** y guardaremos los cambios.



Una vez hemos realizado este cambio refrescaremos el frontpage de nuestro sitio, y veremos algo similar a lo siguiente:



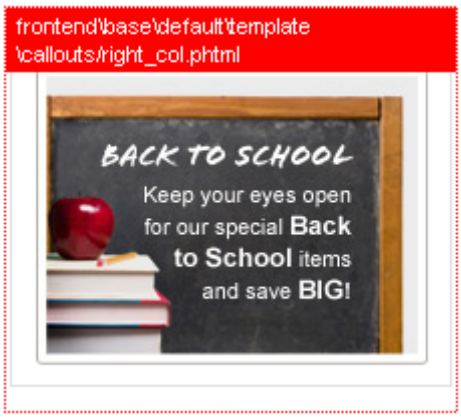
En rojo podemos ver como Magento nos está indicando la ruta hacia los ficheros que conforma los diferentes bloques, veamos algunos un poco más en detalle:



Este bloque se carga de **frontend** -> **default** -> **blank_seo** -> **template** -> **catalog** -> **product** ->

view -> description.phtml

Ese es uno de los ficheros que vienen incluidos en nuestra plantilla, y es ahí donde lo podríamos modificar.



Sin embargo este otro bloque se carga desde **frontend -> base -> default -> template -> callouts -> right_col.phtml**

Este fichero se carga desde base, tal y como comentábamos antes. ¿Que pasaría si copiásemos este fichero en **frontend -> default -> blank_seo -> template -> callouts -> right_col.phtml**?

Vamos a hacerlo, lo copiamos e introducimos alguna pequeña modificación, el resultado será el siguiente:

Vemos como, automáticamente, al estar disponible el fichero en el tema activo, Magento se encarga de utilizarlo en lugar de usar el de base. Vemos también como podemos fácilmente modificar el contenido de dicho fichero.

No olvidéis deshabilitar la cache, en caso contrario no veréis estos cambios.

Configuraciones básicas

Ahora que ya tenemos nuestra tienda instalada y con una apariencia gráfica seleccionada. Pasaremos a centrarnos un poco más en el panel de gestión privado, para saber como manejarnos con el y sacarle el mayor partido posible. Dentro de este apartado vamos a ver los siguientes puntos:

- Generales
- Categorías
- Productos
- Precios
- Pasarela de pago
- Gestión de envió
- Gestión de pedidos (Workflows)

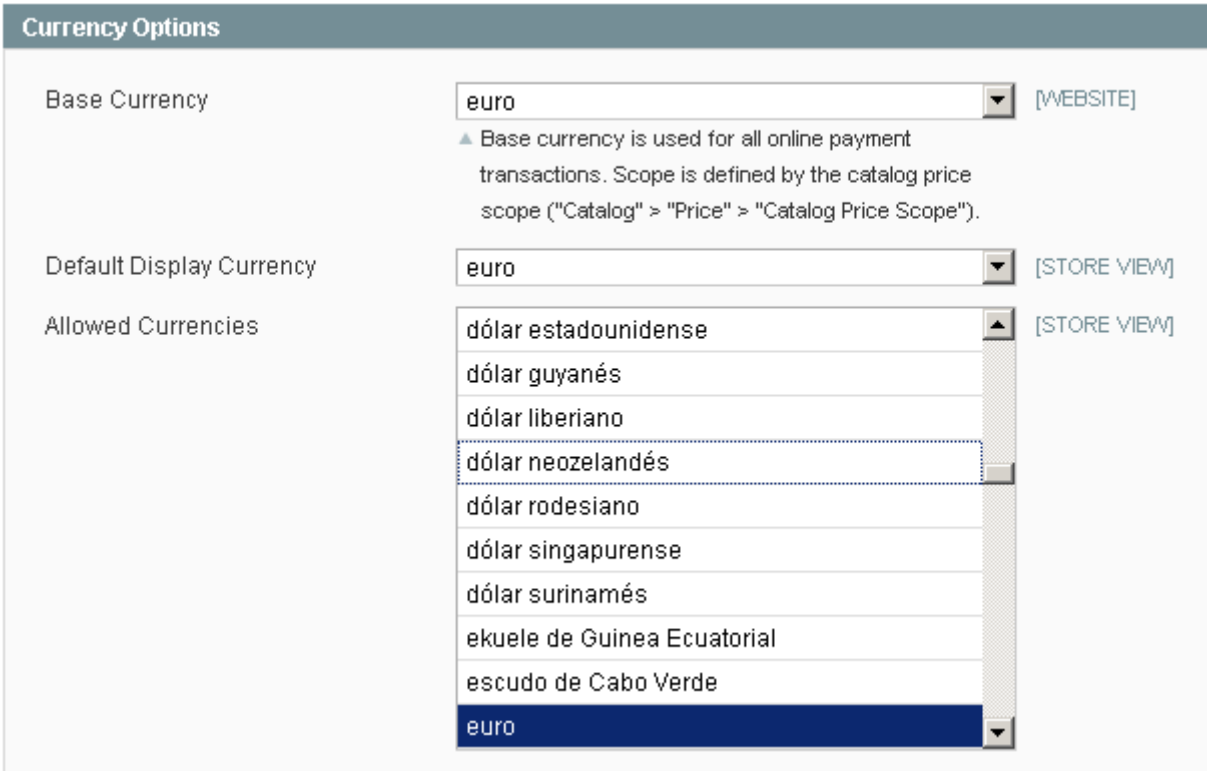
Vamos a ello.

Generales

En esta sección veremos como cambiar el **idioma** de Magento, así como la **moneda** por defecto que utilizará. Aunque estas opciones las seleccionamos durante la instalación es muy posible que necesitemos cambiarlas, veamos como se hace.

Vamos a ver primero donde cambiar la **Moneda** pues es muy sencillo. Accederemos pues a nuestro panel de gestión e iremos al menú **System** -> **Configuration**, luego en la pestaña **General** iremos a **Currency Setup**.

Desplegaremos Currency Options, y veremos una pantalla similar a la siguiente:



Desde aquí podemos seleccionar tanto la Moneda de base, que es la que se usará para los pagos. La que se visualiza por defecto, y las permitidas. Es tan sencillo como modificarlo según nuestras necesidades, guardando las modificaciones después.

Idioma

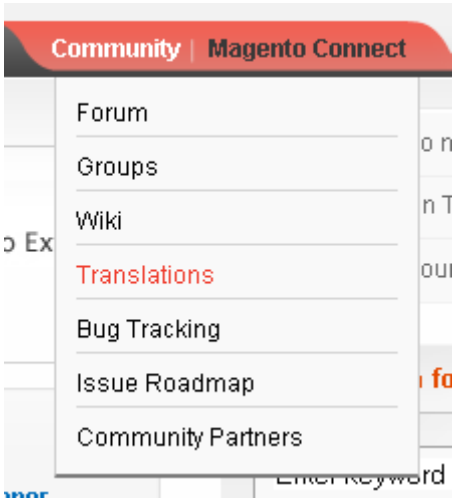
La configuración del idioma requerirá de algunos pasos más, por defecto la instalación de Magento viene con un solo idioma, **inglés de US**. Aunque durante la instalación hayamos seleccionado Español como el idioma por defecto, nuestra instalación no podrá mostrar el idioma, pues no lo tiene. ¿Entonces de que sirve esta selección inicial? De momento solo para que, si miramos el código fuente de nuestra tienda, veamos esto:

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="es" lang="es">
```

Esta declaración indica que nuestro sitio está en Español, aunque esto luego no sea así. *¿Que necesitamos entonces?* Básicamente descargarnos un idioma nuevo, y copiarlo en nuestra instalación de Magento.

Bien primero nos dirigiremos a:

http://www.magentocommerce.com



En el menú Community seleccionaremos Translations, eso nos llevará a una pantalla similar a la siguiente:

Spanish (Peru)	0.55%	totojo	Select
Spanish (Spain)	87.72%	jesteve, insaix, César Gómez	Select
Spanish (Venezuela)	1.27%	Giannino	Select
Swahili (Kenya)	0.52%	boundarycross	Select

Con un listado de los idiomas disponibles, así como su porcentaje de completado. Español de España tiene un porcentaje de finalización del 87,72 lo que nos indica que está traducido en su práctica totalidad. Seleccionamos el idioma, lo que nos llevará a la siguiente pantalla:

Language	Translation Method	Status	Maintainer	Download
Spanish (Spain)	Inline | String list	87.72%	jesteve, insaix, César Gómez	Package

De las tres opciones, Inline, String list y **Package** seleccionaremos la última. Lo cual producirá la descarga de un fichero llamado **es_ES.zip**. Al descomprimirlo veremos que tiene la siguiente estructura:

- app
  - design
    - frontend
      - default
        - default
        - modern

- locale
  - es_ES

Podemos ver como por un lado tenemos la carpeta **design** y por otro la carpeta **locale** . Dentro de la carpeta design incluso vienen dos templates, con las traducciones de los mismos. Y en la carpeta locale están las traducciones generales para nuestra tienda, así como para el **panel de administración**.

Si vamos a la carpeta app\locale\es_ES y abrimos cualquiera de los ficheros CSV, dentro veremos una estructura similar a la siguiente:

```
"Date selection:","Selección  de fecha:"
```

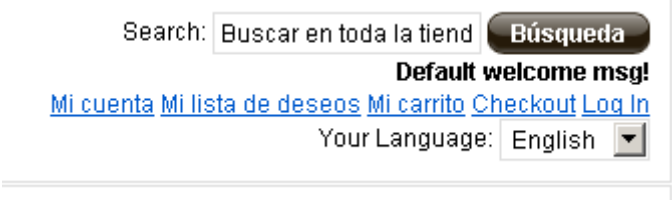
Una frase por linea, en la parte izquierda, entre comillas, el texto original, en la parte derecha la traducción. ¿Como utiliza Magento estos textos? Veamoslo con un ejemplo:

```
<?php echo $this->__('Free Shipping on orders over 50$'); ?>
```

Utilizando el método __() . Se busca una traducción para la frase pasada como parámetro, de no encontrar traducción se mostrará la frase tal cual.

Es recomendable utilizar una herramienta que permita buscar texto entre múltiples ficheros, con tal de facilitar la localización de la frase a traducir (Por ejemplo FreeCommander).

Bien, con esto hemos visto la básico, para tener el lenguaje disponible debemos copiar toda la carpeta app y pegarla en nuestra instalación de Magento, con lo cual el lenguaje estará disponible. Si durante la instalación seleccionamos Español como el idioma principal, automáticamente veremos traducciones tanto en el frontend:



Como en el panel de administración:

Si durante la instalación no seleccionamos Español como el idioma por defecto lo haremos ahora. Iremos a **System** -> **Configuration** -> **General** -> **Locale Options** y Seleccionaremos **Local** -> **Español (España)** y guardaremos la configuración.

El idioma del panel de administración se puede cambiar desde el pie de página del administrador, en un menú similar al siguiente:

Categorías

Las categorías de nuestra tienda serán las herramientas que nos permitirán organizar los productos adecuadamente. Veamos un poco más sobre las categorías, en nuestro panel de control iremos al menú **Catálogo** -> **Gestionar las categorías**.

Todavía no vamos a entrar en el concepto de multitiendas, de momento solo comentaremos que es posible asignar categorías a diferentes vistas de nuestra web, y que solo aparecerán en esa vista en concreto. La selección de la vista se hace desde el siguiente desplegable:

De momento vamos a mantener la selección de "Todas las vistas de tienda". Lo siguiente que podemos hacer es, o bien crear una **categoría padre** o crear una **subcategoría**. Para crear una subcategoría primero deberemos hacer clic para seleccionar a que categoría padre pertenecerá.

Creación de una categoría

Crear una categoría es un proceso rápido y sencillo, una vez nos ponemos a ello veremos cuatro pestañas:

- **Información general**, desde aquí podremos configurar los siguientes parámetros:
  - Nombre de la categoría
  - Descripción de la categoría, en esta versión incluso contaremos con un editor wysiwyg que nos permitirá que los textos de la categoría tengan un mejor aspecto, sin necesidad de saber html.
  - Imagen, que nos permitirá añadir una imagen de categoría
  - Título de la página
  - Palabras clave (Meta Keywords)
  - Descripción SEO (Meta Description)
  - Si esta activa
  - Si aparece en el menú
  - URL de la categoría (aquí definiremos solo el trocito perteneciente a la categoría)
- **Display settings**, desde aquí podremos configurar como se visualizará la categoría, tendremos las siguientes opciones:
  - Modo de visualización, aquí podemos elegir si, al entrar en la categoría, el visitante verá un listado de productos, un bloque de html estático, o una mezcla de ambos.

    - CMS block, nos permitirá seleccionar el bloque CMS entre los que tengamos ya creados
    - Is anchor, esta es una interesante opción, lo que hace es, cuando entramos en una categoría, ademas de mostrar los productos de la misma, mostrará los productos de las subcategorías pertenecientes a la categoría principal.
    - La siguiente opción nos permite seleccionar las opciones de filtrada que habrá disponible.
    - Y la siguiente opción nos permite seleccionar el filtro que se aplicará por defecto.
- **Diseño personalizado**, aquí encontraremos diferentes opciones que nos permitirán configurar el aspecto gráfico de nuestra categoría.
    - Diseño especifico para la categoría, lo que nos permitirá seleccionar un tema distinto para esta categoría. Lo podremos seleccionar entre todos los que tengamos instalados.
    - La siguiente opción nos permitirá definir como se aplicará este diseño.
    - Desde que fecha estará activo
    - Hasta cuando estará activo
    - El layout a utilizar
    - O incluso insertar nuestro propio código
- **La última pestaña nos permite ver los productos asignados a esta categoría.**

Productos

Una vez tenemos nuestra estructura de categorías preparada, el siguiente paso es insertar nuestros productos. Esto lo haremos desde **Catálogo** -> **Gestionar los productos**. Lo primero que vemos al llegar es un listado de todos los productos de nuestra tienda.

Desde este listado podremos filtrar los productos, editarlos o borrarlos, también podremos añadir un nuevo producto:

Una vez hacemos clic en ese botón Magento nos llevará a otra pantalla donde podremos seleccionar el conjunto de **atributos**, y el **tipo de producto**:

Vamos a ver primer los tipos de producto.

Tipos

Existen varios tipos de producto, lo que de hecho nos permite vender casi cualquier tipo de producto. Veamos que opciones tenemos:

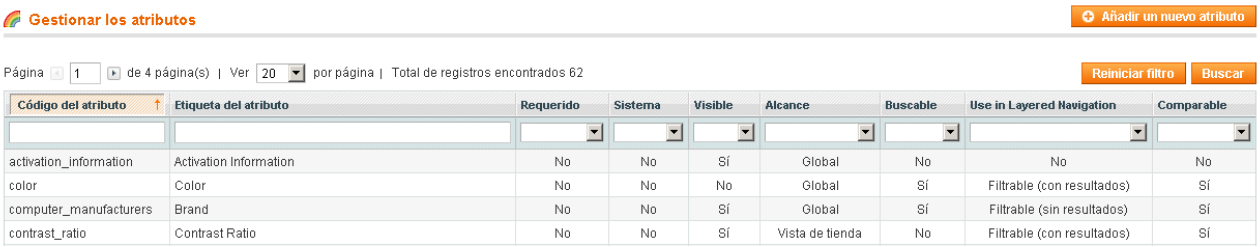
- **Producto simple** -> la opción más básica, sin opciones ni configuraciones. Este tipo de productos se venden tal cual.
- **Producto agrupado** -> en este caso, estamos juntando varios productos simples, y vendiéndolos como un único producto. No se puede modificar el precio, para que sea menor que al comprarlos por separado.
- **Productos configurables** -> esta opción es ideal para productos que tengan variaciones, ya sea de talla, color etc
  Para poder crear productos configurables primero deberos crear los necesarios productos simples, uno por cada opción. Creándolos de manera que no sean visibles en el catalogo. Después, cuando tengamos todas las opciones creadas, procederemos a generar el producto configurable, asociándolo a los otros productos simples. Del resto se encargará Magento.
- **Producto virtual** -> aquí puede haber un poco de confusión con los productos descargables. Un producto virtual no es un producto descargable, ni de ningún otro tipo, el comprador no recibirá nada. Esto es útil para vender licencias, u otros servicios.
- **Paquete de productos** -> esto nos permite agrupar varios productos, que a su vez pueden ser productos configurables.
- **Producto descargable** -> con esta opción podremos vender ficheros de música, fotografías, software etc.

Atributos

Vamos ahora a comentar un poco sobre los atributos. Los conjuntos de atributos nos permiten configurar todavía más nuestros productos. Podemos tener atributos que definan no solo el color, peso o tamaño, podríamos tener atributos que incluso definieran los megapixels (útil para una cámara de fotos), el material del que está hecho el producto y un largo etc de opciones.

Todos estos atributos se pueden agrupar en conjuntos de atributos , de manera que, a la hora de crear nuestros productos, podamos seleccionar fácilmente el conjunto de atributos más apropiado.

Si vamos al menú **Catálogo** -> **Atributos** -> **Gestionar los atributos**, podemos ver todos los atributos que tenemos creados, así como crear nuevos atributos.

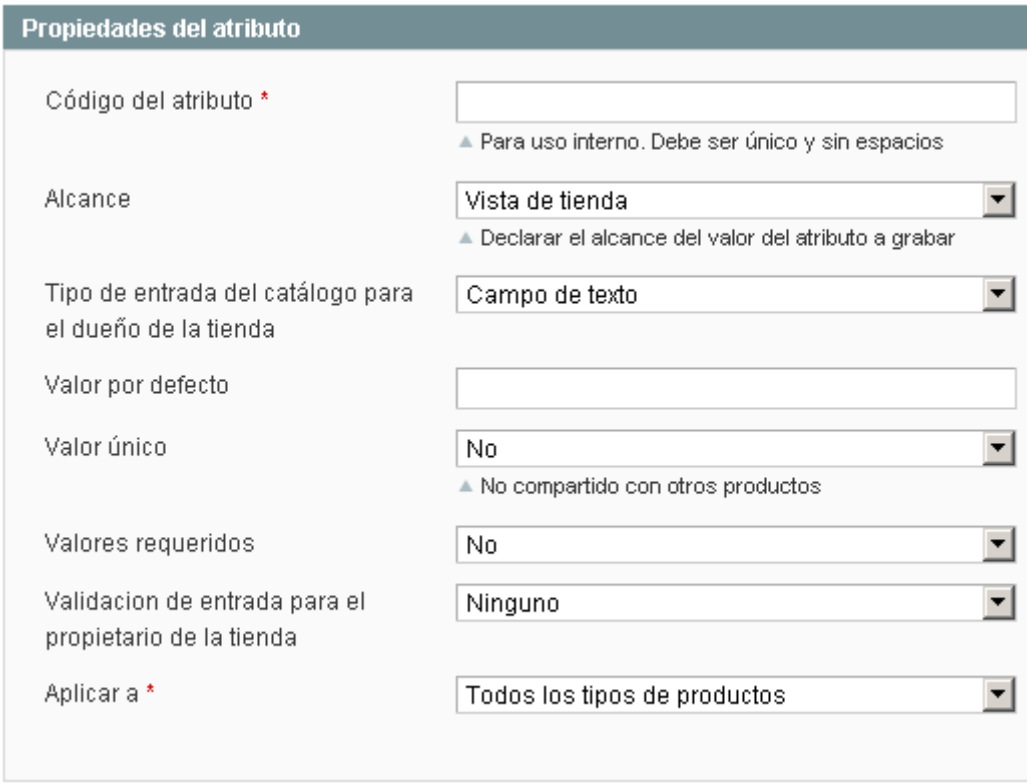


Código del atributo	Etiqueta del atributo	Requerido	Sistema	Visible	Alcance	Buscable	Use in Layered Navigation	Comparable
activation_information	Activation Information	No	No	Sí	Global	No	No	No
color	Color	No	No	No	Global	Sí	Filtrable (con resultados)	Sí
computer_manufacturers	Brand	No	No	Sí	Global	Sí	Filtrable (sin resultados)	Sí
contrast_ratio	Contrast Ratio	No	No	Sí	Vista de tienda	No	Filtrable (con resultados)	Sí

Si deseamos borrar alguno de los atributos, podemos hacerlo entrando en su pantalla de edición. Desde ahí veremos un nuevo botón que nos permitirá borrar el atributo.

Creación de un atributo

Al crear un nuevo atributo veremos una pantalla con una gran cantidad de opciones, vamos a echar un vistazo rápido a algunas de ellas:



- El código del atributo será el nombre del mismo, tal y como lo veremos más tarde.
- El alcance del atributo permite seleccionar donde estará visible el mismo.
- Y el tipo de atributo, ya sea un texto, un checkbox, un desplegable etc.
- En valor por defecto podemos establecer exactamente eso, el valor que tendrá el atributo en

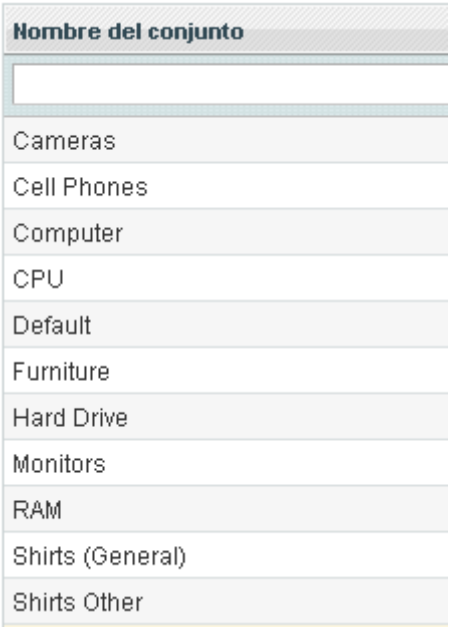
caso de que no lo modifiquemos.

- Valor único es un campo interesante, que nos permite establecer si el valor que tenga este atributo no se podrá repetir en ningún otro producto.
- Requerido establece si es necesario rellenar este campo, para poder crear el producto.
- Validación del valor introducido
- A que productos se aplica el atributo.

Además de estas opciones, tenemos otra pestaña que nos permite establecer las opciones de visualización, filtrado y navegación.

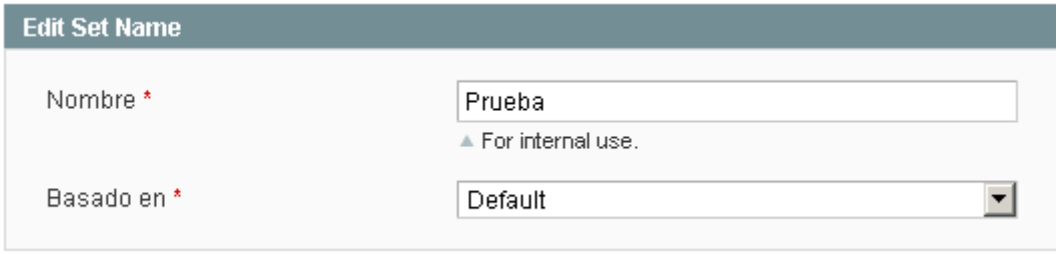
Conjuntos de atributos

Los conjuntos de atributos nos permiten agrupar los atributos, para que podamos utilizarlos cuando sea necesario. Vayamos a **Catalogo** -> **Atributos** -> **Gestionar los conjuntos de atributos**

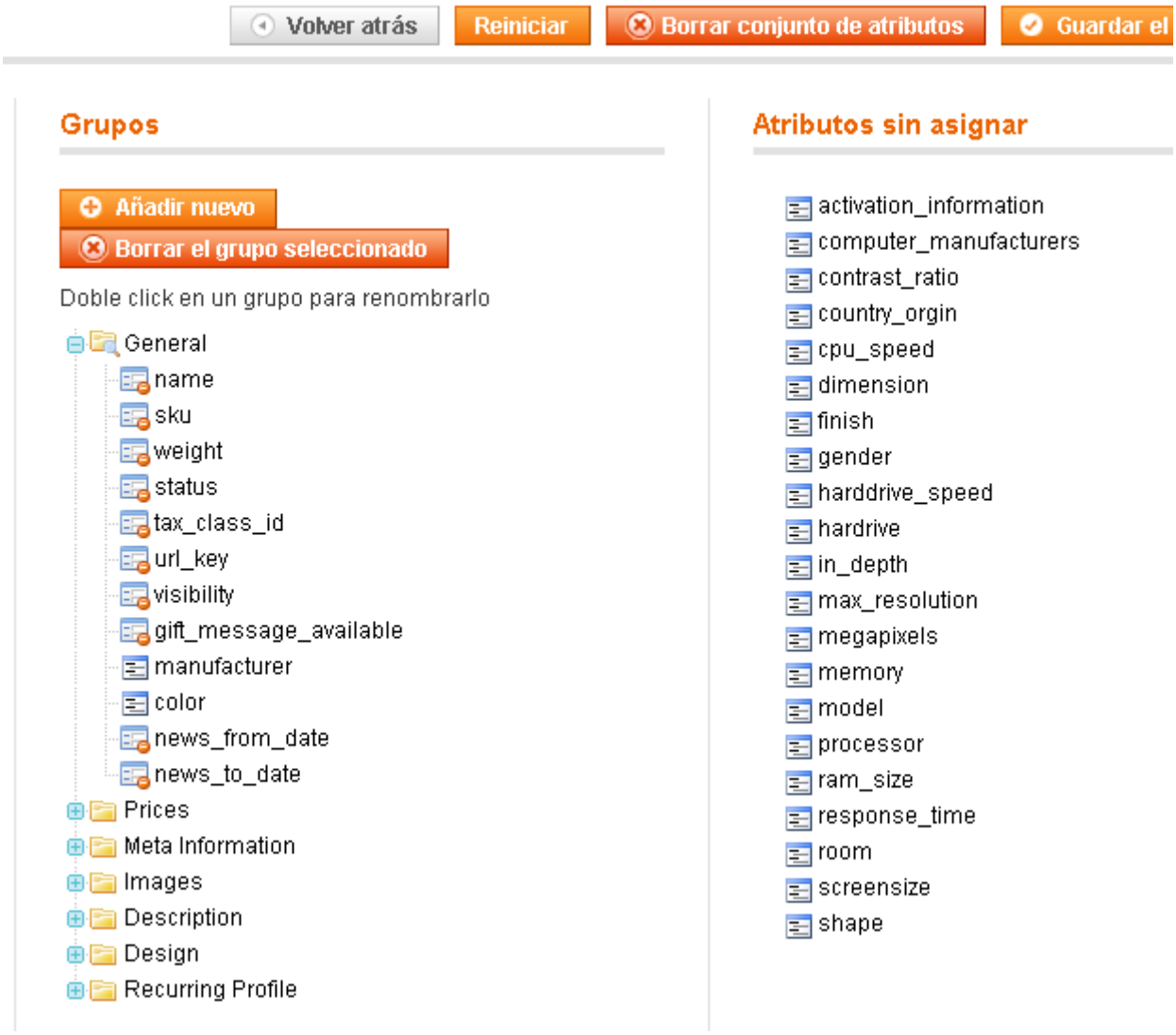


Aquí podemos ver algunos de los conjuntos existentes actualmente, cada uno de ellos con su conjunto de atributos. Los atributos pueden pertenecer a uno o mas conjuntos de atributos. A la hora de crear un nuevo conjunto de atributos daremos los siguiente pasos, primero haremos clic en el siguiente botón:

Eso nos llevará a una pantalla que nos permitirá darle un nombre al conjunto, y decidir en que conjunto estará basado, por ejemplo en **Default**:



Una vez hecho eso, guardaremos el conjunto de atributos haciendo clic en el botón **Guardar el conjunto de atributos**. La pantalla final de edición del conjunto es como la que sigue:

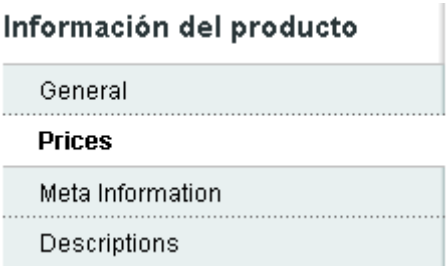


En la columna izquierda podemos ver los grupos dentro del conjunto de atributos. Estos grupos coinciden con las diferentes configuraciones de un producto. Podemos asignar los atributos que vemos a la derecha a cualquiera de estos grupos. Una vez hecho esto solo nos queda guardar el conjunto de atributos, y utilizarlo cuando lo necesitemos.

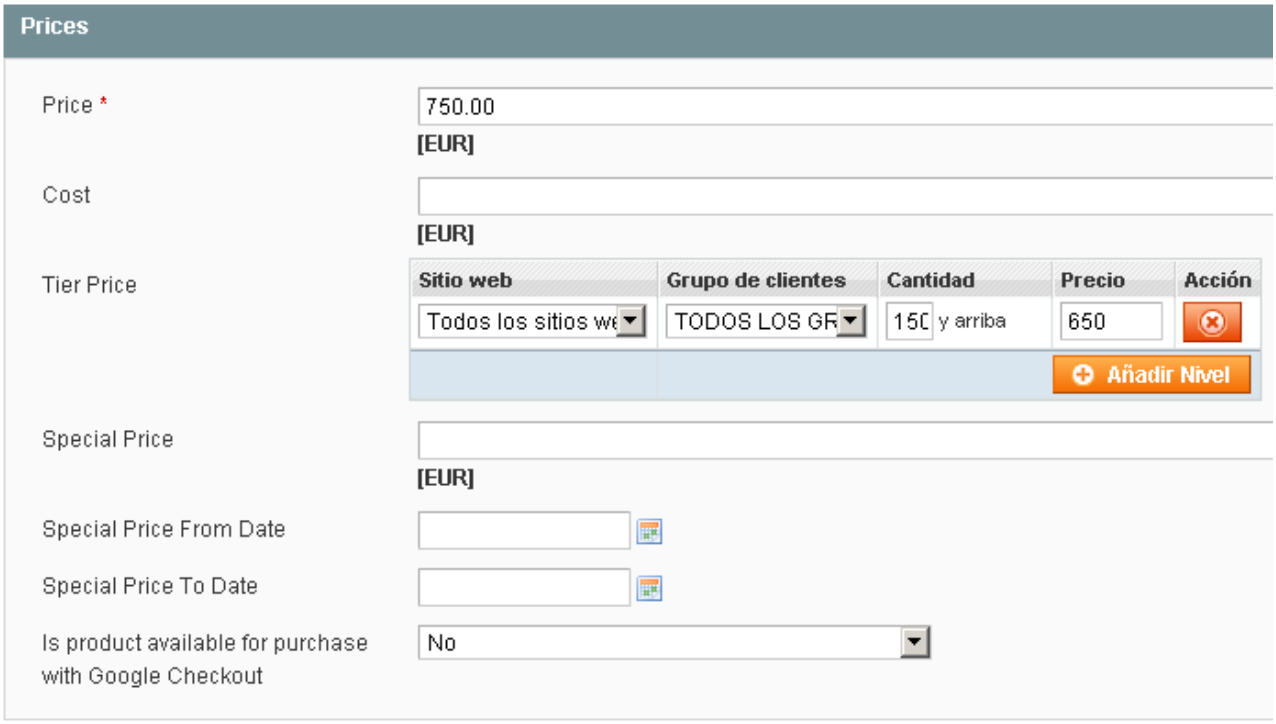
Precios

Por supuesto todos los productos y servicios que aparezcan en nuestra tienda tienen que tener un precio, nuestro siguiente objetivo será ver donde podemos añadir y configurar estos precios. Iremos a **Catálogo** -> **Gestionar los productos**, una vez ahí o bien creamos un producto nuevo o editamos uno de los existentes.

Accederemos a la pestaña **precios**:



La pantalla de gestión de precios es la siguiente:



Y cuenta con las siguientes opciones:

- Precio -> este es el precio base del producto, y el que aparece por defecto.
- Coste -> coste del producto, en caso de que dispongamos de el
- Tier Price -> en este caso podemos establecer descuentos en el precio del producto una vez el cliente ha alcanzado cierta cantidad de gasto. Por ejemplo si compra más de 1500 € podemos hacer que este producto en lugar de costar 750 le cueste 650, promocionando así las ventas.

- Special Price -> este es el precio de descuento del producto, que sirve para establecer ofertas.
- Fecha de inicio de la oferta
- Fecha de fin de la oferta
- Y si el producto está disponible para su compra en Google checkout

Pasarelas de pago

Ahora que tenemos tanto los productos, como sus precios, es el momento de dar a nuestros clientes la posibilidad de pagar. Encontraremos varias opciones ya disponibles para ello, pero también podremos instalar más si es lo que necesitamos.

Opciones por defecto

Veamos primero las opciones que están disponibles nada más instalar Magento. Para ello iremos a **Sistema** -> **Configuración** y bajaremos hasta la pestaña **Ventas**. Seleccionaremos **Métodos de Pago**, ahí veremos las siguientes opciones:

- **Guardado CC** -> esta opción guarda en nuestra instalación de Magento los datos de la tarjeta de crédito del cliente. Posteriormente, nosotros como dueños de la tienda, haremos el cargo a la tarjeta de crédito.
- **El subtotal de la caja es cero** -> con esta opción permitimos a los clientes realizar el pago cuando el coste total es cero.
- **Check / Money order** -> envío de un cheque en pago del pedido.
- **Purchase order** -> en este caso tampoco se realiza un pago, solo queda la orden de pedido. Serviría para clientes habituales, con los que tengamos acordados otros métodos de pago.
- **Authorize.net** -> pasarela de pago para pagos con tarjeta de crédito.

Todas estas opciones las podemos habilitar o deshabilitar según nos convenga. Además de estas opciones, si seguimos mirando en la pestaña de Ventas, veremos alguna más:

- **PayPal** -> desde esta opción podemos configurar todos los parámetros necesarios para este conocido método de pago.
- **Moneybookers** -> sistema de pago similar a paypal, con soporte para una gran cantidad de países.

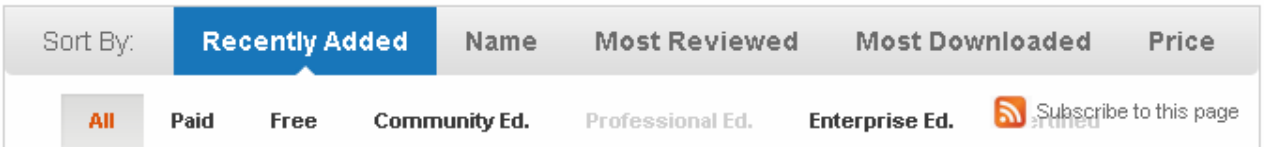
Pasarelas vía Magento Connect

Con todo esto, si todavía necesitamos otro método de pago, lo podemos buscar en Magento Connect. Para ello volveremos a:

http://www.magentocommerce.com/

Y haremos clic en el enlace "**Magento Connect**", lo que nos llevará, de nuevo, al centro de extensiones de Magento. En el menú de la derecha, iremos a la opción Payments & Gateways.

Una vez llegamos a la página destino, tendremos diferentes opciones de filtrado y ordenación, como podemos ver en la siguiente imagen:



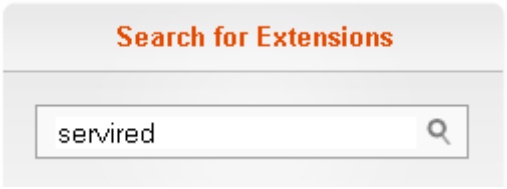
Por un lado podemos ordenar las extensiones por:

- Las que han sido recientemente añadidas
- Por nombre
- Las más valoradas
- Más descargadas
- O por precio

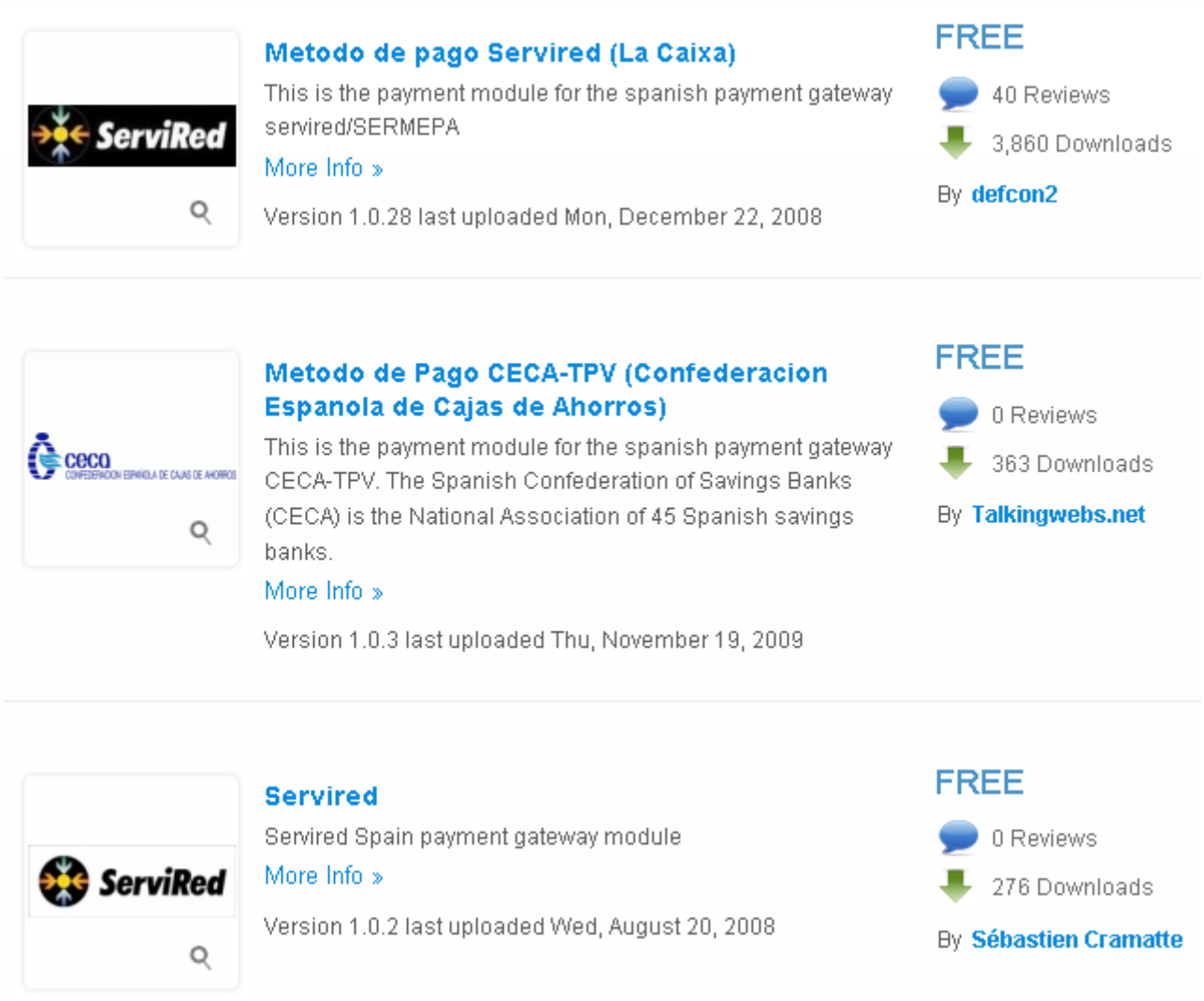
Además podemos aplicar diversos filtros:

- Mostrar todos
- Extensiones de pago
- Extensiones gratuitas
- Extensiones para la versión "Community Edition" de Magento
- Extensiones para la versión "Professional Edition" de Magento
- Extensiones para la versión "Enterprise Edition" de Magento

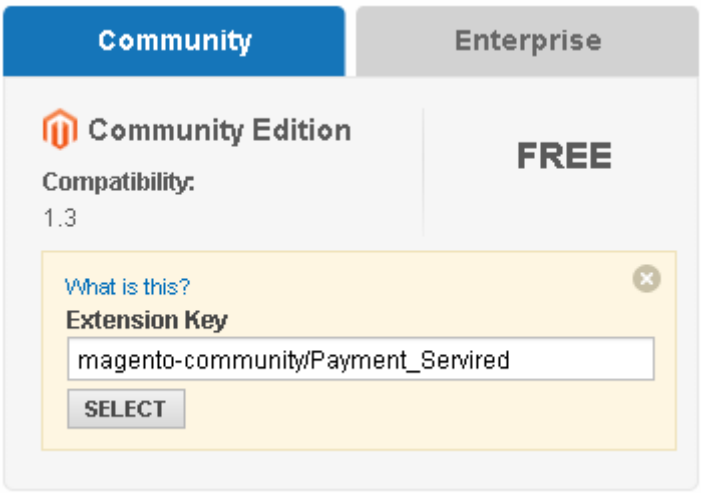
Con todas estás opciones tendremos mucho más fácil encontrar la extensión que mejor sirva a nuestras necesidades. En caso de que conozcamos el nombre de la extensión que estamos buscando, podemos utilizar el buscador. Vamos a probar introduciendo el termino **servired**:



El resultado nos mostrará diversas opciones, que podremos examinar haciendo clic en ellas:



Una vez tengamos la que queremos, bastará con que copiemos su código de extensión:



El proceso de instalación será idéntico al que utilizamos para instalar el tema gráfico.

Gestión de pedidos (Workflows)

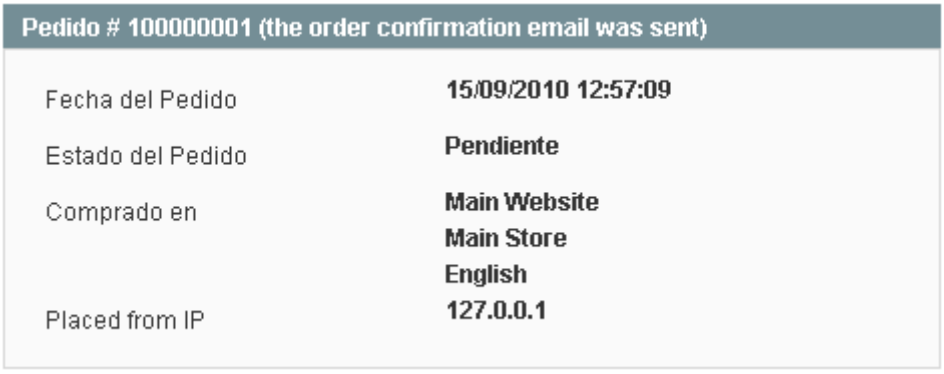
Bien, en estos momentos ya tenemos todo listo para que nuestra tienda funcione, nuestros clientes pueden visitar nuestro sitio, ver nuestros productos, seleccionarlos y pagarlos, finalizando la compra.

¿Que sucede cuando los clientes realizan los pedidos, y estos son registrados en nuestra tienda? Una vez tengamos pedidos los podremos ver en el panel de administrator de nuestra tienda, navegaremos al menú **Ventas** -> **Pedidos**.

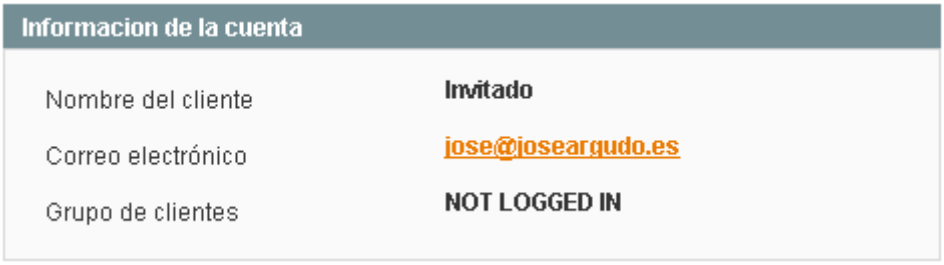
Ahí veremos una pantalla similar a la siguiente:

En la que tendremos listados todos los pedidos, así como filtrarlos y ordenarlos. Si hacemos clic en cualquiera de los pedidos accederemos a su pantalla de detalle. Donde podremos ver un buen número de información referente a dicho pedido.

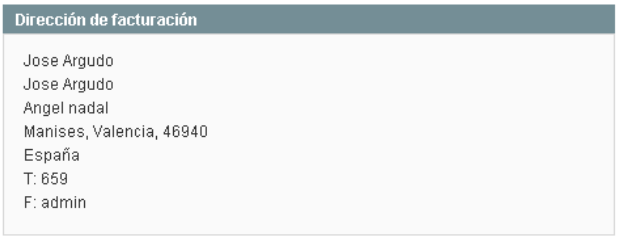
Primero podemos ver información básica del pedido, y su estado:

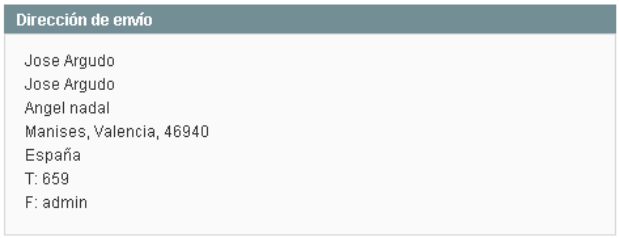


A la derecha tendremos información sobre la cuenta desde la que fue creada el pedido:

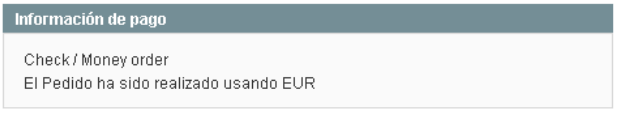


Lo siguiente que podemos ver son los detalles de dirección de envío y facturación:





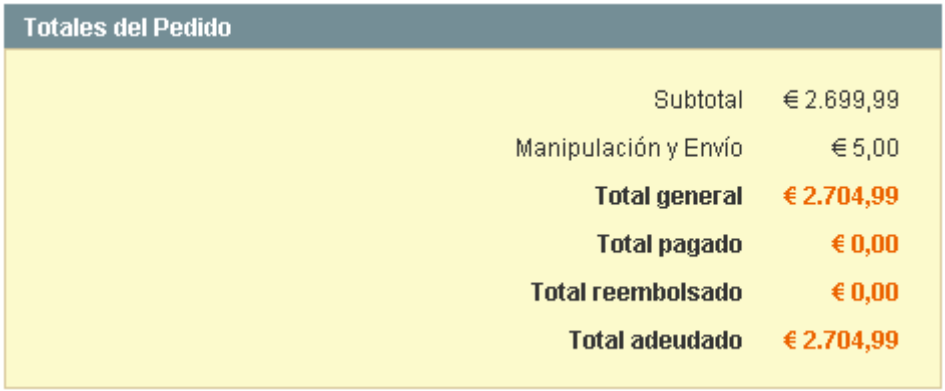
y, más abajo, la información de pago y los gastos de envío:



Por supuesto también tenemos información sobre los artículos que fueron adquiridos en ese pedido:

Además de toda esta información también contamos con un historial de comentarios, que nos permitirá ir añadiendo nuestros propios comentarios al pedido, así como notificar al cliente por email de estos comentarios y hacerlos visibles en la parte pública.

El último detalle al que tenemos acceso es a la información económica, el subtotal, el coste de envío, lo que ha sido pagado, y lo pueda haber sido reembolsado.



Además de toda esta información esta pantalla nos permite actuar con el pedido, tenemos las siguientes opciones:

Vamos a comentar cada una de ellas:

- **Editar** -> con esta opción, podremos realizar modificaciones en el pedido, aunque con ciertas limitaciones.
- **Cancelar** -> con esta opción cancelaríamos el pedido
- **Send Email** -> enviará un email informativo al cliente
- **Suspender** -> dejará el pedido en pausa, posteriormente lo podremos reanudar
- **Factura** -> en ciertos tipos de pago (cheque, transferencia) no emitiremos la factura hasta que realmente se haya efectuado el mismo.
- **Enviar** -> nos permitirá introducir los datos de envío, aunque, mientras el pedido no se facture, no se cerrará.

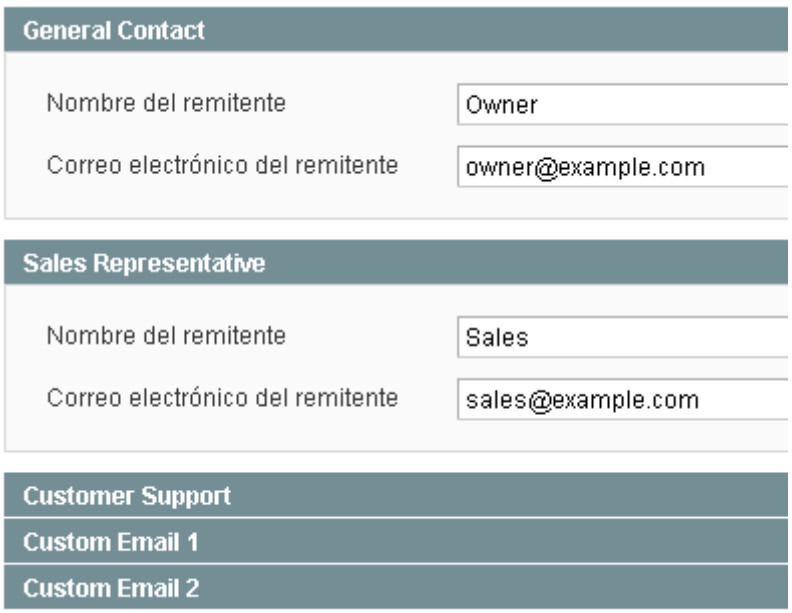
Gestión de envíos

Los pedidos que han sido enviados podemos verlos en **Ventas** -> **Envíos**, de forma cómoda y separada del resto de pedidos. El funcionamiento es bastante similar al de la pantalla de pedidos, un listado filtrable, en el que al hacer clic en los pedidos accederemos a su pantalla de detalle.

Además desde esta pantalla podremos enviar información de seguimiento a los clientes, así como añadir comentarios al pedido.

Configuración de las cuentas de correo

Otra de las configuraciones básicas que debemos realizar en nuestra tienda es la configuración de las cuentas de correo. Para ello iremos a **sistema → configuración**, y ahí pestaña **general** y menú **Direcciones de correo electrónico de la tienda**. Veremos una pantalla similar a la siguiente:



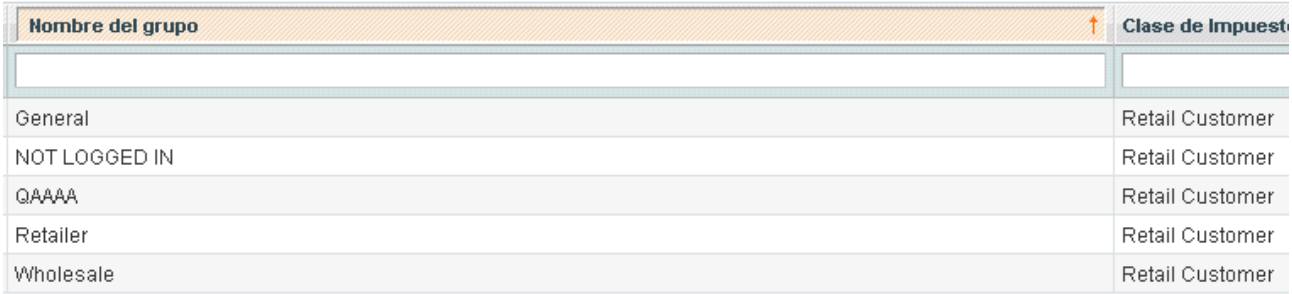
Aquí podemos configurar las cuentas de salida de correo de Magento, que utilizará en los diferentes contactos que tengamos con el cliente. Para configurar el formulario de contacto, desde el cual los clientes se pueden comunicar con nosotros, en esa misma pestaña, **General**, iremos al menú

Contactos, que nos presentará la siguiente pantalla:

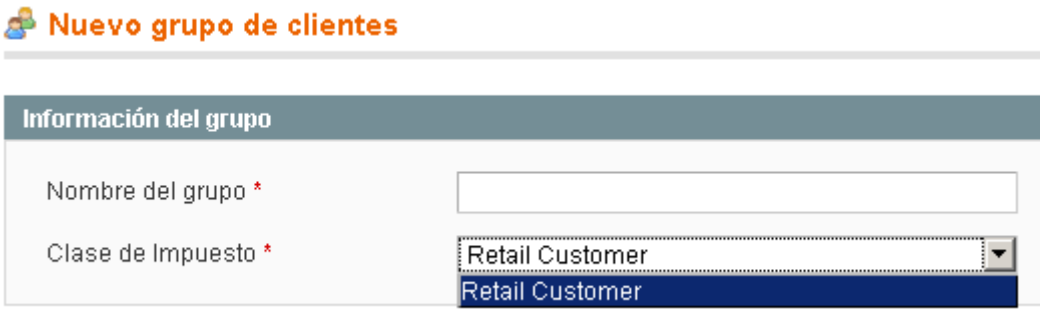
Donde podemos configurar si está habilitado el formulario de contacto, desde que cuenta se envia, el remitente etc

Clientes

Punto clave en nuestra tienda, la gestión de clientes. Podremos gestionar los **clientes**, y sus **grupos**, bajo el menú **Clientes** de nuestro panel de administración. Empecemos por **Clientes → Grupos de Clientes**. La pantalla que veremos será similar a la siguiente:

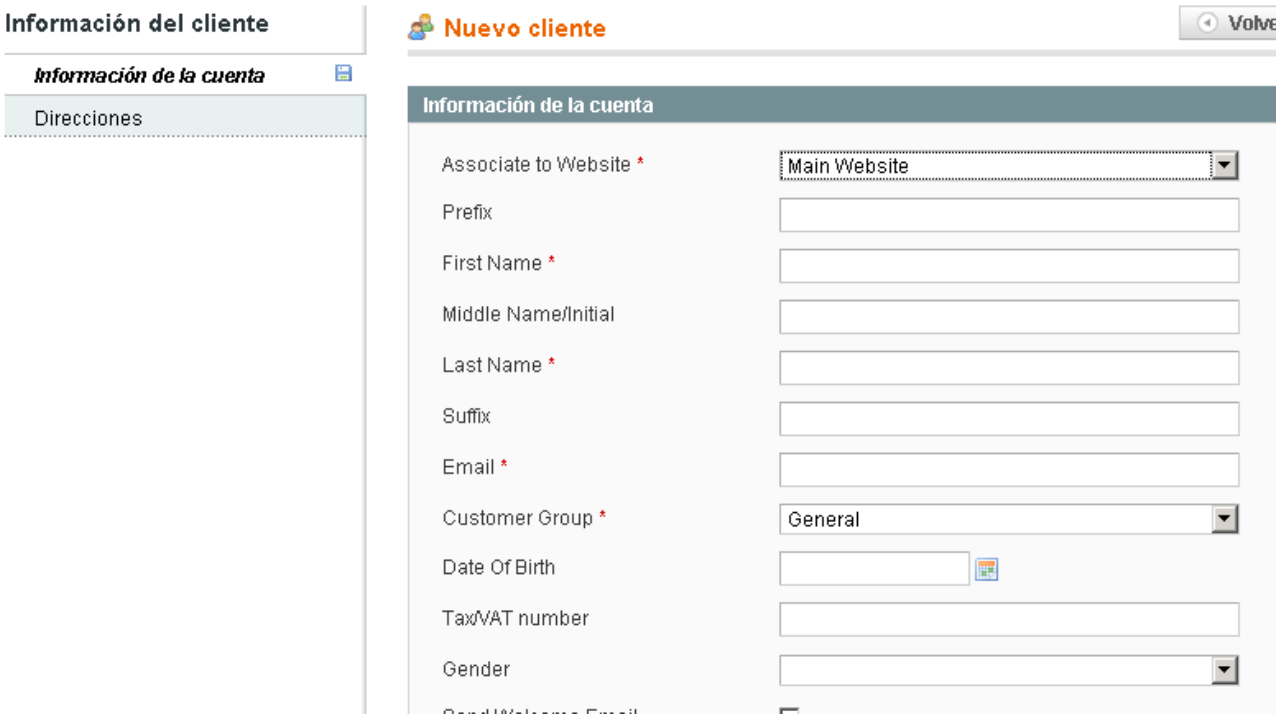


Donde podemos ver los diferentes grupos, junto con la clase de impuesto que se aplica al grupo, más tarde veremos más en profundidad el tema de los impuestos. Para añadir un nuevo grupo de clientes haremos clic en el botón **Añadir un nuevo grupo de clientes**.



La pantalla no podía ser más sencilla, el nombre del grupo, y la clase de impuesto asociada a dicho grupo.

Si ahora vamos al menú **Clientes → Gestionar clientes**, podremos ver un listado con todos los clientes, e incluso añadir uno:



No solo podemos añadirle toda la información necesaria, si no todas las direcciones de envio que deseemos, más tarde se podrán seleccionar a la hora de efectuar los pedidos.

Si en lugar de crear un nuevo cliente, editamos uno existente, podremos acceder a gran cantidad de información sobre el mismo, información de la cuenta, direcciones, pedidos, carritos de la compra que tenga actualmente, listas de deseos, si está suscrito a boletines de noticias, y cuales ha recibido, incluso que comentarios y etiquetas de producto ha creado.

Configuraciones de los clientes

Además de crear grupos de clientes, y clientes, tambien tendremos que configurar diferentes aspectos generales de los mismos. Esto lo haremos desde **Sistema → Configuración**, pestaña **Clientes → Configuración del cliente**. Examinemos por encima los diferentes apartados de que disponemos:

- Opciones online del cliente:
  - Minutos online → que define cuanto tiempo durará la sesión del cliente.
- Opciones para compartir la cuenta:
  - Compartir las cuentas de clientes → si se comparten por sitio web, o por el global de nuestra instalación de Magento.
- Crear nuevas opciones para las cuentas (más bien, opciones para la creación de cuentas nuevas)
  - Grupo por defecto → al que se asociará la nueva cuenta creada
  - Dominio de correo electrónico por defecto → esto solo es útil cuando creemos pedidos nosotros mismos, a encargo de un cliente que no quiere utilizar su cuenta de correo. Imaginemos que efectua el pedido de forma telefónica. En este caso, de no disponer de email, se generará una cuenta ficticia, que mezclará el identificador de usuario, y el dominio que establezcamos aquí.
  - Correo de bienvenida

    - Remitente de correo electronico
    - Correo de confirmación requerido → para activar la cuenta
    - Enlace de correo electrónico de confirmación
    - Correo de bienvenida
    - Generate Human-Friendly Customer ID
- Opciones de la contraseña
    - Plantilla para recordatorio de correo electrónico
    - Remitente para recordatorio de correo electrónico
- Opciones nombre y dirección
    - Number of Lines in a Street Address
    - Show Prefix
    - Prefix Dropdown Options
    - Show Middle Name (initial)
    - Show Suffix
    - Suffix Dropdown Options
    - Show Date of Birth
    - Show Tax/VAT Number
    - Show Gender
- Login Options
    - Redirect Customer to Account Dashboard after Logging in

Impuestos

Normalmente, cuando compramos un producto, el impuesto que pagamos viene definido por una serie de parámetros, entre ellos:

- Donde compramos el producto
- El tipo de producto
- El tipo de comprador (empresa, particular etc)
- Cantidad (dependiendo del pais puede ser que se aplique el impuesto a partir de cierta cantidad)

Veamos como podemos definir todo esto en Magento.

Reglas de impuestos

Una regla de impuestos es la combinación de:

- tasa de impuesto
- direccion de envio
- clase de producto
- clase de cliente
- cantidad

Esto lo podemos ver en el menu **Ventas → Impuestos → Gestionar reglas de impuestos**, donde se nos mostrará una pantalla similar a la siguiente:

Desde esa pantalla podemos hacer clic en el boton **Añadir regla de impuesto**, donde veremos una pantalla en la que rellenar los siguientes campos:

- **Nombre** → este será el nombre que distinga esta regla
- **Clase de impuesto al cliente** → antes vimos que al crear un grupo de clientes se le asociaba una clase de impuesto, aquí defimos que clase de impuesto tendrá esta regla, en relación a los clientes.
- **Clase de impuesto al producto** → similar pero relacionado con los productos, cuando creamos un producto indicamos la clase de impuesto que tendrá ese producto, aquí lo relacionamos.
- **Tasa de impuesto** → es una combinación de zonas de impuestos y porcentajes
- **Prioridad** → esto define como actuará Magento al encontrarse con diferentes reglas de impuestos, si ambas tienen la misma prioridad se sumaran sus porcentajes y se aplicaran al tocal. Si tienen diferentes prioridades se aplicará la de mayor prioridad al total, y al resultado se le aplicará la siguiente regla.
- **Orden** → el orden en el que se muestran en el carrito, es solo visual.

Con esto ya tendríamos creada nuestra regla de impuesto que relaciona la clase de impuesto del cliente, la clase de impuesto del producto y la tasa de impuesto.

Impuestos al cliente

Esta pantalla es muy simple, y al crear uno nuevo solo debemos introducir su nombre.

Impuestos al producto

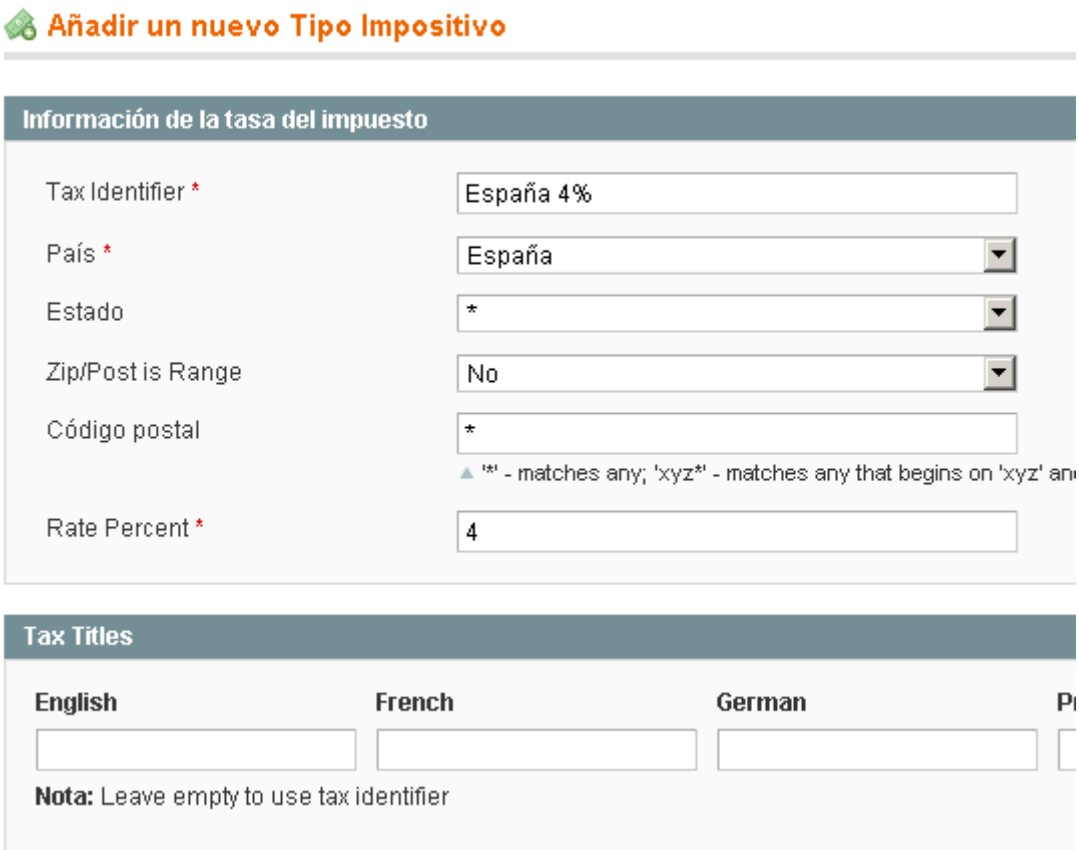
Funciona exactamente igual que los impuestos al cliente.

Manage tax rates and zones

Esta es la pantalla que definitivamente nos va a permitir introducir el porcentaje de impuesto. Accederemos a ella desde el menu **Ventas → Impuestos → Manage Tax Zones and Rates**, ahí podremos ver las que ya hay creadas y crear una nueva, haciendo clic en el botón **Añadir un nuevo tipo impositivo**.

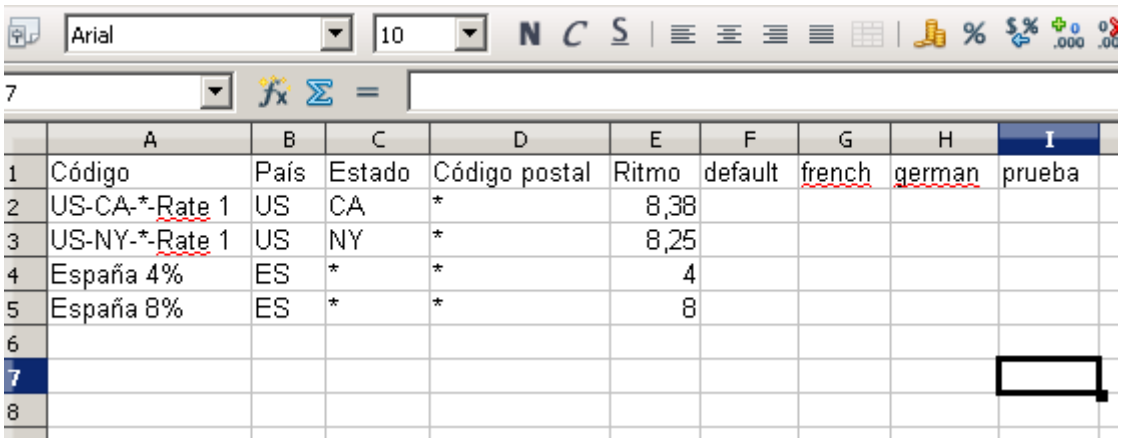
Nota: Magento calcula el impuesto a partir de la dirección de facturación , no la dirección de envio

La pantalla de creación se ve así:



Y nos permite introducir un identificador, el pais, estado (provincia) y los códigos postales a los que aplica esta regla, ademas del porcentaje en cuestión.

Tambien es posible exportar e importar las tasas de impuesto, desde el menú importar/exportar, de manera que podamos editar las tasas desde una hoja de cálculo:



	A	B	C	D	E	F	G	H	I
1	Código	País	Estado	Código postal	Ritmo	default	french	german	prueba
2	US-CA-*-Rate 1	US	CA	*	8,38				
3	US-NY-*-Rate 1	US	NY	*	8,25				
4	España 4%	ES	*	*	4				
5	España 8%	ES	*	*	8				
6									
7									
8									

Desde la que poder trabajar comodamente creando las reglas, y posteriormente importarlas sin necesidad de crearlas una a una.

Envios

Configuración -> configuraciones de envio

Configuración -> métodos de envio

Alcance -> main website -> Table rates -> exportar e importar

Configuraciones avanzadas

Seguimos avanzando en nuestro paseo por Magento. En este punto vamos a ver algunas de las opciones más avanzadas de las que disponemos, tales como la capacidad multitienda, multiidioma o multimoneda.

Veremos también como realizar una importación avanzada de productos, y reglas de catálogo y carrito que nos permitirán manejar mejor las ofertas y los productos de nuestra tienda.

Multitienda

Magento nos ofrece la capacidad de gestionar varias tiendas dentro de un único sitio web. ¿Para que sería esto útil? Bien, esto nos permitiría tener una tienda por cada idioma que utilicemos, o con diferentes monedas, pero, además, también podríamos tener diferentes catálogos y productos.

Es algo muy versátil que vamos a ir examinando detenidamente, para empezar vamos a estudiar el ámbito de los diferentes niveles que tenemos:

- **Global** -> configuraciones e información que será utilizada en todos los escaparates de tiendas.
- **Website** -> esto hace referencia a un conjunto de tiendas, que comparten cuentas de cliente, pedidos y carritos de la compra.
- **Store** -> una tienda es un grupo de escaparates que hacen referencia a la misma categoría, o catalogo.
- **Store view** -> los escaparates son representaciones visuales de una tienda, y nos servirán, por ejemplo, para ofrecer la tienda en múltiples idiomas.

¿Como se traduce esto en nuestra instalación de Magento? Lo podemos ver de una forma muy sencilla desde nuestro panel de administrator, si vamos a **Sistema** -> **Configuración**, podremos ver a la izquierda un desplegable donde podemos seleccionar el alcance de la configuración.

La **configuración por defecto** equivale al nivel **global** de alcance. Los cambios y selecciones que hagamos aquí afectaran pues a todo el sitio, y, a no ser que en los niveles inferiores indiquemos lo contrario, serán los que se apliquen.

Después tenemos **Main Website**, que estaría al nivel de **Website**, es decir, las tiendas que hay dentro comparten información de clientes, pedidos etc.

En este caso solo tenemos una tienda, **Main Store**, que contiene tres escaparates, o **Store Views**, cada uno, como hemos comentado antes, equivale a un idioma.

¿Como podemos crear y modificar estos elementos? Para ello iremos a **Sistema** -> **Gestionar tiendas,** donde podremos ver una pantalla similar a la siguiente:

En el centro podemos ver un listado, en tres columnas, que nos muestra tanto los sitios web de que disponemos, como tiendas, como vistas de tienda (escaparates). En este caso lo que vemos se

traduce en lo que teníamos en el desplegable anterior:

- Main Website
  - Main store
    - English
    - French
    - German

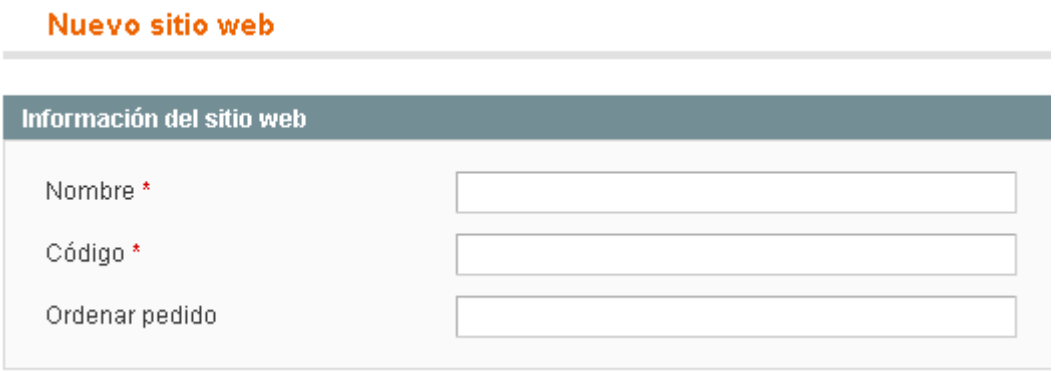
Además de visualizar esta información, en la parte superior de esa pantalla tenemos tres botones:

- Crear un sitio web
- Crear una tienda
- Crear una vista de tienda

Veamos un poco estos apartados

Crear un sitio web

Una vez hacemos clic en el botón crear un sitio web, accederemos a la siguiente pantalla:



Solo hemos de introducir esos tres valores, y guardar, para que nuestro sitio web sea creado.

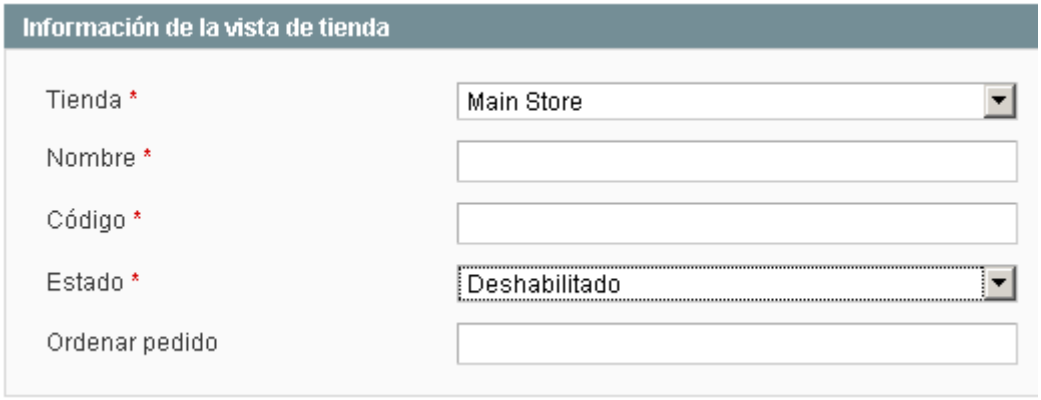
Crear una tienda

Crear una tienda es un proceso igual de sencillo que el anterior, veamos el aspecto que tiene la pantalla de creación:

La variación es mínima, por un lado deberemos seleccionar el sitio web al que se asociará la tienda, el nombre que queremos que tenga, y la categoría raíz del catálogo. Este último parámetro es el que nos permitirá tener un catálogo por cada tienda.

Crear una vista de tienda

Por último tenemos que crear los escaparates para nuestra tienda, con un proceso muy similar al que hemos seguido hasta ahora:

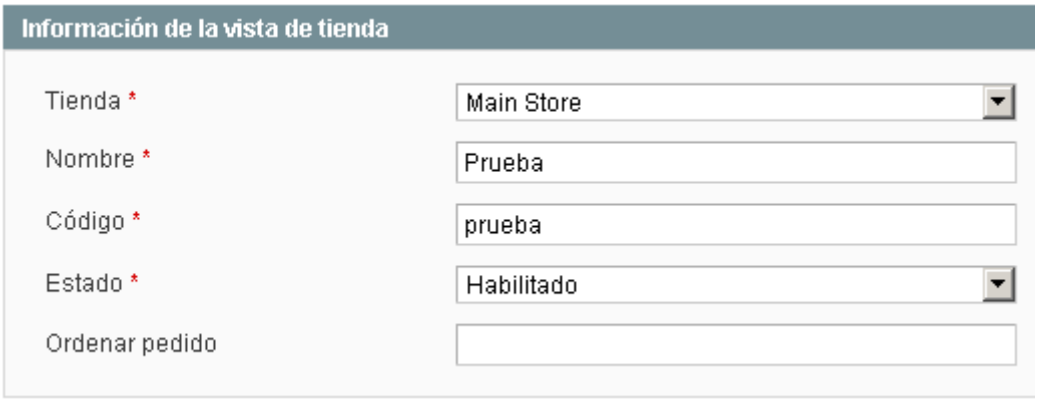


Seleccionando en este caso la tienda a la que pertenecerá la vista, su nombre y código, y si está habilitada. Así como el orden en que aparecerán los productos en el pedido.

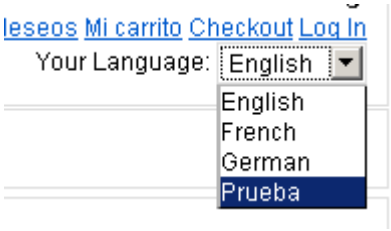
Multiidioma

Una vez que tenemos las diferentes vistas de nuestra tienda, dotarla de capacidades multiidioma es bastante sencillo. Y es que ya vimos los pasos necesarios para instalar un nuevo idioma, ahora veremos como asociar los idiomas a las vistas de tiendas.

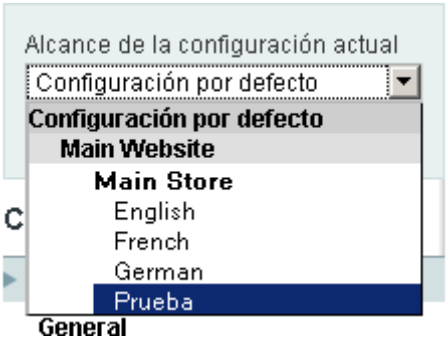
Para hacer la prueba, vamos a crear una nueva vista de tienda, desde la pantalla Sistema -> Gestionar tiendas -> Nueva vista de tienda:



Una vez hecho esto guardamos, y en nuestro frontend veremos algo similar a lo siguiente en el desplegable de selección de idiomas:



Con lo que tenemos nuestra nueva vista disponible. ¿Como asociar un idioma a esta vista? Para eso iremos a **Sistema** -> **Configuración**, seleccionamos el alcance correspondiente:



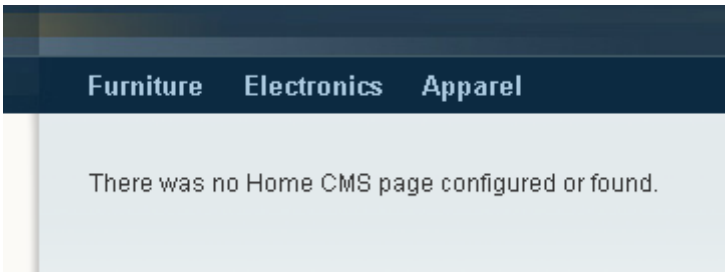
Iremos a la pestaña general:

Después de eso iremos a las opciones de localización, deseleccionaremos el checkbox "Use Website" (indicativo del uso de las opciones globales) y seleccionaremos el idioma que nos interese, en este caso, inglés.

Y con esto al seleccionar la pestaña en el frontend se cambiará el idioma de esa vista en concreto.

Posibles problemas, y solución

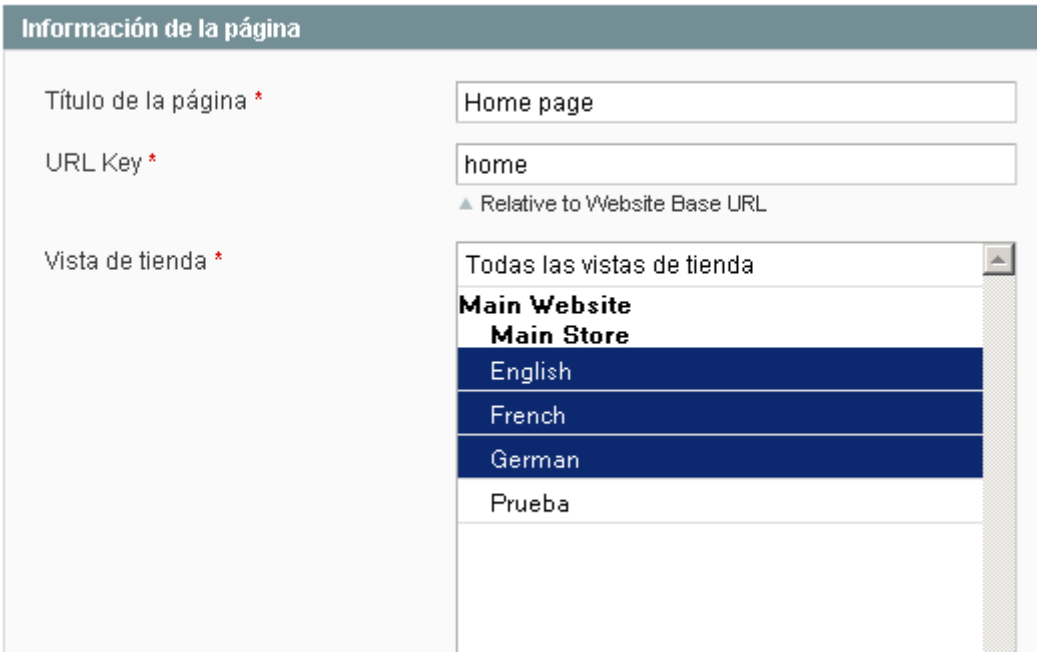
Es posible que al crear esta nueva vista, al intentar acceder a ella desde el frontend veamos el siguiente mensaje:



Este mensaje nos indica que no hemos seleccionado ninguna página CMS para mostrar. Bien, seleccionemosla ahora, para ello iremos a nuestra página de administración, **CMS** -> **Pages**. Si nos fijamos en la línea de la home page:

Home page	home	1 column	Main Website Main Store English French German

Podemos ver como en la cuarta columna aparece las tres vistas de tienda, pero no hay ni rastro de nuestra nueva vista. Haremos clic en esta linea, y veremos una pantalla similar a la siguiente:



Como podemos ver la opción **Prueba** no aparece seleccionada, por lo tanto, está página no se muestra en dicha vista, mostrando por lo tanto el mensaje de error. Vamos a seleccionar la opción **Todas las vistas de tienda**, de manera que si en el futuro añadimos más vistas no tengamos este mismo problema. Guardaremos los cambios, y con esto se habrá solucionado nuestro pequeño problema.

Desde que fichero se genera el selector de idioma

El selector de idioma se genera desde el fichero **app** -> **design** -> **frontend** -> **base** -> **default** -> **template** -> **page** -> **switch** -> **languages.phtml** y tiene este contenido:

```
<?php if(count($this->getStores())>1): ?>
<div class="form-language">
   <label for="select-language"><?php echo $this->__('Your Language:') ?></label>
   <select id="select-language" title="<?php echo $this->__('Your Language') ?>"
onchange="window.location.href=this.value">
   <?php foreach ($this->getStores() as $_lang): ?>
      <?php $_selected = ($_lang->getId() == $this->getCurrentStoreId()) ? ' selected="selected"' : '' ?>
      <option value="<?php echo $_lang->getCurrentUrl() ?>"<?php echo $_selected ?>><?php echo $this-
>htmlEscape($_lang->getName()) ?></option>
   <?php endforeach; ?>
   </select>
</div>
<?php endif; ?>
```

Por supuesto podríamos duplicar este fichero y colocarlo en la carpeta de la plantilla que estemos utilizando, por ejemplo **app** -> **design** -> **frontend** -> **default** -> **blank_seo** -> **template** -> **page** -> **switch** -> **languages.phtml**

De esa manera podremos hacer las modificaciones que queramos sin la necesidad de modificar al fichero base. Por ejemplo vamos a añadirle el siguiente código, al principio:

```
<?php if(count($this->getGroups())): ?>
<div class="store-switcher">
   <label for="select-store"><?php echo $this->__('Select Store:') ?></label>
   <select id="select-store" title="<?php echo $this->__('Select Store') ?>" onchange="location.href=this.value">
   <?php /*foreach ($this->getStores() as $_store): ?>
      <option value="<?php echo $_store->getUrl('') ?>"<?php if($_store->getId()==$this->getCurrentStoreId()): ?>
selected="selected"<?php endif; ?>><?php echo $_store->getName() ?></option>
   <?php endforeach;*/ ?>
   <?php foreach ($this->getGroups() as $_group): ?>
      <?php $_selected = ($_group->getId()==$this->getCurrentGroupId()) ? ' selected="selected"' : '' ?>
      <option value="<?php echo $_group->getHomeUrl() ?>"<?php echo $_selected ?>><?php echo $this-
>htmlEscape($_group->getName()) ?></option>
   <?php endforeach; ?>
   </select>
</div>
<?php endif; ?>
```

Es una versión levemente modificada de lo que podemos encontrar dentro del fichero **app** -> **design** -> **frontend** -> **base** -> **default** -> **template** -> **page** -> **switch** -> **stores.phtml** Pero hemos cambiado esta linea:

```
<?php if(count($this->getGroups())>1): ?>
```

por esta otra

```
<?php if(count($this->getGroups())): ?>
```

Para mostrar el desplegable incluso aunque solo tengamos una tienda:

Ahí tenemos, un desplegable para tiendas, y otro para vistas.

Multimoneda

Al igual que podemos tener una tienda con múltiples idiomas, también podemos tener una tienda con múltiples monedas. Para ello deberemos seleccionar que monedas, y para que vistas, queremos que estén disponibles. Iremos pues a nuestro panel de administración, menú **Sistema** -> **Configuración**. Seleccionaremos el alcance desde el desplegable de la izquierda, y, en la pestaña **General**, seleccionaremos **Configuración de la Moneda**. La pantalla será similar a la siguiente:

De ahí podemos seleccionar las monedas que estarán permitidas. Así como la opción por defecto. Pero, ¿Es suficiente con esto para mostrar el desplegable de selección de moneda? Puede que no, y eso dependerá de nuestra plantilla, y de nuestra instalación de Magento.

Vamos a solucionar primero el tema de nuestra instalación de Magento. Una vez hemos seleccionado las monedas que queremos que estén disponibles, iremos, en nuestro panel de administración, al menú **Sistema** -> **Gestionar tipos de cambio**, ahí veremos una pantalla como la siguiente:

En esa pantalla deberemos hacer clic en importar, para que Magento importe los tipos de cambio. **Si no tenemos esta importación realizada**, no se mostrará la moneda en el desplegable, pues Magento no sabrá cambiar de un tipo monetario a otro. Es pues un paso muy importante.

Luego podemos trabajar en el tema de la plantilla, para ahora sí mostrar el desplegable de selección de moneda, por ejemplo en nuestro fichero **app** -> **design** -> **frontend** -> **default** -> **blank_seo** -> **layout** -> **page.xml** añadiremos en este código:

```
        <block type="page/html_header" name="header" as="header">
            <block type="page/template_links" name="top.links" as="topLinks"/>
            <block type="page/switch" name="store_language" as="store_language"
template="page/switch/languages.phtml"/>
            <block type="core/text_list" name="top.menu" as="topMenu"/>
```

```
        <block type="page/html_wrapper" name="top.container" as="topContainer" translate="label">
          <label>Page Header</label>
          <action method="setElementClass"><value>top-container</value></action>
        </block>
      </block>
```

la siguiente linea:

```
      <block type="page/html_header" name="header" as="header">
        <block type="directory/currency" name="currency" template="directory/currency.phtml"/>
        <block type="page/template_links" name="top.links" as="topLinks"/>
        <block type="page/switch" name="store_language" as="store_language"
template="page/switch/languages.phtml"/>
        <block type="core/text_list" name="top.menu" as="topMenu"/>
        <block type="page/html_wrapper" name="top.container" as="topContainer" translate="label">
          <label>Page Header</label>
          <action method="setElementClass"><value>top-container</value></action>
        </block>
      </block>
```

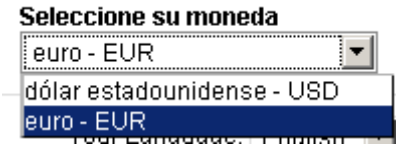
Luego en el fichero **app** -> **design** -> **frontend** -> **default** -> **blank_seo** -> **template** -> **page** -> **html** -> **header.phtml** donde tenemos:

```
    <div class="quick-access">
      <?php echo $this->getChildHtml('topSearch') ?>
      <p class="welcome-msg"><?php echo $this->getWelcome()?></p>
      <?php echo $this->getChildHtml('topLinks') ?>
      <?php echo $this->getChildHtml('store_language') ?>
    </div>
```

Añadiremos la siguiente línea:

```
    <div class="quick-access">
      <?php echo $this->getChildHtml('currency') ?>
      <?php echo $this->getChildHtml('topSearch') ?>
      <p class="welcome-msg"><?php echo $this->getWelcome()?></p>
      <?php echo $this->getChildHtml('topLinks') ?>
      <?php echo $this->getChildHtml('store_language') ?>
    </div>
```

Ya solo nos queda copiar, si deseamos modificar el aspecto del desplegable, la carpeta **app** -> **design** -> **frontend** -> **base** -> **default** -> **template** -> **directory** a **app** -> **design** -> **frontend** -> **default** -> **blank_seo** -> **template** -> **directory** y con eso nuestro desplegable de selección de monedas aparecerá:

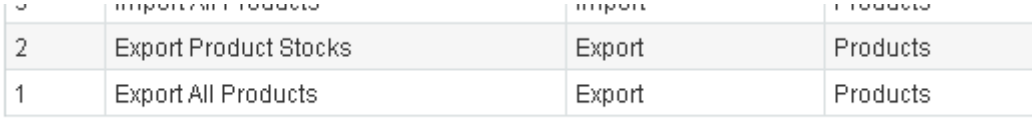


Permitiendo a nuestro visitantes cambiar de divisa cómodamente.

Importación avanzada de productos

Vamos ahora a ver un poquito sobre la exportación de importación de productos. Importar productos nos puede ayudar a insertar una gran cantidad de productos a una tienda recién creada. Sin necesidad de introducirlos uno a uno. Vamos a ver algunos pasos tanto para exportar, como para importar.

Empezaremos con la **exportación**, esto nos ayudará a ver el formato necesario para la importación. En nuestro panel de administración iremos a **Sistema → Importar/Exportar → Perfiles**. En esa pantalla haremos clic en **Export all products**:

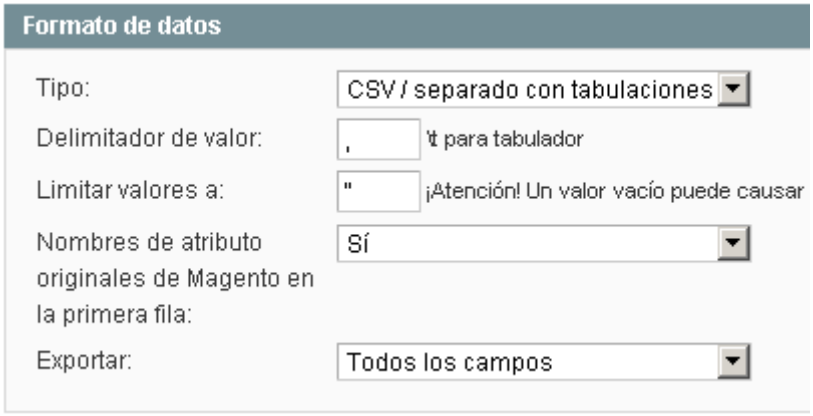


2	Export Product Stocks	Export	Products
1	Export All Products	Export	Products

Eso nos llevará a una pantalla donde podremos editar el perfil de exportación, con gran cantidad de opciones, veamos algunas de ellas. Por ejemplo, tenemos información del perfil:

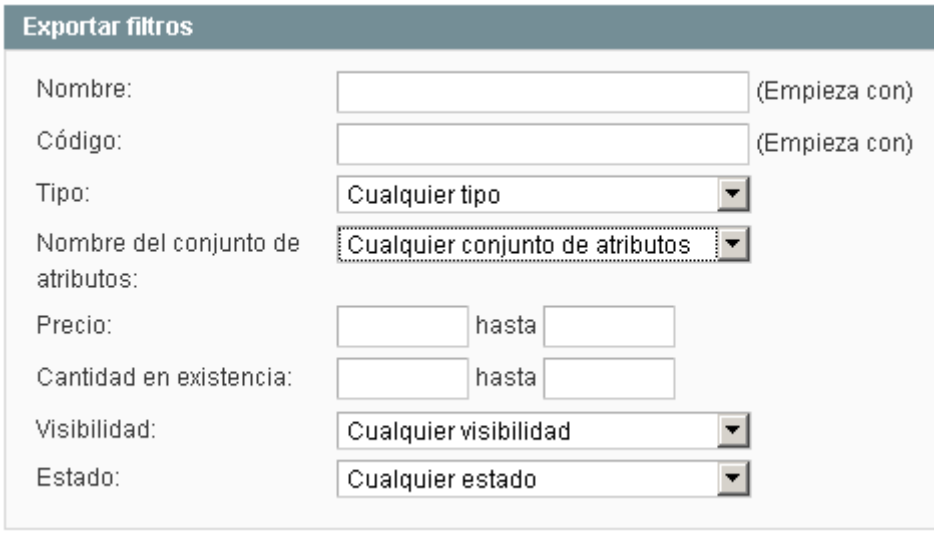
Aquí tenemos el nombre del perfil, el tipo de entidad (productos o clientes), la dirección (si es exportación o importación) así como la tienda y su ámbito. Más abajo tenemos información del archivo:

Con estas opciones indicaremos donde, y bajo que nombre, se crea el archivo. Y justo debajo tenemos opciones relacionadas con su formato:



Podemos adaptar estas configuraciones para abrir el archivo bajo Excel, Open Office etc Especificando los campos delimitadores, el tipo de fichero etc

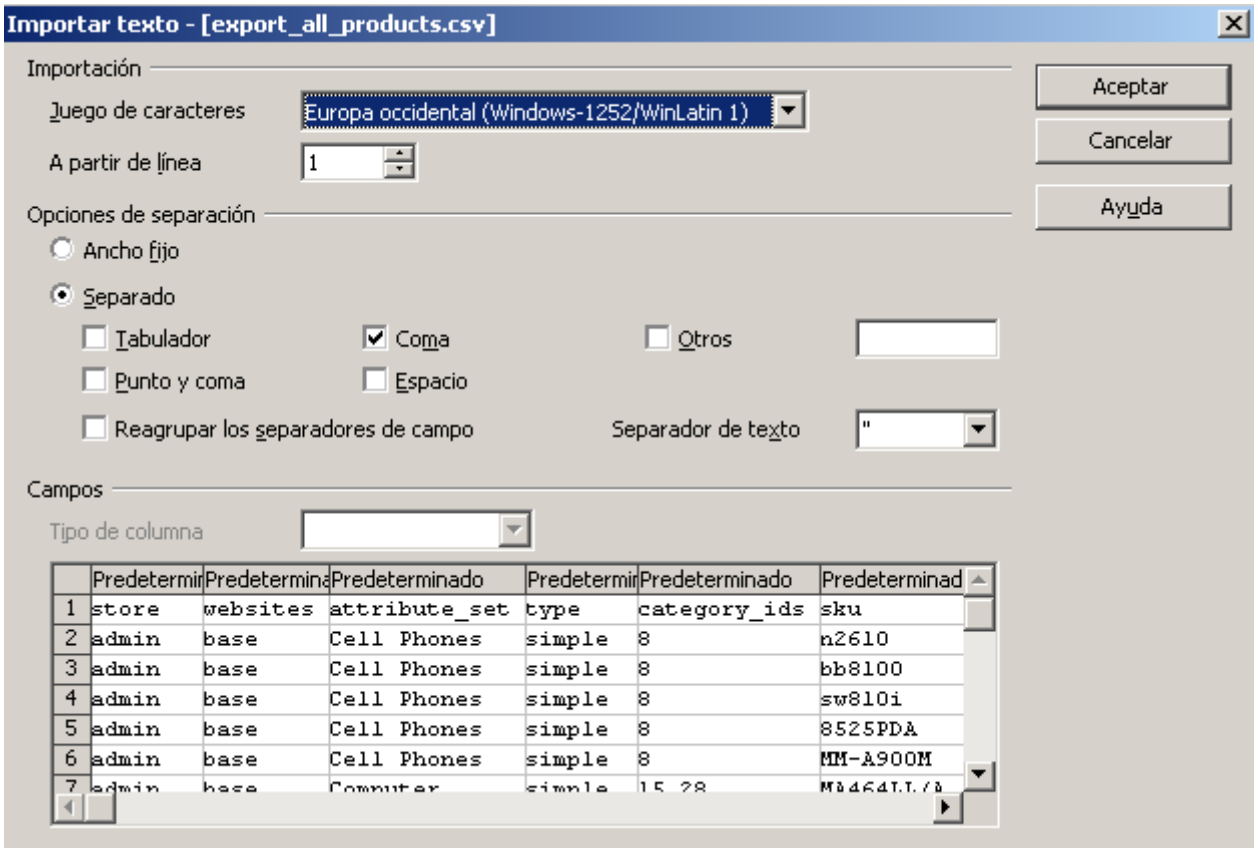
Por último tenemos algunos filtros, que nos permitirán especificar aún más que productos queremos exportar:



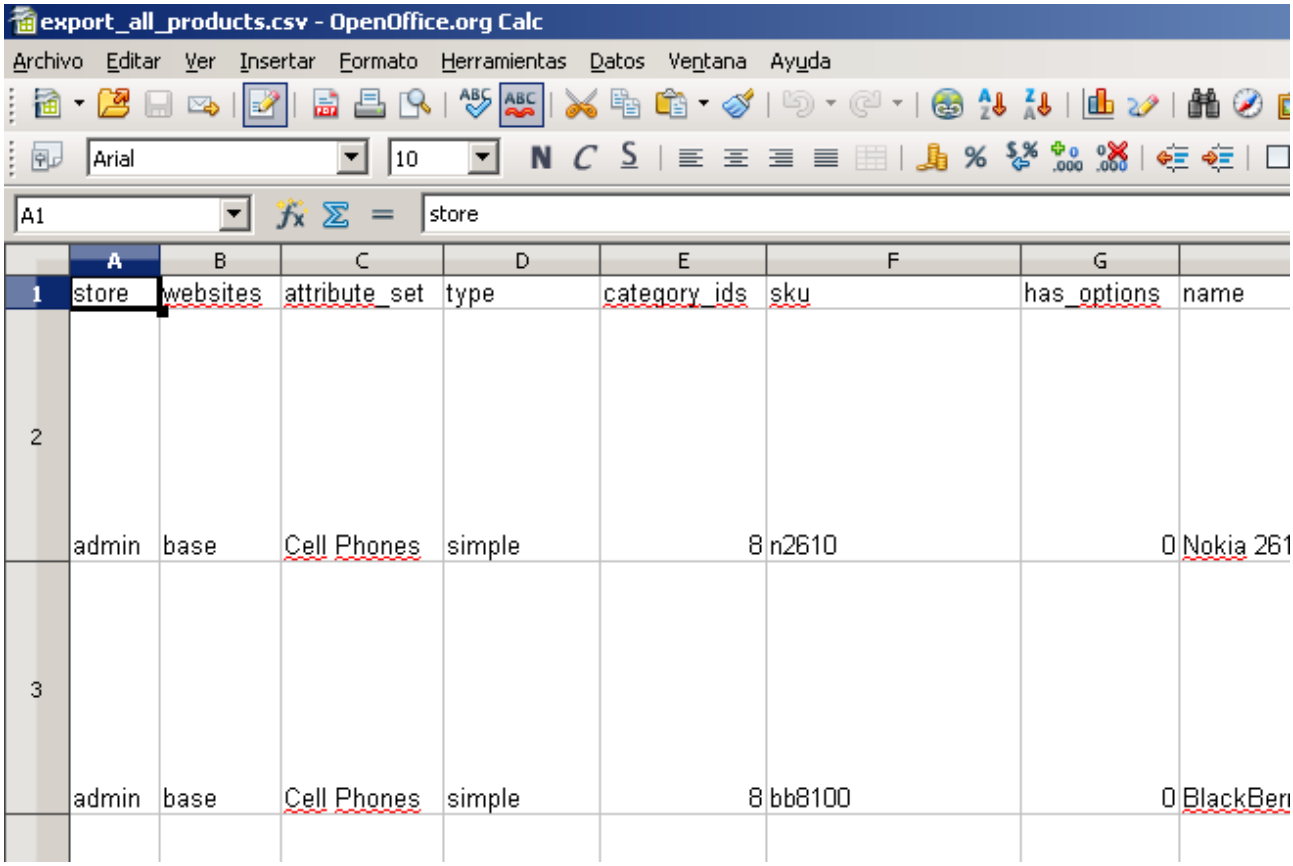
Bien, ahora para realizar la exportación haremos clic en la pestaña **Ejecutar perfil**:

Y posteriormente en el botón **Run profile in popup**. Una vez el proceso de exportación haya finalizado podremos ver un fichero que se habrá generado en **proyecto-mg → var → export**, el fichero tendrá el nombre que nosotros le hayamos dado en la configuración, en este caso **export_all_products.csv**

Podemos abrir este fichero, por ejemplo con Open Office Calc, al hacerlo se nos presentará una pantalla de opciones:



Aquí deberemos configurar las opciones, para que correspondan con las que seleccionamos en el momento de la exportación, dentro de nuestra instalación de Magento. Al abrir el fichero nos encontraremos con una estructura similar a la siguiente:



	A	B	C	D	E	F	G	
1	store	websites	attribute_set	type	category_ids	sku	has_options	name
2	admin	base	Cell Phones	simple	8	n2610	0	Nokia 261
3	admin	base	Cell Phones	simple	8	bb8100	0	BlackBer

La primera fila es ocupada por los nombres de los campos, y, a partir de ahí, cada fila será uno de los productos de nuestra tienda.

Vayamos ahora con la **importación** de productos. Para ir ahí iremos a **Sistema → Importar/Exportar → Perfiles**. Así como antes hicimos clic en **Export all products** ahora haremos clic en **Import all products**. Las opciones serán más o menos las mismas, pero podremos importar los ficheros de dos maneras:

- Subiendo los ficheros, mediante un formulario, para lo cual tendremos la configuración de esta manera:

- La otra opción sería subir el fichero por ftp, y configurar los siguientes campos:

Esto funciona un poco a la inversa que con la exportación. Este proceso de puede tanto insertar, como actualizar productos, los campos básicos requeridos para la importación son los siguientes:

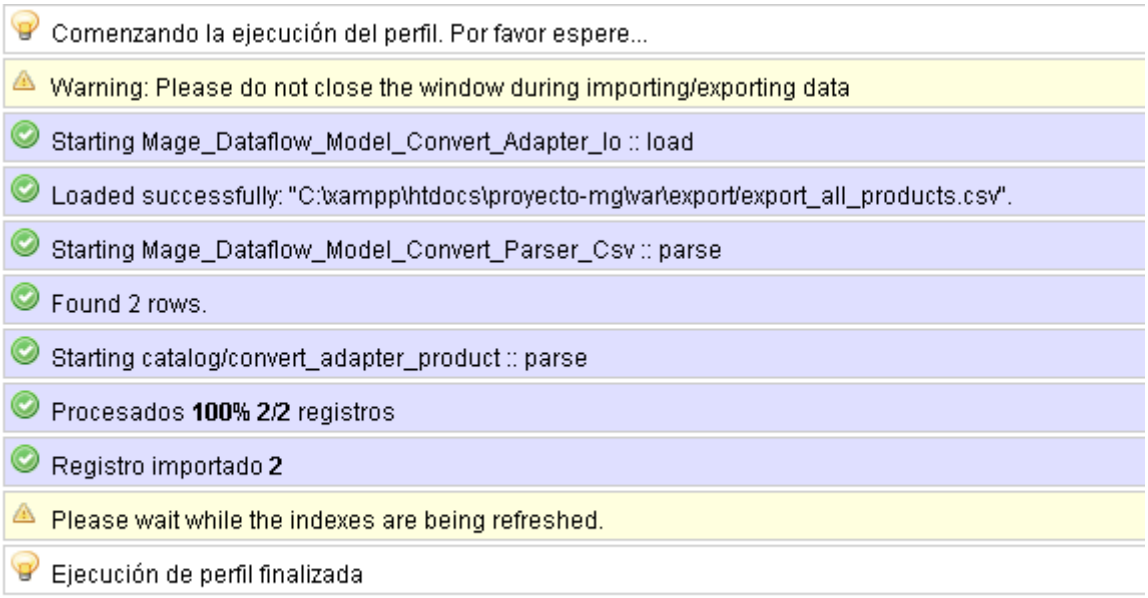
- type (es el tipo de producto, como simple)
- attribute_set (grupo de atributos, por defecto Default)
- tax_class_id
- status (enabled or disabled)
- weight
- sku
- name
- price
- description
- short_description

Como hemos dicho estos campos son indispensables para la importación de los productos, pero, además, si queremos que estén visibles deberemos proporcionar los siguientes campos:

- visibility (”Catalogue,Search”)
- category_ids (las categorías deben ser existentes, así que primero deberemos averiguarlas)
- qty
- image, small_image, thumbnail (las imagenes las dejaremos dentro de la carpeta media/import, precediendo el nombre por una /)

Nota: antes de poder ejecutar el importador, despues de seleccionar el fichero, deberemos de hacer clic en el boton guardar y continuar. De esta manera el importados sabrá que fichero debe importar.

Ejecutamos igual que al exportar y veremos una imagen similar a:



Nota sobre las imagenes en los templates

Independientemente del tamaño con el que subamos las imagenes, a la hora de utilizarlas en el template, por ejemplo **app → design → frontend → base → default → template → catalog → product → list.phtml**, utilizando el método resize:

```
<img src="<?php echo $this->helper('catalog/image')->init($_product, 'small_image')->resize(135); ?>" width="135" height="135" alt="<?php echo $this->stripTags($this->getImageLabel($_product, 'small_image'), null, true) ?>" />
```

En caso de tener más tiempo, podemos seguir indagando el tema de las importaciones con este artículo de la wiki:

http://www.magentocommerce.com/wiki/3_-_store_setup_and_management/import_export/how_to_automatically_import_simple_grouped_and_configurable_products

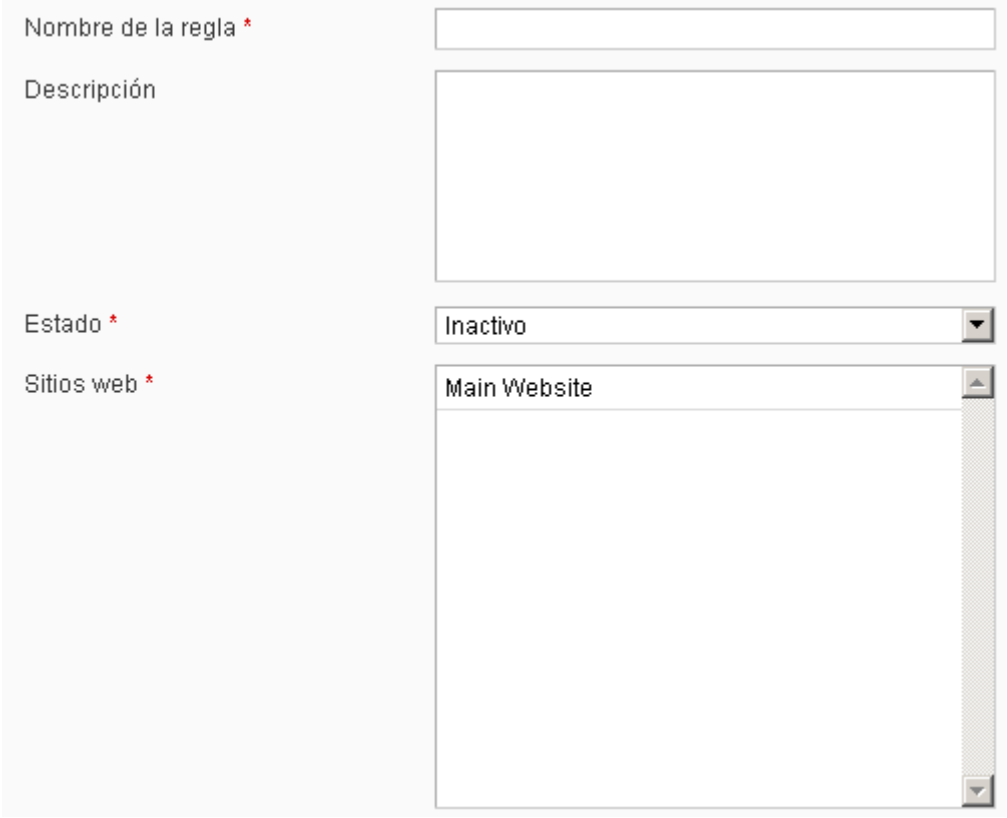
Reglas

Vamos ahora con el tema de las promociones, estos menús los podemos encontrar en nuestro panel de control, en el menú Promociones, ahí tenemos Reglas de precios del catálogo y Reglas de precios del carrito de compras. Comenzaremos con las opciones del catálogo.

Catálogo

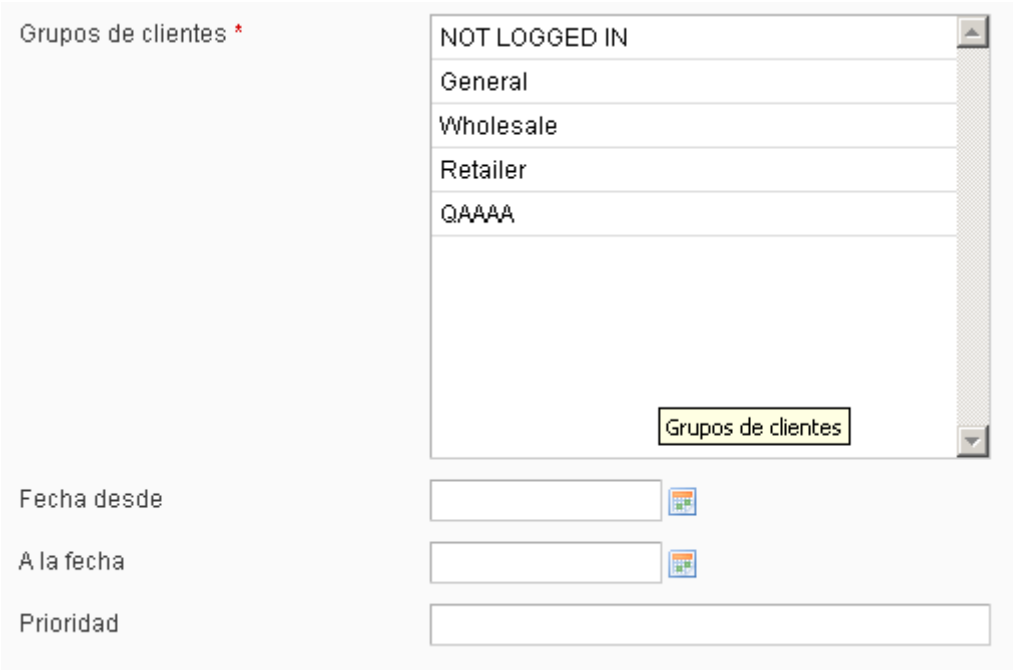
Dentro de nuestro panel de administración iremos a Promociones → Reglas de precios del catálogo, donde veremos una pantalla similar a la siguiente:

Aquí tenemos algunas reglas de ejemplo, podemos estudiarlas para ver como se crean, y como funcionan las reglas de catálogo. Para crear una regla, haremos clic en el botón **Añadir una nueva regla**, veremos una pantalla como la siguiente:

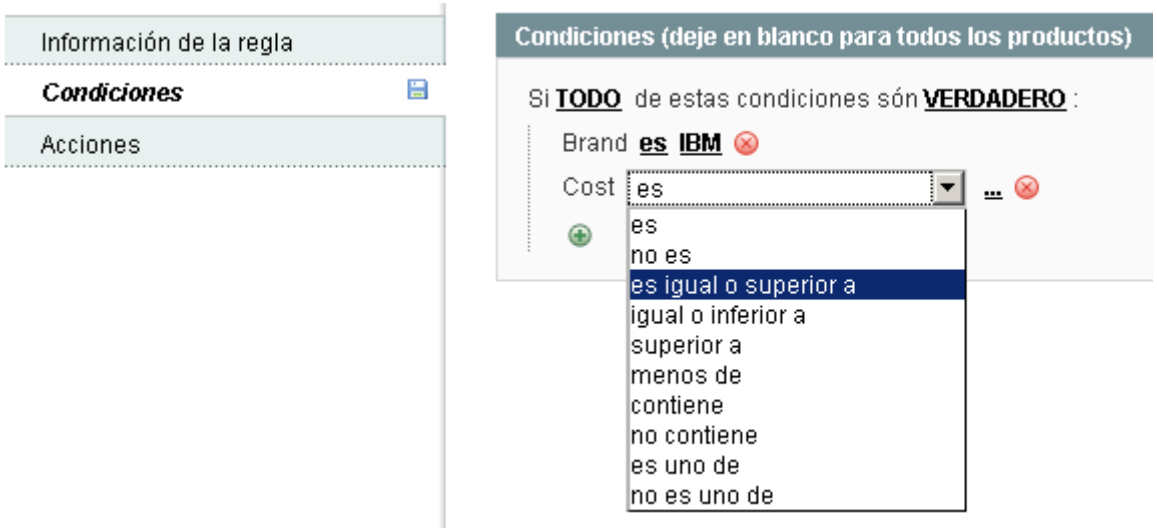


Aquí vemos algunos de los campos que debemos insertar, entre ellos el **nombre de la regla**, su **descripción**, el **estado**, y a que **sitios web** de nuestra tienda se aplica. Estos datos en principio nos ayudarán a identificar la regla, y saber si está o no activa. Sigamos viendo los campos que

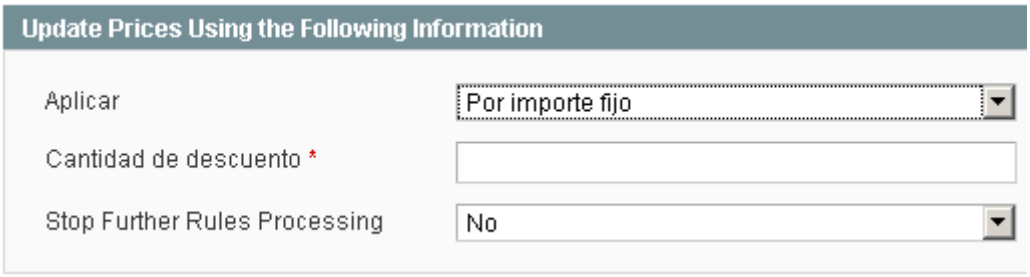
deberemos rellenar:



Aquí podemos indicar a que grupos de clientes se aplicará esta regla, desde que fecha hasta que fecha, y con que preferencia sobre otras reglas. Después de eso a podemos acceder a la pestaña condiciones:



Y desde aquí **construir** las condiciones que harán cumplir nuestra regla. Podemos definir si debe cumplir todo o parte, si debe cumplir las condiciones (VERDADERO) o si no debe cumplirlas (FALSO). Después haciendo clic en el símbolo de + (verde) podremos crear todas las condiciones que nos interesen. Por último accederemos a la pestaña **Acciones:**



En esta pantalla podemos definir el tipo de descuento a aplicar, en que cantidad, y si a partir de aplicar este descuento se dejarán de aplicar el resto. Esto unido con la prioridad que vimos en la primera pantalla nos permitirá combinar las reglas como queramos.

Carrito

Ahora veremos como funcionan las reglas de carrito, iremos a **Promociones → Reglas de precios del carrito de compras**. Haremos clic en Añadir nueva regla, y accederemos a una pantalla muy similar a la anterior, vamos a centrarnos en las diferencias:

- **Información de la regla**:
  - Coupon → por si queremos que la regla se aplique como un cupón de descuento.
  - Uses per costumer → cuantas veces se podrá beneficiar de la regla un mismo cliente.
  - Público en Feed RSS → si queremos que se publique en el RSS, a modo de publicitar el descuento.
- **Condiciones** → funciona igual que en el caso anterior.
- **Acciones** → es una pantalla similar, con algunas diferencias:
  - Maximum Qty Discount is Applied To → cantidad máxima para la que se aplica el descuento.
  - Apply to Shipping Amount → aplicar también al coste de envio.
  - Envío gratuito
- **Labels** → aquí podemos definir las etiquetas, que será la forma en que los visitantes de nuestra tienda verán nombrar la regla/promoción. De manera que noten que se está utilizando.

SEO en Magento

Este es uno de los últimos puntos que vamos a tratar, y es también uno de los más importantes. Hasta ahora los puntos que hemos visto tienen más relación con la creación de nuestra tienda, pero, una vez la tienda este lista y preparada, necesitaremos darla a conocer al mayor número de navegantes posible.

La **S**earch **E**ngine **O**ptimization trata de eso justamente, de hacer que nuestra tienda sea encontrada por los buscadores, en base a las posibles búsquedas que realicen los navegantes.

En estos dos puntos de este tema nos vamos a centrar en las configuraciones que podemos realizar desde el panel de administración de Magento, por supuesto el SEO es mucho más amplio y

necesitaría también que nuestro tema gráfico fuese adaptado en consecuencia. Es, como hemos comentado, un campo muy amplio, de manera que nos centraremos en las configuraciones que se pueden realizar desde el panel.

Configuración general

Nota: para algunas de estas configuraciones es necesario tener instalado el tema Yoast Blank SEO Theme:

http://www.magentocommerce.com/magento-connect/Yoast/extension/974/yoast-blank-seo-theme

Primero vamos a dar los pasos más esenciales, vayamos a nuestro panel de administración, **Sistema** -> **Configuración**, ahí iremos a la pestaña **Web** -> **Optimización** para motores de búsqueda. Ahí tenemos unas cuantas opciones, la más importante de las cuales es:

Usar reescrituras del servidor web -> eso transformará nuestras urls de algo similar a http://www.localhost.com/proyecto-mg/index.php/sony-vaio-vgn-txn27n-b-11-1-notebook-pc.html a algo parecido a http://www.localhost.com/proyecto-mg/sony-vaio-vgn-txn27n-b-11-1-notebook-pc.html

Esa sería la opción principal que debemos activar, pero además tenemos otras opciones:

- Delay loading of print CSS -> esto hará que el fichero de estilos para impresión se cargue más tarde.
- Use delayed javascript image loader - LazierLoad (PrototypeJS) -> esto hará una carga lenta de las imágenes, es decir, se cargará toda la página primero, y por último las imágenes.

Sigamos estudiando las opciones que tenemos disponibles, ahora, en la misma pestaña **Web**, seleccionaremos **Opciones URL**, ahí nos encontramos con:

- **Añadir el código de tienda a las URLs** -> esto nos permite, tal y como indica, añadir el código de tienda a la URL. Si lo habilitamos en lugar de tener URLs como http://www.localhost.com/proyecto-mg/furniture/living-room.html tendríamos URLs como http://www.localhost.com/proyecto-mg/default/furniture/living-room.html Es pues importante que los códigos de tienda que creemos sean significativos.
- **Redirect to Base URL if requested URL doesn't match it** -> si lo habilitamos, en caso de no existir la URL nos llevará a la URL de base.

Sigamos ahora con más configuraciones, seguiremos en **Sistema** -> **Configuración** -> **General**, pero esta vez iremos a la pestaña **Diseño**. Dentro de esa pestaña encontraremos una opción denominada **Cabecera HTML**, vamos a ver que podemos hacer aquí:

- **Título por defecto** -> es el título por defecto de nuestra web, en caso de que no se indique otro desde otra configuración.

- **Title Prefix** y **Title Suffix** -> aquí podemos escribir textos que queramos que aparezcan delante y detrás del título. Es útil por ejemplo, si queremos que siempre aparezca el nombre de nuestra tienda, independientemente del título que se haya auto generado.
- **Descripción por defecto** -> normalmente se generará una descripción a partir del producto, pero en caso de que no haya ninguna se mostrará lo que aquí escribamos.
- **Default Keywords** -> similar a descripción por defecto, pero en este caso para palabras clave.
- **Default Robots** -> opciones por defecto para los robots indexadores
- **Miscellaneous Scripts** -> por si queremos cargar nuestros scripts.
- **Display Demo Store Notice** -> esto mostrará un texto indicando que la tienda no se encentra en funcionamiento, es practico mientras estamos trabajando en su desarrollo, en caso de que este cara al público.

En esta misma etiqueta tenemos otras opciones, por ejemplo las de **Cabecera**, donde podremos establecer la imagen de nuestro logotipo, así como su atributo **alt**, además del texto de bienvenida. Además tendremos opciones para el Pie de página, donde podremos modificar los textos y otro código **HTML** que queremos que aparezca.

Además de estas opciones generales también podemos configurar el catálogo, para ello iremos a **Sistema -> Configuración -> Catálogo -> Catálogo -> Optimizaciones para el motor de búsqueda.** Aquí las más importantes serán:

- Autogenerated Site Map -> para que automáticamente se genere el mapa del sitio de nuestra tienda. Este es un mapa, normalmente en formato XML o texto plano, que los spiders de los buscadores pueden leer, y de esa manera indexar nuestra web de forma más rápida y eficiente.
- Sufijo en la URL de los productos y Categoría Sufijo URL -> aquí podremos indicarle que nuestras páginas terminen, por ejemplo en .html, en lugar de .php, puede parecer cuestión de estética, pero favorece que terminen en .html

Configuración detallada

Hasta ahora hemos visto configuraciones globales que se aplican al conjunto de nuestra tienda, de manera que con un mínimo esfuerzo nuestra tienda mejorara ampliamente. Pero existen tareas que, si bien requerirán de mayor esfuerzo por nuestra parte, ayudarán todavía más a conseguir nuestros objetivos. ¿Donde mirar para esto? Básicamente en:

- **Catálogo -> Gestionar las categorías.** Aquí tenemos muchas opciones de configuración, desde la descripción de la categoría, pasando por los keywords, description, y la URL key.
- **Catálogo -> Gestionar los productos**. Las opciones aquí son similares, tenemos la URL key, y dentro de la pestaña Meta Information, tenemos Meta Title, Meta Keywords y Meta Description.
- **CMS -> Pages**. Nuevamente tenemos la página de Meta Datos, donde podremos configurar

los Meta Keywords y la Meta Description

MVC

http://es.wikipedia.org/wiki/Modelo_Vista_Controlador

Extensiones e Integraciones

Vamos ahora, en este último punto, a intentar ver brevemente los puntos del desarrollo de Magento mas orientados al desarrollo.

Magento Connect

Durante el curso ya hemos visto como trabajar con Magento Connect, como buscar, y como instalar las extensiones que necesitemos. Vale la pena, cuando necesitemos añadir alguna función a nuestra tienda, que primero realicemos una visita a Magento Connect.

Versiones y compatibilidades

Hay que tener en cuenta que no todas las extensiones son aplicables a todas las instalaciones de Magento. Cambios en el API pueden hacer que las extensiones no funcionen en absoluto, o no lo hagan correctamente. Como norma instalaremos extensiones que estén indicada para nuestra versión de Magento.

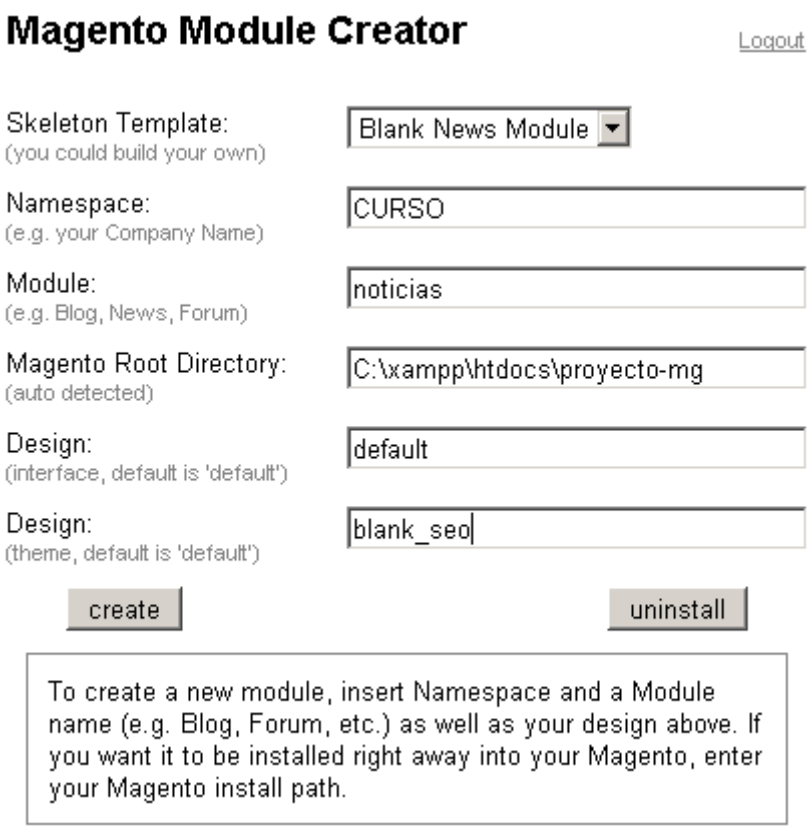
Introducción al desarrollo de extensiones Magento

La creación de extensiones para Magento es un proceso bastante más complejo que la creación de temas. Afortunadamente podemos instalar, vía Magento Connect, la extensión Module Creator de Daniel Nitz. Esta extensión crea la base necesaria para generar nuestros modulos de Magento.

Una vez instalada la extensión podemos acceder a ella a través de la URL:

http://www.localhost.com/proyecto-mg/moduleCreator/

Y utilizando nuestro usuario y contraseña, los mismos del panel de administración, veremos una pantalla similar a la siguiente:



De momento solo tenemos un esqueleto de plantilla, **Blank News Module**, el Namespace que queramos utilizar, el nombre del modulo, el interface y el tema que hay en uso. Haciendo clic en el botón **create** se generarán los ficheros base del modulo:

app/etc/modules/CURSO_Noticias.xml

app/code/local/CURSO/Noticias/Block/Noticias.php

app/code/local/CURSO/Noticias/controllers/IndexController.php

app/code/local/CURSO/Noticias/etc/config.xml

app/code/local/CURSO/Noticias/Model/Noticias.php

app/code/local/CURSO/Noticias/Model/Mysql4/Noticias.php

app/code/local/CURSO/Noticias/Model/Mysql4/Noticias/Collection.php

app/code/local/CURSO/Noticias/Model/Status.php

app/code/local/CURSO/Noticias/sql/noticias_setup/mysql4-install-0.1.0.php

app/design/frontend/default/blank_seo/layout/noticias.xml

app/design/frontend/default/blank_seo/template/noticias/noticias.phtml

app/code/local/CURSO/Noticias/Block/Adminhtml/Noticias.php

app/code/local/CURSO/Noticias/Block/Adminhtml/Noticias/Edit.php

app/code/local/CURSO/Noticias/Block/Adminhtml/Noticias/Grid.php

app/code/local/CURSO/Noticias/Block/Adminhtml/Noticias/Edit/Form.php

app/code/local/CURSO/Noticias/Block/Adminhtml/Noticias/Edit/Tabs.php

app/code/local/CURSO/Noticias/Block/Adminhtml/Noticias/Edit/Tab/Form.php

app/code/local/CURSO/Noticias/controllers/Adminhtml/NoticiasController.php

app/code/local/CURSO/Noticias/Helper/Data.php

app/design/adminhtml/default/blank_seo/layout/noticias.xml

Funciones importantes a la hora de crear extensiones

Vamos ahora a ver un poco por encima los diferentes métodos que nos ayudarán a trabajar con las extensiones.

getModel

Este método creará una instancia del modelo que le pasemos como parámetro, por ejemplo:

```
$noticia = Mage::getModel('noticias/noticias');
```

Esto cargará el modelo noticias, y luego podemos trabajar con el, por ejemplo cargando el elemento cuyo id es 1:

```
$noticia->load(1);
```

Y luego mostrando sus contenidos:

```
echo 'Title: '. $noticia->getTitle();
echo 'Content: ' . $noticia->getContent();
```

Esto podemos hacerlo directamente en el template, **app → design → frontend → default → f002 → template → noticias → noticias.phtml**:

```
    $noticia = Mage::getModel('noticias/noticias');
    $noticia->load(1);

    echo 'Title: '. $noticia->getTitle();
    echo 'Content: ' . $noticia->getContent();
```

O en caso de que lo hicieramos desde el controlador **app → code → local → Curso → Noticias → controllers → IndexController.php**:

```
  $noticia = Mage::getModel('noticias/noticias');
  $noticia->load(1);
  Mage::register('noticia_1', $noticia);
```

Podemos ver como usamos **Mage::register** para “registrar” una variable, y que esté disponible posteriormente, por ejemplo en **noticias.phtml**:

```
$noticia = Mage::registry('noticia_1');
echo $noticia->getTitle();
```

La variable la cogemos del registro con **Mage::registry**, y la podemos utilizar con normalidad, cada campo de la tabla accesible desde su método **getter**, en este caso **getTitle()**.

Para hacerlo todo aún más correcto podríamos incluso utilizar nuestro modelo **app → code → local → curso → noticias → model → noticias.php**, ahí podemos añadir un nuevo método:

```
public function getNoticia($id = 0)
{
    $noticia = $this->load($id);

    return $noticia;
}
```

Hay que notar que dentro del modelo usamos **$this** para hacer referencia al propio modelo. **El modelo está directamente relacionado con la tabla que lleve su propio nombre**. Después en el controlador haríamos lo siguiente:

```
$noticiam = Mage::getModel('noticias/noticias');
Mage::register('noticia_3', $noticiam->getNoticia(1));
```

Y en el template:

```
$noticia = Mage::registry('noticia_3');
echo $noticia->getTitle();
```

Repaso por los ficheros más importantes

Una vez tenemos todos los ficheros creados, vamos a examinarlos más en detalle. Empezaremos en **app → etc → modules**, ahí veremos un fichero llamado **CURSO_Noticias.xml**, desde este fichero podemos activar, o desactivar nuestro modulo:

```
<config>
  <modules>
    <CURSO_Noticias>
      <active>true</active>
```

```
        <codePool>local</codePool>
      </CURSO_Noticias>
   </modules>
</config>
```

Modificando la etiqueta active, a false, desactivará nuestro modulo y no se utilizará. El siguiente paso será examinar el controlador de nuestro modulo. El controlador es donde está situada toda la lógica de programación de nuestro modulo. Este fichero está situado en **app → code → local → CURSO → Noticias → controllers**, lo veremos bajo el nombre de **IndexController.php**

Por defecto el modulo viene rellenado con datos de ejemplo, comentados, lo descomentaremos de manera que tenga el siguiente aspecto:

```
<?php
class CURSO_Noticias_IndexController extends Mage_Core_Controller_Front_Action
{
   public function indexAction()
   {

          /*
           * Load an object by id
           * Request looking like:
           * http://site.com/noticias?id=15
           *  or
           * http://site.com/noticias/id/15
           */

                $noticias_id = $this->getRequest()->getParam('id');

                if($noticias_id != null && $noticias_id != '')          {
                        $noticias = Mage::getModel('noticias/noticias')->load($noticias_id)->getData();
                } else {
                        $noticias = null;
                }


                 /*
           * If no param we load a the last created item
           */

          if($noticias == null) {
                        $resource = Mage::getSingleton('core/resource');
```

```
                $read= $resource->getConnection('core_read');
                $noticiasTable = $resource->getTableName('noticias');

                $select = $read->select()
                  ->from($noticiasTable,array('noticias_id','title','content','status'))
                  ->where('status',1)
                  ->order('created_time DESC') ;

                $noticias = $read->fetchRow($select);
            }
            Mage::register('noticias', $noticias);



            $this->loadLayout();
            $this->renderLayout();
    }
}
```

Este controlador extiende a Mage_Core_Controller_Front_Action, y, en definitiva, al controlador de Zend Framework. De hecho, podemos ver métodos propios del Zend Framework:

```
$this->getRequest()->getParam('id');
```

Así como métodos propios de Magento:

```
Mage::getModel('noticias/noticias')->load($noticias_id)->getData();
```

Este controlador funcionará como controlador del **frontend**, pero también necesitaremos un controlador para la zona de administración, **app → code → local → CURSO → Noticias → controllers → Adminhtml** será el fichero llamado **NoticiasController.php** Este controlador permitirá gestionar las acciones de creación, edición, y borrado de noticias.

El siguiente archivo importante es **config.xml**, donde se guarda la configuración general de nuestro modulo, lo podemos encontrar en **app → code → local → CURSO → Noticias → etc**

Además del fichero de configuración, tambien podemos utilizar helpers, para modificar la presentación del modulo. Los helpers se situan en **app → code → local → CURSO → Noticias → Helper**, y de momento tendremos uno, vacío, llamado **Data.php**

Ahora que hemos visto el controlador, ficheros de configuración etc el siguiente paso es el modelo, los modelos realizan el acceso a datos, es decir, trabajan entre el controlador y nuestra base de datos. Se generan en **app → code → local → CURSO → Noticias → Model**, por ejemplo

Noticias.php

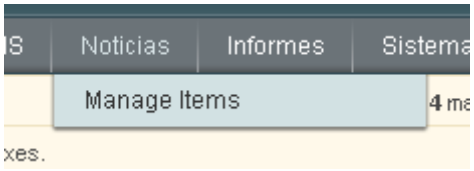
El modelo, sin embargo, solo realiza el acceso a datos, para crear nuestra tabla en base de datos, existe un fichero llamado **mysql4-install-0.1.0.php** situado en **app → code → local → CURSO → Noticias → sql → noticias_setup**, en ese fichero podemos ver la tabla que se generará, así como sus campos.

Por último necesitamos una plantilla para mostrar los contenidos, lo crearemos en **app → design -> frontend → default → blank_seo → template → noticias** , por ejemplo **noticias.phtml** Hay que notar la relación de nuestro modulo, con nuestra plantilla. Junto con la plantilla necesitaremos crear el bloque, en **app → code → local → CURSO → Noticias → Block** y llamado **Noticias.php**

En última instancia podremos acceder a nuestro modulo en esta URL:

http://www.localhost.com/proyecto-mg/noticias

Así como desde el panel de administración:



En caso de que el panel de administración no funcione

Dependiendo de la versión del module creator que estemos utilizando puede ser que al intentar acceder a la gestión de noticias desde el panel de administración, solo veamos una pantalla en blanco. Si esto sucede, para corregirlo iríamos a app → code → local → Curso → noticias → controllers → Adminhtml → NoticiasController.php y modificaremos el siguiente método:

```
public function indexAction() {
        $this->_initAction()
                ->renderLayout();
}
```

Para que quede tal que así:

```
public function indexAction() {
        $this->_initAction()
                ->_addContent($this->getLayout()->createBlock('noticias/adminhtml_noticias'))
                ->renderLayout();
}
```

Creación de un widget

Los widgets se añadieron en la versión 1.4 de Magento, y nos posibilitan añadir pequeños bloques en las páginas CMS sin necesidad de tener conocimientos de programación, en este ejemplo vamos a crear un widget básico. El widget, como el resto de módulos necesita de una estructurá de carpetas determinada. Vamos a crear esta carpeta en **app → code → local → Curso → Cursowidget** y dentro crearemos las siguientes carpetas:

- Block
- etc
- Helper
- Model

El siguiente paso es crear un fichero xml dentro de app → etc → modules, CURSO_Cursowidget.xml, con el siguiente contenido:

```
<config>
  <modules>
    <Curso_Cursowidget>
      <active>true</active>
      <codePool>local</codePool>
      <depends>
        <Mage_Cms />
      </depends>
    </Curso_Cursowidget>
  </modules>
</config>
```

Este el fichero que le indicará a Magento que nuestra instalación cuenta con un modulo más, es igual que los que hemos utilizado hasta ahora excepto por:

```
      <depends>
        <Mage_Cms />
      </depends>
```

Que indica su dependencia con Mage/Cms. Después dentro de **app → code → local → Curso → Cursowidget → etc** crearemos el fichero **config.xml**:

```
<?xml version="1.0"?>
<config>
  <modules>
    <Curso_Cursowidget>
      <version>0.0.1</version>
```

```
        </Curso_Cursowidget>
    </modules>
    <global>
        <helpers>
            <cursowidget>
                <class>Curso_Cursowidget_Helper</class>
            </cursowidget>
        </helpers>
        <blocks>
            <cursowidget>
                <class>Curso_Cursowidget_Block</class>
            </cursowidget>
        </blocks>
        <models>
            <cursowidget>
                <class>Curso_Cursowidget_Model</class>
            </cursowidget>
        </models>
    </global>
</config>
```

El siguiente fichero que crearemos será **app → code → local → Curso → Cursowidget → etc → widget.xml:**

```
<?xml version="1.0"?>
<widgets>
    <cursowidget_list type="cursowidget/list" translate="name description" module="cursowidget">
        <name>Widget de prueba</name>
        <description>Añade un texto</description>
        <parameters>
            <enabled_services>
                <label>Banners</label>
                <visible>1</visible>
                <required>1</required>
                <type>multiselect</type>
                <source_model>cursowidget/services</source_model>
            </enabled_services>
            <template translate="label">
                <label>Plantilla a utilizar</label>
                <visible>1</visible>
                <required>1</required>
```

```
            <type>select</type>
            <values>
               <text translate="label">
                  <value>cursowidget/list.phtml</value>
                  <label>Una opcion</label>
               </text>
               <icons translate="label">
                  <value>cursowidget/list.phtml</value>
                  <label>Otra opcion</label>
               </icons>
            </values>
         </template>
      </parameters>
   </cursowidget_list>
</widgets>
```

Crearemos tambien el fichero **app → code → local → Curso → Cursowidget → Helper → Data.php**:

```
<?php

class Curso_Cursowidget_Helper_Data extends Mage_Core_Helper_Abstract
{

}
```

Despues crearemos el fichero **app → code → local → curso → cursowidget → model → Services.php**:

```
<?php

class Curso_Cursowidget_Model_Services
{



   public function toOptionArray()
   {
      return array(
         array('value' => '1', 'label' => 'Banner 1'),
         array('value' => '2', 'label' => 'Banner 2')
      );
```

```
    }

}
```

Despues **app → code → local → curso → cursowidget → block → List.php**:

```
<?php

class Curso_Cursowidget_Block_List extends Mage_Core_Block_Template implements
Mage_Widget_Block_Interface
{

    /**
     * A model to serialize attributes
     *
     * @var Varien_Object
     */
    protected $_serializer = null;

    /**
     * Constructor
     */
    protected function _construct()
    {
        $this->_serializer = new Varien_Object();
        parent::_construct();
    }

    /**
     * Produce links list and render it as html
     *
     * @return string
     */
    protected function _toHtml()
    {
        $html = '';
        $config = $this->getData('enabled_services');
        if (empty($config)) {
            return $html;
        }
        $services = explode(',', $config);
```

```
        $list = array();
        foreach ($services as $service) {
            $item = $this->_generateServiceLink($service);
            if ($item) {
                $list[] = $item;
            }
        }
        $this->assign('list', $list);
        return parent::_toHtml();
    }

    protected function _generateServiceLink($service)
    {


        switch ($service) {
            case '1':
                $item = "Banner 1";
                break;
            case '2':
                $item = "Banner 2";
                break;
            default:
                return array();
                break;
        }

        return $item;
    }

}
```

Y por ultimo **app → design → frontend → default → f002 → template → cursowidget → list.phtml**:

```
<?php

foreach($list as $item){

    echo $item;
```

```
}
?>
```

Creación de un modulo de envio (Shipping module)

En el siguiente ejemplo vamos a ver los pasos básicos para crear un modulo de envio propio. El primer paso será crear una carpeta dentro de **app → code → local → Curso** que se llame **Enviosm**, con lo cual la ruta completa quedará **app → code → local → Curso → Enviosm**

Dentro de esta carpeta tendremos dos más, etc y model, y dentro de model, carrier:

app → code → local → Curso → Enviosm

- **etc**
- **Model**
  - **Carrier**

Dentro de la carpeta **etc** crearemos un fichero llamado **config.xml**, con el siguiente contenido:

```
<?xml version="1.0"?>
<config>
  <modules>
    <Curso_Enviosm>
      <version>1.0.0</version>
      <depends>
        <Mage_Shipping />
      </depends>
    </Curso_Enviosm>
  </modules>

  <global>
    <models>
      <Enviosm>
        <class>Curso_Enviosm_Model</class>
      </Enviosm>
    </models>
    <resources>
      <Enviosm_setup>
        <setup>
          <module>Curso_Enviosm</module>
        </setup>
        <connection>
```

```
            <use>core_setup</use>
        </connection>
    </Enviosm_setup>
</resources>
<sales>
    <shipping>
        <carriers>
            <enviosm>
                <class>Curso_Enviosm_Model_Carrier_ShippingMethod</class>
            </enviosm>
        </carriers>
    </shipping>
</sales>
</global>

<default>
    <carriers>
        <enviosm>
            <active>1</active>
            <sallowspecific>1</sallowspecific>
            <cutoff_cost>50</cutoff_cost>
            <model>curso_enviosm_model_carrier_shippingmethod</model>
            <name>Envios Magento</name>
            <name>Envios Magento</name>
            <specificerrmsg>This shipping method is currently unavailable. If you would like to ship using this shipping
method, please contact us.</specificerrmsg>
        </enviosm>
    </carriers>
</default>

</config>
```

El siguiente fichero que crearemos será **system.xml**, también dentro de la carpeta **etc**:

```
<?xml version="1.0"?>
<config>
<sections>
    <carriers>
        <groups>
            <enviosm translate="label" module="shipping">
```

```
<label>Envios Magento</label>
<frontend_type>text</frontend_type>
<sort_order>13</sort_order>
<show_in_default>1</show_in_default>
<show_in_website>1</show_in_website>
<show_in_store>1</show_in_store>

<fields>

  <active translate="label">
    <label>Enabled</label>
    <frontend_type>select</frontend_type>
    <source_model>adminhtml/system_config_source_yesno</source_model>
    <sort_order>1</sort_order>
    <show_in_default>1</show_in_default>
    <show_in_website>1</show_in_website>
    <show_in_store>1</show_in_store>
  </active>

  <handling_type translate="label">
    <label>Calculate Handling Fee</label>
    <frontend_type>select</frontend_type>
    <source_model>shipping/source_handlingType</source_model>
    <sort_order>10</sort_order>
    <show_in_default>1</show_in_default>
    <show_in_website>1</show_in_website>
    <show_in_store>0</show_in_store>
  </handling_type>

  <handling_action translate="label">
    <label>Handling Applied</label>
    <frontend_type>select</frontend_type>
    <source_model>shipping/source_handlingAction</source_model>
    <sort_order>11</sort_order>
    <show_in_default>1</show_in_default>
    <show_in_website>1</show_in_website>
    <show_in_store>0</show_in_store>
  </handling_action>

  <handling_fee translate="label">
    <label>Handling fee</label>
```

```
    <frontend_type>text</frontend_type>
    <sort_order>12</sort_order>
    <show_in_default>1</show_in_default>
    <show_in_website>1</show_in_website>
    <show_in_store>1</show_in_store>
</handling_fee>

<sort_order translate="label">
    <label>Sort order</label>
    <frontend_type>text</frontend_type>
    <sort_order>100</sort_order>
    <show_in_default>1</show_in_default>
    <show_in_website>1</show_in_website>
    <show_in_store>1</show_in_store>
</sort_order>

<title translate="label">
    <label>Title</label>
    <frontend_type>text</frontend_type>
    <sort_order>2</sort_order>
    <show_in_default>1</show_in_default>
    <show_in_website>1</show_in_website>
    <show_in_store>1</show_in_store>
</title>

<sallowspecific translate="label">
    <label>Ship to applicable countries</label>
    <frontend_type>select</frontend_type>
    <sort_order>90</sort_order>
    <frontend_class>shipping-applicable-country</frontend_class>
    <source_model>adminhtml/system_config_source_shipping_allspecificcountries</source_model>
    <show_in_default>1</show_in_default>
    <show_in_website>1</show_in_website>
    <show_in_store>1</show_in_store>
</sallowspecific>

<specificcountry translate="label">
    <label>Ship to Specific countries</label>
    <frontend_type>multiselect</frontend_type>
    <sort_order>91</sort_order>
    <source_model>adminhtml/system_config_source_country</source_model>
```

```
                <show_in_default>1</show_in_default>
                <show_in_website>1</show_in_website>
                <show_in_store>1</show_in_store>
            </specificcountry>

            <showmethod translate="label">
                <label>Show method if not applicable</label>
                <frontend_type>select</frontend_type>
                <sort_order>92</sort_order>
                <source_model>adminhtml/system_config_source_yesno</source_model>
                <show_in_default>1</show_in_default>
                <show_in_website>1</show_in_website>
                <show_in_store>1</show_in_store>
            </showmethod>

            <specificerrmsg translate="label">
                <label>Displayed Error Message</label>
                <frontend_type>textarea</frontend_type>
                <sort_order>80</sort_order>
                <show_in_default>1</show_in_default>
                <show_in_website>1</show_in_website>
                <show_in_store>1</show_in_store>
            </specificerrmsg>
        </fields>
    </enviosm>
  </groups>
 </carriers>
</sections>
</config>
```

Y por último crearemos el fichero **Model -> Carrier -> ShippingMethod.php**, dentro de la carpeta **Model -> Carrier**, será este fichero el que gestione la lógica de nuestro modulo de envio:

```
<?php


class Curso_Enviosm_Model_Carrier_ShippingMethod extends Mage_Shipping_Model_Carrier_Abstract
{

    protected $_code = 'enviosm';
```

```
    public function collectRates(Mage_Shipping_Model_Rate_Request $request)
    {
        if (!$this->getConfigData('active')) {
            Mage::log('The ' . $this->_code . 'my shipping module is not  active.');
            return false;
        }

        $handling = $this->getConfigData('handling');
        $result = Mage::getModel('shipping/rate_result');

        foreach ($response as $method) {
            $rMethod = Mage::getModel('shipping/rate_result_method');

            $method->setCarrier($this->_code);
            $method->setCarrierTitle($this->getConfigData('title'));

            $method->setMethod($method['code']);
            $method->setMethodTitle($method['title']);

            $method->setCost($method['amount']);

            $method->setPrice($method['amount'] + $handling);

            $result->append($method);
        }

        return $result;
    }

}
```

Una vez que tenemos todo esto, nos queda generar el fichero **app** -> **etc** -> **modules** -> **CURSO_Enviosm.xml** con el siguiente contenido:

```
<config>
    <modules>
        <Curso_Enviosm>
            <active>true</active>
            <codePool>local</codePool>
        </Curso_Enviosm>
    </modules>
```

</config>

Nos quedará activar el modulo en **Sistema** -> **configuración** -> **avanzado** -> **avanzado**

Y despues configurar sus opciones en **Sistema** -> **configuración** -> **ventas** -> **metodos de envio**

API de Magento

El desarrollo de extensiones de Magento involucra el uso de funciones ya existentes, que Magento nos proporciona a través de su API, para saber más sobre el API podemos visitar esta página, que realmente es una de las mejor documentadas de Magento:

http://www.magentocommerce.com/support/magento_core_api

Para poder trabajar con el api primero tendremos que permitir el acceso a los scripts al api, para ello iremos a **Sistema → Servicios web → roles**. Haremos clic en **añadir nuevo rol**. Lo llamaremos **Acceso** por ejemplo. Y en la pestaña de **Recursos del rol** elegiremos **Acceso de recursos → Todos**. **Guardaremos** el rol.

Luego iremos a **Sistema → Servicios web → usuarios** y haremos clic en **Añadir nuevo usuario**. En la pestaña **info del usuario** escribiremos los siguientes valores:

- nombre del usuario → usu
- nombre → usu
- apellido → usu
- correo electronico → usu@gmail.com
- api key → usu123
- api key confirmation → usu123
- la cuenta es → activo

Y en la pestaña **rol del usuario** elegiremos **Acceso**. Guardamos.

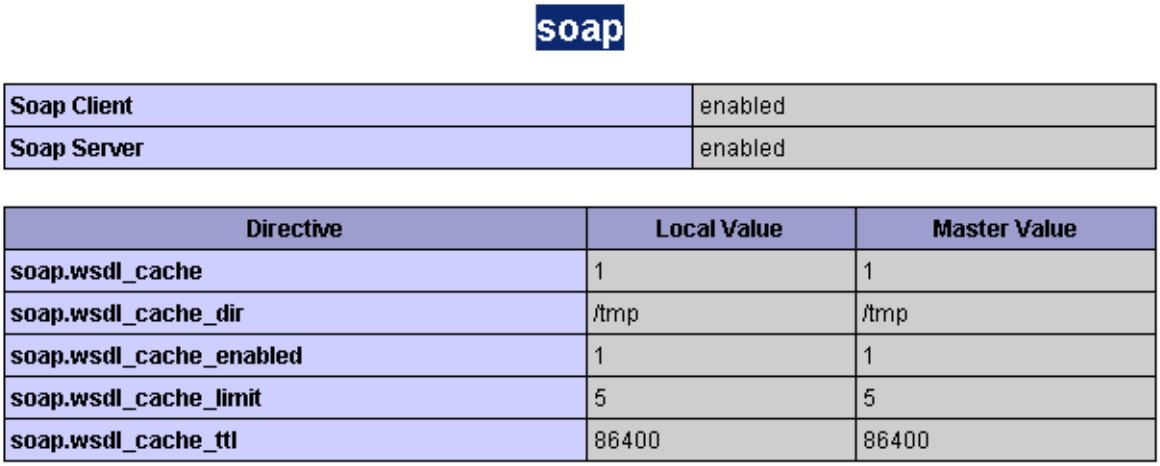
Para trabajar con el API de magento podemos utilizar dos protocolos:

- SOAP
- XML RPC

En este ejemplo vamos a usar SOAP, y para comprobar que esta activo, crearemos un fichero php que incluya el siguiente codigo:

```
<?php
phpinfo();
?>
```

Deberemos ver una imagen similar a la siguiente:



soap

Soap Client	enabled
Soap Server	enabled

Directive	Local Value	Master Value
soap.wsdl_cache	1	1
soap.wsdl_cache_dir	/tmp	/tmp
soap.wsdl_cache_enabled	1	1
soap.wsdl_cache_limit	5	5
soap.wsdl_cache_ttl	86400	86400

Y crearemos por ejemplo, en una carpeta separada de la de magento un fichero api.php:

```
<?php

$client = new SoapClient('http://www.localhost.com/proyecto-mg/api/soap/?wsdl');

$session = $client->login('usu', 'usu123');

/*Crear un usuario*/
/*
$customerInfo = array(
    'firstname'  => 'First',
    'lastname'   => 'Last',
    'email'      => 'test@example.com',
    'password_hash'   => md5('password'),
    'store_id'   => 0,
    'website_id' => 0
);
```

```
$newCustomerId = $client->call($session, 'customer.create', array($customerInfo));
*/



/*Mostrar un cliente*/
/*
$customerId = 2;
$customerInfo = $client->call($session, 'customer.info', $customerId);
print var_dump($customerInfo);
*/



$customerId = 4;
$newCustomerInfo = array(
   'firstname'  => 'Updated',
   'lastname'   => 'Customer'
);

$client->call($session,
                        'customer.update',
                        array($customerId, $newCustomerInfo)
                        );



$client->endSession($session);

?>
```

Herramientas

A la hora de trabajar con Magento, podemos hacer uso de un gran número de herramientas que nos faciliten el desarrollo. Ahora vamos a nombrar algunas de ellas.

XHTML / CSS

Todos los temas de Magento se crean utilizando una mezcla de XHTML, CSS, JavaScript y pequeñas llamadas a funciones de Magento. Realmente cualquier editor nos serviría para crear nuestras plantillas, desde editores simples como notepad++ (http://notepad-plus-plus.org/), a grandes editores como Dreamweaver.

En cuanto a navegadores, Firefox ofrece un gran número de extensiones, como:

- Firebug: https://addons.mozilla.org/es-es/firefox/addon/1843/
- Yslow: http://developer.yahoo.com/yslow/
- Barras de desarrollo

Y mucho más, con todo lo cual, nuestro desarrollo será más sencillo.

Magento y Zend Studio

El uso de un IDE como Zend Studio, con Magento, puede realmente ayudarnos con el desarrollo. Si bien para trabajar con temas no sería necesario, para desarrollar extensiones si sería de gran utilidad, aunque no cuenta con automatizaciones que generen extensiones, o su base, de forma automática.

Otra opción a tener en cuenta sería utilizar un IDE como NetBeans.

Magento y Zend Application Server

Si necesitamos que nuestra instalación de Magento funcione mejor y de forma más confiable, podemos instalar Magento sobre el Zend Application Server (http://www.zend.com/en/solutions/packaged-php-applications/zend-server-magento), esto nos permitirá mejorar el rendimiento de nuestra instalación de Magento, así como realizar diagnosticos de cuellos de botella y errores.

Conceptos avanzados sobre Layouts

Como ya hemos visto anteriormente, los temas de Magento se dividen en tres partes:

- Layout
- Template
- Locale

En locale encontraremos los ficheros de traducción, y, quizá, sea en template donde encontremos ficheros que contengan algo a lo que estamos más acostumbrado, código html. Sin embargo, es está tercera parte, layout, la que quizá nos cause un poco más de dificultad para entender su funcionamiento.

En este apartado vamos a avanzar un poco más en el concepto de layout, e intentar aclarar todas las dudas que podamos tener.

Tal y como vimos, los ficheros de layout se encuentran en:

- **app** -> **design** -> **frontend** -> **interfaz** -> **tema** -> **layout**

En realidad, cada modulo de Magento, puede definir cual será su fichero de layout. ¿Donde podemos ver de que modulos de Magento disponemos? Bien, en realidad lo podemos ver en tres

sitios, si navegamos a **app → code**, veremos estos tres directorios:

- **community** → donde podemos encontrar modulos que hayamos instalado en Magento a través de Magento Connect. Son modulos de third-parties.
- **core** → (mage) aquí podemos ver los modulos que vienen instalados en Magento por defecto, y, en realidad, son necesarios para el correcto funcionamiento de la instalación.
- **local** → normalmente cuando desarrollemos nuestros propios modulos, los crearemos dentro de esta carpeta.

Por ejemplo, tomemos el modulo que hemos creado anteriormente, el modulo de **noticias**, que podemos encontrar dentro de local, en el espacio **CURSO**. Dentro de **noticias → etc → config.xml**

Este fichero de configuración contiene los parámetros de configuración del modulo, entre ellos que layout utilizará. Esto lo podemos ver en:

```
<frontend>
    <routers>
        <noticias>
            <use>standard</use>
            <args>
                <module>CURSO_Noticias</module>
                <frontName>noticias</frontName>
            </args>
        </noticias>
    </routers>
    <layout>
        <updates>
            <noticias>
                <file>noticias.xml</file>
            </noticias>
        </updates>
    </layout>
</frontend>
```

Hay que notar como define el layout **noticias.xml**, y este fichero de layout lo podemos encontrar dentro de un tema, por ejemplo **app → design → frontend → default → blank_seo → layout → noticias.xml**:

```
<layout version="0.1.0">
    <default>
    </default>
    <noticias_index_index>
```

```
        <reference name="content">
            <block type="noticias/noticias" name="noticias" template="noticias/noticias.phtml" />
        </reference>
    </noticias_index_index>
</layout>
```

Y ya dentro del layout, indicamos que template va a utilizar, y donde lo encontrará, en este caso **template → noticias → noticias.phtml**

Durante el proceso de carga de cualquier página, Magento utilizará todos los ficheros de layout necesarios para montar toda la estructura y en que posición deberá aparecer cada modulo.

Layout Handles

Los handles, o manejadores, son los elementos que indican que parte del fichero de layout se debe de cargar en cada página. Normalmente el handle a utilizar se sabe en base al controlador, y a la acción del mismo, tomemos como ejemplo nuestro layout de noticias:

```
<?xml version="1.0"?>
<layout version="0.1.0">
    <default>
    </default>
    <noticias_index_index>
        <reference name="content">
            <block type="noticias/noticias" name="noticias" template="noticias/noticias.phtml" />
        </reference>
    </noticias_index_index>
</layout>
```

En este caso podemos ver dos handles, por un lado **default**, y por otro, **noticias_index_index**. El handle **default** se carga siempre, en todas las páginas. Y, el otro handle, noticias_index_index solo en el caso de que se esté utilizando el modulo noticias, con su controlador index, en su acción index.

¿Cuando se cargará este handle? Por ejemplo cuando se llame a esta URL:

http://www.localhost.com/proyecto-mg/noticias

Vemos por un lado el modulo, noticias, pero no tenemos un controlador, ni una acción, ese caso, podría ser el siguiente caso:

http://www.localhost.com/proyecto-mg/catalogsearch/term/popular/

En este caso vemos como se cargaría el modulo **catalogsearch**, controlador **term**, acción **popular**. Si indagamos un poco en **app → design → frontend → base → default → layout → catalogsearch.xml**, podemos navegar y encontrar el bloque html donde se especifica el handle para dicha página:

```
<catalogsearch_term_popular translate="label">
    <label>Popular Search Terms</label>
    <remove name="right"/>
    <remove name="left"/>

    <reference name="root">
        <action method="setTemplate"><template>page/1column.phtml</template></action>
    </reference>
    <reference name="content">
        <block type="catalogsearch/term" name="seo.searchterm" template="catalogsearch/term.phtml"/>
    </reference>
</catalogsearch_term_popular>
```

Layout elements

Cada layout handle puede estar formado por los siguientes elementos:

- **label** → introducido a partir de la versión 1.4 de Magento, se utiliza como descripción en algunas partes de la zona de administración de Magento.
- **Reference** → este elemento se utiliza para enlazar con bloques anteriormente definidos en cualquier fichero de layout. Para añadirle más bloques, modificarlos etc El elemento reference debe tener un atributo name que se refiera al bloque ya existente.

  Esto se ve más fácilmente con un ejemplo, veamos, si miramos dentro de **page.xml** (de blank_seo template por ejemplo), podemos encontrar el siguiente código:

  ```
  <block type="core/text_list" name="content" as="content" translate="label"> <label>Main Content Area</label>
  </block>
  ```

  Y dentro de **noticias.xml** podemos ver:

  ```
  <reference name="content">
      <block type="noticias/noticias" name="noticias" template="noticias/noticias.phtml" />
  </reference>
  ```

  De manera que desde noticias.xml estamos añadiendo más bloques a los que ya se cargaban dentro del bloque de page.xml con nombre igual a content.
- **Block** → este elemento se utiliza para definir un nuevo bloque. Debe tener un atributo **name**, y un atributo **type** que viene dado por el nombre de la clase del bloque. Además

puede tener el atributo **template** (con algunas restricciones) que indicará el template a utilizar.

```
<block type="noticias/noticias" name="noticias" template="noticias/noticias.phtml" />
```

- action → define una acción para que sea ejecutada en el bloque referenciado, o creado. Equivale a un método de la instancia del bloque que se está ejecutando. Por ejemplo:

```
<catalog_product_view>
<reference name="root">
<action method="setTemplate">
<template>page/1column.phtml</template>
</action>
</reference>
```

En este caso estaríamos cambiando el template de root, definiendo que ahora debe usar 1column.phtml

Existen otros elementos, como **remove y update**. Pero nos quedaremos con los anteriores por ser los más comunes.

El proceso de interpretación (rendering)

Durante el proceso de carga e interpretación de las páginas se cargán todos los bloques, bloques anidados con sus elementos hijos. Para que un bloque salga por pantalla debe tener el atributo **output**, por ejemplo, tal y como lo encontramos en el layout **page.xml**:

```
<block type="page/html" name="root" output="toHtml" template="page/3columns.phtml">
```

En el layout page.xml este bloque se encuentra en la posición de bloque principal, por lo tanto todos los bloques hijos serán renderizados también, no será necesario añadirles el atributo output a todos los bloques hijos.

Tipos de bloque

Los tipos de bloque más comunes son los siguientes:

- **core/template** → este bloque muestra un template definido por su atributo **template**. La mayoria de bloques definidos en el layout son de tipo, o subtipo **core/template**.
- **page/html** → es un subtipo de core/template y define el bloque raiz (root). Todos los demas bloques son bloques hijos de este bloque. Como podemos ver en layout.xml:

  ```
  <block type="page/html" name="root" output="toHtml" template="page/3columns.phtml">
  ```

- **page /html_head** → define la seccion head de HTML, conteniendo los elementos JavaScript, CSS etc
- **page/html_header** → define la parte de la página que contendrá el logo, links de la parte

superior etc

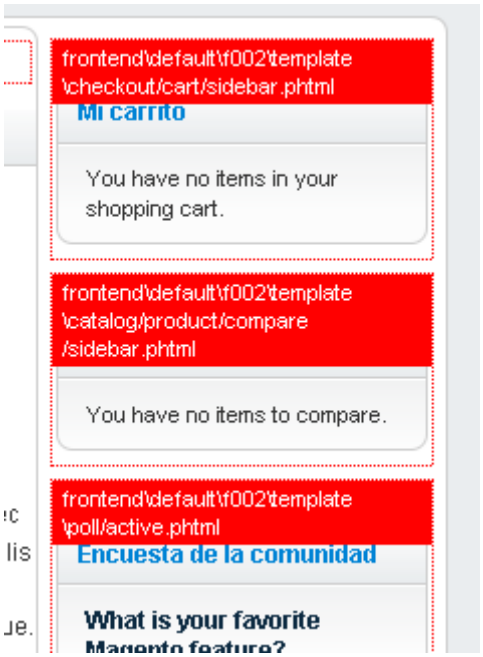
- **page/template_links** → este bloque muestra una lista de links, visible en el pie y la cabecera.
- **core/text_list** → algunos bloques, como content, left, right son de tipo core/text_list. Cuando estos bloques son mostrados, todos sus bloques hijos son mostrados automáticamente, sin necesidad de utilizar el método getChildHtml().
- **page/html_wrapper** → este bloque se utiliza para crear un bloque contenedor, que muestra sus bloques hijo dentro de una etiqueta HTML definida en la acción setHtmlTagName. Por defecto utiliza un
- **page/html_breadcrumbs** → muestra los breadcrumbs de la página.
- **page/html_footer** → muestra el area del pie, que contiene los links del pie, el mensaje del copyright etc
- **core/messages** → este bloque muestra mensajes de error, noticias informativas etc
- **page/switch** → este bloque puede ser utilizado para mostrar el selector de idioma, tienda etc

Ejercicios

Vamos ahora a hacer unas pequeñas pruebas que resuman un poco todos estos conceptos, y nos ayuden a comprender mejor la lógica que relaciona todos los elementos.

Localizar donde están los elementos de la columna derecha

Si miramos las ayudas de magento no se ve que estén dentro de un template mayor, veamos una imagen:



Podemos ver varios modulos, ¿Podemos esperar encontrar un template **right.phtml** con instrucciones como **<?php echo $this->getChildHtml('sidebar') ?>** ? Eso es lo que tenemos que ver ahora.

Solución

Como hemos comentado antes, las columnas son de tipo **core/text_list** lo que quería decir que los bloques hijos de este bloque se cargarán sin necesidad de tener que hacer llamadas getChildHtml. Si vemos por ejemplo **app → design → frontend → default → f002 → layout → page.xml**, si buscamos encontraremos el siguiente código:

```
<block type="core/text_list" name="right" as="right" translate="label">
    <label>Right Column</label>
</block>
```

Vemos como se define el bloque, pero no hay nada en su interior. Ahora veamos **app → design → frontend → default → f002 → template → page → 3columns.phtml**, aquí tenemos el siguiente código:

```
<div class="col-right sidebar"><?php echo $this->getChildHtml('right') ?></div>
```

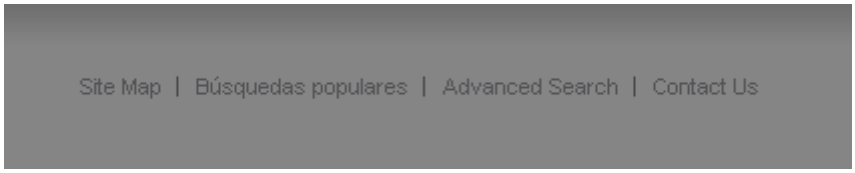
No será necesario que busquemos algo similar a right.phtml, ya que no existe. Pensemos otra cosa. Hemos dicho que podemos añadir bloques de un layout a otro anteriormente creado, referenciandolo. Si hacemos una búsqueda de **reference name="right"**, veremos que aparece en varios layouts, por ejemplo en **app → design → frontend → default → f002 → layout → tag.xml**:

```
<default>
    <!-- Mage_Tag -->
    <reference name="right">
        <block type="tag/popular" name="tags_popular" template="tag/popular.phtml"/>
    </reference>

</default>
```

Aquí vemos como hacemos referencia (**reference**) al bloque anteriormente definido con nombre right, y le añadimos un bloque, de tipo **tag/popular**, que utilizará el template **tag/popular.phtml**. Y así con el resto de layouts que además añaden bloques al bloque **right** de **page.xml**

Localizar donde se crean los enlaces del pie

Si echamos un vistazo al tema que estamos utilizando, en el pie podremos ver la siguiente lista de enlaces:



¿De donde salen estos enlaces? Esa es la tarea, ¡vamos localizarlos!

Solución

En realidad estos enlaces tienen muchas fuentes de origen, así, a primera vista, podemos pensar que estarían en **CMS → Bloques estáticos**, ahí tenemos un bloque llamado **Footer Links**, si abrimos ese bloque, solo veremos dos enlaces:

- About Us
- Customer Service

¡Pero estos enlaces ni siquiera aparecen en el pie! Vamos a indagar un poquito en nuestro template, el primer vistazo, por lógica sería en un layout, por ejemplo **app → design → frontend → default → f002 → layout → page.xml**, navegando por el fichero podemos encontrar el siguiente código:

```
        <block type="page/html_footer" name="footer" as="footer" template="page/html/footer.phtml">
            <block type="page/html_wrapper" name="bottom.container" as="bottomContainer" translate="label">
                <label>Page Footer</label>
                <action method="setElementClass"><value>bottom-container</value></action>
            </block>
            <block type="page/switch" name="store_switcher" as="store_switcher"
template="page/switch/stores.phtml"/>
            <block type="page/template_links" name="footer_links" as="footer_links"
template="page/template/links.phtml"/>
        </block>
```

Por un lado vemos el bloque que generará la zona del pie **page/html_footer**, este bloque indica el uso de un template, **footer.phtml**, si abrimos este fichero, podemos ver el siguiente código:

```
<div class="informational">
    <?php echo $this->getChildHtml('footer_links') ?>
    <?php echo $this->getLayout()->createBlock('cms/block')->setBlockId('payments')->toHtml();?>
</div>
```

Aquí estamos cargando footer_links, que había sido definido dentro del bloque **page/html_footer,** como un bloque hijo:

```
        <block type="page/template_links" name="footer_links" as="footer_links"
template="page/template/links.phtml"/>
```

Entonces, cuando llamemos a getChildHtml('footer_links'), se cargará el fichero page/template/links.phtml, ¿Estarán ahí nuestros enlaces? Veamoslo, abriremos el fichero **app → design → frontend → base → default → template → page → template → links.phtml**:

```
<?php $_links = $this->getLinks(); ?>
<?php if(count($_links)>0): ?>
<ul class="links"<?php if($this->getName()): ?> id="<?php echo $this->getName() ?>"<?php endif;?>>
   <?php foreach($_links as $_link): ?>
      <li<?php if($_link->getIsFirst()||$_link->getIsLast()): ?> class="<?php if($_link->getIsFirst()): ?>first<?php endif;
?><?php if($_link->getIsLast()): ?> last<?php endif; ?>"<?php endif; ?> <?php echo $_link->getLiParams() ?>><?php
echo $_link->getBeforeText() ?><a href="<?php echo $_link->getUrl() ?>" title="<?php echo $_link->getTitle() ?>" <?
php echo $_link->getAParams() ?>><?php echo $_link->getLabel() ?></a><?php echo $_link->getAfterText() ?></li>
   <?php endforeach; ?>
</ul>
<?php endif; ?>
```

Pues no, no encontramos los links aquí, sin embargo tenemos un código que realiza un bucle, y muestra unos enlaces. ¿Entonces estos links provienen de la base de datos? No parece existir en el panel de administración ninguna zona donde insertar y gestionar links. Vamos a retroceder un poco, concretamente volveremos al fichero app → design → frontend → default → f002 → layout → page.xml:

```
        <block type="page/html_footer" name="footer" as="footer" template="page/html/footer.phtml">
           <block type="page/html_wrapper" name="bottom.container" as="bottomContainer" translate="label">
              <label>Page Footer</label>
              <action method="setElementClass"><value>bottom-container</value></action>
           </block>
           <block type="page/switch" name="store_switcher" as="store_switcher"
template="page/switch/stores.phtml"/>
           <block type="page/template_links" name="footer_links" as="footer_links"
template="page/template/links.phtml"/>
        </block>
```

Vemos que el bloque carga los links es de tipo **page/template_links**, como es de tipo page estará situado en **app → code → core → Mage → Page**, luego el tipo nos indica template, así que si miramos dentro de la carpeta → **Page → Block**, veremos una carpeta llamada **template**, y dentro un fichero llamado **Links.php**, abramos dicho fichero.

Si recordamos dentro de nuestro fichero links.phtml teníamos el siguiente código:

```
<?php $_links = $this->getLinks(); ?>
```

Y en links.php podemos ver un método getter:

```
  public function getLinks()
  {
```

```
    return $this->_links;
}
```

Que devuelve el valor del atributo protegido $_links:

```
protected $_links = array();
```

Si indagamos un poco más por el fichero links.php no veremos ningún método desde el que se pudiera acceder a la base de datos y recoger links, pero si veremos un metodo add:

```
public function addLink($label, $url='', $title='', $prepare=false, $urlParams=array(),
    $position=null, $liParams=null, $aParams=null, $beforeText='', $afterText='')
```

Si buscásemos addLink, veríamos que aparece en gran número de layouts, por ejemplo en **app → design → frontend → base → default → layout → contacts.xml**:

```
    <reference name="footer_links">
        <action method="addLink" translate="label title" module="contacts"
ifconfig="contacts/contacts/enabled"><label>Contact Us</label><url>contacts</url><title>Contact
Us</title><prepare>true</prepare></action>
    </reference>
```

Vemos como hace referencia al bloque footer_links, de manera que el código que haya dentro acabará ejecutándose en aquel bloque. En este caso es un bloque de tipo **action**, que ejecutará el método addLink, el modulo es contacts (contacts.xml), y le indicamos el label, la url, el title etc

Cuando se está renderizando la página, se ejecutan todos estos trocitos de xml que van añadiendo links, que más tarde serán mostrados. Un poco enrevesado, pero es un buen ejemplo de la relación de todos los elementos de Magento.

De todas maneras, nos queda por aclarar que pasa con el bloque de **footer links** que había en **CMS –> Bloques estáticos**. Imaginemos que queremos añadir ese bloque, en nuestro template, por ejemplo en el fichero footer.phtml, justo después de los otros links:

```
<div class="informational">
    <?php echo $this->getChildHtml('footer_links') ?>
    <?php echo $this->getLayout()->createBlock('cms/block')->setBlockId('payments')->toHtml();?>
</div>
```

Lo haríamos tal que así:

```
<div class="informational">
    <?php echo $this->getChildHtml('footer_links') ?>
<?php echo $this->getLayout()->createBlock('cms/block')->setBlockId('footer_links')->toHtml();?>
    <?php echo $this->getLayout()->createBlock('cms/block')->setBlockId('payments')->toHtml();?>
```

```
</div>
```

De manera que, además de recoger los links insertados en los layouts, también mostraria los links del bloque estático.

Añadir un bloque con una imágen a la columna izquierda

En este pequeño ejercicio vamos a añadir una pequeña imagen en la columna izquierda , cuando estemos dentro de las categorías de catálogo, por ejemplo en:

http://www.localhost.com/proyecto-mg/furniture.html

Para añadir esta imagen vamos a crear un nuevo bloque, que llame a un template phtml, y, en última instancia muestre la imágen.

Solución

En este caso, como queremos que el banner o imagen, se muestre cuando estemos en las páginas de catálogo, trabajaremos en el fichero **app → design → frontend → default → f002 → layout → catalog.xml**

Vamos a buscar la etiqueta <default>, y añadiremos ahí nuestro bloque, podemos basarnos en bloques de otros banners como ejemplo, y colocaremos dentro de <reference name="left"> un código similar al siguiente:

```
        <block type="core/template" name="store_banner" as="store_banner"
template="page/html/banner.phtml">
            <action method="setImgSrc"><src>images/media/col_left_callout-1.jpg</src></action>
            <action method="setImgAlt" translate="alt" module="catalog"><alt>Our customer service is available 24/7.
Call us at (800) DEMO-NUMBER.</alt></action>
            <action method="setLinkUrl"><url>checkout/cart</url></action>
        </block>
```

Aquí estamos definiendo el bloque de **tipo** core/template, que nos servirá para crear bloques de html simple. El siguiente parámetro es el **name**, que nos servirá por si posteriormente necesitamos referenciar el bloque. El parametro **as**, en caso de que queramos invocar el bloque con getChildHtml, y, por último el **template** a utilizar.

Si miramos el template, **app → design → frontend → default → f002 → template → page → html → banner.phtml**:

```
banner 1

<div class="block block-banner">
   <?php if (strtolower(substr($this->getLinkUrl(),0,4))==='http'): ?>
      <a href="<?php echo $this->getLinkUrl() ?>">
   <?php elseif($this->getLinkUrl()): ?>
```

```
        <a href="<?php echo $this->getUrl($this->getLinkUrl()) ?>">
    <?php endif ?>
    <img src="<?php echo $this->getSkinUrl($this->getImgSrc()) ?>" width="195" alt="<?php echo $this->__($this->getImgAlt()) ?>" style="display:block;" />
    <?php if ($this->getLinkUrl()): ?>
    </a>
    <?php endif ?>
</div>
```

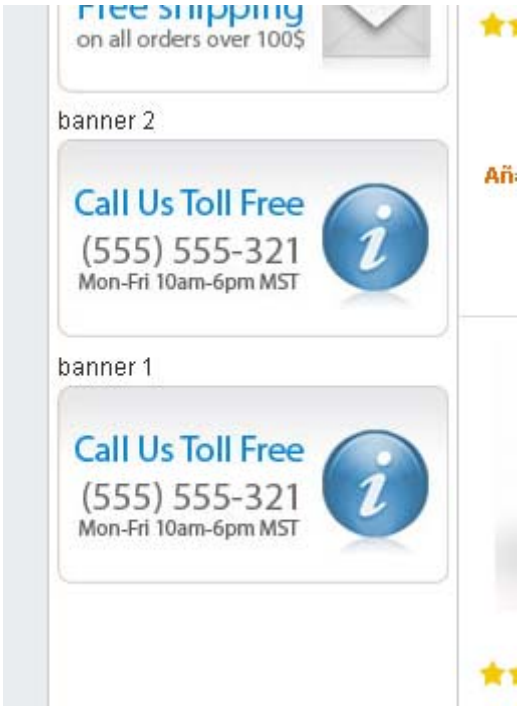
Con esto ya tendríamos nuestro banner creado, pero, no habría una forma más sencilla de crearlo, ya que solo queremos mostrar una imágen. Volvamos a catalog.xml, justo debajo del bloque que acabamos de crear, añadiremos el siguiente:

```
<block type="core/template" before="store_banner" name="store_banner2" as="store_banner2" template="page/html/banner2.phtml"/>
```

Mucho más corto, y con una novedad, **before="store_banner"**, con esto le indicamos que este bloque, deberá aparecer sobre el bloque con **name=”store_banner”**. Y, si la definición del bloque es más corta, que pasará con el template? Veamoslo:

```
banner 2

<div class="block block-banner">
    <img src="<?php echo $this->getSkinUrl() ?>images/media/col_left_callout-1.jpg" width="195" alt="" style="display:block;" />
</div>
```

Simplificado también, ¿Y el resultado? Veamoslo:



Justo lo que queríamos.

Añadir un bloque de noticias

En este ejercicio vamos a ver como podemos añadir un bloque de noticias en nuestra columna izquierda, visible cuando los visitantes accedan a la pantalla de catálogo, por ejemplo:

http://www.localhost.com/proyecto-mg/furniture.html

Más o menos debe quedar así:

Solución

La creación de este bloque se divide en varios pasos:

- Primero, crear el bloque, lo haremos en **app → code → local → curso → noticias → block → lista.php** y tendrá el siguiente contenido:

```
<?php
class Curso_Noticias_Block_Lista extends Mage_Core_Block_Template
{
        public function _prepareLayout()
    {
```

```
                return parent::_prepareLayout();
    }

     public function getLista()
     {

        return Mage::getModel('noticias/noticias')->getCollection();

    }
}
```

- Segundo, preparar un template para el bloque, **app → design → frontend → default → f002 → template → noticias → lista.phtml**:

```
<h4>Listado de noticias</h4>
<?php

    $noticias = $this->getLista();
    foreach ($noticias as $item) {
        echo $item->getTitle();
        echo "<br/>";
        echo $item->getContent();
        echo "<br/><br/>";
    }

?>
```

- Tercero, invocar el modulo, desde **app → design → frontend → default → f002 → layout → noticias.xml**:

```
    <catalog_category_layered translate="label">
        <reference name="left">
            <block type="noticias/lista" name="noticias" as="noticias" template="noticias/lista.phtml"/>
        </reference>
    </catalog_category_layered>
```

- O desde app → design → frontend → default → f002 → layout → catalog.xml:

```
    <catalog_category_layered translate="label">
        ...
        </reference>
```

```
    <reference name="left">
      <block type="noticias/lista" name="noticias" as="noticias" template="noticias/lista.phtml"/>
    </reference>
  </catalog_category_layered>
```

Posibilitar que desde el bloque de noticias se acceda al detalle de la noticia, y utilizar un template de una sola columna

El primer paso que vamos a dar para solucionar este tema será crearnos un nuevo controlador, en este caso lo emplazaremos en **app** -> **code** -> **local** -> **curso** -> **noticias** -> **controllers** -> **DetalleController.php** con el siguiente contenido:

```
<?php
class Curso_Noticias_DetalleController extends Mage_Core_Controller_Front_Action
{
   public function verAction()
   {

      /* Listado de métodos */
      /*
      $read = Mage::getSingleton('core/resource')->getConnection('core_read');

      echo '<pre>';
      print_r(get_class_methods($read));
      echo '</pre>';
      */


                  $id = $this->getRequest()->getParam('id');

      $read = Mage::getSingleton('core/resource')->getConnection('core_read');

      $select = $read->select()->from('noticias', array('noticias_id','title','filename','content'))->where('noticias_id =?', $id);
```

```
        $noticia = $read->fetchRow($select);

        Mage::register('noticia', $noticia);



        $this->loadLayout();
        $this->renderLayout();
    }

}
```

Notar que la clausula **where** guarda el formato tipico de Zend framework, **('noticias_id =?', $id);**, de hecho conocer como trabaja Zend Framework para generar las consultas nos será útil:

http://framework.zend.com/manual/en/zend.db.select.html

También podriamos hacer queries de tipo RAW, es decir creando nosotros toda la query sin usar Active Record:

http://www.sycha.com/magento-database-models-helpers-raw-sql-queries

El siguiente paso será modificar el fichero **app** -> **design** -> **frontend** -> **default** -> **f002** -> **template** -> **noticias** -> **lista.phtml** y lo modificaremos para que los titulos de las noticias sean enlaces:

```
<h4>Listado de noticias</h4>
<?php

    $noticias = $this->getLista();
    foreach ($noticias as $item) {

        ?>
        <a href="<?php echo $this->getUrl('noticias/detalle/ver'); ?>id/<?php echo $item->getNoticias_id();?>">
        <?php
        echo $item->getTitle();
        echo "</a>";
        echo "<br/>";
        echo $item->getContent();
```

```
        echo "<br/><br/>";
    }

?>
```

Necesitaremos tambien un template de detalle, por ejemplo, **app** -> **design** -> **frontend** -> **default** -> **f002** -> **template** -> **noticias** -> **detalle.phtml** , con el siguiente contenido:

```
<h4>Detalle noticia</h4>

<?php

$noticia = Mage::registry('noticia');

echo 'Id: ' . $noticia['noticias_id'] . "<br/><br/>";
echo 'Title: ' . $noticia['title'] . "<br/><br/>";
echo 'Content: ' . $noticia['content'] . "<br/><br/>";



?>
```

El último paso será modificar el fichero app -> design -> frontend -> default -> f002 -> layout -> noticias.xml, añadiremos el código en negrita:

```
<?xml version="1.0"?>
<layout version="0.1.0">
    <default>
    </default>
    <catalog_category_layered translate="label">
        <reference name="left">
            <block type="noticias/lista" name="noticias" as="noticias" template="noticias/lista.phtml"/>
        </reference>
    </catalog_category_layered>

    <noticias_detalle_ver>
        <reference name="root">
            <action method="setTemplate">
                <template>page/1column.phtml</template>
            </action>
        </reference>
        <reference name="content">
```

```
        <block type="noticias/noticias" name="noticias" template="noticias/detalle.phtml" />
      </reference>
    </noticias_detalle_ver>

    <noticias_index_index>
      <reference name="content">
        <block type="noticias/noticias" name="noticias" template="noticias/noticias.phtml" />
      </reference>
    </noticias_index_index>
</layout>
```

Mini recetas

A continuación veremos algunos pequeños códigos que nos pueden ayudar a personalizar nuestra instalación de Magento.

Añadir boton Me gusta, de facebook

Esta receta es muy sencilla, bastará con que vayamos a **app → design → frontend → default → f002 → template → catalog → product → view.phtml** y añadamos, más o menos por la línea 56 el siguiente código:

```
        <!---Facebook like --->
        <?php $src = urlencode($this->helper("core/url")->getCurrentUrl()); ?>
        <iframe src="http://www.facebook.com/plugins/like.php?href=<?php echo $src; ?
>&amp;layout=button_count&amp;show_faces=true&amp;width=450&amp;action=like&amp;font=arial&amp;colorsc
heme=light&amp;height=21" scrolling="no" frameborder="0" style="border:none; overflow:hidden; width:450px;
height:21px;" allowTransparency="true"></iframe>
        <!---// --->
```

Muy sencillo.

Añadir un producto al carrito con una URL

Esta tambien es muy sencilla, bastará con buscar el id del producto desde el administrador de Magento y crear una URL Similar a la siguiente:

http://www.localhost.com/proyecto-mg/checkout/cart/add/product/**171**/qty/**1**

Añadir productos desde las categorias

Lamentablemente desde la pantalla de **Catálogo → Gestionar los productos**, no es posible separar los productos por categorías, y, además, desde la pantalla **Catálogo → Gestionar las categorías → Productos de la categoría** no es posible acceder a la edición de los productos. ¿Que solución tenemos pues? Bien, editaremos el fichero **app → code → core → Mage → AdminHtml → Block → Catalog → Category → Tab → Product.php** y ahí buscaremos el método siguiente:

protected function _prepareColumns()

añadiremos el siguiente código:

```
        'type'      => 'number',
        'index'     => 'position',
        'editable'  => !$this->getCategory()->getProductsReadonly()
        //'renderer'  => 'adminhtml/widget_grid_column_renderer_input'
    ));
    $this->addColumn('action',
        array(
            'header' => Mage::helper('catalog')->__('Action'),
            'width' => '50px',
            'type' => 'action',
            'getter' => 'getId',
            'actions' => array(
                array(
                    'caption' => Mage::helper('catalog')->__('Edit'),
                    'url' => array(
                        'base'=>'*/catalog_product/edit',
                        'params'=>array('store'=>$this->getRequest()->getParam('store'))
                    ),
                    'field' => 'id'
                )
            ),
            'filter' => false,
            'sortable' => false,
            'index' => 'stores',
    ));

    return parent::_prepareColumns();
}
```

Y ya está, con eso tendremos un nuevo enlace “Edit” que nos permitirá editar los productos desde la página de categorías.

Modificar la ruta del panel de administración

Normalmente para acceder a nuestro panel de administración de Magento accedemos a la siguiente URL:

http://www.localhost.com/proyecto-mg/index.php/admin

Pero tenemos la posibilidad de cambiarla en caso de que lo necesitemos, lo haremos desde el siguiente fichero **app → etc → local.xml**:

```
<admin>
   <routers>
      <adminhtml>
         <args>
            <frontName><![CDATA[admin]]></frontName>
         </args>
      </adminhtml>
   </routers>
</admin>
```

Modificando el texto resaltado en negrita modificaremos la ruta hacia nuestro panel de administración.

Borrar pedidos

Por defecto los pedidos que podemos ver en Ventas → pedidos no se pueden borrar, o para hacerlo debemos hacerlo desde la base de datos. Podemos instalar esta extensión:

http://www.magentocommerce.com/magento-connect/hedererjs/extension/4072/asperience_deleteallorders

Que nos permitirá borrar los pedidos cancelados, cerrados o terminados. Aunque para limitar riesgos es mejor utilizarlo solo en los pedidos de prueba. Elimina también los comentarios, y actualiza estadisticas e informes.

Webs utiles

- Wiki: http://www.magentocommerce.com/wiki/index/pages/
- Nettuts: http://net.tutsplus.com/?s=magento
- Dzone: http://www.dzone.com/links/search.html?query=magento&x=0&y=0
- http://themeforest.net/category/magento
- http://magsvento.se/magento-references/magento-layout-files-reference
- http://blog.chapagain.com.np/magento-how-to-select-insert-update-and-delete-data/
- http://sree.cc/magento-ecommerce-tips/database-concepts-how-to-write-a-query

Métodos de pago asociados a grupos de usuario

Por defecto no es posible asociar métodos de pago a determinados grupos de usuarios, pero, en el siguiente foro:

http://www.magentocommerce.com/boards/viewthread/6710/P0/

Indican que la siguiente extensión (PaymentFilter for Products and Customer Groups

):

http://www.magentocommerce.com/magento-connect/Rico+Neitzel/extension/764/paymentfilter-for-products-and-customer-groups

ofrece dicha posibilidad correctamente.