



webStore

Documentación

Pedro Pablo Carranza

A large, abstract 3D graphic at the bottom of the page. It consists of several overlapping, semi-transparent blue and grey rectangular blocks arranged in a complex, geometric pattern. The blocks are oriented at various angles, creating a sense of depth and dimension. The overall shape is roughly rectangular but with many internal facets and edges visible.

2.8.2

Tabla de contenido

Carga y ejecución de webStore	3
Secuencia de carga de webStore	3
Fichero index.php.....	3
Clases y controladores:	4
Clases.....	4
Objeto.....	4
Controladores.....	5
ArmiSync.....	5
BaseDatos	5
BasicObject	5
Component.....	5
ConfigFile.....	5
FileSystem	5
Getter	5
JShrink	5
JSqueeze	5
Minify	5
Module	5
PHPWee.....	6
Price.....	6
RemoteRequest.....	6
Sections	6
Sesion	6
Theme.....	6
WSLocale	6
WSplFileObject	6
Librerías externas	6
Funciones remotas.....	6
Eventos Javascript	7
Evento de carga.....	7
Acciones	8
Módulos	8
Caché	8

cache	9
clases	9
controlFiles	9
controllers	9
file_index.....	9
media.....	9
sections.....	9

Carga y ejecución de webStore

Para ejecutar webStore tenemos que tenerlo copiado en nuestro hosting de manera que tengamos acceso al fichero `initWebStore.php` que se encuentra dentro de la carpeta “core”. Incluyendo este fichero, estaremos iniciando el core de webStore.

Secuencia de carga de webStore

El fichero `initWebStore.php` realiza las siguientes acciones y en este orden:

1. Se definen las constantes públicas de la web.
 - a. Aquellas constantes que identifiquen rutas del core lo harán siempre de manera relativa a la ruta desde la que hemos hecho el include.
2. Lanzamos los “_defines” del usuario. ****Esto acabará por desaparecer, en su lugar tendremos una tabla llamada “tienda” encargada de la configuración propia de la tienda. Para esto habrá que hacer una reestructuración de la información almacenada en webStore. La idea es que termine siendo multi-tienda. ****
3. Creamos una función que capturará las excepciones que se hayan lanzado para que, sea el propio webStore quien las capture.
4. Por compatibilidad con webStore viejo ejecutamos un `set_include_path`
5. Se crean los directorios dinámicos (caché)
6. Ejecutamos el autoload de webStore
7. Iniciamos el motor de base de datos
8. Cargamos la configuración local. *****Por ahora aquí sólo se carga el tema actual, esto también hay que cambiarlo, el tema se cargará de la tienda activa*****
9. Iniciamos la tienda y la sesión de PHP
10. Se comprueban y lanzan los controles de acciones controladas por módulos
11. Se lanza la función de captura de pantalla *****esto se debe hacer de otra manera*****
12. Lanzamos una función de retro-compatibilidad con la vieja versión de webStore *****tiene que desaparecer en el futuro*****
13. Cargamos el índice de ficheros activos para el tema actual. *****Esto hay que revisarlo también. La idea es que el admin cargue un tema de webStore con sus propios módulos y que esté almacenado en otro sitio.*****
14. Por si se ha lanzado webStore de manera remota, hacemos un commit de la sesión. *****No cierra la base de datos, en funciones remotas hay que revisar que no se quede abierta*****

Fichero `index.php`

Este fichero será el fichero raíz de la web. Ahora mismo hace lo siguiente:

1. Inicia webStore
2. Carga el control de secciones. *****Esto debería hacerse desde un controlador*****
3. Recarga la página si es necesario
4. Ejecuta el fichero de la plantilla que corresponda según la sección actual
5. Añade el script básico que SIEMPRE debe tener webStore cargado.
6. Hace un commit en la sesión

7. Cierra la conexión en la base de datos. ***Ver cómo se puede automatizar esto.***

Clases y controladores:

Las clases y controladores que utiliza webStore son clases PHP que gestionan la funcionalidad, casi por completo, de la web. En la mayoría de los casos utilizan funciones estáticas para poder ser llamadas sin instanciar la clase. Su principal diferencia es que utilizamos las clases para representar objetos cargando datos desde la base de datos, mientras que los controladores son clases que afectan al comportamiento de la web, a ficheros y a otros elementos que no tienen nada que ver con la base de datos. Esto no quiere decir que un controlador no se comunique con la base de datos para funciones de uso propio. Las clases PHP están definidas como súper clases para poder ser extendidas después. Así si tenemos la clase `SuerArticulo`, podemos extenderla después en la clase `Articulo` que será la que se utilizará. **Nunca** se utilizan las súper clases de manera directa. WebStore buscará la clase sobre escrita en su carpeta correspondiente, en caso de no encontrarla creará automáticamente la clase y la incluirá en la web. La carpeta de sobre escritura se almacena en la variable `_WS_OVERRIDE_DIR_` y dentro de dicha carpeta tendremos la carpeta “classes” que almacena las clases y “controllers” que almacena los controladores

- Clases:
 - Ruta de la carpeta: `_WS_DIR_/core/classes/`
 - Variable de sistema: `_WS_CLASSES_DIR_`
 - Carpeta de sobre escritura: `_WS_DIR_/override/classes/`
- Controladores
 - Ruta de la carpeta: `_WS_DIR_/core/controllers/`
 - Variable de sistema: `_WS_CONTROLLERS_DIR_`
 - Carpeta de sobre escritura: `_WS_DIR_/override/controllers/`

Clases

Por ahora no vamos a describir todas las clases, basta con decir que los objetos así instanciados se corresponden con entidades de la base de datos. ***Tenemos objetos que ya se cargan utilizando la nueva metodología y objetos que aún usan la antigua. Es necesario actualizarlos todos***. Vamos a destacar por ahora sólo la clase `Objeto`. Todas las clases que realicen una carga directa de una tabla de la base de datos (artículos, clientes, etc.).

Objeto

Esta clase es la que nos da las funciones de carga para que cualquier clase que la extienda pueda cargar datos de la base de datos. La idea es que cada clase se corresponda con una tabla de la base de datos. En algunas ocasiones no se llaman igual y esta clase contiene el traductor que identificará la tabla desde la clase y viceversa. Las funciones que provee deben ser sobre escritas en el caso de clases complejas.

Controladores

ArmiSync

Es el encargado de realizar la sincronización con ficheros generados por programas Arminet. Al margen de sobrescribir métodos propios de esta clase podemos personalizar la lectura de los ficheros teniendo en cuenta estas 2 funciones:

1. *nombreTabla_procesarFichero*(\$ruta_del_fichero, \$linea_de_inicio);
2. *post_nombreTabla_procesarFichero*(\$ruta_del_fichero, \$linea_de_inicio);

En *nombreTabla* indicaremos el nombre de la tabla de la Base de Datos que estamos procesando. Si escribimos la primera función para una tabla concreta, se ejecutará esta función en vez de la función nativa (por defecto se usa en la tabla categorías-productos). En la segunda función podemos lanzar un proceso que se ejecutará justo después del procesado de la tabla (por defecto se usa para reindexar productos después de haberlos leído).

BaseDatos

Controla el acceso a la base de datos

BasicObject

Únicamente se utiliza como superclase. Los únicos métodos que tiene son unos que permiten crearle propiedades estáticas para leer y escribir en ellas.

Component

Por ahora no se usa uno sabemos aún si se usarán o no. Su propósito será ejecutar componentes declarados en los módulos.

ConfigFile

Extiende a BasicObject. Lo que hace es que el objeto "BasicObject" se almacene de manera serializada en un fichero. Pudiendo abrirlo y para su lectura en cualquier momento.

FileSystem

Controlador que se utiliza para gestionar funciones complejas de ficheros y directorios que no existan en PHP. Está en crecimiento aún.

Getter

Con este controlador recogemos la información recibida por GET y/o POST.

JShrink

Se usa para cargar esta librería externa que permite minificar javascript.

JSqueeze

Se usa para cargar esta librería externa que permite minificar javascript.

Minify

Se usa para cargar esta librería externa que permite minificar css.

Module

Controlador que se encargará de gestionar los módulos. Está por escribir.

PHPWee

Se usa para cargar esta librería externa que permite minificar css.

Price

Con este controlador hemos añadido una función estática que calcula el precio de un artículo. ***No es un controlador definitivo porque debe terminar por desaparecer cuando encontremos la mejor forma de hacer esto.***

RemoteRequest

Controlador que gestiona las peticiones remotas, las ejecuta, lanza los módulos necesarios y devuelve la información.

Sections

Controlador que se usa para gestionar las secciones de la web

Sesion

Este controlador gestiona las sesiones. En webStore no se almacenan las sesiones con la propia sesión de PHP. En su lugar se usa este objeto que escribe y lee los datos de una tabla de la base de datos

Theme

Controlador que se encarga de coordinar los componentes del tema.

WSLocale

Este controlador se encarga de gestionar la configuración regional de la web. ***Por ahora webStore no es multi-idioma, pero en cuanto lo sea, este controlador se encargará de las traducciones con un método estático.****

WSplFileObject

La clase SplFileObject contiene una serie de funciones muy útiles. Sin embargo no todas funcionan como queremos. Es por eso que con esta clase hemos modificado las funciones para adaptarlas a webStore.

Librerías externas

Dentro de la carpeta del core hay una carpeta que contendrá las librerías externas junto con su lógica. El nombre de esta carpeta es "lib". La manera de cargar estas librerías ya depende del uso que se le quiera dar y de la lógica de cada librería. Por ejemplo para las librerías de compresión de código se han creado unas super clases PHP para que se puedan instanciar desde donde queramos en el momento que queramos. Si no hay que minificar código no se instanciarán y evitaremos sobrecargar el sistema. En el caso de la librería de envío de email sólo se instancia cuando se va a enviar un mail.

Funciones remotas

Desde JavaScript podemos ejecutar funciones remotas definidas en las clases. Para ello tienen que estar declaradas como estáticas y públicas y su nombre debe empezar por "static".

Además el nombre de la función debe ser “camel case”. De esta forma, si queremos obtener los datos de un producto, tendríamos que crear una función dentro de la ficha de esta manera:

```
public static function remoteDatosProducto($data){  
    .....  
}
```

Esta función recibe la variable \$data que es un objeto que contiene en sus propiedades las variables recibidas vía javascript.

Para llamar a estas funciones remotas, debemos componer un objeto Json con la información que vamos a mandar, la clase/controlador a llamar y la función remota a ejecutar:

```
var dataToSend = {  
    controller:"CabeceraCesta",    //clase o controlador a utilizar  
    method:"emptyCart",           //Método a llamar  
    data: data,                   //Datos a enviar  
};
```

Vemos que el método no lleva la palabra remote ni la primera letra en mayúscula.

El campo “data” es un objeto que contiene los campos con su respectivo valor.

Esta información ha de enviarse al servidor como texto plano. Para ello convertimos el objeto JavaScript a Json y lo pasamos a base64 utilizando una función en javascript propia porque la nativa no lo hace bien con las tildes

Cuando enviemos por ajax la información, recibiremos la respuesta en el mismo formato, en base64 y Json.

Eventos Javascript

Evento de carga

Se declara un evento llamado “WSLE” (Web Store Load Event) que es el encargado de lanzarse cuando se haya cargado la página. Para asignarlo haremos lo siguiente en javascript:

```
document.addEventListener('WSLE', function(){  
  
});
```

Y dentro de la función especificaremos el código a ejecutar

Acciones

En webStore una acción se lanza cuando se recibe la variable “action” por GET o por POST (el método es indiferente pero siempre tendrá prioridad el POST. El valor de la variable será el nombre de la acción que se ha lanzado (login, registro, etc.).

Se puede usar para recibir datos de un formulario que se envíe por GET o POST. Hay que tener en cuenta que en webStore se pueden enviar datos por Ajax mediante las funciones remotas.

Ahora mismo se cargará el fichero de acciones de cada módulo para comprobar si carga acciones o no. En el futuro las acciones se registrarán de otra manera para que puedan ser lanzadas desde directamente sin la necesidad de que la acción venga por GET o POST.

Módulos

Los módulos son los encargados de añadir funcionalidades a webStore. Los módulos han de instalarse para que se configuren. Tendrán las siguientes características:

- Añaden campos y/o tablas a la base de datos
- Capturan acciones
- Capturan llamadas a funciones remotas
- Añaden nuevas clases y/o controladores
- Sobrescriben las clases y/o controladores actuales
- Tienen vistas que pueden ser personalizadas en el tema

Para que webStore pueda gestionar los módulos, han de ser instalados. Cada módulo tiene su propio script de instalación que realizará las tareas pertinentes.

webStore tendrá en su caché almacenada la información que necesita para mostrar el módulo. En caso de no existir en la caché, se regenerará de la base de datos.

****hasta que no esté todo esto programado, por ahora esto no funcionará así. Ahora lo único que vamos a hacer es que webStore haga un “include” de todos los ficheros “action” que tengamos dentro de los módulos y ya está ****

Caché

En webStore se utiliza la caché en diversas situaciones para agilizar la carga de datos. Normalmente la caché consiste en un fichero .php que se “incluye” en cualquier momento para tener disponibles sus datos. De esta manera evitamos que el motor tenga que procesar datos. Cuando se generó el fichero ya se procesaron esos datos. Por ejemplo tenemos un fichero en caché que almacena la ruta a los diferentes ficheros del tema que utilizamos en un array. Así webStore consulta el array para poder lanzar el fichero específico. Si fuese un fichero de texto (en formato json, XML, csv, etc.) habría que procesar el contenido con alguna propia o nativa. En casos como este hay un segundo fichero de control que es el que se asegura de que haya habido cambios en la plantilla para reindexar el fichero de manera automática. La caché

se almacena en la carpeta “dynamic”. Vamos a ver las carpetas que, por ahora, se almacenan aquí y para qué sirven.

cache

Aquí se guardan los ficheros de caché más genéricos.

- Árbol de escaparates
- Estructura de la base de datos

clases

Aquí almacenamos las clases que cargará webStore. Para crear estas clases lo que hace es tener en cuenta los overrides y la propia superclase. Así compone una nueva clase totalmente operativa.

controlFiles

Los ficheros de control contienen, de manera serializada, objetos del tipo ConfigFile (ver en controladores). Los utilizamos para llevar un control de los ficheros cacheados junto con la fecha de última modificación y así saber si es necesario reindexarlos o no.

controllers

Aquí almacenamos los controladores que cargará webStore. Para crear estas clases lo que hace es tener en cuenta los overrides y la propia superclase. Así compone una nueva clase totalmente operativa para cada controlador.

file_index

Almacena un fichero .php que genera un array con las rutas de los ficheros del tema. De esta manera, cuando haya que hacer un “Theme::include_file(‘xxx.xxx’);” webStore sabrá dónde se guarda el fichero sin tener que buscarlo. Este fichero se reindexa sólo cuando detecta que hay cambios en la plantilla. Esto se hace a través de un fichero de control (ver controlFiles).

media

Los recursos javascript y js de cada sección se generan aquí como ficheros únicos y minificados. Así para cada sección, webStore sólo cargará un fichero de hojas de estilo o javascript. Nuevamente se controla que estos ficheros se modifiquen utilizando un fichero de control (ver controlFiles).

sections

Por ahora sólo tiene un fichero del tipo “ConfigFile” serializado con las secciones de la web que se hayan configurado en el tema activo. *****En el futuro las secciones se deben gestionar desde un apartado en el admin y no desde un fichero .php.****