Universidad Pontificia Bolivariana

Guiá rápida de configuración de una tienda online usando la plataforma de E-commerce OXID

Autor:
Juan Diego Echeverri
c.c. 1.037.613.749
Id. 000174133
Tel. (+57) 300 615 2202
Programa. Ingeniería Electrónica
juaneche@msn.com

Director:
MSc. José Valentín A. Restrepo
c.c. 71.751.993
Id. 000008329
Tel. (+57) 300 620 1331
josev.restrepo@upb.edu.co

Facultad de Ingeniería Eléctrica y Electrónica
Universidad Pontificia Bolivariana
15 de noviembre de 2016
Medellín

Índice

1. PARTICIPANTES

Estudiante

Nombre: Juan Diego Echeverri Isaza

Facultad: Ingeniería Electrónica

Fecha de terminación de cursos: Noviembre de 2014

Director

Nombre: José Valentín Antonio Restrepo Laverde

Institución: Universidad Pontificia Bolivariana

Títulos: Ingeniero Electrónico / Magister en Finanzas

Fecha de terminación de pregrado: 1997

Fecha de terminación de posgrado: 2009

2. GLOSARIO

Framework: es una estructura conceptual y tecnológica de soporte definido, normalmente con artefactos o módulos concretos de software, que puede servir de base para la organización u desarrollo de software. Los frameworks tienen como objetivo principal, ofrecer una funcionalidad definida, auto contenida, siendo construidos usando patrones de diseño, y su característica principal es si alta cohesión y bajo acoplamiento.

E-commerce: También llamado comercio electrónico, es definido como cualquier forma de transacción o intercambio de información con fines comerciales en la que las partes interactúan utilizando tecnologías de la información y la comunicación, en lugar de hacerlo por intercambio o contacto físico directo.

Plugin: Un plugin es aquella aplicación que, en un programa informático, añade una funcionalidad adicional o una nueva característica al software. Lo habitual es que el plugin sea ejecutado mediante el software principal, con el que interactúa a través de una cierta interfaz.

Base de datos: es un banco de información que contienen datos relativos a diversas temáticas y categorizados de distinta manera, pero que comparten entre sí algún tipo de vínculo o relación que busca ordenarlos y clasificarlos en conjunto. El modelo utilizado en la actualidad es el postulado por Edgar Frank Codd, el modelo relacional, debido a su capacidad de representar problemas reales y administrar datos dinámicamente.

Servidor web: es un programa informático que procesa una aplicación del lado del servidor, realizando conexiones bidireccionales y/o unidireccionales, y síncronas o asíncronas con el cliente. El código recibido por el cliente suele ser compilado y ejecutado por un navegador web.

Frontend: es la parte del software que interactúa con el o los usuarios. En general, es responsable de recoger las entradas del usuario, como los valores en un formulario, y enviárselos al backend en el formato correcto para ser procesados.

Backend: es la parte del software que procesa las entradas que vienen desde el frontend. El backend es básicamente la capa de acceso a la base de datos. Este le responde al frontend en base a las entradas recibidas, por ejemplo, con un nombre de usuario y su información de contacto.

3. MODALIDAD

Guías para práctica de laboratorio.

4. TEMA DEL PROYECTO

Este proyecto se enfocará en demostrar la instalación de la plataforma de E-commerce OXID, a la vez que se enfoca en detallar los posibles métodos de extensión de la misma.

5. JUSTIFICACIÓN

La inspiración para realizar esta guía nace del conocimiento adquirido durante una práctica profesional, realizada entre los meses Junio de 2015 y Mayo de 2016, donde el estudiante, Juan Diego Echeverri Isaza, trabajó con la empresa Nexus Netsoft GmbH, ubicada en la ciudad de Langenfeld en Alemania, realizando diferentes tareas de integración, desarrollo, y apoyo en general, junto al equipo de desarrolladores.

Las tareas dentro de la empresa consistían, en un principio, en realizar la integración de plantillas, usando el *Template Engine* Smarty. En la sección 7.1 se pueden encontrar descripciones básicas sobre las tecnologías mencionadas y utilizadas a lo largo de esta guía.

Luego de terminar estas primeras tareas como integrador de plantillas, se comenzaron a escribir módulos de extensión para la tienda online OXID. Con este conocimiento se hizo posible realizar tareas mas complejas, que involucraban la lectura y escritura de información en la base de datos, la extracción de datos para mostrar en el Frontend, el control de las relaciones entre diferentes tablas en la base de datos, la creación y definición de nuevas funcionalidades, entre otras.

El lenguaje de programación utilizado dentro de la empresa es PHP. Otros lenguajes como HTML, CSS, y Javascript también son usados para la presentación de las páginas web.

Durante este año de práctica, se participó en varios proyectos con importantes empresas de renombre mundial. Algunos de estos proyectos eran páginas completamente nuevas, donde se tuvo que desarrollar todas las funcionalidades para el cliente, así como la integración del diseño generado por el departamento de servicios creativos de la empresa; otros fueron paginas de antiguos clientes de la compañía, que requerían mantenimiento, resolución de errores, desarrollo de nuevas funciones; y otros fueron con nuevos clientes que ya tenían una página establecida usando una tienda OXID, pero requerían modificaciones.

En esta guía se pretende mostrar de una manera clara, como se puede configurar la plataforma de E-commerce OXID, explicando las diferentes partes de las que consta, mostrando las múltiples opciones que le permiten al usuario configurar su tienda online, y mostrar algunos plugins que extienden el funcionamiento de la misma, para incluir, por ejemplo, Paypal como un medio de pago.

6. OBJETIVOS

6.1. Objetivo General

Presentar una guía corta de cómo configurar y poner en funcionamiento una tienda en linea usando la plataforma de E-commerce OXID, mirando también qué posibilidades de expansión y personalización tiene la misma.

6.2. Objetivos Específicos

1. Explicar las ventajas y desventajas de utilizar estas tiendas.
2. Presentar una guía de cómo configurar una tienda en linea usando la plataforma de E-commerce OXID.
3. Mostrar de forma clara, cómo se puede personalizar la página web.
4. Demostrar un ejemplo de programación de un plugin, módulo, o extensión para la tienda.

7. ANTECEDENTES

7.1. MARCO TEÓRICO

7.1.1. Lenguajes de programación

HTML, CSS y Javascript

Estos 3 lenguajes de programación son los bloques de construcción básicos de la web. En pocas palabras, estos se pueden entender de la siguiente manera.

- **HTML:** es un lenguaje que determina la estructura y el contenido de la página web. Usando este lenguaje se pueden definir títulos, listas, encabezados, pies de página, entre muchas otras; y también permite separar la página en partes mas pequeñas, como columnas y bloques. Tener una buena estructura tiene el beneficio añadido, de que la página estará mejor rankeada en los motores de búsqueda como Google, ya que estos usan la estructura para identificar títulos y palabras claves, que le permiten realizar una mejor conexión entre un término de búsqueda y la página en cuestión.

```
1 <!DOCTYPE html>
2 <html>
3   <head>
4     <title>Título</title>
5   </head>
6   <body>
7     Contenido
8   </body>
9 </html>
```

- **CSS:** este lenguaje le permite al programador darle un estilo a la página. El CSS tiene muchas diferentes opciones, por ejemplo establecer el tipo de fuente a utilizar, darle a los títulos un tamaño mas grande con un color diferente, establecer una imagen de fondo para la página, entre muchas otras. Todo lo que se ve en una página web, es como es, básicamente por una serie de estilos escritos en el lenguaje CSS.

```
1 body {
2   background-color: #FF0000;
3 }
```

- **Javascript:** este lenguaje le permite al programador hacer cambios en la página luego de que ésta haya sido cargada. Ejemplos de uso de javascript pueden ser, mostrar un submenu cuando el mouse

se encuentra sobre un título, verificar que una contraseña ingresada cuente con la cantidad y tipo de caracteres necesarios para aumentar la seguridad, mostrar una lista de sugerencias a medida que el usuario ingresa un termino de búsqueda, etc.

```
1 function writeToTheConsole(message) {
2   console.log(message);
3 }
```

PHP

PHP: Hypertext Preprocessor es un lenguaje de programación del lado del servidor, diseñado en un principio para el desarrollo web, pero ha evolucionado en un lenguaje de programación de propósito general. El desarrollo del PHP comenzó en el año 1994, creado por Rasmus Lerdorf, como una solución escrita en C para realizar mantenimiento en su página web. (Lerdorf 2007) Hoy en día mas del 80 % de las páginas web usan PHP como su lenguaje base. De este porcentaje, aproximadamente el 98 % usan la versión 5 del lenguaje, siendo la 7 la versión actual. (W3Techs 2016) PHP 7 fue lanzado al público en Diciembre del 2015, y pruebas han mostrado que puede presentar una mejoría de hasta 100 % en rendimiento, y 50 % en uso de memoria, permitiéndole al usuario expandir la capacidad de su página, sin tener que cambiar el hardware. (Zend 2016)

El código PHP se ejecuta antes de mostrar la página, y se encarga principalmente de obtener la información relevante de la base de datos, como nombres de usuarios, direcciones, artículos, precios, etc; y pasársela a una plantilla, que en últimas es mostrada en el explorador en forma de HTML. Otra importante función del código PHP es el control de acceso a ciertas páginas, verificando que el usuario haya ingresado sus credenciales, para por ejemplo, modificar sus detalles de cuenta.

```
1  <?php
2  class oxArticle extends oxI18n implements oxIArticle, oxIUrl
3  {
4    public function setId($sId = null)
5    {
6      $sId = parent::setId($sId);
7
8      $this->oxarticles__oxnid = $this->oxarticles__oxid;
9
10     return $sId;
11   }
12 }
```

SQL

SQL: Structured Query Language es un lenguaje declarativo para el acceso a bases de datos relacionales, que permite especificar diversos tipos de operaciones en ellas. El alcance de este lenguaje incluye la inserción, la edición, la eliminación, y la lectura, de información de la base de datos. A pesar de ser definido en gran parte como un lenguaje declarativo, también incluye elementos procedimentales.

SQL fue uno de los primeros lenguajes comerciales para el Modelo Relacional de Edgar F. Codd, cuyo modelo fue descrito en su influencial publicación de 1970. A pesar de no adherirse al 100 % al modelo de Codd, SQL es hoy en día el lenguaje de bases de datos mas usado. (Wikipedia 2016c)

```
1 SELECT  Book.title AS Title,
2         count(*) AS Authors
3   FROM  Book
4   JOIN  Book_author
5     ON  Book.isbn = Book_author.isbn
6   GROUP BY Book.title;
```

7.1.2. Frameworks y plugins

OXID

OXID es una solución de E-commerce desarrollada por la empresa OXID eSales AG, basada en Freiburg, Alemania. Establecida en el 2003, su principal producto, OXID, ofrece una plataforma de E-commerce que es escalable y modular.

OXID cuenta con una gran comunidad de usuarios y programadores, que han puesto a disposición del público muchos de sus propios módulos, los cuales se pueden encontrar en la tienda OXID eXchange.

OXID, como tienda, ofrece al usuario muchas posibilidades, e incluye por defecto:

- manejo de descuentos y promociones
- control de direcciones y métodos de envío
- múltiples lenguajes de interfaz
- control de impuestos
- control de acceso de usuarios
- gestión de varias subtiendas en la misma interfaz
- uso de diferentes tipos de pago

- entre otras muchas opciones

Dentro del código de las tiendas OXID se pueden apreciar ciertas prácticas de programación. La mas notoria es el uso de prefijos al momento de nombrar las variables, y el uso del camelCase. Estos prefijos pueden ser:

- $o- para objetos
- $s- para cadenas o strings en inglés
- $bl- para booleanos
- $a- para arreglos

OXID cuenta con 3 versiones, *Enterprise, Professional y Community*. *Enterprise* está pensada para tiendas con mucho tráfico, y tiene un muy alto rendimiento, disponibilidad y seguridad. (OXID 2016b) También ofrece la posibilidad de tener y administrar varias tiendas en el mismo servidor y bajo la misma interfaz. *Professional* está apuntada a suplir tanto pequeñas start-ups como experimentados minoristas online. (OXID 2016c) Estas dos versiones cuentan con soporte técnico por parte de OXID.

Community es una versión de código libre de la tienda, que en la práctica tiene todas las mismas funcionalidades de las otras dos versiones, pero sin costo para el usuario. Ésta, como las otras, también puede ser expandida y personalizada casi sin límites. Miles de clientes usan esta versión, para llevar a cabo sus negocios en linea. (OXID 2016a)

La versión *Community* será la que se utilizará durante el desarrollo de este proyecto.

Smarty

Smarty es un motor de plantillas basado en PHP. Su principal labor es la de la separación entre la lógica de una página y su presentación. Esto permite que el Frontend de una pagina, es decir su presentación, cambie, sin cambiar también la lógica que esta detrás. Esto minimiza, idealmente, los costos y el esfuerzo necesario para llevar a cabo el mantenimiento de una página.

Smarty genera contenido web, usando tags especiales dentro de archivos con extensión '.tpl'. Estas tags son interpretadas y reemplazadas mas adelante por código PHP, que en su lugar será interpretado por el motor PHP instalado en el servidor.

El codigo Smarty se puede encontrar encapsulado entre llaves '{codigo smarty}', pero su definición ha sido modificada dentro de las tiendas OXID, y dentro de ellas se debe usar una combinación de llaves y corchetes, de la siguiente forma '[{codigo smarty}]'.

```
1 [{* Esto es un comentario *}]
2 [{if $blIsLoggedIn}]
3   <p>Welcome [{$sUserName}]</p>
4 [{/if}]
```

En este ejemplo, la variable $sUserName contiene el nombre del usuario, y pudo haber sido definida en el controlador de la página, o anteriormente en la plantilla. Lo mismo ocurre con la variable $blIsLoggedIn.

jQuery

jQuery es una librería multiplataforma de Javascript, diseñada para simplificar la programación del lado del cliente o el explorador. A diferencia de PHP, que es un lenguaje de programación del lado del servidor.

jQuery es la librería de Javascript más popular del mundo, siendo instalada en el 65 % de las páginas con mayor tráfico de la web. (jQuery 2016)

Esta librería no juega un papel muy importante en las tiendas OXID, por defecto, ya que todas las páginas se recargan nuevamente una vez que el usuario cambia algún parámetro. Por ejemplo, cuando el usuario aumenta la cantidad de artículos que desea comprar, la página se recarga completamente para mostrar los nuevos totales. Usando jQuery, todos estos cambios se pueden hacer sin necesidad de recargar la página, usando peticiones ajax. Esto resulta en una experiencia de usuario mucho mas limpia y agradable.

7.2. Estado del arte

7.2.1. Magento

Magento es una plataforma de E-commerce de código abierto, escrita en PHP. El software fue desarrollado por Varien Inc. y su primer lanzamiento general fue en el año 2008. Posteriormente la compañía fue adquirida en su totalidad por eBay.

De acuerdo a una investigación conducida en el 2015, Magento tiene una cuota de mercado del 29.8 %, de entre las 30 plataformas mas usadas. (Works 2016)

Magento utiliza una base de datos relacional MySQL/MariaDB, esta escrito en PHP, y utiliza elementos del framework Zend. Aplica las convenciones de la programación orientada a objetos, y la arquitectura Modelo-Vista-Controlador. (Wikipedia 2016a)

7.2.2. PrestaShop

PrestaShop es una solución de E-commerce de código abierto, escrita en PHP. Es usada actualmente por 250.000 tiendas alrededor del mundo, y cuenta con una cuota de mercado del 9 % para páginas de E-commerce. El desarrollo comenzó en el 2005 como un proyecto estudiantil. (PrestaShop 2016)

PrestaShop soporta el uso de temas y módulos, que le permite al usuario la integración de nuevos visuales y agregar nuevas funcionalidades. PrestaShop también cuenta con una tienda de extensiones que provee a los desarrolladores vender sus propios módulos a los mercaderes. (Wikipedia 2016b)

7.2.3. OpenCart

OpenCart es un sistema de E-commerce basado en PHP, de código abierto. Fue lanzado por primera vez en 1999, basado en PERL, para suplir las necesidades de varios negocios, y posteriormente fue lanzado al público general. (OpenCart 1999)

El desarrollo se estancó a principio del nuevo milenio, y fue retomado en el 2005 por Daniel Kerr, un desarrollador británico, que lo usó como base para su propio software de E-commerce en PHP. En el 2014 la versión 2.0 fue lanzada. En ese momento OpenCart era la opción preferida por los mercaderes en China, y en Marzo del 2016 OpenCart contaba con una cuota de mercado del 6.65 %, por debajo sólo de WooCommerce. (BuiltWith 2016)

7.2.4. Oscommerce

osCommerce Online Merchant es una solución de tiendas online completamente auto alojado que contiene tanto un catalogo en el front-end, como una herramienta de administración en el back-end, la cual puede ser fácilmente instalada y configurada a través de un proceso de instalación basado en web. (osCommerce 2016b)

osCommerce cuenta con una comunidad de mas de 280.000 tiendas, desarrolladores y proveedores de servicios, y cuenta también con mas de 7.000 plugins que pueden ser usados para personalizar su tienda en linea. (osCommerce 2016a)

7.2.5. WooCommerce

WooCommerce es una extensión para WordPress, que permite convertir su sitio de WordPress en una tienda online hecha y derecha. Tanto WordPress como WooCommerce son de codigo abierto. Esto empodera a miles de desarrolladores alrededor del mundo para construir sobre y mejorar WooCommerce. (Media 2015)

Al 16 de Marzo de 2016, WooCommerce contaba con el 31.89 % de cuota de mercado, entre las soluciones de tiendas online en el mundo, convirtiéndolo en la opción con mayor acogida. (BuiltWith 2016)

8. METODOLOGÍA

En el transcurso de este proyecto se mostrará, en una forma sencilla, como instalar y configurar una tienda básica usando la plataforma de E-commerce OXID.

De manera introductoria se describirán las diferentes opciones disponibles para el usuario, tanto las ventajas y desventajas de cada uno, como una lista de características para cada una de las opciones. De esta forma se cumple con el objetivo número 1.

En este proceso se comenzará con la descarga del paquete de instalación, se discutirá los requerimientos de la misma, y paso a paso se describirá el proceso completo de instalación. Así se cumplirá con el objetivo número 2.

Luego de la instalación, se demostrará las maneras que la tienda se puede personalizar, incluyendo las posibilidades de configuración, sin necesidad de modificar el código, y también las opciones de personalización, editando directamente las plantillas de la tienda, modificando así la estructura y el estilo de las páginas generadas. De esta manera se cumple con el objetivo numero 3.

Finalmente se ejemplarizarán varios modos en que la tienda se puede extender, añadiendo a ella nuevas funcionalidades, y también modificando las existentes en la misma. En este punto se mirará en detalle ejemplos de código PHP. Con este punto se cumple con el objetivo 4.

9. ENTREGABLES

Los entregables de este proyecto serán los siguientes:

- Una guía detallada de instalación de tiendas en linea, usando el solución de E-commerce OXID.
- Varios ejemplos claros de cómo se puede personalizar la tienda, tanto visual como funcionalmente.
- Una lista de consideraciones necesarias, si se deseara implementar esta tienda en el mercado colombiano.
- Una lista de uso rápido, conteniendo las principales funciones de OXID y Smarty que se usan dentro de la tienda.

10. DESARROLLO

En esta sección se describirá el paso a paso de la instalación, configuración y personalización de una tienda online, usando la plataforma de E-commerce OXID.

Se comenzará con una descripción de pre-requisitos y recomendaciones para el desarrollo de la tienda. Luego se describirá el proceso de instalación y configuraciones básicas de la tienda. En tercer lugar se demostrarán las formas de personalización de la tienda usando el motor de plantillas Smarty, que viene incluido con la tienda. Finalmente se expondrán dos casos de estudio, y como se pueden implementar usando módulos o extensiones para la tienda.

10.1. Pre-requisitos, configuraciones y programas recomendados

Todos los siguientes programas serán instalados en una computadora corriendo Windows. La mayoría de éstos tienen versiones disponibles para MacOS X como para Linux, aunque las configuraciones específicas pueden variar de programa en programa.

10.1.1. XAMPP

XAMPP se encargará de alojar nuestra tienda localmente durante el proceso de desarrollo. Éste aplicativo incluye un servidor APACHE con una base de datos basada en MySQL.

El instalador se puede descargar directamente y de forma gratuita desde la página de Apache Friends ("`www.apachefriends.org`") (Apache-Friends 2016). Para el desarrollo de esta guía es suficiente con usar las configuraciones por defecto.

10.1.2. PhpStorm

PhpStorm es un IDE (**I**ntegrated **D**evelopment **E**ngine) para el desarrollo de aplicaciones escritas en PHP. Es uno de los mejores en el mercado, y la gran ventaja que tiene en este caso específico, es que existen extensiones para el programa que agregan soporte para el desarrollo de tiendas OXID, como para el uso del motor de plantillas Smarty. Esto ahorra al programador la escritura de muchas lineas de código y le ofrece sugerencias en vivo y autocompletación. Este IDE también brinda excelente soporte para la escritura de código HTML, CSS y JavaScript.

PhpStorm se ofrece en forma de prueba gratuita por un mes, y el archivo de descarga se puede encontrar en la página de JetBrains ("`www.jetbrains.com/phpstorm`") (JetBrains 2016).

Luego de la instalación se procede a descargar e instalar las extensiones anteriormente mencionadas. Desde el menú "Archivo → Configuraciones" (File → Settings) se selecciona "Plugins" y luego "Explorar repositorios" (Browse repositories). En la ventana emergente se busca **"OXID Plugin"** y se instala. Se hace lo mismo con el plugin **"Symfony2 Plugin"** .

Adicionalmente es recomendable configurar la definición del lenguaje de Smarty, para adaptarlo a la

implementación utilizada en las tiendas OXID. En la ventana de configuraciones se selecciona “Lenguajes y Frameworks → PHP → Smarty”, y allí se definen los delimitadores como “[{” y “}]” como se puede ver en la figura 1.

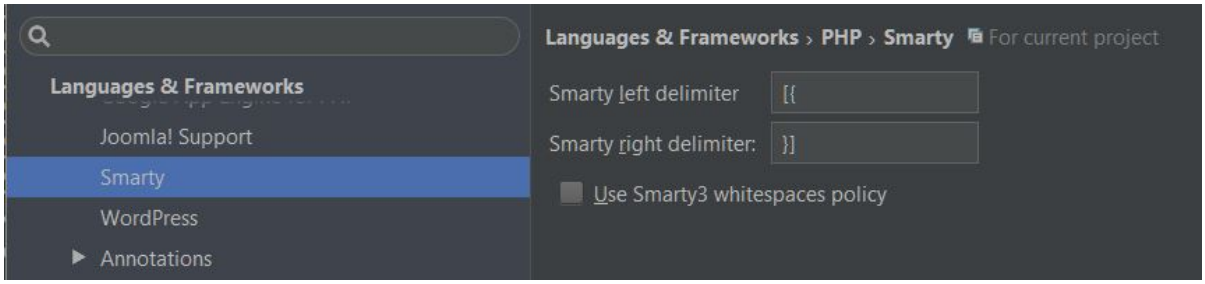


Figura 1: Configuración del lenguaje Smarty para las tiendas OXID usando PhpStorm

Claro está que cualquier otro IDE nos puede servir de la misma manera, y depende de la preferencia personal de cada desarrollador.

10.1.3. HeidiSQL

HeidiSQL permite al usuario una conexión directa con la base de datos instalada en el servidor, proveyendo una manera fácil y clara de manipular la información en la misma. Este será especialmente útil en la sección 10.5, cuando sea necesaria la manipulación directa de entradas en la base de datos.

El archivo de instalación está disponible de forma gratuita para los sistemas operativos Windows, Linux y MacOS X, en la página de HeidiSQL (“`www.heidisql.com/download.php`”) (HeidiSQL 2016).

Una vez la tienda esté instalada y configurada (sección 10.2), se puede crear la conexión con su base de datos de la siguiente forma. Una vez abierto HeidiSQL, se crea una nueva sesión, con los datos de la base de datos, como se puede ver en la figura 2.

- Hostname / IP: 127.0.0.1
- User: root
- Password: ——
- Port: 3306

10.1.4. WinSCP

WinSCP es un cliente SFTP, FTP, WebDAV y SCP para Windows, que nos permite tener una conexión directa con el sistema de archivos del servidor. Durante el desarrollo de esta guía no será muy

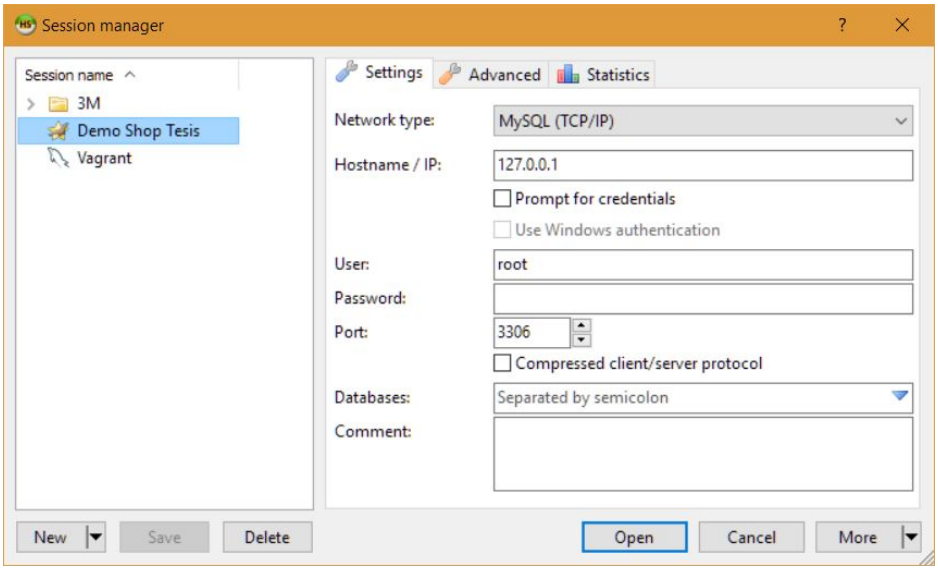


Figura 2: Configuración de la conexión con la base de datos usando HeidiSQL

necesario, ya que se tiene acceso directo a los archivos del servidor usando el explorador de Windows, pero en una etapa posterior, cuando la tienda este alojada en un servidor remoto, este programa puede ser muy útil.

Se puede descargar de forma gratuita desde la página de WinSCP ("`winscp.net/eng/download.php`") (WinSCP 2016).

10.1.5. PuTTY y MTPuTTY

PuTTY es una utilidad que permite establecer una conexión SSH con el servidor donde esta alojada nuestra tienda, dando acceso a la consola de comandos para ejecutar todo tipo de instrucciones en el servidor. **MTPuTTY** es una utilidad que permite encapsular varias sesiones de PuTTY en una sola interfaz gráfica.

Al igual que WinSCP, esta aplicación no es muy importante durante el proceso de desarrollo de la tienda mientras ésta esté alojada localmente. Al momento de que la tienda esté alojada remotamente, esta aplicación se vuelve imprescindible.

Es gratis y descargable desde la página de TTY Plus ("`www.ttyplus.com/multi-tabbed-putty`") (TTY-Plus 2012).

Se necesita adicionalmente descargar un ejecutable de PuTTY, *putty.exe*, disponible para descarga en la página de Greenend ("`www.chiark.greenend.org.uk/~sgtatham/putty/download.html`") (PuTTY 2016). Este archivo se debe ubicar en la carpeta donde se instaló MTPuTTY. En Windows, el sitio de instalación por defecto es "`C:/ProgramFiles(x86)/MTPuTTY/putty.exe`".

10.1.6. Hosts

Para mejorar el proceso de desarrollo, se puede modificar el archivo de hosts del sistema, para crear una URL para la tienda mientras se esta trabajando localmente. Esto permite usar, por ejemplo, la URL "`www.my-shop.dev`" para visitar nuestra tienda local, en vez de usar la URL "`localhost:8080`", que es el caso por defecto.

Primero hay que abrir el bloc de notas con permisos de administrados (click derecho en el ícono, y seleccionar "abrir como administrador"). Usando el bloc de notas, abrimos el archivo (Archivo → Abrir) "hosts" localizado en "`C:/Windows/System32/drivers/etc`". En este archivo vamos a agregar la siguiente linea, donde se define la URL que se usará localmente para acceder a la tienda (Wiebe 2015).

```
1 127.0.0.1 www.my-shop.dev
```

10.2. Instalación

10.2.1. Descarga

El primer paso para comenzar la instalación es descargar los archivos base de la tienda. Como se vio en la sección 7.1.2, OXID cuenta con 3 versiones, *Community*, *Professional* y *Enterprise*. Para el desarrollo de esta guía se utilizará la versión *Community*.

El archivo a descargar se puede encontrar visitando la página principal de OXID eSales ("`www.oxid-esales.com/en`"). El link a la descarga se encuentra al fondo de la página, como se puede ver en la figura 3. Este link redirecciona a una página donde se da la posibilidad de registrarse como miembro de la comunidad OXID, pero este es un paso completamente opcional. Debajo del formulario para la inscripción se encuentra un link que redirige directamente a la descarga (*"Go directly to the download"*) como se puede ver en la figura 4.

Una vez el archivo termine de descargarse (aproximadamente 83MB), se extrae usando WinZip o Winrar. Todos los archivos en la raíz de este comprimido se deben copiar al ambiente de desarrollo creado usando XAMPP (sección 10.1.1). Los archivos se deben copiar a la carpeta "htdocs" ubicada en "`C:/xampp/htdocs`".

Una vez los archivos estén copiados en el ambiente de desarrollo, se puede proceder a hacer la instalación.

10.2.2. Instalación

Antes de comenzar la instalación, hay que asegurarse que el ambiente de desarrollo esta corriendo. Para hacer esto, se abre el panel de control de XAMPP, y se le da en iniciar a los módulos de Apache y MySQL. Una vez los módulos estén activos (figura 5), se visita la URL definida en la sección 10.1.6

para acceder al ayudante de instalación de la tienda, usando el explorador de preferencia. Si no se definió ninguna URL en la sección 10.1.6, se visita la URL "`localhost:8080`".

Al visitar esta URL, se muestra el ayudante de instalación de las tiendas OXID, que configurará la base de datos, instalará algunos productos de muestra y configuraciones adicionales que permitan saltar al desarrollo al instante. El primer paso del ayudante de instalación se ve como en la figura 6. Este paso muestra si el sistema puede alojar la tienda para el desarrollo. Si en este paso hay algún elemento que tenga una "X" roja, la tienda no se puede instalar en este sistema. En caso contrario se le da click en "Proceed with setup".

El segundo paso del ayudante, visto en la figura 7 permite modificar el mercado objetivo de la tienda, así como el lenguaje. Por defecto, la tienda sólo incluye Inglés y Alemán como lenguajes disponibles, pero en la sección 10.3.1 se mostrará como descargar e instalar un nuevo paquete de idiomas. Luego de escoger un mercado y un país principal, se le da click en "Start installation".

El tercer paso del ayudante, visto en la figura 8, nos muestra las condiciones de instalación. Luego de haber leído las condiciones, seleccionamos "I accept license conditions" y le damos click en "Continue".

El cuarto paso del ayudante, visto en la figura 9, permite configurar algunos aspectos básicos de la base de datos, como su ubicación, nombre, y las credenciales de acceso. Se usa "localhost" como ubicación, ya que está corriendo localmente dentro de la utilidad de XAMPP. Se le da el nombre y las credenciales deseadas.

En este caso es importante marcar la opción de "Install demodata". Lo que esto hará es instalar diferentes productos, categorías, métodos de envío y de pago, atributos y demás, que permitirán saltar a la personalización y desarrollo de nuevas funcionalidades, de inmediato. Finalmente se le da click en "Create database now".

El quinto paso del ayudante, visto en la figura 10, permite crear un usuario con derechos de administrador, lo que le permitirá modificar varios aspectos de la tienda usando la plataforma de administración. Es importante anotar estas credenciales para no perder el acceso a dicha plataforma.

Finalmente se le da click en "Save and continue", lo que lleva al sexto y último paso de instalación, visto en la figura 11. Desde esta ventana se puede visitar directamente la tienda instalada, vista en la figura 12, o la plataforma de administración. La plataforma de administración es accesible visitando la URL ""`<URL elejida>/admin`"", e ingresando los datos del paso 5.

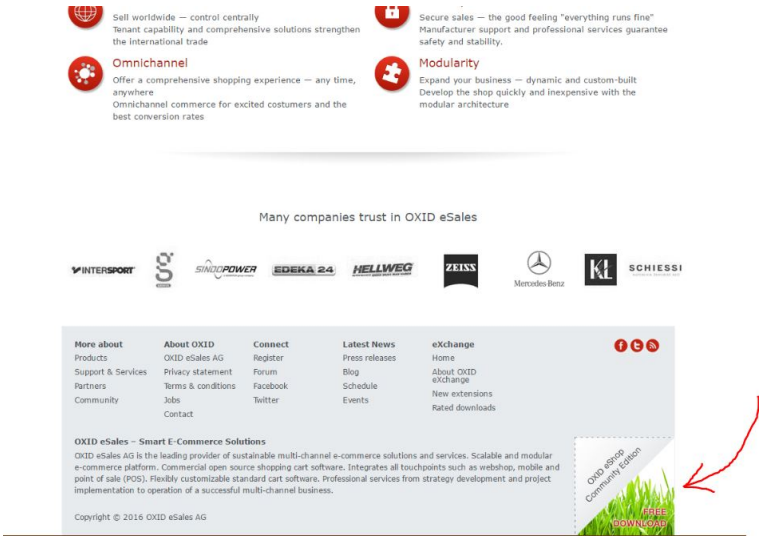


Figura 3: Descarga de la versión de comunidad de la tienda OXID

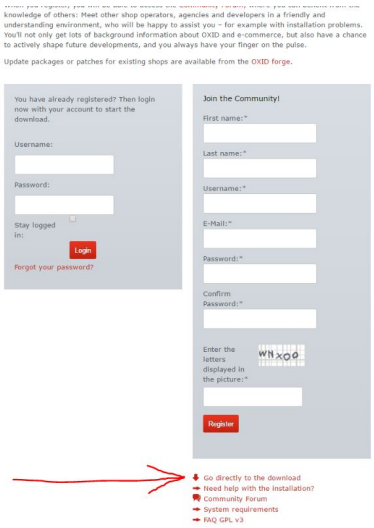


Figura 4: Descarga de la versión de comunidad de la tienda OXID

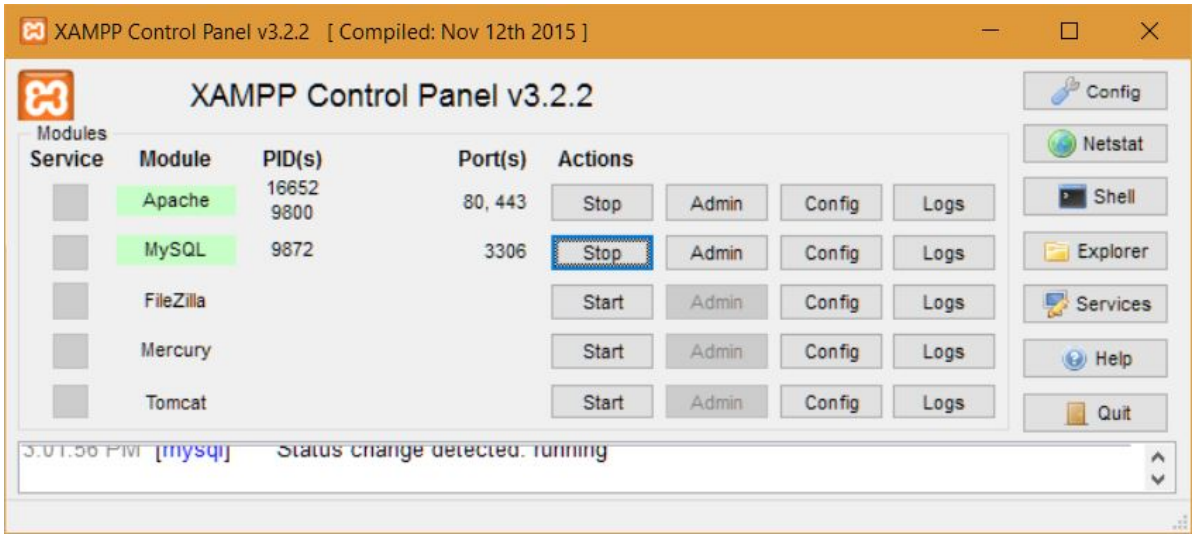


Figura 5: Panel de control de XAMPP

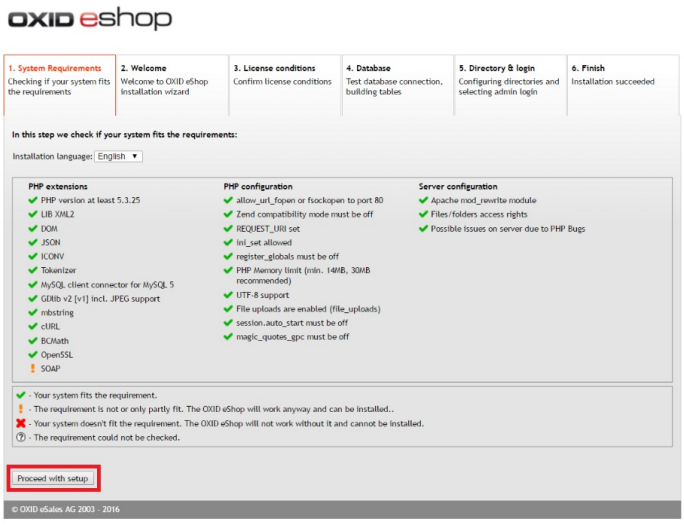


Figura 6: Paso 1. Requerimientos del sistema

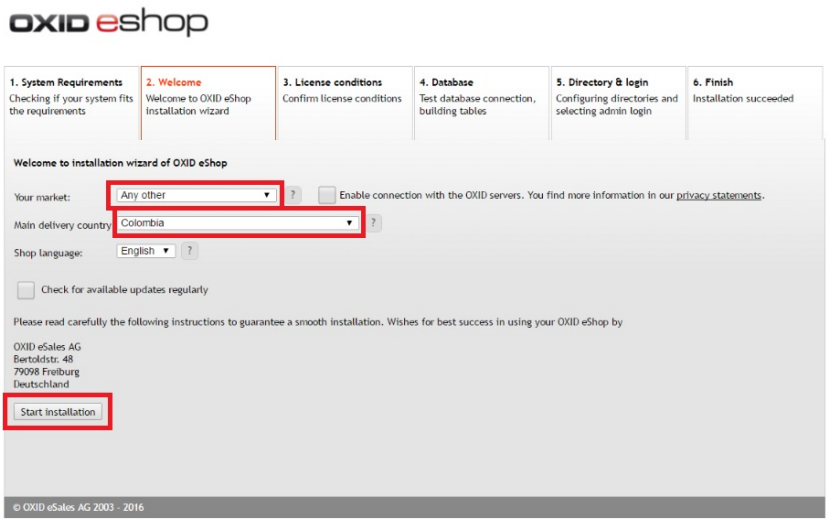


Figura 7: Paso 2. Configuraciones básicas de la tienda y el mercado objetivo

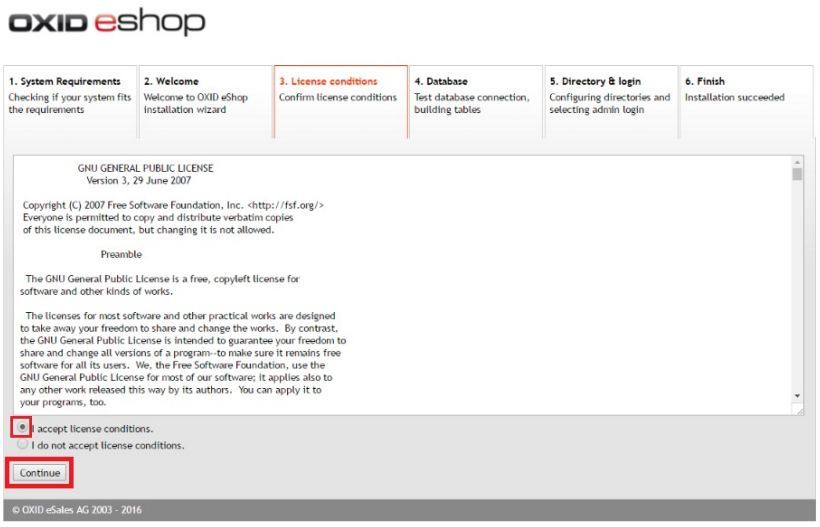


Figura 8: Paso 3. Condiciones de licencia

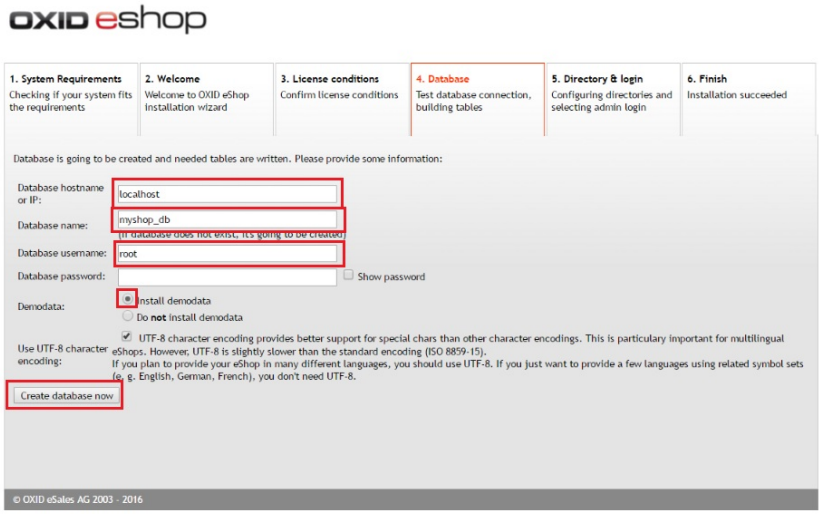


Figura 9: Paso 4. Configuraciones de la base de datos

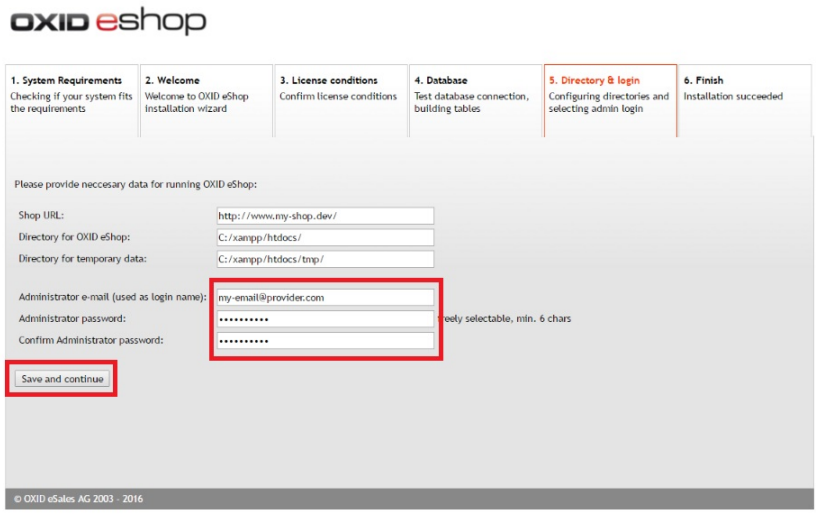


Figura 10: Paso 5. Configuración de cuenta de administrador

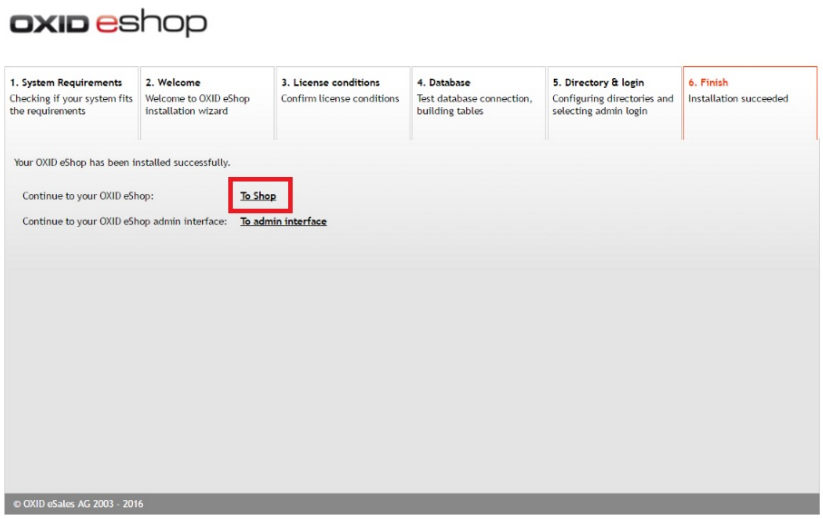


Figura 11: Paso 6. Tienda instalada

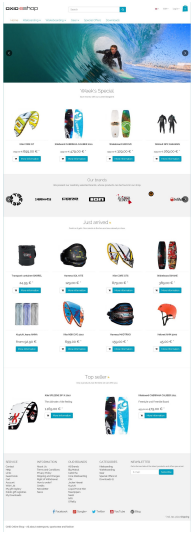


Figura 12: Paso 7. Pagina inicial de la tienda por defecto

10.3. Personalización

10.3.1. Instalación de un paquete de idiomas

Por defecto, las tiendas OXID solo ofrecen los idiomas Inglés y Alemán. Afortunadamente, en la tienda OXID eXchange, se puede descargar un paquete de idiomas para instalar español de latinoamerica.

La tienda se encuentra en el siguiente link, "`exchange.oxid-esales.com/en/home/`". Allí se busca *spanish* y se descarga el paquete llamado "***Latin American Spanish for OXID eShop***", siguiendo las instrucciones de la página. La descarga es gratuita, pero aún así es necesario proporcionar un correo donde recibir el link de descarga.

Al descargar el archivo, se descomprime usando WinZip o WinRar, y la carpeta "application" debe ser copiada en la raíz del proyecto. En este caso, la raíz del proyecto es la carpeta "htdocs" ubicada en "`C:/xampp/htdocs`".

Para activar el idioma primero hay que activarlo en la base de datos. Se entra a la plataforma de administrador yendo a la siguiente URL "`www.my-shop.dev/admin`" y se ingresan las credenciales definidas en el paso 5 de la sección 10.2.2 (figura 10). Una vez adentro, se da click en "Master settings", y luego en "Languages".

Se muestra un formulario para agregar nuevos idiomas, como se ve en la figura 13. Se rellena de la forma en que se ve en la figura y se da click en "'save". Nótese que la abreviatura esta escrita en minúsculas.

Luego se deben actualizar las vistas de la base de datos. Para hacer esto se da click en "Service" y luego en "Tools", en la barra de navegación a la izquierda. Finalmente se da click en "Update DB Views now". Lo que hace esto es generar nuevas vistas en la base de datos, para el nuevo idioma. En este momento el idioma ya debe estar activo y disponible para ser seleccionado tanto en la tienda como en la plataforma de administrador.

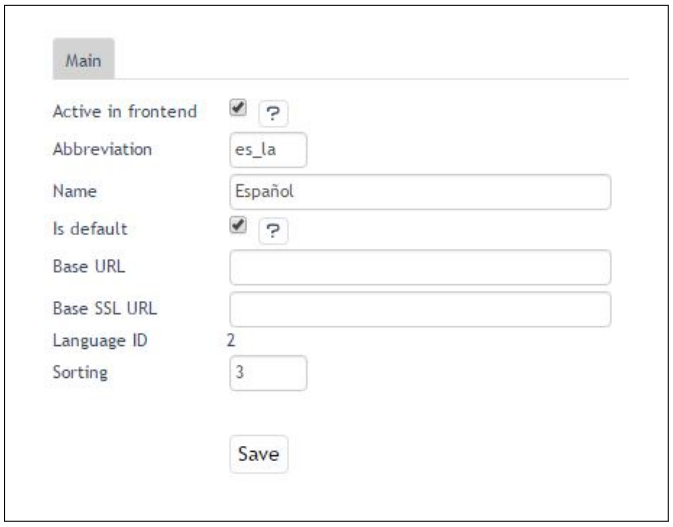


Figura 13: Formulario de creación de un nuevo idioma

Ahora solo falta el último detalle para continuar con el desarrollo. Ninguno de los artículos, categorías, atributos, y otros elementos tienen una traducción al español, por lo que en el front-end no se muestran los títulos de los mismos. Una solución rápida a este problema es copiar todos los valores en inglés al español. De esta manera la tienda seguirá funcionando normalmente, con todos los menús en español; y los artículos y categorías en inglés.

Para hacer esto de forma rápida se usará un archivo SQL previamente preparado para este propósito.

Se puede descargar desde la siguiente URL "`tinyurl.com/zw3zk6a`". Este archivo se puede correr usando HeidiSQL (ver sección 10.1.3). En el programa se selecciona la base de datos correspondiente, luego se selecciona la pestaña llamada "Query", y se arrastra el archivo descargado hacia la ventana de texto.

Luego de que el archivo es cargado en la interfaz, se presiona el botón "Execute SQL" (▷) o se presiona "F9". Una vez el archivo ha sido ejecutado, todos los artículos, categorías, atributos, y demás elementos, tendrán un nombre en español igual a su nombre en inglés.

Personalizando el idioma

Es importante notar que este paquete de idiomas soporta hasta la versión 4.8 de la tienda OXID, con soporte para el tema *Azure*. En esta guía se está usando la versión 4.10 de la tienda, con el tema *Flow* (mas sobre temas en la sección 10.3.2), por lo que existen nuevas cadenas en la tienda que no están definidas en el paquete.

Adicionalmente no esta de más revisar las traducciones incluidas en el paquete, para garantizar la calidad y coherencia del mismo.

Por ejemplo, cuando se agrega un artículo a la canasta, se muestra el siguiente texto. *"ERROR: Translation for DD_MINIBASKET_MODAL_TABLE_TITLE not found!"*. Esto quiere decir que la cadena identificada por el nombre *DD_MINIBASKET_MODAL_TABLE_TITLE* no ha sido definida.

Para crear nuevas cadenas se usa el archivo "`cust_lang.php`". Debido a que este archivo no existe para nuestro tema actual, lo copiaremos del tema *Azure*. Se copia todo el contenido de la carpeta "`application/views/azure/es_LA`" a la carpeta "`application/views/flow/es_LA`".

Dentro de esta carpeta se encuentra el archivo "`cust_lang.php`" con el siguiente contenido.

```
1 <?php
2   $sLangName  = 'Español de América';
3   $aLang = array(
4     'charset' => 'utf-8',
5   );
```

Se agrega la siguiente linea dentro del arreglo,

```
1 'DD_MINIBASKET_MODAL_TABLE_TITLE' => 'Articulo',
```

Y el resultado final debe ser algo así,

```php
<?php
  $sLangName  = 'Espanol de America';
  $aLang = array(
    'charset' => 'utf-8',
    'DD_MINIBASKET_MODAL_TABLE_TITLE' => 'Articulo',
  );
```

De la misma manera se pueden crear nuevas cadenas que se quieran usar dentro de la tienda, y sobre-escribir las previamente definidas. La recomendación es, en caso de que se quiera cambiar un aspecto de la tienda, sobre escribir las cadenas predefinidas, dado de que se mantenga una correlación entre el identificador de la cadena y el texto de la misma. De esta manera, en caso de que la tienda sea actualizada y las plantillas por defecto cambien, los cambios hechos se mantendrán.

También es recomendable reutilizar los mas posible cadenas ya existentes. De esta manera aumenta la congruencia a través de la página, y el mantenimiento de la misma disminuye.

Si las cadenas no cambian de valor en la página, es necesario vaciar la carpeta de archivos temporales, que incluye plantillas previamente procesadas y la cache de la tienda. Esta carpeta es "`/tmp/`". De ella se borran todos los archivos y subcarpetas, dejando unicamente el archivo "`.htaccess`".

10.3.2. Temas

Las tiendas OXID soportan el manejo de temas para personalizar el frontend de las mismas. Estos temas pueden ser desarrollados o descargados como paquetes listos para su uso.

El sistema de temas de OXID utiliza el pre-compilador de plantillas Smarty, y permite heredar plantillas de otros temas, de manera que el tamaño del proyecto se minimice. En estos casos se define un tema "padre" para el nuevo tema, y cuando una plantilla no se encuentra dentro del tema actual, se usará la versión del tema padre, en caso de que exista.

Es recomendable siempre crear temas nuevos cuando se desean modificar aspectos del frontend. La razón para esto siendo que, si en algún momento la tienda se actualiza y los temas predefinidos cambian, esto no sobre-escriba los cambios hechos por el desarrollador.

Creación de un nuevo tema

Se comienza por definir un nuevo tema, y se usará el tema "Flow", predeterminado de OXID, como su tema padre. De esta manera se puede modificar sólo algunos aspectos de la tienda, sin tener que preocuparse por todos los demás archivos necesarios dentro del tema.

Para crear el nuevo tema se crean dos carpetas con el nombre del tema. Una en la carpeta "`/application/views/nombredeltema`" para alojar plantillas y archivos de idiomas. La segunda en la car-

peta "`/out/nombredeltema`" para alojar todos los recursos necesarios específicos para este tema, como imágenes, scripts, hojas de estilos, etc. El nombre del tema debe ser en minúsculas y sin espacios.

Una vez creadas estas carpeta se crea un archivo llamado "`theme.php`" dentro de la primera. Este archivo contendrá los detalles del nuevo módulo, como su nombre, descripción, versión, herencias y demás. En el archivo se escribe el siguiente código.

```php
<?php
  $aTheme = array(
    'id'             => 'mytheme',
    'title'          => 'My Custom Theme',
    'description'    => 'Custom Theme created for this guide',
    'thumbnail'      => 'theme.jpg',
    'version'        => '0.1',
    'author'         => 'Juan Diego Echeverri',
    'parentTheme'    => 'flow',
    'parentVersions' => array('1.0.0-beta', '1.1.0-beta'),
  );
```

Los parámetros aquí definidos son respectivamente:

- **id**: Es el id del tema. Debe ser igual al nombre de la carpeta.
- **title**: Define el nombre del tema como tal. Se muestra en la plataforma de administrador de la tienda.
- **description**: Define la descripción del tema. Se muestra en la plataforma de administrador de la tienda.
- **thumbnail**: Es el nombre de la imagen que se mostrará para identificar al nuevo tema dentro de la plataforma de administrador. Esta imagen debe ser guardada en la carpeta "`out/mytheme/theme.jpg`".
- **version**: Define la versión actual del tema.
- **author**: Define el autor del tema.
- **parentTheme**: Define el tema padre, de donde se tomarán las plantillas y recursos que no se hayan definido dentro del tema actual.
- **parentVersions**: Permite definir las versiones del tema padre a utilizar. Esta versión se puede encontrar en el archivo "`theme.php`" del tema padre correspondiente.

Luego de crear los archivos para el tema hay que activarlo. En la plataforma del administrador se hace click en **"Extensiones → Temas"**, luego se selecciona el nuevo tema de la lista, y se presiona en **"Activar"**.

Debido a que no se ha incluido ninguna plantilla personalizada al tema, éste hereda todo del tema padre, por lo que la tienda en este momento no muestra ningún cambio.

Ejemplo

Como ejemplo se modificará la página de detalles de los productos. Se eliminará el sistema de estrellas y se agregará la descripción corta a los artículos relacionados.

El sistema de estrellas se encuentra en el encabezado del producto, debajo del título del mismo, como se puede ver en la figura 14. Los artículos relacionados se encuentran al fondo de la página. Actualmente éstos muestran una imagen del artículo, su nombre, su precio y un link a su página de detalles, como se puede ver en la figura 15a.

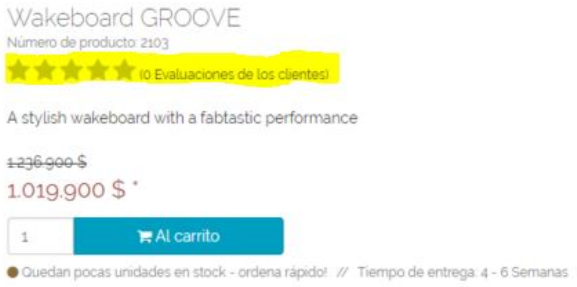


Figura 14: Sistema de estrellas presente por defecto en el tema “Flow”, en la página de detalles del producto.

Analizando la estructura y las clases que se encuentran en la página, se encuentra que el sistema de estrellas esta definido en la plantilla “`application/views/flow/tpl/page/details/inc/productmain.tpl`” alrededor de la linea 104.

```
104 [{* ratings *}]
105 <div class="star-ratings">
106   [{if $oView->ratingIsActive()}]
107     [{block name="details_productmain_ratings"}]
108       [{include file="widget/reviews/rating.tpl" itemid="anid=`
              $oDetailsProduct->oxarticles__oxnid->value`" sRateUrl=
              $oDetailsProduct->getLink()}]
109     [{/block}]
110   [{/if}]
111 </div>
```

Aquí se puede ver una llamada al método “ratingIsActive()” de la instancia de la vista, en este caso *details* (Para mas información con respecto al uso de Smarty en la plataforma OXID y su funcionamiento ver la sección 14.8). Esto indica que hay una configuración en la plataforma del administrador que permite desactivar el sistema de estrellas sin tener que modificar la plantilla.

Esta configuración se encuentra bajo **“Maestro Configuración → Valores del núcleo”**, en la pestaña de “Rendimiento”. Se desactiva la segunda opción en el subgrupo “Ajustes de rendimiento mejorado”, llamada “Carga de usuarios Comentario”.

Al volver a mirar la pagina de detalles del producto, se observa que el sistema de estrellas ya no esta, al igual que el sistema de calificaciones y comentarios que se encontraba en la parte inferior de la página.

Para modificar los artículos relacionados hay que identificar a que plantilla en el tema padre están definidos. Para esto se observa la estructura HTML y sus clases. Se encuentra que la plantilla usada para generar los artículos relacionados es "`application/views/flow/tpl/widget/product/listitem_grid.tpl`".

Lo que se debe hacer es copiar esta plantilla completa en nuestro tema, con la misma estructura de carpetas. Por lo tanto se obtendrá el siguiente archivo "`application/views/mytheme/tpl/widget/product/listitem_grid.tpl`", con el mismo contenido. Dentro de este nuevo archivo ya se pueden hacer todos los cambios necesarios.

Se desea agregar la descripción corta entre el título y el precio del artículo. Para esto hay que identificar exactamente donde están definidos estos dos elementos, y agregar el nuevo código entre ellos. Tras estudiar la plantilla anteriormente mencionada, se encuentra que el título y los precios están definidos alrededor de la linea 62, como se puede ver a continuación:

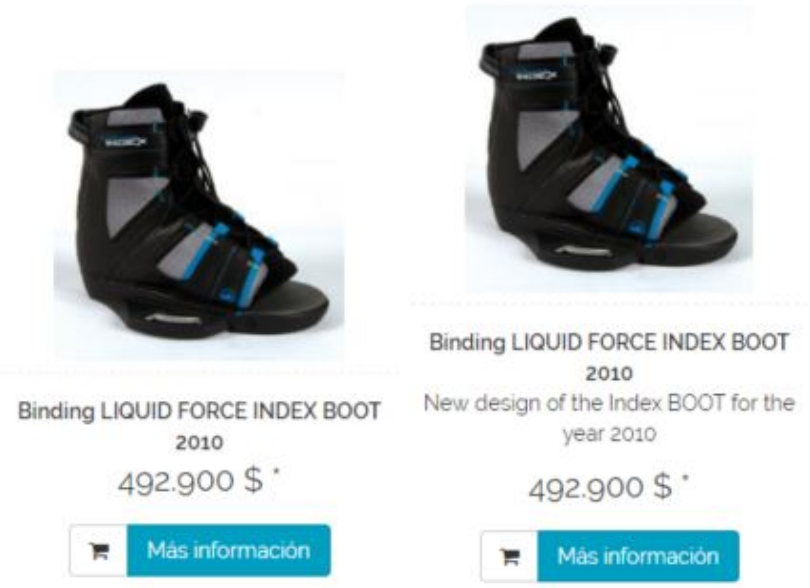
```
62 <div class="title">
     ...
   </div>
69 <div class="price text-center">
     ...
   </div>
```

Entre estos dos bloques, alrededor de la linea 69, se agregará el siguiente código.

```
67 [{/block}]
68
69 <div class="short-description">
70   <p>[{$product->oxarticles__oxshortdesc->value}]</p>
71 </div>
72
73 <div class="price text-center">
```

Lo que esta ocurriendo en la linea 70, es que se está utilizando el objeto *$product*, instancia de la clase *oxArticles*, definida previamente en la linea 2 de la misma plantilla, para obtener su descripción corta. El elemento *oxarticles__oxshortdesc* hace referencia a una tabla de la base de datos, y a una columna en específico. En este caso la tabla *oxarticles* y la columna *oxshortdesc*. "value" retorna el valor almacenado en esta columna de la base de datos para este artículo.

Esta estructura, *tabla__columna*, es muy común en toda la tienda. Los objetos también tienen métodos para obtener ciertos campos que requieren de cierto pre-procesamiento, como el precio actual, que puede



(a) Por defecto (b) Modificado

Figura 15: Disposición de artículos relacionados

depender del usuario, promociones activas, y muchos otros parámetros. Por esto, siempre es recomendable tratar de obtener estos valores usando los métodos específicos, y en caso de que estos no existan, usar esta estructura. Para conocer qué métodos existen para todas las clases dentro de la tienda, ver la documentación de la tienda en la URL "`docu.oxid-esales.com/CE/sourcecodedocumentation/4.10.0/`".

En las imágenes 15a y 15b se puede ver el resultado final.

10.3.3. Inclusión de nuevas monedas

La tienda soporta la utilización de cualquier moneda. Estas deben ser definidas en la plataforma del administrador.

Para hacerlo se selecciona **"Maestro Configuración → Valores del núcleo"**, luego se entra en la pestaña **"Configuración"**. En el subgrupo **"Otros ajustes"** se encuentra una entrada de texto marcada como **"Agregar o quitar monedas"**. En esta entrada te texto se pueden definir todas las monedas que se deseen soportar en la tienda, siguiendo el siguiente formato.

[Abreviación]@[Tasa]@[Separador decimales]@[Separador miles]@[Símbolo]@[Precisión decimal]

Cada uno de estos elementos se puede definir así:

- **Abreviación:** Es el nombre abreviado de la moneda. En el estándar internacional suele ser de 3 letras en mayúsculas. Por ejemplo, el dólar americano es USD, el euro es EUR y el peso colombiano

es COP.

- **Tasa:** Es la tasa con la cual se calculan los precios en la tienda al usar esta moneda. Esta tasa se define respecto a la moneda que tenga una tasa de **1**. Por ejemplo, si el dolar americano se define con tasa de 1, el peso colombiano tendría una tasa de 2933 aproximadamente. En el caso contrario, si el peso colombiano tiene una tasa de 1, el dolar americano tendría una tasa de 0.000341 aproximadamente.

  Es importante decidir cual moneda lleva la tasa de 1, porque con respecto a esta moneda se definen los precios dentro de la tienda. Es decir, al crear un nuevo artículo, su precio se ingresa en la moneda que tenga la tasa unitaria.

- **Separador de decimales:** Éste define el caracter que se usará para separar los decimales de la parte entera en los precios.

- **Separador de miles:** Éste define el caracter que se usará para separar grupos de miles en los precios dentro de la tienda. Se usa recursivamente para separar cada 3 dígitos. Es decir, también se usa para separar millones, miles de millones, etc.

- **Símbolo:** Es el caracter o la cadena usada para mostrar que moneda se esta usando en el frontend. Es mostrado al lado de todos los precios dentro de la tienda. Por ejemplo para el dolar se usa el símbolo **$** y para el euro **€**.

- **Precisión decimal:** Determina el numero de decimales a mostrar dentro de la tienda. Afecta todos los precios mostrados.

Es importante notar que la tienda usa la moneda actualmente elegida para generar las facturas y correos al usuario y a la empresa. Por esta razón es vital mantener estas tasas de cambio actualizadas, ya sea manualmente, o con algún script que consulte las tasas vigentes y actualice la tienda.

Para definir el peso colombiano, usando el dolar americano como moneda alternativa, se reemplaza el contenido de la entrada de texto por las siguientes lineas, y se da click en “Guardar”.

```
1 COP@ 1@ @ .@ $@ 0
2 USD@ 0.000341@ .@  @ $us@ 2
```

Una vez las monedas han sido activadas, se vuelven disponibles en el frontend de la tienda.

10.4. Módulos

Una parte esencial del desarrollo de tiendas OXID es el manejo de módulos. Por módulo se entiende un paquete de funcionalidades, que puede ser activado o desactivado al gusto del usuario.

Un módulo debe ser, en la medida de lo posible, completamente independiente, e incluir en sus archivos todo lo que necesita para funcionar. Entiéndase por esto, que éste incluya sus propios scripts u hojas de estilos de ser necesario, sus propias cadenas de texto, archivos de instalación y desinstalación en caso de que el módulo utilice una nueva tabla en la base de datos, o modifique una de las tablas existentes, y demás recursos que pueda necesitar. La filosofía detrás de esto es que el módulo funcione instantáneamente luego de ser activado, sin necesidad de hacer configuraciones posteriores o cambios en otras partes de la tienda, y que pueda usarse en otras tiendas sin modificar el módulo en si.

10.4.1. ¿Qué puede hacer un módulo?

Una de las funcionalidades mas básicas de un módulo es extender una clase ya existente. Esto puede ser en la forma de crear un nuevo método para la clase, o sobre-escribir y/o modificar un método ya existente. Por ejemplo, si se desea modificar los requisitos para una contraseña segura, como requerir una mayúscula, un número y 8 caracteres como mínimo, se puede modificar el método "registeruser()" del controlador *register* para incluir esta validación.

La pregunta en este caso sería, ¿por qué crear un módulo cuando se puede modificar directamente el archivo donde la función está definida? La respuesta, como ya se ha mencionado antes, es prevenir que en el caso de que la tienda sea actualizada, se pierdan funcionalidades porque los nuevos archivos sobre-escriben los cambios previamente hechos.

Otra funcionalidad de un módulo es la creación de páginas completamente nuevas, que introducen abanicos completos de nuevas funcionalidades. Ya no se habla de extender una funcionalidad existente, sino de crear una funcionalidad completamente nueva para el usuario. Por ejemplo, crear una página donde el usuario pueda agregar artículos directamente a la canasta, ingresando unicamente el numero de artículo o referencia. En la sección 10.5.2 se mirará este ejemplo mas a fondo.

Otra funcionalidad crucial para un módulo es la modificación de las plantillas de la tienda, sin modificar directamente los archivos donde las plantillas están definidas. La forma en que OXID hace esto es mediante el uso de bloques. Si se observa de nuevo el ejemplo discutido en la sección 10.3.2, en el archivo "`application/views/flow/tpl/page/details/inc/productmain.tpl`", linea 107, se encuentra la definición de un bloque con nombre "`details_productmain_ratings`". Desde el módulo se puede definir una pequeña plantilla que sobre-escriba el contenido de este bloque. También se puede obtener el contenido del bloque original haciendo una llamada a su padre, si lo que se desea es anexar algo al principio o al final del bloque.

Ahora, la pregunta en este caso es, ¿por qué modificar la plantilla desde un módulo si se puede modificar directamente el archivo donde el bloque esta definido? La respuesta para esta pregunta no es tan obvia, como es el caso con la modificación de métodos, ya que debido a que se ha definido un nuevo tema, éste no se verá afectado por una posible actualización de la tienda. En este caso, depende de si lo que se va a incluir en el bloque contiene funcionalidades específicas de este módulo.

Si el nuevo contenido incluye una llamada a un método definido en el mismo módulo, por ejemplo "$oUser->getPhoneNumber()" (que es un método que la clase *oxUser* no incluye por defecto), todo funcionará muy bien, siempre y cuando el módulo esté activo. Si el módulo llega a estar desactivado en

algún momento, esto causará que se dé una excepción, o error de ejecución, y la página no se mostrará como se planeaba. Por lo tanto, la llamada a este método debe estar incluida dentro del módulo, de manera que al desactivarlo, el método no sea llamado.

10.4.2. ¿Cómo se crea un módulo?

Crear un módulo para la plataforma OXID es relativamente sencillo. Lo primero que se debe hacer es crear un carpeta en la carpeta de módulos, "`modules/`", con el nombre del módulo que se desea crear. **Este nombre no debe contener espacios y debe estar escrito en minúsculas en su totalidad.**

Si se desean agrupar todos los módulos en una misma carpeta, por ejemplo para separar los módulos creados por una compañía y los módulos de terceros, se puede hacer. Se crea una carpeta con el nombre de la empresa en la carpeta de módulos, "`modules/nombredecompania/`", y dentro de ésta se crea un archivo llamado "`vendormetadata.php`". Este archivo es una mera formalidad, pero es necesario para que la tienda interprete esta carpeta como un repositorio de módulos, y no como un módulo en sí. El contenido de este archivo debe ser el siguiente:

```php
<?php
/**
* Metadata version
*/
$sVendorMetadataVersion = '1.0';
```

Habiendo hecho esto, ya se pueden crear las carpetas de módulos dentro de ésta.

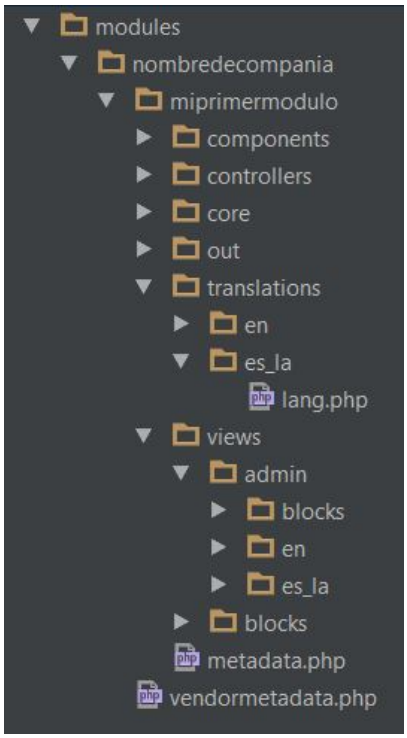


Figura 16: Estructura general de un nuevo módulo en las tiendas OXID

En la figura 16 se puede ver una estructura general de carpetas a la hora de desarrollar un nuevo módulo para las tiendas OXID. Nótese que se trata de mantener la misma estructura de carpetas que la tienda usa por defecto, y aunque esto es lo recomendado, no es necesario organizar el módulo de esta manera. **Todas** estas carpetas son opcionales, y se pueden utilizar según la necesidad o según lo que el módulo haga.

Los únicos archivos que deben mantener esta estructura, en caso de ser usados, son los archivos de idiomas. Para las cadenas que se usarán en el frontend, los archivos de idiomas, "`lang.php`", se guardan en la carpeta "`modules/nombredecompania/miprimermodulo/translations`" y luego el idioma correspondiente.

Para las cadenas que se usarán en el backend, cuando se desarrollan funcionalidades configurables desde éste, se utiliza la carpeta "`modules/nombredecompania/miprimermodulo/view/admin/`" y el idioma correspondiente.

Como se puede ver en la figura 16, se usa un archivo llamado "`metadata.php`". Este archivo es de suma importancia, ya que éste contiene las configuraciones del módulo, y sin este archivo el módulo no funcionará.

A continuación se muestra un ejemplo del contenido del archivo "`metadata.php`", y como se pueden usar cada uno de las diferentes propiedades para definir diferentes aspectos del módulo.

```php
<?php
  /**
  * Metadata version
  */
  $sMetadataVersion = '1.0';

  /**
  * Module information
  */
  $aModule = array(
    'id'          => 'miprimermodulo',
    'title'       => array(
      'es_la' => 'Mi primer módulo',
      'en'    => 'My first module',
    ),
    'description' => array(
      'es_la' => 'Esta es la descripción de mi módulo',
      'en'    => 'This is the description of my module',
    ),
    'thumbnail'   => 'out/pictures/pmlogo.png',
    'version'     => '1.0.0',
    'author'      => 'Juan Diego Echeverri',
    'url'         => 'http://www.mypersonalwebsite.com/',
    'email'       => 'myemail@provider.com',
    'extend'      => array(
      'oxuser'           => 'nombredecompania/miprimermodulo/models/extensionoxuser',
      'oxbasket'         => 'nombredecompania/miprimermodulo/models/extensionoxbasket',
    ),
    'files'       => array(
      'nuevocontrolador'     => 'nombredecompania/miprimermodulo/controllers/
          nuevocontrolador.php',
    ),
    'templates'   => array(
      'nuevaplantilla.tpl'   => 'nombredecompania/miprimermodulo/views/page/checkout/
          nuevaplantilla.tpl',
    ),
    'blocks'      => array(
      array(
        'template' => 'module_config.tpl',
        'block'    => 'admin_module_config_form',
        'file'     => 'views/admin/blocks/miprimermodulo_module_config.tpl'
      ),
      // Pages
      array(
```

```
43          'template' => 'page/checkout/payment.tpl',
44          'block'    => 'select_payment',
45          'file'     => 'views/blocks/miprimermodulo_select_payment.tpl'
46        ),
47      ),
48      'settings'    => array(
49        array(
50          'group' => 'main',
51          'name'  => 'miprimermoduloSandboxMode',
52          'type'  => 'bool',
53          'value' => '1'
54        ),
55      ),
56      'events'      => array(
57        'onActivate'   => 'nombreDeClase::onActivate',
58        'onDeactivate' => 'nombreDeClase::onDeactivate',
59      ),
60    );
```

A continuación se muestra una explicación breve de cada uno de los parámetros que se pueden definir en este archivo.

- **id:** Es el identificador del módulo. Debe ser exactamente igual al nombre de la carpeta donde está guardado el módulo.
- **title:** Es el título del módulo que será mostrado en la plataforma de administrador. Como se ve en el ejemplo, se puede definir en los diferentes lenguajes en los que se puede mostrar la tienda.
- **description:** Es la descripción del módulo que se mostrará en la plataforma de administrador. Al igual que el título, se puede definir en los diferentes idiomas.
- **thumbnail:** Define una imagen que se mostrará en la plataforma de administrador. Puede ser el logo de la empresa que desarrolló el módulo, o cualquier otra cosa.
- **version, author, url, email:** Definen información extra del módulo y sus autores. Esta información también se mostrará en la plataforma del administrador.
- **extend:** Define las diferentes clases de la tienda que este módulo extiende. Como índice se utiliza el nombre de la clase, *oxUser* por ejemplo, y en el valor se determina el camino donde está el archivo con las extensiones de la clase. Este camino se define dentro del contexto de la carpeta "`modules`", y el archivo no lleva la extensión.
- **files:** Define nuevas clases que se hayan creado para este módulo. En el índice se determina el nombre de la clase y en el valor se define el camino al archivo donde la clase esta definida. En este caso, el archivo sí lleva la extensión.
- **templates:** Define plantillas que se hayan creado específicamente para este módulo. En el índice se introduce el nombre de la plantilla, incluyendo su extensión. En el valor se define el camino donde se puede encontrar la plantilla, de nuevo, incluyendo la extensión del archivo.

- **blocks:** Define los bloques que sobre-escriben otros bloques dentro de la tienda. Estos se definen como arreglos, que constan de 3 elementos, que son:
  - **template:** Aquí se define la plantilla original donde se encuentra el bloque a sobre-escribir. En caso de querer sobre-escribir un bloque de la plataforma de administrador, se ingresa unicamente el nombre de la plantilla. En caso de sobre-escribir un bloque del frontend de la tienda, es necesario definir el camino exacto donde encontrar la plantilla original. En el código de ejemplo se pueden ver ambos casos.
  - **block:** Aquí se define el nombre del bloque a sobre-escribir, tal cual como aparece en la plantilla original.
  - **template:** Define el camino del archivo donde el nuevo bloque está definido, dentro del contexto de la carpeta del módulo actual.
- **settings:** Permite definir configuraciones accesibles desde la plataforma de administrador. Cada configuración se define como un arreglo con 4 partes, que son:
  - **group:** Define el grupo de configuraciones al cual ésta pertenece. En la plataforma de administrador todas las configuraciones son agrupadas basadas en este parámetro
  - **name:** Define el nombre de la configuración, con el cual se podrá obtener su valor una vez se quiera usar dentro de los archivos del módulo.
  - **type:** Define el tipo de variable que esta configuración sera. Los tipos de variables disponibles son booleanos (bool), cadenas (str), y arreglos simples (arr) o asociativos (aarr). Estos son mostrados como un checkbox, una entrada de texto corta y bloques de texto, respectivamente.
  - **value:** Define el valor predeterminado de la configuración.
- **events:** Permite definir métodos a ejecutar cuando ciertos eventos son activados. Los principales eventos son *onActivate*, que se ejecuta cuando el módulo es activado; y *onDeactivate*, que se ejecuta una vez el módulo es desactivado. Estos eventos son extremadamente útiles si el módulo necesita modificar la estructura de la base de datos, crear una tabla o columna en la misma, o cualquier otra pre-configuración. De igual manera, se pueden restablecer los cambios realizados al activar el módulo con el evento *onDeactivate*. Para definir estos métodos hay que definir una clase, en el ejemplo llamada *nombreDeClase*, y dentro de ella los métodos estáticos "onActivate()" y "onDeactivate()". El archivo donde se definió la clase se debe incluir en la lista de archivos (files) en el archivo "`metadata.php`" del módulo.

En la sección 10.5 se mostrarán ejemplos donde todos estas opciones se utilizarán.

10.5. Casos de estudio

10.5.1. Caso 1

Definición

Desarrollar un contador que permita mostrar en el frontend de la página la cantidad de veces que un artículo específico ha sido visitado. Este contador se mostrará en la página principal del artículo, debajo del título del mismo. La frase a usar será *"Visto XX veces"*.

En este ejemplo se mostrará como extender clases, modificar la base de datos y acceder a la información usando un módulo, la definición de cadenas dentro del mismo, y la sobre-escritura de bloques.

Plan de desarrollo

- Modificar la tabla *oxarticles* en la base de datos, agregando la columna *views*, para almacenar el contador de vistas de cada artículo. Esta columna tendrá un valor por defecto de "0".
- Extender el modelo *oxArticle*, agregando un método para aumentar el conteo de vistas dentro de la base de datos.
- Extender la vista *details*, que es la encargada de cargar y mostrar la página principal de los artículos. En esta extensión se sobre-escribirá el método "render()", que se ejecuta justo antes de mostrar la página, y allí se usará el método anteriormente definido para aumentar el conteo de vistas del artículo. Adicionalmente se creará un nuevo método público, para poder obtener el conteo actual de vistas desde la plantilla.
- Crear el bloque que mostrará el contador en la página principal del artículo.
- Crear las cadenas de texto necesarias para mostrar en el frontend.

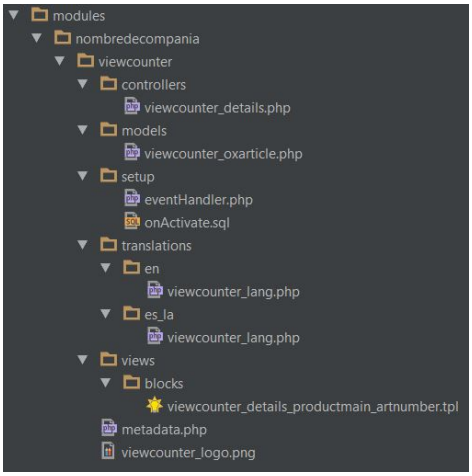


Figura 17: Estructura de archivos del módulo "*viewcounter*"

Basándose en estos requerimientos y pasos, se puede crear la estructura de archivos necesaria para el módulo. Ésta se muestra en la figura 17.

Paso seguido se crea el archivo "`metadata.php`" con el siguiente contenido. Los detalles con respecto a de donde salió el bloque a sobre-escribir se mirará mas adelante.

```php
<?php
  /**
  * Metadata version
  */
  $sMetadataVersion = '1.0';

  /**
  * Module information
  */
  $aModule = array(
    'id'          => 'viewcounter',
    'title'       => array(
      'es_la'   => 'Contador de vistas',
      'en'      => 'View counter',
    ),
    'description' => array(
      'es_la'   => 'Agrega un contador de vistas a la página principal de cada artículo',
      'en'      => 'Adds a view count to the details page of each article',
    ),
    'thumbnail'   => 'viewcounter_logo.png',
    'version'     => '0.1',
    'author'      => 'Juan Diego Echeverri',
    'url'         => '',
    'email'       => 'juaneche@msn.com',
    'extend'      => array(
      'details'       => 'nombredecompania/viewcounter/controllers/viewcounter_details',
      'oxarticle'     => 'nombredecompania/viewcounter/models/viewcounter_oxarticle',
    ),
    'files'       => array(
      'viewcounter_eventHandler'  => 'nombredecompania/viewcounter/setup/eventHandler.php'
    ),
    'templates'   => array(),
    'blocks'      => array(
      array(
        'template'  => 'page/details/inc/productmain.tpl',
        'block'     => 'details_productmain_artnumber',
        'file'      => 'views/blocks/viewcounter_details_productmain_artnumber.tpl'
      ),
    ),
    'settings'    => array(),
    'events'      => array(
      'onActivate'    => 'viewcounter_eventHandler::onActivate',
      'onDeactivate'  => 'viewcounter_eventHandler::onDeactivate',
    ),
  );
```

Para modificar la base de datos, se escribe una instrucción de SQL, y se guarda en el archivo "`onActivate.sql`". La instrucción es la siguiente:

```sql
ALTER TABLE `oxarticles` ADD COLUMN `VIEWS` INT UNSIGNED
  NOT NULL DEFAULT '0';
```

A continuación se debe crear el método encargado de ejecutar esta instrucción al momento de activar el módulo. Éste es el método "onActivate()" de la clase *viewcounter_eventHandler*, declarada en el archivo "`eventHandler.php`".

El contenido de este archivo se puede ver a continuación

```
1  <?php
2    class viewcounter_eventHandler
3    {
4      static function onActivate(){
5        $sFilename = __DIR__  . '/onActivate.sql';
6        if (is_file($sFilename)) {
7          $sSql = file_get_contents($sFilename);
8          if ($sSql) {
9            $oDb = oxDb::getDb();
10           $oDb->execute($sSql);
11         } // if
12       } // if
13     }
14   }
```

Para evitar errores y alertas, primero se verifica que el archivo "`onActivate.sql`" exista, en la linea 6. Luego se verifica que el archivo no esté vacío, en la linea 8. A continuación, en la linea 9, se obtiene el objeto encargado de realizar y ejecutar instrucciones de SQL. Finalmente se ejecuta las instrucciones guardadas en el archivo.

Una vez el archivo este en la tienda, se puede activar el módulo y éste ejecutará el método "onActivate()", generando la nueva columna en la base de datos.

Es muy importante, **cada vez que se modifique la estructura de la base de datos**, reconstruir las vistas de la misma. Esto se hace de manera sencilla, en la plataforma del administrador, yendo a **"Servicio → Herramientas"**, y hacer click en "**Actualizar vistas de BBDD ahora**" (*Update DB Views now*).

Si las vistas no son actualizadas, es posible que la nueva información guardada en la base de datos no sea reflejada en el frontend, o al cargar nuevas instancias del objeto modificado. *oxArticle* en este caso.

A continuación se creará la extensión de la clase *oxArticle*. De forma general, las extensiones se definen de la siguiente manera.

```
1  <?php
2    class viewcounter_oxarticle extends viewcounter_oxarticle_parent
3    {
4    }
```

El nombre de la nueva clase es arbitrario, pero es recomendable usar nomenclatura expresiva, que indique a que módulo pertenece y que clase extiende. En este caso se usa el nombre *viewcounter_oxarticle*.

La clase a extender es un aspecto importante de la creación de módulos en OXID, y es importante notar que la tienda no maneja las clases y sus extensiones de la manera tradicional, donde tradicionalmente la clase a extender en este caso sería *oxArticle*. Por esta razón se define una forma diferente para crear extensiones. En general, cuando se desea extender otra clase, se usa la forma *nombredeclase_**parent***. En este caso, la clase a extender es *viewcounter_oxarticle_parent*. Finalmente, la clase que será extendida será *oxArticle*, pero esto se define en el archivo "`metadata.php`", y no en el archivo de la extensión en si.

Ahora, para crear la funcionalidad, se agrega el siguiente código en la extensión del *oxArticle*, el archivo "`models/viewcounter_oxarticle.php`".

```
1  <?php
2    class viewcounter_oxarticle extends viewcounter_oxarticle_parent
3    {
4      public function addViewToCounter()
5      {
6        $iCurrentCount = $this->getFieldData('views');
7        $aNewValues = array('views' => ++$iCurrentCount);
8        $this->assign($aNewValues);
9        $this->save();
10     } // function
11   }
```

Aquí se ha creado el método "addViewToCounter()", que se encargará de actualizar la base de datos, aumentando el conteo de vistas para el artículo correspondiente. Paso a paso, lo que se está haciendo en este método es lo siguiente.

- En la linea 6 se está obteniendo el valor actual de las vistas del artículo. Se usa el método "getFieldData()", que es un método estándar para todos los modelos que permite obtener el valor de cualquier columna para un objeto de la base de datos. En este caso, se obtiene el valor guardado en la columna *views*, creada anteriormente para este módulo.
- En la linea 7 se crea un arreglo con los nuevos valores que se guardaran en la base de datos. En este caso, se actualizará el valor en la columna *views*, con el valor anterior aumentado en 1 unidad. En este arreglo se puede definir cualquier otro elemento que se desee actualizar, como el título (*oxtitle*), el precio (*oxprice*), la descripción (*oxshortdesc*), y demás campos.
- En la linea 8 se asigna el arreglo con los valores actualizados a la instancia actual del objeto en cuestión, usando el método "assign()".
- Finalmente, en la linea 9, se usa el método "save()" para actualizar la base de datos con los nuevos valores que han sido asignados al objeto. Usar estos métodos, "assign()" y "save()", facilita mucho la manipulación de información que debe ser introducida a la base de datos, sin tener que preocuparse por escribir las instrucciones SQL correspondientes.

A continuación se creará la extensión a la vista o controlador *details*. Este controlador se encarga de cargar toda la información necesaria para mostrar un producto correctamente, y de pasársela a la plantilla. Esta extensión será escrita en el archivo "`viewcounter_details.php`", con el siguiente contenido.

```
1  <?php
2    class viewcounter_details extends viewcounter_details_parent
3    {
4      public function render()
5      {
6        $oArticle = $this->getProduct();
7        $oArticle->addViewToCounter();
8        return parent::render();
9      } // function
10   }
```

El método "render()" es el encargado de preparar la información para mostrar en la plantilla, y retorna una cadena con el nombre de la plantilla a cargar. Este método es ejecutado cada vez que cualquier página de la tienda se muestra. Por esta razón es el punto ideal para aumentar el conteo de vistas de cada artículo.

Paso a paso, lo que hace esta extensión, es:

- En la linea 6, se obtiene una instancia de la clase *oxArticle* con la información del artículo que esta siendo actualmente cargado. Esto se hace usando el método "getProduct()", de la clase *details*.
- Usando este objeto, en la linea 7, se ejecuta el método "addViewToCounter()", escrito con anterioridad en la extensión de la clase *oxArticle*.
- Finalmente, se hace un llamado al método padre y se retorna su valor.

Es importante llamar siempre el método padre cuando se está escribiendo una extensión, a menos que lo que se desee es sobre-escribir completamente el método.

Habiendo escrito esta última extensión, se habrá terminado con los archivos encargados de la funcionalidad del módulo. Ahora hace falta escribir los bloques de plantillas que mostrarán la información en el frontend de la tienda. Para esto primero hay que identificar qué bloque se desea sobre-escribir.

Figura 18: Disposición de los detalles principales de un artículo

En la figura 18 se puede ver el estado actual de la página principal del artículo. Se identifica la clase que tiene el texto "Número de producto XXXX", justo debajo del título, y usando el IDE se

encuentra que este texto está definido en la plantilla "`productmain.tpl`", alrededor de las lineas 100 a 102. El código del bloque, en esta plantilla, es el siguiente:

```
100 [{block name="details_productmain_artnumber"}]
101   <span class="small text-muted">[{oxmultilang ident="ARTNUM" suffix="
        COLON"}] [{$oDetailsProduct->oxarticles__oxartnum->value}]</span>
102 [{/block}]
```

De este extracto de código se puede apreciar que el nombre del bloque a extender, visto en la linea 100, es *details_productmain_artnumber*.

Ahora se procede a crear el archivo "`viewcounter_details_productmain_artnumber.tpl`". En éste se incluye el siguiente contenido.

```
1 [{$smarty.block.parent}]
2 <br>
3 <span class="small text-muted">
4   [{oxmultilang ident="NUMBER_OF_VIEWS" args=$oDetailsProduct->
        oxarticles__views->value}]
5 </span>
```

En la linea 1 se hace una llamada al bloque padre, con el objetivo de no perder esa funcionalidad original. En la linea 3 se crea un "span" con las mismas clases que la linea en el bloque padre, de manera que sean estéticamente idénticas. Finalmente, la parte más importante es el bloque de Smarty "oxmultilang", en la linea 4.

Éste mostrará la cadena identificada por el identificador "NUMBER_OF_VIEWS", y a ésta se la pasará el contador de vistas como un argumento. Esta cadena aún no existe, por lo que debe ser creada en los archivos de lenguajes del módulo. Estos son los archivos "`viewcounter_lang.php`", dentro de las carpetas "`en`" y "`es_la`". El contenido de estos archivos es, respectivamente:

```
1 <?php
2   $sLangName = "English";
3   $aLang = array(
4     "charset"                     => "UTF-8",
5     "NUMBER_OF_VIEWS"             => "Viewed %s times",
6   );
```

```
1 <?php
2   $sLangName = "Español";
3   $aLang = array(
4     "charset"                     => "UTF-8",
5     "NUMBER_OF_VIEWS"             => "Visto %s veces",
6   );
```

En la linea 5 se define la cadena con el identificador respectivo. Nótese el uso de "**%s**" en la cadena definida. Este es el punto donde el argumento enviado al bloque "oxmultilang" será reemplazado. Estos marcadores permiten adaptar más fácilmente cadenas en diferentes idiomas, ya que es posible que en algunos idiomas el orden gramatical de las oraciones cambie, y de esta manera se puede ubicar el valor a mostrar en cualquier posición de la frase de forma sencilla.

Con esto ya se ha terminado de construir el módulo, pero antes de ver el resultado final es necesario un paso anterior. Las tiendas OXID dependen mucho en el uso de caches y objetos precargados y almacenados en memoria. Esto incluye plantillas, objetos, estructuras de bases de datos, cadenas de idiomas, etc. Esto es hecho para mejorar el desempeño de la tienda, y se puede hacer porque estos elementos no cambian muy seguido. Por esta razón la tienda aún no tiene guardada en su memoria la nueva estructura de la base de datos, ni el bloque de plantilla, ni la nueva cadena que se definió para el frontend.

Por esto es necesario limpiar la cache, para que la tienda cargue nuevamente todos estos elementos en su última versión. Esto se hace eliminando todos los archivos dentro de la carpeta "`tmp/`", incluyendo la carpeta "`smarty/`", y obviando el archivo "`.htaccess`".

Habiendo hecho esto, y asegurándose que el módulo esté activo, ya se puede ver el resultado final. Se visita la página de un artículo cualquiera y debajo del título se verá el conteo de visitas que ha tenido el mismo. Se debe ver algo como en la figura 19.

Figura 19: Resultado final del módulo "*viewcounter*"

10.5.2. Caso 2

Definición

Desarrollar una nueva funcionalidad en la página, que permita ingresar en un campo de texto el número de referencia de un artículo, y que éste sea agregado directamente a la canasta. La nueva página debe tener una tabla con dos campos de texto en cada fila, uno para ingresar la referencia del artículo, y el otro para ingresar la cantidad. Hay que validar que el número de productos a agregar no sea menor que 1. La página debe contar con un botón que agregue filas a la tabla, para ingresar más artículos si así se desea. Se debe mostrar un link en el header de la página, que permita visitar directamente la nueva funcionalidad. Finalmente se debe mostrar un texto modificable por el usuario antes de la tabla con los campos de texto.

En este ejemplo se mostrará cómo crear un nuevo controlador; cómo extender clases con nuevos métodos; como incluir scripts, estilos y cadenas de texto; y cómo incluir configuraciones para personalizar el módulo desde la plataforma de administración.

Plan de desarrollo

- Escribir un nuevo controlador para manejar la página de las ordenes rápidas.
- Crear la plantilla que se mostrará en la nueva página.
- Extender el modelo *oxArticle*, para incluir un método que permita cargar un artículo usando únicamente su numero de referencia. La instancia de este artículo es la que se usará posteriormente para agregarlo a la canasta.
- Escribir el script encargado de manejar el frontend de la página, principalmente encargado de agregar nuevas filas al presionar un botón.
- Escribir el archivo de estilos para mostrar un diseño agradable en la nueva funcionalidad.
- Crear un bloque para sobre-escribir la barra de navegación de la página, para mostrar en ella un link que redirija al usuario a la nueva funcionalidad.
- Incluir las nuevas cadenas utilizadas tanto en el frontend como en el backend.

Basándose en estos requerimientos y pasos, se puede crear la estructura de archivos necesaria para el módulo. Ésta se muestra en la figura 20.

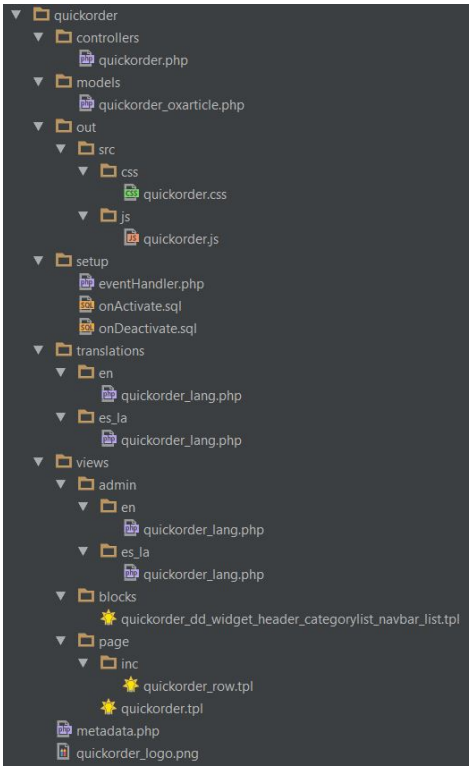


Figura 20: Estructura de archivos del módulo "*quickorder*"

En este caso **NO** se mostrará el código al lado de la explicación del mismo, debido a la longitud de éste. Todo el código creado para este módulo se puede encontrar en la sección 14.7 de los anexos. Para mejor entendimiento se recomienda tener el código a la mano o a fácil acceso, para fácil referencia a lo largo de esta sección.

metadata.php

Para empezar, se escribe el archivo "`metadata.php`" (ver sección 14.7.1, de los anexos). En este archivo es importante notar que ya se utiliza el parámetro "settings", y el parámetro "templates", para incluir las configuraciones del módulo como las nuevas plantillas que se utilizarán en el mismo. Para la configuración se escoge el nombre *sQuickOrderInitialLines*, visto en la linea 46, que se utilizará luego para obtener el valor de la misma. Ésta se define como una cadena.

quickorder.php

A continuación se crea el controlador *quickOrder* dentro de la carpeta "`controllers`" (ver sección 14.7.2, en los anexos). Este controlador extiende la clase *oxUBase*, que es la misma extendida por todas las vistas por defecto de la tienda OXID. Esta clase (*oxUBase*) contiene todas los métodos necesarios para cargar y mostrar una nueva vista. Lo primero que hay que definir en este controlador es la plantilla, usando el atributo *$_sThisTemplate*, como se puede ver en la linea 5.

Ya que esta clase extiende a *oxUBase*, no es necesario crear un método "render()" para mostrar la plantilla seleccionada. Éste se ejecuta automáticamente, y procesa la plantilla definida en la linea 5. A pesar de la herencia de métodos, si existe un método que hace falta crear. Este es el método "getBreadCrumb()", definido en la linea 32, que se encarga de mostrar la ubicación actual del usuario en la página. Esta es una copia exacta del método "getBreadCrumb()" de otra vista, y se modificó la linea 38, para mostrar la cadena definida por el identificador "QUICKORDER". Es necesario crearlo, porque diferentes vistas pueden generar su ubicación de formas diferentes. Por ejemplo:

Home → *Orden Rápida*

Home → *Categoría 1* → *Subcategoria 1* → *Subsubcategoria 1* → *Artículo*

Esto es mostrado en la cabecera de todas las páginas.

En la página se mostrará un formulario que contiene los campos de texto con la información del artículo a agregar (linea 38, del archivo ""`quickorder.tpl`"", sección 14.7.14). Este formulario enviará a este mismo controlador dicha información, por lo que se debe escribir un método que se encargue de procesar esta información y que retorne el resultado. Se define el método "add()", en la linea 10. Éste hace 3 tareas, obtener la información del formulario, procesarla, y poner a disposición de la plantilla el resultado de este procesamiento. Esto se logra definiendo y llamando a los 3 siguientes métodos privados:

- **getQuickOrderParameters():** este método, definido en la linea 52, obtiene la información del formulario, y elimina los campos vacíos.
- **addArticlesToBasket():** este método, definido en la linea 102, recibe los números de referencia ingresados en el formulario y sus cantidades respectivas. Luego procede a verificar que la información sea correcta, revisando que el contenido de cada numero de referencia no esté vacío (linea 109), revisando que un artículo con dicho número de referencia exista (linea 110) y revisando que dicho artículo se pueda comprar (linea 111). Si todas estas condiciones se cumplen agrega el artículo a la canasta usando la instancia de la clase *oxBasket*, almacenada en la sesión (linea 104 y 115). A medida que se procesa cada número, se clasifica cada uno como exitosos o erróneos, para luego mostrar un mensaje en el frontend.
- **setViewMessages():** este método, definido en la linea 88, pone a disposición de la plantilla tanto las referencias que fueron exitosas como las erróneas, para poder mostrar mensajes de retroalimentación al usuario.

Finalmente se define un método publico que retorne el valor de configuración *sQuickOrderInitialLines* a la plantilla. Este método se denomina "getInitialLines()" y está definido en la linea 22. El método es usado en la plantilla ""`quickorder.tpl`"", en la linea 4 (ver sección 14.7.14).

quickorder.tpl

Éste archivo define la plantilla que se mostrará cuando el controlador se cargue. Algo muy importante para notar aquí, es que todo el contenido se encapsula dentro de un bloque de Smarty "capture", y como última instrucción se incluye el archivo "`page.tpl`" en la linea 69. Lo que esto hace es cargar todos los demás elementos de la página que serán ignorados en esta plantilla, como la navegación del sitio, el header y el footer. Ésto hace que todas las nuevas páginas que sean creadas sigan el mismo patrón de diseño, incluyendo siempre los mismo elementos básicos.

En las lineas 2-4 se asignan algunas variables y objetos que se utilizarán más adelante, usando la instrucción de Smarty "assign". Esta instrucción tiene dos parámetros, "*var*", que define el nombre de la nueva variable, y "*value*" que define su valor. En este segundo parámetro se pueden usar valores fijos, objetos, y llamadas a métodos de objetos; aunque es importante notar que Smarty no permite encadenar métodos, por lo que por cada llamada a un método que retorne un objeto hay que guardarlo en una variable temporal.

A continuación se muestran los mensajes de error y éxito, bajo la condición que los respectivos arreglos existan. Se usan las instrucciones "if" para verificar la existencia, y "foreach" para recorrerlos y mostrarlos uno por uno. Finalmente se usa la función "oxmultilang" para mostrar una cadena predefinida en los archivos "`quickorder_lang.php`". Más sobre estos archivos mas adelante.

En la sección 14.8 de los anexos se puede encontrar una referencia extendida de las diferentes funciones, métodos y filtros de **Smarty** que se ofrecen dentro de las tiendas OXID.

Mas adelante, en la linea 32, se usa la instrucción "oxifcontent", que trata de cargar una página de contenido del CMS, y si existe, muestra lo que sea que esté definido dentro del bloque. Los parámetros usados para esta instrucción son "ident" y "object", que definen el identificador de la página a cargar, y

donde se guardará la página cargada, respectivamente. Ésto se usa usualmente cuando se desea mostrar un bloque de texto que debe ser fácilmente modificado por el administrador del sitio, en este caso pueden ser instrucciones sobre como usar la página o cualquier otro mensaje que se le desee dar al usuario.

EL CMS (Content Management System) se puede acceder dentro de la plataforma de administración, en "Información del cliente → CMS de páginas". Aquí se puede ver que ya existen numerosas otras páginas de contenido, para otros elementos como correos, textos informativos, términos y condiciones, y muchas otras cosas.

A continuación se define el formulario, y se usa el método "getShopCurrentUrl()" para obtener la página de la tienda a la cual se le enviará la petición de este formulario. El método retorna la dirección de la pagina, incluyendo el script "`index.php`", por ejemplo "`https://www.my-shop.dev/index.php?`".

Se define el controlador a usar y el método a llamar usando los parámetros "cl" y "fnc", en las lineas 40 y 41.

En la linea 50 se utiliza el bloque "section", que dentro de OXID permite hacer iteraciones de una forma similar a como si se tuviera un "for", ya que smarty para OXID no cuenta con el bloque "for". En el parámetro loop se define la cantidad de veces que se desea repetir el código en su interior. En este se incluye otra plantilla que contiene los campos de ingreso para cada fila. Ésta está definida en el archivo "`quickorder_row.tpl`".

Posteriormente se definen botones para agregar nuevas filas y para enviar la solicitud con la información ingresada. Nótese que se utilizan clases de otros botones preexistentes en la tienda, lo que promueve la uniformidad en el diseño.

Finalmente, se crea un clon de la fila en la linea 62, con el objetivo de poder copiarla y agregar más filas usando JavaScript.

En el archivo "`quickorder_row.tpl`" (ver sección 14.7.15) se define la fila de campos de texto, usando de igual manera clases de otros campos que ya existen en la tienda.

quickorder_oxarticle.php

En este archivo se crea el método "loadByArtNum()", que permite cargar la instancia de un artículo conociendo unicamente su número de referencia. Esto se hace escribiendo una instrucción SQL, en la linea 16, que se encargue de encontrar el registro correspondiente en la base de datos, y posteriormente se utiliza el método "assignRecord()" con esta instrucción como parámetro, lo que carga en la instancia actual toda la información correspondiente al artículo deseado. Es importante notar que el número de referencia debe ser escapado, para prevenir inyección SQL. Cualquier cadena de texto se puede escapar usando el método "quote()", también usado en la linea 16.

quickorder.js

En este archivo se le agrega la funcionalidad al botón de agregar nuevas lineas. Primero se crea la función "addNewInputsRow()" que se encargará de clonar la fila de muestra, y mostrarla en la página.

Posteriormente, se agrega un evento cuando el usuario haga click en el botón, y éste llama a la función previamente definida.

quickorder_lang.php

Hay 4 archivos donde se incluyen las cadenas de texto usadas por este módulo, dos para el frontend (inglés y español), y dos para el backend. Los archivos para el frontend se construyen de la misma forma que en el caso 1, y es importante recordar que se guardan en la carpeta "`translations/idioma`".

Los archivos para el backend si vale la pena mirarlos mas a fondo. Estos se guardan en la carpeta "`views/admin/idioma`" y contienen varias cadenas importantes. Debido a que se definieron configuraciones para este módulo (el parámetro "settings" en el archivo "`metadata.php`"), se generan algunas cadena para mostrar en la plataforma de administrador.

Como se puede ver en la linea 7 de las cadenas de la plataforma del administrador (secciones 14.7.11 y 14.7.12), se debe definir la cadena "`SHOP_MODULE_sQuickOrderInitialLines`". Para cada configuración que se halla definido en los metadatos se debe crear esta cadena, anteponiendo "`SHOP_MODULE_`" al identificador de la configuración.

Adicionalmente se puede definir una cadena que muestra mas información en cada configuración. Ésta se define anteponiendo "`HELP_`" a la cadena anterior, como se puede ver en la linea 8. Cuando esta cadena es definida, se mostrará un signo de pregunta en el que se puede hacer click, para mostrar la información adicional con respecto a la configuración.

También se debe definir una cadena al grupo de configuraciones que se creó. El identificador para los grupos se construye anteponiendo "`SHOP_MODULE_GROUP_`" al nombre del grupo, como se puede ver en la linea 6.

El resultado final de estas cadenas se muestra en la figura 21.

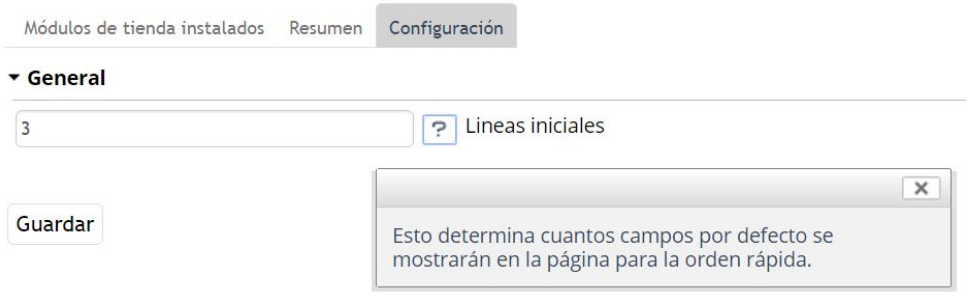


Figura 21: Ventana de configuración del módulo "quickorder"

quickorder_dd_widget_header_categorylist_navbar_list.tpl

Ahora lo único que hace falta para terminar el módulo es crear un link en la navegación que permita dirigir al usuario a la página. Identificando las clases de los elementos de la navegación, se puede localizar que estos están definidos en la plantilla "`categorylist.tpl`", alrededor de las lineas 25, en el tema *Flow*. Aquí se puede ver que todos los elementos están definidos en el bloque *dd_widget_header_categorylist_navbar_list*, por lo tanto éste será el bloque usado donde se incluirá el link a la página.

Antes de crear el link es importante notar un problema que existe con este bloque en el tema *Flow*,

versión "1.0.0-beta". El bloque anteriormente mencionado está definido en dos posiciones, lo cual sería problemático para luego sobre-escribirlo. En el archivo "`application/views/flow/tpl/widget/header/categorylist.tpl`" está definido en las lineas 13 y 26. Una solución para este problema es cambiarle el nombre a uno de estos bloques, idealmente a el definido en la linea 13, ya que el otro es el que nos interesa. Ésto se puede hacer directamente en la plantilla del tema *Flow*, lo cual no es recomendado, o se puede sobre-escribir la plantilla completa en el tema personalizado que se creó en la sección 10.3.2.

Habiendo arreglado este problema ya se puede proceder a crear el bloque que sobre-escribirá el del tema (ver sección 14.7.13). Este bloque es bastante sencillo. Como se vio en el caso 1, lo primero que se hace es incluir el contenido del bloque padre, en la linea 1. A continuación se procede a incluir el link para la nueva página. Para obtener el link se usa el método del configurador "getShopCurrentUrl()", y concatenándole a éste el controlador en cuestión. Para concatenar se utiliza el modificador de Smarty "|cat:". Mas información sobre modificadores en la sección 14.8.

Producto final

En la figura 22 se puede ver el resultado final de la nueva página, incluyendo el link en la navegación

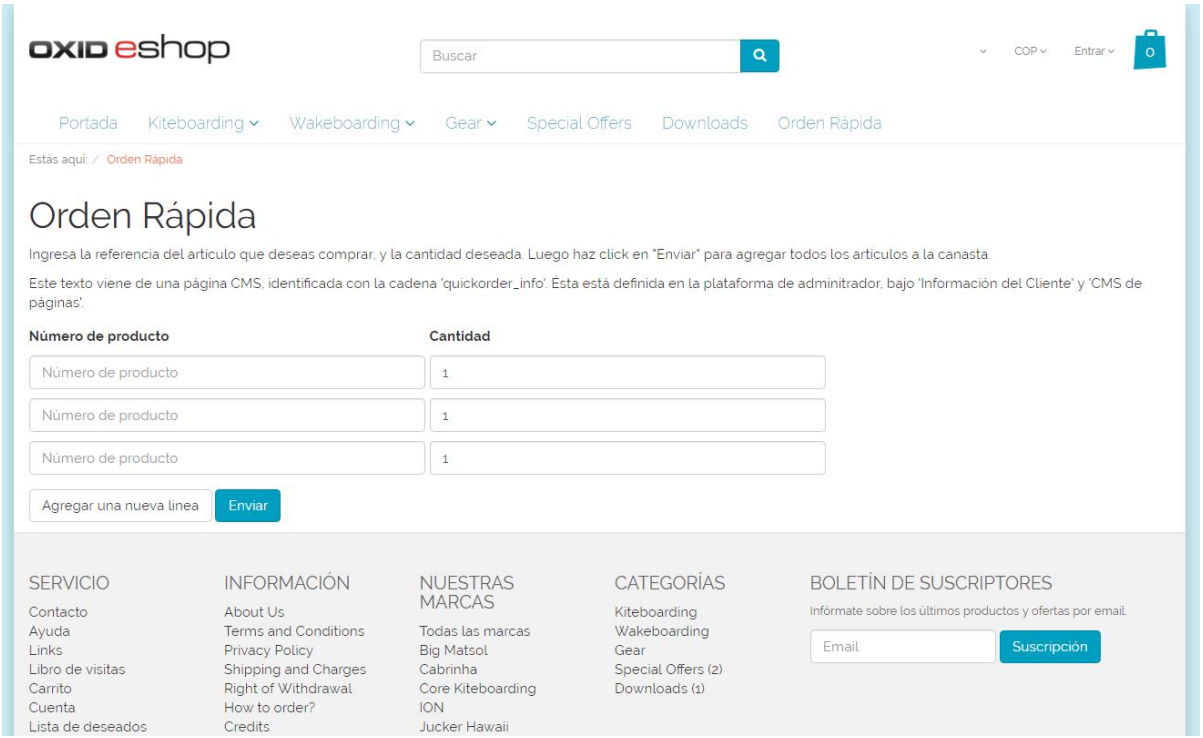


Figura 22: Resultado final del módulo "quickorder"

11. SOLUCIÓN DE PROBLEMAS

La página redirecciona continuamente a la página de inicio, y en el url está definido el parámetro "redirect=1" luego de la redirección

Este problema suele deberse a que hay clases y/o métodos no definidos dentro de la tienda. Esto se puede dar cuando se usa un nuevo método creado en un módulo, y éste no esta activo.

Para conocer exactamente cual es la clase o método problemático se puede consultar el archivo "`EXCEPTION_LOG.txt`", que se encuentra en la carpeta "`log/`" dentro de la raíz del proyecto.

La página redirecciona continuamente a una página en blanco, y en el url esta definido el párametro "redirect=1"

En este caso el problema es probablemente el mismo que en el caso anterior, donde un método o clase no están definidos. La diferencia en este caso es que la clase o método están siendo usados en la página de inicio, lo que no permite que se cargue, mostrando una página en blanco en su lugar.

Consultar el archivo "`EXCEPTION_LOG.txt`", que se encuentra en la carpeta "`log/`" dentro de la raíz del proyecto.

La página redirecciona continuamente a una página en blanco que dice "shop offline", y en la url la página cargada es "/offline.html"

Esto suele deberse a un problema con la base de datos, o con algún método que accede a la misma. Los problemas suelen ser que una columna no existe o que una vista no esta actualizada.

Para conocer exactamente cual es la clase o método problemático se puede consultar el archivo "`EXCEPTION_LOG.txt`", que se encuentra en la carpeta "`log/`" dentro de la raíz del proyecto. Si el problema prueba ser con una vista, (las vistas se identifican porque el nombre de la tabla comienza con "oxv_") estas se deben actualizar. Esto se hace en la plataforma del administrador, bajo "Servicio → Herramientas" y haciendo click en "Actualizar vistas de BBDD ahora".

Cambios realizados en las plantillas no se reflejan en la página

Las tiendas OXID dependen mucho del contenido guardado en la cache, por cuestiones de rendimiento. Esto incluye las plantillas pre-procesadas por Smarty. Cuando esto sucede, se debe vaciar la carpeta "`tmp/`", que es donde se guardan todas las plantillas pre-procesadas. Esto forzará a la tienda a procesar las plantillas una vez mas, incluyendo los últimos cambios realizados en éstas.

Para prevenir tener que hacer esto cada vez que se hacen cambios en las plantillas, se debe desactivar el modo de producción de la tienda y activar la verificación de cambios en las plantillas. El modo de producción se puede desactivar en la plataforma de administrador, bajo "Maestro Configuración → Valores del núcleo", y desactivando la opción "Productivo modo". La verificación de cambios se puede activar en esta misma página, en la pestaña "Rendimiento", activando la primera opción "Detección automática de la nueva plantilla necesaria la compilación".

Con estos cambios no será necesario borrar la cache al hacer cambios en las plantillas de los temas. Sin embargo, al hacer cambios en plantillas que se encuentran dentro de un módulo, sí es necesario vaciar la cache. Esto aplica para plantillas y bloques definidos dentro de los módulos.

Se han modificado o creado nuevas cadenas de texto en los archivos de idiomas, pero estos cambios no se ven reflejados en la página

Al igual que en el punto anterior, las cadenas de texto se guardan en la cache de la tienda, para mejorar su rendimiento. Por esta razón se debe vaciar la carpeta "`tmp/`", que es donde se guarda la cache de las cadenas de texto, cada vez que hay algún cambio que no esté siendo reflejado en la tienda.

Otra posible razón para este problema, es que el archivo de idiomas siendo editado no se encuentre en el lugar correcto. Este problema sólo se ve con los archivos de idiomas de los módulos. Para conocer como manejar los archivos de idiomas dentro de un módulo, ver sección 10.4.2, figura 16. En esta figura se puede ver la estructura correcta para construir un módulo, y donde guardar los archivos de idiomas tanto para el frontend como para el backend.

La página lanza un error 500, de error interno en el servidor

Hay muchas razones por las cuales puede estar ocurriendo esto, y generalmente no es específico de la tienda. Agrega las siguientes lineas en el archivo "`config.inc.php`" para activar el reporte de errores. Esto mostrará todos los errores y advertencias que puedan estar causando problemas en la página.

```
1 error_reporting( E_ALL & ~E_NOTICE );
2 ini_set("display_errors", 1);
```

¿Donde puedo encontrar la documentación completa de OXID?

La documentación completa para las tiendas OXID, versión de comunidad se puede ver en la siguiente página, "`docu.oxid-esales.com/CE/sourcecodedocumentation/`". Allí se selecciona la versión de la tienda que se esté utilizando.

En la documentación se puede ver una lista completa de las clases utilizadas dentro de la tienda, y de los métodos que cada una contiene.

12. BIBLIOGRAFÍA

Apache-Friends (2016). *XAMPP Installers and Downloads for Apache Friends*. URL: https://www.apachefriends.org/index.html.

BuiltWith (2016). *Lists of Ecommerce Websites and Internet Retailers Sales Trends*. URL: http://builtwith.com/ecommerce/.

HeidiSQL (2016). *Download HeidiSQL*. URL: http://www.heidisql.com/download.php?download=installer.

JetBrains (2016). *PhpStorm IDE :: JetBrains PhpStorm*. URL: https://www.jetbrains.com/phpstorm/.

jQuery (2016). *What is jQuery?* URL: https://jquery.com/.

Lerdorf, Rasmus (2007). *PHP on hormones*. URL: http://web.archive.org/web/20130729204354id_/http://itc.conversationsnetwork.org/shows/detail3298.html.

Media, Voltage New (2015). *Shopify vs WooCommerce*. URL: http://www.voltagenewmedia.com/shopify-vs-woocommerce/.

OpenCart (1999). *What Is OpenCart?* URL: http://web.archive.org/web/19991022054418/http://www.opencart.com/.

osCommerce (2016a). *Creating Online Stores Worldwide*. URL: https://www.oscommerce.com.

— (2016b). *Online Merchant*. URL: https://www.oscommerce.com/Products.

OXID (2016a). *OXID eSales — Product Information — OXID eShop Community Edition — Facts — Products*. URL: https://www.oxid-esales.com/en/products/facts/oxid-eshop-community-edition/product-information.html.

— (2016b). *OXID eSales — Product Information — OXID eShop Enterprise Edition — Facts — Products*. URL: https://www.oxid-esales.com/en/products/facts/oxid-eshop-enterprise-edition/product-information.html.

— (2016c). *OXID eSales — Product Information — OXID eShop Professional Edition — Facts — Products*. URL: https://www.oxid-esales.com/en/products/facts/oxid-eshop-professional-edition/product-information.html.

PrestaShop (2016). *About PrestaShop open source shopping cart*. URL: https://www.prestashop.com/en/about-us.

PuTTY (2016). *PuTTY Download Page*. URL: http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html.

TTY-Plus (2012). *TTY Plus*. URL: http://www.ttyplus.com/multi-tabbed-putty/.

W3Techs (2016). *Usage statistics and market share of PHP for websites*. URL: https://w3techs.com/technologies/details/pl-php/all/all.

Wiebe, Brian (2015). *Modifying Hosts File — Okanagan Web Developer - Kelowna Penticton Website Development*. URL: https://okanaganwebdeveloper.com/2015/03/04/modifying-hosts-file/.

Wikipedia (2016a). *Magento*. URL: https://en.wikipedia.org/wiki/Magento.

— (2016b). *PrestaShop*. URL: https://en.wikipedia.org/wiki/PrestaShop.

— (2016c). *SQL*. URL: https://en.wikipedia.org/wiki/SQL.

WinSCP (2016). *WinSCP :: Official Site :: Download*. URL: https://winscp.net/eng/download.php.

Works, Ahead (2016). *Ecommerce Platform Popularity, May 2015: Two Platforms Take Half*. URL: https://blog.aheadworks.com/2015/05/ecommerce-platforms-popularity-may-2015-two-platforms-take-half/.

Zend (2016). *PHP 7 makes powering the web a whole lot better*. URL: http://www.zend.com/en/resources/php-7.

13. PROPIEDAD INTELECTUAL Y DESTINACIÓN DEL PROYECTO

La autoría del proyecto pertenece al estudiante, con participación del director, donde éste proporciona ayuda y asesoría al estudiante que trabaja dentro de un objetivo total y definido por el alumno, y obtiene en consecuencia un reconocimiento como desarrollador del proyecto con reconocimiento para el director en la elaboración del proyecto.

La propiedad patrimonial y moral corresponde a las entidades y unidades que financian el proyecto, la propiedad moral es compartida además con los autores.

A continuación se encuentran incluidas las cartas de intención y apoyo del presente proyecto, junto con el acta de propiedad intelectual.

14. ANEXOS

14.1. Equipo y Material Bibliográfico

El estudiante y director, se comprometen a devolver a la Universidad Pontificia Bolivarianatodo el equipo y material bibliográfico, obtenido con recursos proporcionados por las dependencias académicas y administrativas de la Universidad, una vez termine el proyecto.

14.2. Derechos Morales

Los derechos morales de autor corresponden al estudiante Juan Diego Echeverri Isaza y al director José Valentín Antonio Restrepo Laverde y a toda persona que a criterio de éstos, haga aportes originales intelectuales en los avances y en el resultado final del proyecto. En cualquier tipo de divulgación se dará crédito a los autores y la Universidad Pontificia Bolivariana.

14.3. Derechos Patrimoniales

Los derechos sobre los resultados derivados del presente trabajo de grado se rigen por el Estatuto de Propiedad Intelectual de la Universidad.

14.4. Constancia

Todos los partícipes declaran conocer el Estatuto de Propiedad Intelectual de la Universidad Pontificia Bolivariana. En caso que algún participante se retire antes del 80 % de ejecución del cronograma del proyecto, perderá todos los derechos sobre los resultados de la misma.

En caso de presentarse alguna circunstancia que altere los términos de la presente acta, deberá anexarse al presente documento la respectiva modificación aprobada por la respectiva facultad. Para constancia se firma en Medellín, el _____.

_____

José Valentín Antonio Restrepo Laverde

Director del Proyecto

_____

Juan Diego Echeverri Isaza

Estudiante

14.5. Carta de Presentación de Proyecto

Medellín, noviembre de 2016.

Señores
Consejo de Facultad
Ingeniería Electrónica,
Medellín.

Asunto: Proyecto de Grado.
Cordial saludo;

La presente tiene como objetivo presentar para su estudio por parte de Consejo de Facultad, el Proyecto de Grado titulado: Guiá rápida de configuración de una tienda online usando la plataforma de E-commerce OXID.

Manifiesto además que, conozco el manual de Proyecto de Grado de la Escuela de Ingenierías de la UPB y los deberes y derechos que como Estudiante ésto implica. El desarrollo del proyecto se hará de conformidad con lo estipulado en dicho manual.

Atentamente,

Juan Diego Echeverri Isaza

Estudiante

14.6. Carta del Director

Medellín, noviembre de 2016.

Señores
Consejo de Facultad
Ingeniería Electrónica,
Medellín.

Asunto: Dirección de Trabajo de Grado.
Cordial saludo;

La presente tiene como objetivo confirmar mi intención de participar como Director en el desarrollo del Proyecto de Grado: Guiá rápida de configuración de una tienda online usando la plataforma de E-commerce OXID, que será desarrollado por el estudiante Juan Diego Echeverri Isaza con ID 000174133.

Declaro además que, conozco el manual de Proyecto de Grado de la Escuela de Ingenierías de la UPB y los deberes y derechos que como Director esto implica. El desarrollo del proyecto se hará de conformidad con lo estipulado en dicho manual.

Atentamente,

José Valentín Antonio Restrepo Laverde

Director del Proyecto

14.7. Código fuente

14.7.1. modules/nombredecompania/quickorder/metadata.php

```php
<?php
/**
 * Metadata version
 */
$sMetadataVersion = '1.0';

/**
 * Module information
 */
$aModule = array(
    'id'          => 'quickorder',
    'title'       => array(
        'es_la' => 'Orden rápida',
        'en'    => 'Quickorder',
    ),
    'description' => array(
        'es_la' => 'Agrega la posibilidad de agregar artículos directamente a la canasta ingresando unicamente la referencia del mismo',
        'en'    => "Adds the possibility to add articles directly to the basket, using only the article's reference number" ,
    ),
    'thumbnail'   => 'quickorder_logo.png',
    'version'     => '0.1',
    'author'      => 'Juan Diego Echeverri',
    'url'         => '',
    'email'       => 'juaneche@msn.com',
    'extend'      => array(
      'oxarticle'     => 'nombredecompania/quickorder/models/quickorder_oxarticle',
    ),
    'files'       => array(
        'quickOrder'                => 'nombredecompania/quickorder/controllers/quickorder.php',
        'quickorder_eventHandler'   => 'nombredecompania/quickorder/setup/eventHandler.php',
    ),
    'templates'   => array(
      'quickorder.tpl'        => 'nombredecompania/quickorder/views/page/quickorder.tpl',
      'quickorder_row.tpl'    => 'nombredecompania/quickorder/views/page/inc/quickorder_row.tpl',
    ),
    'blocks'      => array(
        array(
            'template'  => 'widget/header/categorylist.tpl',
            'block'     => 'dd_widget_header_categorylist_navbar_list',
            'file'      => 'views/blocks/quickorder_dd_widget_header_categorylist_navbar_list.tpl'
        ),
    ),
    'settings'    => array(
      array(
        'group' => 'main' ,
        'name'  => 'sQuickOrderInitialLines' ,
        'type'  => 'str' ,
        'value' => '3' ,
      )
    ),
```

```
51      'events'      => array(
52          'onActivate'    => 'quickorder_eventHandler::onActivate',
53          'onDeactivate'  => 'quickorder_eventHandler::onDeactivate',
54      ),
55  );
```

14.7.2. modules/nombredecompania/quickorder/controllers/quickorder.php

```
1   <?php
2
3   class quickOrder extends oxUBase
4   {
5     protected $_sThisTemplate = 'quickorder.tpl';
6
7     /**
8      * Handles the adding of the articles in the basket
9      */
10    public function add()
11    {
12      list($aArticleNumbers, $aArticleAmounts) = $this->getQuickOrderParameters();
13      list($aErrors, $aSuccess) = $this->addArticlesToBasket($aArticleNumbers, $aArticleAmounts
            );
14      $this->setViewMessages($aSuccess, $aErrors);
15    } // function
16
17    /**
18     * Returns the amount of lines to be shown initially
19     *
20     * @return int
21     */
22    public function getInitialLines()
23    {
24      return (int)oxRegistry::getConfig()->getConfigParam('sQuickOrderInitialLines');
25    } // function
26
27    /**
28     * Returns Bread Crumb - you are here page1/page2/page3...
29     *
30     * @return array
31     */
32    public function getBreadCrumb()
33    {
34      $aPaths = array();
35      $aPath = array();
36
37      $iBaseLanguage = oxRegistry::getLang()->getBaseLanguage();
38      $aPath['title'] = oxRegistry::getLang()->translateString('QUICKORDER', $iBaseLanguage,
            false);
39      $aPath['link'] = $this->getLink();
40
41      $aPaths[] = $aPath;
42
43      return $aPaths;
44    }
45
46    /**
47     * Fetches the parameters sent via post to this script, containing the article number and
            the amounts for each
48     * article. Additionally filters out the parameters that were left empty.
```

```
49      *
50      * @return array
51      */
52     private function getQuickOrderParameters()
53     {
54       $oConfig = oxRegistry::getConfig();
55       $aArtNum = $oConfig->getRequestParameter('artnum');
56       $aArtAmount = $oConfig->getRequestParameter('amt');
57
58       foreach ($aArtNum as $iIndex => $sArtnum) {
59         if ($sArtnum === '') {
60           unset($aArtNum[$iIndex]);
61           unset($aArtAmount[$iIndex]);
62         } // if
63       } // foreach
64
65       return array($aArtNum, $aArtAmount);
66     }
67
68     /**
69      * Loads an article from the DB, using only the article number
70      *
71      * @param string $sArtNum
72      * @return bool|oxArticle
73      */
74     private function getArticleByArtNum($sArtNum)
75     {
76       $oArticle = oxNew('oxArticle');
77       return $oArticle->loadByArtNum($sArtNum)
78         ? $oArticle
79         : false;
80     }
81
82     /**
83      * Sets the template variables to be able to display the success and error messages in the
           frontend
84      *
85      * @param array $aSuccess
86      * @param array $aErrors
87      */
88     private function setViewMessages($aSuccess, $aErrors)
89     {
90       $this->addTplParam('aSuccess', $aSuccess);
91       $this->addTplParam('aError', $aErrors);
92     }
93
94     /**
95      * Attempts to add all the articles to the basket, and returns two arrays, one containing
           the ones that were added,
96      * and another containing the ones that were not
97      *
98      * @param array $aArticleNumbers
99      * @param array $aArticleAmounts
100     * @return array
101     */
102    public function addArticlesToBasket($aArticleNumbers, $aArticleAmounts)
103    {
104      $oBasket = oxRegistry::getSession()->getBasket();
105      $aErrors = $aArticleNumbers;
106      $aSuccess = array();
```

```
107
108     foreach ($aArticleNumbers as $iIndex => $sArtNum) {
109       if ($sArtNum) {
110         if ($oArticle = $this->getArticleByArtNum($sArtNum)) {
111           if ($oArticle->isBuyable()) {
112             if ((int)$aArticleAmounts[$iIndex] > 0) {
113               $aSuccess[$iIndex] = $sArtNum;
114               unset($aErrors[$iIndex]);
115               $oBasket->addToBasket($oArticle->getId(), $aArticleAmounts[$iIndex]);
116             } // if
117           } // if
118         } // if
119       } // if
120     } // foreach
121     return array($aErrors, $aSuccess);
122   }
123 }
```

14.7.3. modules/nombredecompania/quickorder/models/quickorder_oxarticle.php

```
1  <?php
2
3  class quickorder_oxarticle extends quickorder_oxarticle_parent
4  {
5    /**
6     * Loads an article from the DB, using only the article number.
7     *
8     * @param string $sArtNum
9     * @return bool
10    */
11   public function loadByArtNum($sArtNum)
12   {
13     $oDb = oxDb::getDb();
14     $iShopId = oxRegistry::getConfig()->getShopId();
15     $sViewName = getViewName($this->_sCoreTable);
16     $sSql = "SELECT * FROM {$sViewName} WHERE oxshopid = '{$iShopId}' AND oxartnum = " . $oDb
           ->quote($sArtNum) . " LIMIT 1";
17     return $this->assignRecord($sSql);
18   } // function
19 }
```

14.7.4. modules/nombredecompania/quickorder/out/src/css/quickorder.css

```
1  .quickorder__input-table{
2    width: 100%;
3    max-width: 80rem;
4    margin-bottom: 1rem;
5  }
6
7  .quickorder__input-table th, .quickorder__input-table td {
8    padding: 0.5rem;
9    padding-left: 0;
10 }
11
12 .quickorder__buttons{
13   margin-bottom: 1rem;
14 }
```

14.7.5. modules/nombredecompania/quickorder/out/src/js/quickorder.js

```
$(function quickorder(){
  function addNewInputsRow() {
    var oClone = $('tbody', '#clone_quickorder_row');
    $('tbody', '.quickorder__input-table').append(oClone.html());
  }

  $(document).ready(function(){
    $('#quickorder--add-new-line').click(function(){
      addNewInputsRow();
    })
  })
});
```

14.7.6. modules/nombredecompania/quickorder/setup/eventHandler.php

```
<?php

class quickorder_eventHandler
{
  static function onActivate()
  {
    $sFilename = __DIR__ . '/onActivate.sql';
    self::executeSQLFile($sFilename);
  }

  static function onDeactivate() {
    $sFilename = __DIR__ . '/onDeactivate.sql';
    self::executeSQLFile($sFilename);
  }

  private static function executeSQLFile($sFilename)
  {
    if (is_file($sFilename)) {
      $sSql = file_get_contents($sFilename);
      if ($sSql) {
        $oDb = oxDb::getDb();
        $oDb->execute($sSql);
      } // if
    } // if
  }
}
```

14.7.7. modules/nombredecompania/quickorder/setup/onActivate.sql

```
REPLACE INTO `oxcontents` (`OXID`, `OXLOADID`, `OXSHOPID`, `OXSNIPPET`, `OXTYPE`, `OXACTIVE`, `OXACTIVE_1`, `OXPOSITION`, `OXTITLE`, `OXCONTENT`, `OXTITLE_1`, `OXCONTENT_1`, `OXACTIVE_2`, `OXTITLE_2`, `OXCONTENT_2`, `OXACTIVE_3`, `OXTITLE_3`, `OXCONTENT_3`, `OXCATID`, `OXFOLDER`, `OXTERMVERSION`, `OXTIMESTAMP`) VALUES ('6155f9057ca0f8ebe2df5778a9636f08', 'quickorder_info', 'oxbaseshop', 1, 0, 1, 0, '', 'Quickorder Info', '<p>Enter the article number and the desired amount for the article that you wish to purchase. After that, click in the "Submit" button to add them directly to your basket.</p><p>This text is coming from a CMS page, identified by the ident \'quickorder_info\'. It is defined in the administration platform, under \'Customer Info\' and \'CMS pages\'.</p>', '', '', 1, 'Quickorder Info', '<p>Ingresa la referencia del artículo que deseas comprar, y la cantidad deseada. Luego haz click en "Enviar" para agregar todos los artículos a la canasta.</p><p>Este texto viene de una página CMS, identificada con la cadena \'quickorder_info\'. Ésta está definida en la plataforma de adminitrador, bajo \'Información del Cliente\' y \'CMS de páginas\'.</p>', 0, '', '', '943a9ba3050e78b443c16e043ae60ef3', '', '', '2016-10-23 13:40:27');
```

14.7.8. modules/nombredecompania/quickorder/setup/onDeactivate.sql

```
# Vacio
```

14.7.9. modules/nombredecompania/quickorder/translations/en/quickorder_lang.php

```
<?php
$sLangName = "English";
$aLang = array(
  "charset"                          => "UTF-8",

  "QUICKORDER"                       => "Quick Order",
  "QUICKORDER_ARTICLE_ADDED"         => "Article added: %s",
  "QUICKORDER_ARTICLE_NOT_AVAILABLE" => "This article does not exist or is not buyable: %s",
  "QUICKORDER_ADD_NEW_LINE"          => "Add a new line",
);
```

14.7.10. modules/nombredecompania/quickorder/translations/es_la/quickorder_lang.php

```
<?php
$sLangName = "Español";
$aLang = array(
  "charset"                          => "UTF-8",

  "QUICKORDER"                       => "Orden Rápida",
  "QUICKORDER_ARTICLE_ADDED"         => "Artículo añadido: %s",
  "QUICKORDER_ARTICLE_NOT_AVAILABLE" => "Este artículo no esxiste o no se puede comprar: %s",
  "QUICKORDER_ADD_NEW_LINE"          => "Agregar una nueva linea",
);
```

14.7.11. modules/nombredecompania/quickorder/views/admin/en/quickorder_lang.php

```
<?php
$sLangName = "English";
$aLang = array(
  "charset"                                    => "UTF-8",

  "SHOP_MODULE_GROUP_main"                     => "General",
  "SHOP_MODULE_sQuickOrderInitialLines"        => "Initial lines",
  "HELP_SHOP_MODULE_sQuickOrderInitialLines"   => "Determines how many input lines will be
      shown by default in the page.",
);
```

14.7.12. modules/nombredecompania/quickorder/views/admin/es_la/quickorder_lang.php

```
<?php
$sLangName = "Español";
$aLang = array(
  "charset"                                    => "UTF-8",

  "SHOP_MODULE_GROUP_main"                     => "General",
  "SHOP_MODULE_sQuickOrderInitialLines"        => "Lineas iniciales",
  "HELP_SHOP_MODULE_sQuickOrderInitialLines"   => "Esto determina cuantos campos por defecto
      se mostrarán en la página para la orden rápida.",
);
```

14.7.13. .../quickorder/views/blocks/quickorder_dd_widget_header_categorylist_navbar_list.tpl

```
[{$smarty.block.parent}]
[{assign var="oConfig" value=$oView->getConfig()}]

<li>
  <a href="[{oxgetseourl ident=$oConfig->getShopCurrentUrl()|cat:'cl=quickorder'}]">[{
      oxmultilang ident="QUICKORDER"}]</a>
</li>
```

14.7.14. modules/nombredecompania/quickorder/views/page/quickorder.tpl

```
[{capture append="oxidBlock_content"}]
  [{assign var="sModulePath" value=$oViewConf->getModuleUrl('quickorder')}]
  [{assign var="oConfig" value=$oView->getConfig()}]
  [{assign var="iInitialLines" value=$oView->getInitialLines()}]

  [{oxstyle include=$sModulePath|cat:'out/src/css/quickorder.css'}]

  [{if $aSuccess}]
    <p class="alert alert-success">
      [{foreach from=$aSuccess item="sArtnum"}]
        [{if $sArtnum}]
          [{oxmultilang ident="QUICKORDER_ARTICLE_ADDED" args=$sArtnum}]
          <br>
        [{/if}]
      [{/foreach}]
    </p>
  [{/if}]

```

```
19    [{if $aError}]
20      <p class="alert alert-warning">
21        [{foreach from=$aError item="sArtnum"}]
22          [{if $sArtnum}]
23            [{oxmultilang ident="QUICKORDER_ARTICLE_NOT_AVAILABLE" args=$sArtnum}]
24            <br>
25          [{/if}]
26        [{/foreach}]
27      </p>
28    [{/if}]
29
30    <h1>[{oxmultilang ident="QUICKORDER"}]</h1>
31
32    [{oxifcontent ident="quickorder_info" object="oContent"}]
33      <div>
34        [{$oContent->oxcontents__oxcontent->value}]
35      </div>
36    [{/oxifcontent}]
37
38    <form action="[{$oConfig->getShopCurrentUrl()}]" method="post">
39      <div>
40        <input type="hidden" name="cl" value="quickorder">
41        <input type="hidden" name="fnc" value="add">
42      </div>
43
44      <table class="quickorder__input-table">
45        <tr class="quickorder__input-row">
46          <th>[{oxmultilang ident="ARTNUM"}]</th>
47          <th>[{oxmultilang ident="QUANTITY"}]</th>
48        </tr>
49
50        [{section name="quickOrder_rows" loop=$iInitialLines}]
51          [{include file="quickorder_row.tpl"}]
52        [{/section}]
53      </table>
54
55      <div class="quickorder__buttons">
56        <a id="quickorder--add-new-line" href="javascript: void(0)" class="btn btn-default">[{
              oxmultilang ident="QUICKORDER_ADD_NEW_LINE"}]</a>
57        <button class="btn btn-primary" type="submit">[{oxmultilang ident="SUBMIT"}]</button>
58      </div>
59    </form>
60
61    <table id="clone_quickorder_row" style="display: none;">
62      [{include file="quickorder_row.tpl"}]
63    </table>
64
65    [{oxscript include=$sModulePath|cat:'out/src/js/quickorder.js'}]
66    [{insert name="oxid_tracker" title=$template_title}]
67  [{/capture}]
68
69  [{include file="layout/page.tpl"}]
```

14.7.15. modules/nombredecompania/quickorder/views/page/inc/quickorder_row.tpl

```
<tr class="quickorder__input-row">
  <td>
    <input
        class="form-control"
        type="text"
        name="artnum[]"
        placeholder="[{oxmultilang ident="ARTNUM"}]">
  </td>

  <td>
    <input
        class="form-control"
        type="number"
        min="1"
        name="amt[]"
        value="1"
        placeholder="[{oxmultilang ident="QUANTITY"}]">
  </td>
</tr>
```

14.8. Referencia Smarty

En esta sección se mostrará una referencia general sobre las principales funciones de Smarty que se usan dentro de las tiendas OXID. Para conocer todas las posibilidades se pueden mirar las carpetas "`core/smarty/internals`" y "`core/smarty/plugins`". En estas carpetas están las definiciones de todas las funciones y modificadores disponibles dentro del sistema de plantillas Smarty. Cada funcionalidad tiene su propio archivo correspondiente.

14.8.1. Funciones

assign

La función "assign" permite crear una variable y asignarle un valor, que se puede utilizar mas adelante en la plantilla. Recibe dos parámentros, "var" y "value". "var" define el nombre de la variable a asignar, mientras que "value" su valor.

```
1 [{assign var="currency" value=$oView->getActCurrency()}]
```

include

La función "include" permite inyectar otras plantillas dentro de la actual. Esto facilita la reutilización de código. Esta función recibe un parámetro requerido, "file", que determina el archivo de la plantilla a cargar.

Adicionalmente puede recibir parámetros adicionales que le darán valor a variables dentro de la plantilla usada. Por ejemplo, se puede tener una plantilla que muestre el nombre del usuario, y este se le tenga que pasar a la misma; o una plantilla que muestre errores, y estos se tengan que definir.

```
1 [{include file="inc_error.tpl" Errorlist=$Errors.default}]
```

Dentro de la plantilla se puede usar la variable *$Errorlist* para acceder a los errores definidos en la llamada.

if

Esta función permite evaluar una condición, e incluir el código dentro del bloque en caso de que la condición se cumpla.

```
1 [{if $Errors.default|@count > 0 }]
2   [{include file="inc_error.tpl" Errorlist=$Errors.default}]
3 [{/if }]
```

foreach

Esta función permite recorrer un arreglo u objeto, usando cada elemento para ejecutar el código en el interior del bloque. Recibe dos parámetros requeridos, “from” e “item”. Adicionalmente puede recibir otros parámetros como “key” y “name”.

El parámetro “from” define el objeto a recorrer. El parámetro “item” define el nombre que tendrá cada elemento dentro del bloque, y del cual se podrá referenciar. El parámetro “key” contiene el nombre del indice de cada elemento, en caso de que sea necesario. El parámetro “name” le da un nombre al *foreach* en sí, lo que permite referenciarlo directamente, y obtener ciertos valores como el número de la iteración, si es la primera o la última iteración, entre otras.

```
1  [{foreach from=$aLanguages item=oLang key=iLang name="languages"}]
2    <option value="[{ $iLang }]" [{ if $oLang->selected }]SELECTED[{/if}]>
3      [{ $oLang->name }]
4    </option>
5    [{* Retorna la iteración *}]
6    [{$smarty.foreach.languages.iteration}]
7    [{* Retorna verdadero si es la primera iteración *}]
8    [{$smarty.foreach.languages.first}]
9    [{* Retorna verdadero su es la última iteración *}]
10   [{$smarty.foreach.languages.last}]
11 [{/foreach}]
```

section

Esta función permite repetir código por una cantidad de veces conocida. Tiene una funcionalidad similar a la función *for*, aunque ésta no existe en Smarty para OXID.

Puede recibir cuatro parámetros, “start”, “loop”, “step”, y “name”. El parámetro “name” cumple una funcionalidad similar a aquella del parámetro “name” en la función foreach, que permite referenciar a este ciclo, para obtener el número de la iteración, si es la primera, si es la última, el total de iteraciones, etc. Los parámetros “start”, “loop”, y “step” permiten configurar de que manera se harán las repeticiones, y cuantas se harán. Basicamente comienza con un indice definido por “start”, cada iteración aumenta el valor definido por “step”, y repite este proceso hasta que el índice sea igual o mayor a “loop”. Cuando

esta condición se cumple, el contenido del bloque no se muestra. Es decir, el valor definido en "loop" es excluyente.

```
[{section name="thisloop" start=5 loop=13 }]
  [{$smarty.section.thisloop.index}]
  [{* Retorna 5, 6, 7, ... *}]
  [{$smarty.section.thisloop.first}]
  [{* Retorna true, false, false, ... *}]
  [{$smarty.section.thisloop.last}]
  [{* Retorna ..., false, false, true *}]
  [{$smarty.section.thisloop.iteration}]
  [{* Retorna 1, 2, 3, ... *}]
[{/section}]
```

oxcontent

La función *oxcontent* permite mostrar una página de contenido del sistema CMS. Recibe un parámetro requerido, "ident". Éste define el identificador de la página de contenido a mostrar.

```
[{oxcontent ident="oxemailfooter"}]
```

oxifcontent

Esta función permite cargar una página de contenido del sistema CMS, y ejecutar todas las instrucciones dentro del bloque sólo si la página existe. Recibe dos parámetros, "ident" y "object". "ident" define el identificador de la pagina de contenido a cargar. "object" contendrá una instancia de la clase *oxContent*, que corresponde a la página definida.

La ventaja de esta función con respecto a "oxcontent", es que permite cargar con igual facilidad otros aspectos de la página de contenido, como el título, el link al que corresponde, entre otros atributos.

```
[{oxifcontent ident="oxsecurityinfo" object="oCont"}]
  <p>
    [{oxmultilang ident="MESSAGE_READ_DETAILS"}]
    <a href="[{$oCont->getLink()}]" rel="nofollow">
      [{$oCont->oxcontents__oxtitle->value}]
    </a>
  </p>
```

```
8 [{/oxifcontent}]
```

oxgetseourl

Esta función retorna una la URL SEO de la página que se especifique. Recibe un parámetro, "ident", que contiene la url a interpretar

```
1 [{oxgetseourl ident="index.php?cl=account"}]
2 [{* Retorna 'http://www.my-shop.dev/en/my-account/' *}]
```

oxmultilang

Esta función retorna cadenas de texto definidas en los archivos de idiomas. Recibe un parámetro requerido, "ident", que define el identificador de la cadena a cargar. Adicionalmente se puede definir el parámetro "args", que permite usar valores cargados en la plantilla dentro de la cadena definida, usando el placeholder "%s".

Si se tiene la siguiente cadena, definida en un archivo de idiomas,

```
1 "NUMBER_OF_VIEWS"    =>    "Visto %s veces",
```

Se puede llamar en la plantilla de la siguiente forma,

```
1 [{oxmultilang ident="NUMBER_OF_VIEWS" args=$oDetailsProduct->
    oxarticles__views->value}]
```

oxprice

Esta función se usa para mostrar precios, utilizando el formato correcto dependiendo de la moneda usada, de acuerdo a lo definido en la sección 10.3.3. Recibe dos parámetros, "price" y "currency". "price" puede ser un número o una instancia de la clase *oxPrice*. "currency" puede ser una cadena o una instancia de la clase *oxCurrency*, y define la moneda y el formato del precio que se mostrará. Si "currency" no se define, la tienda usará la primera moneda de la lista de monedas definida en la sección 10.3.3.

```
1 [{oxprice price=$basketitem->getRegularUnitPrice() currency=$currency}]
```

oxscript

Esta función se encarga de recolectar todos los archivos e instrucciones de JavaScript usados, y los ubica al final de la página. Por esta razón se puede definir en cualquier parte de la plantilla, y Smarty se encarga de organizarlos posteriormente. Hay dos formas de usar esta función, una es escribiendo directamente las instrucciones que se desean incluir, usando el parámetro "add". La segunda forma de usarlo es incluyendo archivos de JavaScript, usando el parámetro "include".

```
1 [{oxscript add="alert('test');"}]6
2 [{oxscript include="oxid.js"}]
```

oxstyle

Esta función se encarga de recolectar todos los archivos de estilos, y los carga juntos en el header de la página. Usa el parámetro "include" para definir el archivo que se desea cargar.

```
1 [{oxstyle include="oxid.css"}]
```

14.8.2. Modificadores

Los modificadores permiten modificar los elementos a los que se les aplican, usualmente cadenas, para obtener un elemento procesado. Algunos modificadores reciben parámetros, los cuales se listan luego del modificador, separados por dos puntos (:) (ver "replace" o "cat"). Los modificadores se ubican después de la variable a la cual se le va a aplicar, anteponiéndolos por una barra (|). Adicionalmente los modificadores se pueden encadenar, y son evaluados de izquierda a derecha (ver ejemplo de "oxescape").

var_dump

Este modificador permite mostrar todo el contenido de un objeto en la página. Es una excelente herramienta de desarrollo.

```
[{$oxcmp_user|@var_dump}]
```

capitalize

Este modificador capitaliza todas las palabras de la cadena a la que se le aplique.

```
[{'hola mundo'|capitalize}]
[{* Retorna 'Hola Mundo' *}]
```

lower

Este modificador retorna la cadena a la que se le aplica, con todas sus letras en minúscula.

```
[{'HOlA MunDo'|lower}]
[{* Retorna 'hola mundo' *}]
```

cat

Este modificador permite concatenar varias cadenas, para pasárselas a un método o función.

```
[{section name="month" start=1 loop=13 }]
  <option value="[{$smarty.section.month.index}]">
    [{oxmultilang ident="MONTH_NAME_"|cat:$smarty.section.month.index}]
  </option>
[{/section}]
```

nl2br

Este modificador procesa las cadenas de texto para prepararlas para ser mostradas en una página web, convirtiendo los quiebres de linea estándar (`\n` , `\r` , `\r\n`) por quiebres de linea de HTML (`</br>`). Esto suele ser necesario cuando se manejan bloques de texto ingresados por el usuario, que fueron guardados en la base de datos con los quiebres de linea estándar.

```
1 [{$oxcmp_basket->getCardMessage()|nl2br}]
```

oxescape

Este modificador permite escapar bloques de texto para ciertos lenguajes, de manera que no sean interpretados como tal. Por defecto escapa para HTML, pero también se le puede especificar el lenguaje como su primer argumento. Los valores admitidos son `html`, `htmlall`, `url`, `quotes`, `hex`, `hexentity`, y `javascript`.

```
1 [{$userInfo->send_message|oxescape:'html'|nl2br}]
```

oxmultilangassign

Este modificador procesa el identificador que se le entregue, y retorna la cadena de texto correspondiente, de los archivos de idiomas.

```
1 [{assign var="sFrom" value="PRICE_FROM"|oxmultilangassign}]
```

En este caso, contando con la siguiente definición de la cadena “PRICE_FROM” en los archivos de idiomas,

```
1 'PRICE_FROM' => 'De',
```

el modificador retornaría “De”, y lo guardaría en la variable “sFrom”.

replace

Este modificador permite buscar y reemplazar una parte de una cadena, y reemplazarla por otra. Recibe dos parámetros, primero la cadena a buscar, y segundo la cadena que se usará para reemplazarla.

Nótese que cuando se le necesita enviar parámetros a un modificador, se listan luego de éste separados por dos puntos (:).

```
1 [{$basket->getFVoucherDiscountValue()|replace:"-":""}]
```

regex_replace

Este modificador permite buscar y reemplazar una parte de una cadena, usando una expresión regular para ubicarla, y reemplazarla por otra. Recibe dos parámetros, primero la expresión a buscar, y segundo la cadena que se usará para reemplazarla.

El segundo argumento soporta utilizar placeholders para mostrar grupos que hayan sido capturados. Estos placeholders se definen como "\1", "\2", etc. En general se utiliza la nomenclatura definida para expresiones regulares en PHP.

Nótese que cuando se le necesita enviar parámetros a un modificador, se listan luego de éste separados por dos puntos (:).

```
1 [{assign var="iBirthdayDay" value=$oxcmp_user->oxuser__oxbirthdate->value
    |regex_replace:"/^([0-9]{4})[-]([0-9]{1,2})[-]/":"" }]
```

Este ejemplo toma el cumpleaños del usuario, que se guarda en el formato "2016-10-30", y reemplaza el año y el mes por una cadena vacía, eliminándolos de la cadena. Esto da como resultado "30" y se guarda en la cadena "iBirthdayDay".

truncate

Este modificador corta una cadena hasta un punto definido, 80 caracteres por defecto, y elimina lo restante. Por defecto se agrega una elipsis (...) al final de la cadena, para evidenciar que la cadena fue truncada, pero esto se puede modificar usando el segundo parámetro.

```
1 [{$product->oxarticles__oxshortdesc->value|truncate:160:"..."}]
```

wordwrap

Este modificador permite agregar quiebres de linea a un bloque de texto, cada cierta cantidad de caracteres, 80 por defecto. Como segundo parámetro se puede definir el caracter o la cadena que se desea usar para quebrar las lineas. En este caso, se puede usar "</br>" en lugar del valor predeterminado "\n", cuando se desea mostrar HTML.

```
1 [{$product->oxarticles__oxshortdesc->value|wordwrap:80:"<br>"}]
```