Plug-ins WordPress: primeros pasos

Autor: César Córcoles Briongos

El sistema de gestión de contenidos: WordPress

En los inicios de la web los sitios que se publicaban eran relativamente sencillos y se creaban ‘a mano’, usando editores de texto que se fueron adaptando poco a poco al recientemente creado lenguaje de marcado HTML. Poco a poco fue aumentando la complejidad de muchos de esos sitios web, hasta que se hizo imprescindible algún proceso de gestión automática del contenido. Así nació una nueva categoría de software: los **sistemas de gestión de contenidos** (o CMS por sus siglas en inglés, de *Content Management System*), que permiten que un colectivo administre los contenidos y la presentación de un sitio web con una arquitectura de información sofisticada.

Por otro lado, y también desde los inicios de la web, algunos usuarios comenzaron a editar ‘diarios web’, que fueron evolucionando y dieron lugar a los *weblogs* (posteriormente blogs), un término acuñado por Jorn Barger a finales de 1997. No existe una definición estricta y consensuada de lo que es un blog, pero el término se usa para definir infinidad de sitios web que presentan una colección de ‘entradas’ que se agrupan en orden cronológico inverso en una o más páginas web.

Con la popularización del *blogging* fueron apareciendo, en los últimos pocos años del siglo XX y primeros del XXI, sistemas de gestión de contenidos muy ligeros dedicados exclusivamente a la creación y mantenimiento de blogs. En 2003 uno de esos sistemas era *b2/cafelog*, creado por Michel Valdrighi. Matt Mullenweg y Mike Little tomaron el código de *b2* (de código libre) y crearon una variante bajo el nombre de *WordPress*.

Si bien la primera versión de WordPress apareció en mayo de 2003, con número de versión 0.7, es a partir de 2004, con la aparición de la versión 1.2, que el programa comienza a adquirir popularidad. La evolución de WordPress, junto con un conjunto de hechos ajenos, ha conllevado un aumento continuo de esa popularidad, que ha venido ligada a la aparición de una fuerte comunidad de desarrollo de código abierto alrededor de la plataforma.

Este proceso de crecimiento ha llevado a la aparición paulatina de nuevas funcionalidades en el software, que hacen que hoy en día WordPress merezca el nombre de *sistema de gestión de contenidos*, con mejoras en las características para administrar plantillas para la presentación de contenidos, la interfaz de administración, el editor de contenidos, el rendimiento del sistema, o la definición de flujos de trabajo (que permiten, por un ejemplo, que un editor pueda introducir contenidos pero que sólo un administrador pueda dar el visto bueno para su publicación definitiva).

Hoy en día existe un buen número de sistemas de gestión de contenidos de propósito general que, como WordPress, son de código libre y funcionan sobre ‘plataformas AMP’: servidor web Apache, base de datos MySQL y lenguaje de programación ‘de servidor’ PHP. Podemos destacar Drupal, Joomla!, Mambo o TYPO3. Todos ellos (y muchos otros) son aptos para su personalización para la creación del sitio web de una revista como Mosaic.

Entre los factores que influyeron en la elección de WordPress como CMS para Mosaic se cuentan la familiaridad del equipo con dicha herramienta, su popularidad (tanto entre desarrolladores como entre usuarios), lo bien documentado que está o la facilidad de uso de las herramientas de administración.

Un pequeño recorrido por la interfaz de administración de WordPress

La pantalla principal de administración de WordPress (en su versión 3.1.2) es la siguiente:

En el panel 'Right Now' podemos apreciar la estructura de WordPress (o, más exactamente, la estructura de WordPress que se ha usado en esta instalación particular, que corresponde a Mosaic):

WordPress 3.2.1 is available! [Please update now.](#) Screen Options Help

Posts

[Add New](#)

All (494) | Published (488) | Sticky (1) | Drafts (6)

Bulk Actions [Apply](#) Show all dates View all categories Filter

494 items 1 of 25

☐	Title	Author	Categories	Tags	Comments	Date	Views
☐	Cargador universal ecológico de Idapt, a prueba	visda1	Noticias	No Tags	0	2011/09/29 Published	20 visitas
☐	Cargador universal - Draft	visda1	Sin categoría	No Tags	0	2011/09/29 Last Modified	0 visitas
☐	España vuelve a flaquear en nuevas	visda1	Noticias	No Tags	0	2011/09/28 Published	30 visitas

Podemos apreciar que hay 488 'posts' (o entradas, que son la unidad de contenido que suele usarse en WordPress), que se agrupan en 207 categorías diferentes y a los que se han asignado 233 'tags' (o etiquetas). El lector acostumbrado a trabajar con sistemas de gestión de contenidos apreciará que el número de categorías es inusualmente alto. Si accedemos a la pantalla de administración de entradas veremos el motivo.

Categories

[Add New Category](#)

Name

The name is how it appears on your site.

Slug

The "slug" is the URL-friendly version of the name. It is usually all lowercase and contains only letters, numbers, and hyphens.

Parent

Categories, unlike tags, can have a

Bulk Actions [Apply](#)

207 items 1 of 11

☐	Name	Description	Slug	Posts
☐	Agenda		agenda	65
☐	Artículos		articulos	97
☐	Boletines		boletines	70
☐	— Número 001		numero-001	7
☐	— Número 002		numero-002	5

En la adaptación de WordPress a Mosaic se tomaron una serie de decisiones de diseño sobre la estructura de las entradas. Veamos cómo es una entrada de WordPress y qué particularidades se presentan en esta instalación en particular. Cada entrada se compone de:

- **Título** (en la captura de pantalla, en el primer ítem, 'Reflexiones sobre la narrativa digital'). Es el título del contenido correspondiente. No presenta particularidades.
- **Autor** ('visda1'). Corresponde al usuario de WordPress que ha creado la entrada. En la mayoría de blogs este usuario se corresponde con el autor del contenido. En muchos sitios más complejos (como es el caso), el usuario que crea la entrada.
- **Categorías** ('Antoni Marín', 'Artículos', 'Número 083'). Habitualmente, las categorías son una taxonomía de tamaño pequeño o mediano que agrupan las entradas del sitio web en temáticas. Sin embargo, en el caso de Mosaic se utilizan para una serie de objetivos adicionales. Como puede intuirse, el autor del contenido es una categoría, cada tipo de contenido también lo es (los tipos de contenidos disponibles son Artículos, Entrevistas, Experiencias, Recursos, Noticias y Agenda) y, finalmente, también se usan las categorías para diferenciar cada uno de los números de la revista Mosaic. Actualmente existen mejores maneras de realizar estas categorizaciones (a través de la funcionalidad de taxonomías a medida de WordPress), pero en el momento de creación del sitio WordPress dicha funcionalidad estaba todavía en fase experimental y se decidió no usarla.
- **Etiquetas** ('Gestión_Contenidos'). Las etiquetas o tags se usan para anotar el contenido de cada entrada, habitualmente intentando añadir información semántica. Así, etiquetaremos con 'diseño web' todas las entradas que hagan referencia a esa temática, independientemente de la categoría a la que pertenezca la entrada. Es frecuente que a medida que se vayan añadiendo contenidos al sitio vayan apareciendo etiquetas nuevas, mientras que el número de categorías suele mantenerse estable.
- **Comentarios**. Indica la cantidad de comentarios que han hecho los lectores a esta entrada. Se puede activar o desactivar individualmente para cada entrada la capacidad de los usuarios de escribir comentarios. Como puede apreciarse, en Mosaic actualmente muchas entradas no permiten la realización de comentarios.
- **Fecha**. Indica la fecha y hora de la última edición realizada sobre la entrada, así como el estado de la entrada (que suele ser 'borrador' o 'publicada').
- **Vistas**. Indica el número de vistas que ha tenido el contenido.

La estructura de categorías de Mosaic

Hemos visto que la estructura de categorías de Mosaic es bastante particular y que agrupa diferentes tipos de información haciendo uso de categorías y subcategorías. Estas son:

- Agenda
 - Artículos
 - Boletines (indica el número al que pertenece el contenido específico)
 - Número 001
 - Número 002
 - ...
 - Entrevistas
 - Experiencias
 - Noticias
 - Recursos
 - www-Autores (indica el nombre del autor del contenido, o el nombre de la persona entrevistada en el caso de las entrevistas)
 - Adrià Ferrer Castro
 - Albert Alfonso Moreno
 - ...
-

Veamos, finalmente, el contenido de una entrada. Si accedemos al editor de contenido para la primera entrada de la captura anterior veremos algo como lo siguiente:

Edit Post

El marketing online: la importancia de analizar los resultados

Permalink: <http://multimedia2.uoc.edu/visda1/2009/07/10/el-marketing-o...los-resultados/> Edit View Post Get Shortlink

Upload/Insert Visual HTML

B *I* ABC [List Icons] [Quote Icon] [Table Icon] [Link Icon] [Image Icon] [Media Icon] [More Icon]

La evolución de Internet en estos últimos años ha abierto la puerta a nuevas técnicas de marketing que permiten alcanzar o incrementar los objetivos marcados por las empresas, con el fin de aumentar el retorno de la inversión (ROI) marcado. Al conjunto de estas nuevas técnicas y su utilización es lo que denominamos **"Marketing Online"**.

La evolución del Marketing Online ha ido en paralelo a la evolución de la audiencia de Internet y es por ello que es ilustrativo hacer un breve repaso de la evolución de Internet en estos últimos años. La evolución de usuarios de Internet en España sigue creciendo año tras año, superando según el [EGM \(Estudio General de Medios\)](#) ya los 17,9 millones de usuarios en el mes de Noviembre 2008 y con una penetración en la población española del 46,80%.

Este crecimiento es debido principalmente al desarrollo de las telecomunicaciones que ha permitido que el nivel de interacción de las personas en Internet sea proporcional a la capacidad de transmisión de datos, facilitando el desarrollo de aplicaciones soportadas por ellas.

Path: p

Word count: 2669 Draft saved at 5:04:09 pm. Last edited by on 20/05/10 at 12:37 pm

Tenemos diferentes ítems:

- El *título de la entrada*, que ya hemos visto anteriormente.
- El '*permalink*' (o enlace permanente) bajo el que se publicará el contenido. Se compone de dos partes:
 - Una estructura (en este caso <http://multimedia2.uoc.edu/visda1/2011/01/18/>)
 - Un 'slug' (en este caso reflexiones-sobre-la-narrativa-transmedia). El 'slug' se suele generar de manera automática a partir del título de la entrada (eliminando o modificando los caracteres que podrían resultar problemáticos). Podemos cambiar el 'slug', normalmente para facilitar su memorización o para introducir términos importantes para que los tengan en cuenta los buscadores.
- El *contenido de la entrada*. Disponemos de un editor visual y de uno en el que podemos trabajar directamente en HTML. Sobre la caja del editor disponemos de una botonera desde la que podemos subir todo tipo de archivos (principalmente, imágenes) a asociar a la entrada.
- El *extracto*. Si en algún lugar (la portada, por ejemplo) queremos publicar un texto breve y no todo el contenido de la entrada, podemos optar por dejar que WordPress seleccione automáticamente las primeras palabras de la entrada (podemos configurar cuántas palabras seleccionará) o bien redactar un texto alternativo, como es el caso.

- Los *campos personalizados* de la entrada. Estos suelen utilizarse para añadir información a la entrada que usaremos posteriormente. En breve veremos cuáles son los campos personalizados que se usan en Mosaic.

Junto con la entrada se pueden utilizar imágenes. Veamos cómo gestionar las entradas de la imagen (accedemos a esta funcionalidad desde el correspondiente enlace *Upload/Insert*, sobre la caja de edición del contenido de la entrada).

A medida que vamos subiendo imágenes se crea, para cada entrada, una galería. Tal y como se ha definido, en Mosaic la primera imagen se usa para los destacados en portada, mientras que la segunda se utiliza para el currículum del autor. Naturalmente, es posible cambiar el orden de las imágenes subidas en cualquier momento, si es necesario.

Otros ítems que corresponden a cada entrada son su estado de publicación (principalmente 'publicada' o 'borrador', aunque pueden definirse otros estados si son necesarios para el flujo de trabajo), su visibilidad (pública, privada o protegida por contraseña) y la fecha de publicación (puede disponerse una fecha en el futuro, de forma que el contenido se publique en un momento determinado de manera automática), así como los ya vistos correspondientes a las categorías y etiquetas asignadas a la entrada.

Los campos personalizados de Mosaic

Excerpt

Internet se ha convertido en un medio más en el que invertir en publicidad para muchas empresas. Maribel Morales, experta en marketing online y analítica web, nos ofrece un repaso a las estrategias de marketing más...

Excerpts are optional hand-crafted summaries of your content that can be used in your theme. [Learn more about manual excerpts.](#)

Custom Fields

Name	Value
Autor	<u>Maribel</u> Morales
<input type="button" value="Delete"/> <input type="button" value="Update"/>	
Curriculum	<u>Maribel</u> Morales es directora técnica de <u>OJD</u> Interactiva. Ingeniera Técnica en Informática de Gestión por la <u>UPC</u> y licenciada en Ingeniería Informática por la UOC. donde se ha especializado en
<input type="button" value="Delete"/> <input type="button" value="Update"/>	
lightboxoff	false
<input type="button" value="Delete"/> <input type="button" value="Update"/>	
views	2506
<input type="button" value="Delete"/> <input type="button" value="Update"/>	

En la instalación de WordPress para Mosaic se usan por defecto cinco campos personalizados:

- **Autor.** Contiene el nombre del autor del contenido. Duplica la información ya contenida en una subcategoría. Esto es así porque en función de dónde queremos usar dicha información, es más fácil recuperarla de la subcategoría o del campo personalizado.
- **Curriculum.** Contiene, como puede verse, un breve texto sobre el autor.
- **lightboxoff.** Es un campo generado y usado por uno de los *plug-ins* de WordPress usados en Mosaic, Lightbox 2, que permite la creación de galerías de imágenes.
- **miniintro.** Contiene un breve resumen del contenido. Suele ser ligeramente diferente del extracto visto anteriormente.
- **Views.** Es un campo generado y usado por el plug-in WP-PostViews para contar el número de visualizaciones que ha tenido cada contenido del sitio.

Podemos ver el uso de toda la información que hemos visto en la siguiente captura de pantalla de la entrada tal y como aparece publicada:

Plug-ins WordPress

Hemos visto que una de las maneras de extender la funcionalidad de WordPress es a través del uso y la creación de plug-ins. Los plug-ins permiten la extensión o modificación del código de WordPress sin tocar el código fuente de la aplicación, lo que permite, si se siguen unas buenas prácticas de programación, que una actualización del software de WordPress no afecte a las funcionalidades añadidas.

Un plug-in WordPress es un programa, o un conjunto de una o más funciones, escritas en el lenguaje de *scripting* PHP, que añade un conjunto específico de características o servicios al weblog WordPress, que pueden integrarse transparentemente con el Weblog usando puntos de acceso y métodos ofrecidos por la API de WordPress.¹

En nuestro caso particular, en el que queremos ofrecer una visualización de WordPress (y, en particular, del WordPress sobre el que funciona Mosaic), lo que queremos hacer serán tres cosas:

1. En primer lugar, queremos una manera de acceder con facilidad y de manera eficiente a la información contenida en WordPress y ponerla a disposición de la herramienta de visualización que crearemos.
2. En segundo lugar, pretendemos ofrecer una herramienta para administrar las diferentes opciones de personalización del plug-in (para adaptarlo a una instalación cualquiera de WordPress en función de su estructura, pero también para modificar su aspecto gráfico, por ejemplo).

¹ https://codex.wordpress.org/Writing_a_Plugin

3. Finalmente, necesitamos una forma simple de insertar la visualización allí donde sea conveniente.

El esqueleto de un plug-in

¿Qué elementos componen un plug-in?

- Un nombre único que identifique al plug-in (hay que tener en cuenta que existen, literalmente, miles de plug-ins para WordPress², por lo que es importante asegurarnos de que nuestro plug-in no coincide en el nombre con ningún otro).
- Un archivo, también con nombre único, más una probable carpeta que contenga más archivos, si es necesario. Normalmente ese archivo o grupo de archivos reside en una carpeta estándar de WordPress, `wp-content/plugins`. De todas formas, no podemos asegurar que esa sea la carpeta en que se almacenan los plug-ins en *todas* las instalaciones de WordPress, por lo que no es conveniente asumir que es esa la carpeta, y es mejor utilizar la función `plugins_url()` para obtener la ruta, si es necesario.
- En caso que deseemos publicar el plug-in en el repositorio de wordpress.org, también deberemos incluir un archivo `readme.txt` describiendo el plug-in³.
- No es un requisito, pero si muy conveniente, disponer de una página web para el plug-in en la que se informe sobre sus funcionalidades y desde la que se pueda dar soporte a lo usuarios y recoger sus comentarios y sugerencias.

El archivo.php principal del plug-in debe comenzar de la siguiente forma:

```
<?php
/*
Plugin Name:
Plugin URI: http://
Description:
Version:
Author:
Author URI: http://
License: por ejemplo, GPL2
*/
?>
```

Y debe seguirle la información de la licencia elegida para el plug-in.

² <https://wordpress.org/extend/plugins/>

³ El formato del archivo `readme.txt` se detalla en <https://wordpress.org/extend/plugins/about/readme.txt>

Una consideración sobre la licencia

WordPress se distribuye bajo la licencia de código libre *GNU General Public License* (en su versión 2 o posterior⁴). Según WordPress.org⁵ las obras derivadas de WordPress incluyen tanto los temas como los plug-ins que se desarrollen para WordPress heredan las condiciones de la GPL. Por tanto, deben licenciarse, como mínimo, con una licencia compatible con GPL v2.

Un recurso interesante para generar automáticamente el esqueleto de un plug-in puede encontrarse en <http://themergency.com/generators/wordpress-plugin-boilerplate/>.

Cómo se activa un plug-in

Una vez que hayamos escrito el código para el plug-in, necesitaremos lanzarlo en los momentos convenientes. Disponemos de varias maneras para hacerlo.

- Si queremos que el plug-in se active siempre en determinados puntos de la ejecución de WordPress (por ejemplo, cada vez que se dibuje la cabecera, o cada vez que editemos una entrada), haremos uso de los 'hooks' (ganchos) de WordPress.
- Si deseamos insertarlo 'a mano' dentro de una plantilla, podemos crear una 'template tag' nuevo. Si, además, deseamos que un editor de contenido pueda hacer uso de esa template tag dentro de una entrada, podemos, posteriormente, crear un [comando] que active la etiqueta.

La estructura básica de archivos de WordPress

WordPress.org recomienda que cualquier tema para WordPress respete unos estándares mínimos en su estructura⁶ que facilitan, posteriormente, el funcionamiento universal de los plug-ins que hacen uso de 'hooks'.

Dado que es muy común que los sitios web tengan al menos una cabecera y un pie en cada página, se asume que todas las plantillas contienen, respectivamente, un `header.php` y un `footer.php` que se encargan de dibujarlos. Incluso cuando los sitios web no tienen una cabecera o un pie, se suelen incluir estos archivos, de forma que podemos contar con su existencia.

Las barras laterales no son universales, pero también son muy frecuentes. En caso de existir, su código residirá en un archivo `sidebar.php`. Otros archivos frecuentes son `single.php` (para el contenido principal de páginas que contengan una única entrada), `category.php` (para las páginas con archivos de categorías) o

⁴ Podéis encontrar las condiciones de la licencia en <https://wordpress.org/about/gpl/>

⁵ <https://wordpress.org/about/license/>

⁶ https://codex.wordpress.org/Stepping_Into_Templates

archive.php (para los archivos temporales). No es la intención de este texto entrar en profundidad en esta estructura de archivos y su funcionamiento.

Template tags

Un 'template tag' o etiqueta de plantilla es una etiqueta que le dice a WordPress que haga algo. Existen etiquetas para hacer prácticamente cualquier cosa que se nos pase por la cabeza, desde escribir el título del blog (`<?php bloginfo('name'); ?>`) a listar las categorías que hemos creado (`<?php wp_list_cats(); ?>`)⁷. Si necesitamos una etiqueta inexistente, podemos definirla creando un plug-in.

Tipos de 'hooks'

Disponemos de dos tipos de 'hooks' en WordPress:

- Las *acciones* se lanzan en puntos específicos o como respuesta a determinados eventos.
- Los *filtros* se interponen entre las consultas o actualizaciones a la base de datos, modificando valores.

Un ejemplo de acción sería `get_header`, que se llama justo antes de acceder al archivo `header.php` y que nos podría servir para insertar todo aquello que queremos que pase antes de que el tema escriba la primera línea de HTML.⁸

Por su parte, un ejemplo de filtro sería `the_content`, que se aplica cada vez que recuperamos el contenido de una entrada. Podríamos usar este filtro para buscar apariciones de determinadas palabras y sustituirlas por otras, o para sustituir las imágenes por sus textos alternativos.⁹

Cómo añadir una acción

El código que debemos usar para añadir una acción es

```
add_action ( 'hook_utilizado', 'funcion_a_llamar', [prioridad],  
[argumentos_aceptados] );
```

- `hook_utilizado` y `funcion_a_llamar` deberían ser autoexplicativos.

⁷ Encontraréis el catálogo completo de etiquetas de plantilla de WordPress, con su documentación correspondiente, en https://codex.wordpress.org/Template_Tags.

⁸ Encontraréis la lista de acciones disponibles en https://codex.wordpress.org/Plugin_API/Action_Reference.

⁹ Encontraréis la lista de filtros disponibles en https://codex.wordpress.org/Plugin_API/Filter_Reference.

- `prioridad` es un parámetro optativo. Por defecto su valor es 10, y valores más bajos indican prioridades más altas. Hay que pensar que es probable que haya otros plug-ins en funcionamiento que utilicen el mismo hook que nosotros y que puede ser importante (o no) que nuestro plug-in se ejecute cuanto antes. Es una mala práctica solicitar una prioridad alta si no es imprescindible.
- `argumentos_aceptados`, también opcional, indica el número de valores que acepta `funcion_a_llamar`.

Creación, mantenimiento y uso de opciones

Es muy probable que en un plug-in existan determinados parámetros que queramos que el usuario pueda configurar: el color de fondo del elemento interactivo que queremos crear, su tamaño, el número de categorías que queremos usar (o cuales), si los elementos deben ordenarse de mayor a menor, de menor a mayor o aleatoriamente...

Para hacerlo WordPress dispone de un sofisticado sistema de gestión de opciones.

El primer paso que debemos hacer es **crear la opción**, con

```
add_option($nombre, $valor, $deprecated, $autoload)
```

El único parámetro obligatorio es el nombre de la opción. Podemos especificar, si lo deseamos, un valor por defecto con `$valor`. `$autoload`, que por defecto vale `yes`, indica si la opción se recuperará automáticamente cuando WordPress ejecute `wp_load_alloptions`. `$deprecated` es un parámetro sin valor semántico, que se conserva (sólo si deseamos especificar `$autoload`) por motivos de compatibilidad con versiones antiguas de WordPress.

Para **recuperar el valor de la opción** en cualquier momento debemos usar

```
get_option($opcion);
```

y para **actualizar su valor**

```
update_option($nombre_opcion, $valor_nuevo);
```

Así, por ejemplo, en el caso concreto de la creación de un plug-in para Mosaic, podríamos tener:

```
/* Creación y destrucción de las opciones al activar y desactivar
el plugin */
/* Al activar el plugin se crean las opciones */
function activacion_VisDa() {
    add_option('VisDaWidth', '344','','yes');
    add_option('VisDaHeight', '392','','yes');
}

/* Y al desactivarlo se borran */
function desactivacion_VisDa() {
```

```
delete_option('VisDaWidth');  
delete_option('VisDaHeight');
```

Para hacer que estas funciones se lancen en el momento de instalación y desinstalación del plug-in, usaremos un par de *hooks*:

```
register_activation_hook(__FILE__, "activacion_VisDa");  
register_deactivation_hook(__FILE__, "desactivacion_VisDa");
```

Una vez hemos creado las opciones que necesitaremos debemos crear un mecanismo para poder gestionarlas desde el panel de control de WordPress. Para ello disponemos de tres alternativas: crear una página nueva dentro del menú de administración, crear una subpágina dentro de una ya existente o bien añadir nuestras opciones a una página ya existente.

Para **crear una página nueva** haremos:

```
add_action('admin_menu', 'menu_del_plugin');
```



```
function menu_del_plugin() {  
    add_options_page($titulo_pagina, $titulo_menu,  
        'manage_options', $slug_menu, $funcion);  
}
```

- `titulo_pagina` es el texto que se mostrará como título de página en el navegador al cargarla.
- `titulo_menu` es el texto que se usará en el texto del menú.
- `manage_options` indica a WordPress que esta página sólo debe mostrarse a los usuarios que tengan permisos concedidos para administrar las opciones del sistema¹⁰.
- `slug_menu` es el 'slug' único con que se identificará posteriormente el menú.
- `funcion`, finalmente, es la función que se llamará para generar el contenido de la página.

Si queremos **crear una subpágina** (será lo más conveniente, en general) lo que haremos será:

¹⁰ WordPress dispone de un sistema sofisticado que nos permite tener, por ejemplo, usuarios que puedan administrar el sitio completo, mientras que otros sólo puedan escribir nuevos contenidos que deban ser aprobados por una tercera persona antes de publicarse.

```
add_submenu_page( $slug_padre, $titulo_pagina, $titulo_menu,
'manage_options', $slug_menu, $funcion);
```

La única opción nueva es `slug_padre` que, como podemos imaginar, identifica la página del menú de la que deberá colgar la nuestra y que puede ser, por ejemplo, `plugins.php`, `tools.php` o `options_general.php` (para ubicar la subpágina en el menú 'Settings').

Finalmente, en muchos casos será conveniente **crear una sección dentro de una página o subpágina existente**. Para ello haremos uso de:

```
add_settings_section($id, $titulo, $funcion, $pagina);
```

- `id` es un identificador único para la sección.
- `titulo` es el título de la sección.
- `funcion` es la función encargada de escribir los contenidos de la sección.
- `pagina` es la página a la que debe añadirse la sección. Puede ser `general`, `reading`, `writing`, `media`, o el 'slug' de una página o subpágina que nosotros hayamos creado previamente.

La función para generar el contenido será del tipo:

```
function opciones_plugin() {
?>
<div>
<h2>El título</h2>
Texto descriptivo.

<form action="options.php" method="post">
<?php settings_fields('plugin_options'); ?>
<?php do_settings_sections('plugin'); ?>

<input name="Submit" type="submit" value="<?php
esc_attr_e('Guardar cambios'); ?>" />

</form></div>

<?php
}??>
```

Observad cómo podemos salir y entrar del código en PHP a voluntad (con `?>` y `<?php`) para poder escribir el contenido HTML con facilidad.

`settings_fields` toma como argumento, en el ejemplo, `'plugin_options'`, un nombre arbitrario de un grupo de opciones.

Por su parte, `do_settings_sections` se encarga de escribir todas las secciones que hemos añadido previamente a una página, que en el ejemplo es `'plugin'`.

También tendremos que añadir una acción para inicializar la administración de nuestro plug-in:

```
<?php
add_action('admin_init', 'plugin_admin_init');
```

Dicha acción llama a una función como la siguiente:

```
function plugin_admin_init() {
    register_setting(      'plugin_options',      'plugin_option_name',
    'plugin_option_name_validate' );
}
```

donde `plugin_options` viene del fragmento de código anterior, `'plugin_option_name'` es el nombre de cada una de las opciones a registrar y el tercer parámetro (opcional) es el nombre de la función que emplearemos para limpiar y validar el valor de la opción.

A continuación deberemos añadir cada uno de los campos que necesitemos, con

```
add_settings_field('plugin_text_string', 'Plugin Text Input',
'plugin_setting_string', 'plugin', 'plugin_main');

}??>
```

Los parámetros para `add_settings_field` son, en orden, los siguientes:

- Un identificador para cada campo (en nuestro caso, `'plugin_text_string'`);
- el título del campo (`'Plugin Text Input'`);
- una función que muestra el campo (`'plugin_setting_string'`);
- la página en que debe aparecer el campo (`'plugin'`) y
- la sección en que debe aparecer (es un parámetro opcional, y en el ejemplo le hemos dado el valor `'plugin_main'`).
- A continuación añadiremos, si son necesarios, los argumentos que deban pasarse a la función que cumplimenta el campo. Actualmente el único argumento que puede pasarse es un `label_for`, para asignar al campo una etiqueta diferente al título.

Las funciones que crean cada campo son del tipo:

```
<?php function plugin_setting_string() {
    $options = get_option('plugin_options');

    echo
        "<input                                id='plugin_text_string'
        name='plugin_options[text_string]'      size='40'          type='text'
        value='{ $options['text_string']}' />";
}
```

```
} ?>
```

Finalmente, las opciones de validación y/o 'sanitización' son del tipo:

```
function          plugin_options_validate($input)          {
$newinput['text_string']      =      trim($input['text_string']);

if(!preg_match('/^[a-z0-9]{32}$/i', $newinput['text_string'])) {
    $newinput['text_string']      =      '';
}

return                $newinput;
}
```

Siguiendo con el ejemplo de un plug-in para Mosaic, el código correspondiente a la página de opciones podría ser el siguiente:

```
/* Definición de la página de opciones */

function VisDa_opciones() {

/* Comprueba si hemos entrado en la página y cambiado opciones */
    if( isset($_POST[ancho]) ) {
        $opt_ancho = $_POST['ancho'];
        $opt_alto = $_POST['alto'];
        update_option('VisDaWidth', $opt_ancho );
        update_option('VisDaHeight', $opt_alto );
    }

    <div class="updated">
    <p><strong>Opciones guardadas.</strong></p></div>
    <?php
    }
/* Escribimos el formulario de las opciones */
    print("<div class='wrapper'>");
    screen_icon();
    print("<h2>Opciones del plugin VisDa</h2>");
    ?>
    <form id="opciones_VisDa" method="post" actions="options.php">
        <p>
            <label for="ancho">Ancho:</label>
            <input
                name="ancho"
                type="text"
                id="ancho"
                value="<?php echo get_option('VisDaWidth','false'); ?>"
                size="10" />
        </p>
        <p>
            <label for="alto">Alto:</label>
            <input
                name="alto"
                type="text"
                id="alto"
                value="<?php echo get_option('VisDaHeight','false'); ?>"
                size="10" />
        </p>
        <p>
            <input
                type="submit"
                name="enviar"
                id="enviar"
                value="Guardar cambios" class="button-primary" />
        </p>
    </form>
    <?php
}

/* Crea la página de opciones */
function VisDa_opciones_menu() {
    add_options_page('Opciones', 'Opciones', 'manage_options', 'plugin_VisDa', 'VisDa_opciones');
}
}
```

Algunas recomendaciones finales

Es muy útil y recomendable consultar los estándares de programación de WordPress, en https://codex.wordpress.org/WordPress_Coding_Standards.

Es muy conveniente prefijar todos nuestros nombres de función siguiendo alguna convención relacionada con el nombre del plug-in (i.e. `visda_nombrefuncion()`) para evitar coincidencias con funciones de otros plug-ins.

En cuanto a la base de datos:

- No usemos `wp_` como prefijo para los nombres de tablas, sino `$wpdb->prefix` (esta segunda opción funcionará siempre, incluyendo los casos en que se haya cambiado el prefijo habitual).
- Escribir a la base de datos es una operación muy cara y lenta, por lo que debemos minimizar las escrituras y, cuando lo hagamos, es mucho mejor hacer uso de los métodos nativos de WordPress que acceder a la base de datos directamente, siempre que esto sea posible.
- Leer de la base de datos es bastante más eficiente que escribir, pero continúa siendo una operación costosa: cuantos menos `SELECTs`, mejor.

Otro cuello de botella potencial es la carga de estilos CSS y archivos JavaScript. Para evitarlos es muy conveniente el uso de `wp_enqueue_style()` y `wp_enqueue_script()`.

Finalmente, mientras estamos probando, puede ser útil `define('WP_DEBUG', true);` en el `wp-config.php`. Esto hará, por ejemplo, que aparezcan en la página web cualesquiera mensajes de error que pudieran generarse, incluyendo errores de MySQL y PHP.