Apéndice:
Adaptando software libre

"Un enano subido a los hombros de un gigante, verá más lejos que el mismo gigante".

Reflexionemos un momento sobre el modelo de desarrollo del software libre, para que podamos evaluar las ventajas y desventajas de utilizar sus proyectos como punto de partida de los nuestros.

Con lo que ya sabemos programar en PHP, nada nos impide programar un sistema completo desde cero, identificando las tareas principales que deberán realizar los usuarios de ese sistema (casos de uso), y creando páginas, funciones y/o clases para lograr todas las interacciones necesarias con una base de datos. Pero, con el tiempo, luego de haber pasado repetidamente por la experiencia de crear sistemas desde cero, notaremos que buena parte de los sistemas se suelen parecer bastante entre sí. Llegará un punto en el que pensaremos que no inventaremos nada nuevo, puesto que casi todo está hecho.

Iremos comprendiendo que las necesidades de nuestros clientes no son infinitas, ni tan "especiales" como ellos creen. Todos los negocios de un mismo rubro suelen necesitar aproximadamente las mismas soluciones a los mismos problemas. Las diferencias son escasas. Con pequeños cambios se puede personalizar un sistema completo. Todos los sistemas tienen una base común: requerirán un *back-end* y un *front-end*, un módulo para registro y acceso mediante la identificación de los usuarios, distintos tipos de usuarios con distintos permisos de acceso a los contenidos, gestión de esos contenidos (productos, noticias, mensajes) para luego mostrarlos y navegarlos de distintas maneras, etc.

Por lo tanto, en lugar de programar todo desde cero en cada nuevo proyecto, será más eficiente disponer de un conjunto de **soluciones ya probadas** y perfeccionadas, y solamente "adaptarlas" a cada uno de nuestros clientes. Usaremos como base para nuestro trabajo lo que muchos otros programadores experimentados han creado, y si le encontramos alguna falla, compartiremos la solución (o, al menos, avisaremos cuál fue la falla, para que otro le encuentre solución). De esa manera, el sistema se irá perfeccionando con el tiempo, y cuando volvamos a implementarlo para otro cliente, disfrutaremos de las soluciones que hemos ido aportando todos los usuarios en nuestro testeo cotidiano.

Al usar un sistema libre, al igual que hicimos con las clases en el capítulo anterior, nuestra principal tarea será **seleccionar** el sistema que más se asemeje a lo que necesitamos. Una vez elegido, **analizaremos** sus funciones, sus métodos, sus páginas, y lo **adaptaremos** a las necesidades de nuestro cliente particular realizando pequeños retoques y agregados.

El comenzar nuestro trabajo a partir de una base que ya funciona, posee una serie innegable de ventajas competitivas frente al desarrollo de sistemas a medida desde cero:

- Muchísima rapidez de desarrollo, plazos de entrega cortos.
- Presupuestos mucho más bajos: competitividad.
- Facilidad de adaptación.
- El conocimiento sobre el sistema está distribuido en una comunidad, por lo que no se depende de un solo proveedor para que nos dé soporte y nos solucione cualquier problema.

Como hemos visto en el primer capítulo del libro, nuestro mercado ideal es éste:

LENGUAJES	HTML/CSS	PHP/MySQL	Otros (Java, .Net)
COMPETIDORES	Diseñadores gráficos	Programadores Web	Empresas de sistemas
TAREA PRINCIPAL	Decorar páginas (no saben programar)	Adaptar sistemas prearmados rápida y económicamente	Hacer sistemas a medida desde cero (caros y de largo plazo)
NIVEL DE PRESUPUESTOS	Cientos	Miles/Decenas de miles	Decenas a centenas de miles

Tabla 1. Mercado de los sistemas prearmados.

Desde ya que no competiremos con empresas de sistemas a medida, que manejan presupuestos y plazos enormes, sino que apuntaremos a satisfacer las necesidades de **pequeñas y medianas empresas**, que con relativamente poca inversión y en muy poco tiempo, podrán contar con un sistema Web funcionando eficientemente.

De nuestras capacidades (nuestras, o de las personas con las que nos asociemos para integrar un equipo) dependerá **agregarle valor**, mediante la integración de diseño, multimedia y programación PHP, a estas soluciones prearmadas.

Licencias para no reinventar la rueda

Técnicamente, PHP como lenguaje no tiene demasiadas diferencias con otros lenguajes de programación para la Web. Su **licencia libre** es la clave de su éxito. Si PHP no hubiese tenido una licencia libre (y gratuita), no se habría difundido en los *hostings,* y hubiera pasado a la historia jugando un papel muy secundario.

Pensemos que, cuando a fines de los 90, se popularizó Internet, varios lenguajes de programación del lado del servidor compitieron por llegar a ser populares en los *hostings* que comenzaban a ofrecer programación y bases de datos: en ese momento, los candidatos eran ASP (de Microsoft, solo para servidores que usaran Windows como sistema operativo), ColdFusion (un producto de Allaire, luego de Macromedia, y ahora de Adobe, cuya licencia actualmente cuesta 7500 dólares por servidor) y JSP (de Sun, gratuito, pero increíblemente complejo para programadores Web principiantes).

Estos candidatos, en un principio, parecían tener las mismas probabilidades de ser adoptados masivamente por los programadores Web, pero PHP los superó enormemente por contar con todas las ventajas (y ninguna de las desventajas) de cada uno de ellos:

- Licencia **libre** y gratuita.
- **Multiplataforma** (puede instalarse en Linux, Windows, Mac, Solaris, etc.).
- **Facilidad** de aprendizaje.
- Integración con cualquier gestor de **base de datos** (entre ellos, MySQL, otra clave del éxito de PHP).
- Y, finalmente, la enorme ventaja de poder participar de comunidades de programadores que surgieron en torno a los

principales **proyectos libres** programados en PHP, brindando soporte y mejoras continúas a esas aplicaciones de libre uso.

Todos los sistemas prearmados que utilizaremos poseen una licencia libre. Para simplificar, denominaremos **software libre** a aquel software que cuenta con cualquier **licencia** que permita su libre uso, modificación y distribución e, incluso, la posibilidad de realizarle mejoras y compartirlas nuevamente. Esto incluye numerosas licencias, como la GNU GPL (GNU *General Public License* o Licencia Pública General GNU), LGPL, licencia X11, licencia MIT, Expat, Cryptix, licencia BSD modificada, ZLib, licencia W3C, Berkeley, OpenLDAP, licencia de código abierto de Intel, licencia de Software Abierto (OSL), y también aquellos sistemas que son de dominio público.

No vamos a entrar en cuestiones filosóficas de si debería llamarse software libre u open source: simplemente, creemos que es un modelo de desarrollo sumamente conveniente para programadores independientes (*freelance*), para diseñadores Web que están aprendiendo programación, y para empresas pequeñas a medianas de desarrollo Web.

¿Por qué PHP tiene tantos proyectos listos para usar?

Lo que más sorprende de PHP en cuanto comenzamos a investigar es la cantidad de **proyectos libres** que existen, a diferencia de los escasos que hay en otros lenguajes de servidor, donde casi siempre estamos obligados a programar todo desde cero. En PHP, contamos con (literalmente) miles de proyectos libres, sistemas completos, listos para que los descarguemos gratuitamente, los adaptemos y los vendamos a nuestros clientes.

La clave está en el estilo de desarrollo colaborativo: esa modalidad crea **grandes comunidades** de programadores en torno a cada proyecto, que se benefician de aportar apenas un granito de arena cada uno (reportar algún fallo, sugerir alguna mejora) y, a cambio, utilizan el sistema completo cada vez más perfeccionado (tarea muy difícil para un equipo de pocas personas trabajando aisladas dentro de una empresa de desarrollo de código cerrado).

Sistemas para todos los rubros y clientes

Cuando visitemos un cliente para relevar las necesidades que busca satisfacer con una aplicación Web, es conveniente que conozcamos de antemano cuáles son los sistemas libres existentes en el área específica de trabajo de nuestro cliente, para orientar su demanda hacia alguno de esos software que conozcamos.

Vamos entonces a conocer los principales proyectos existentes en los rubros más comunes.

Principales CMS: portales, e-commerce, gestión de proyectos

En cada área del mercado de las aplicaciones Web que examinemos, siempre encontraremos entre los sistemas líderes alguna aplicación realizada con PHP y MySQL de código abierto, lista para utilizar y mejorar.

Para elegir cuál de todas ellas utilizaremos, primero debemos conocerlas y, para eso, podemos recurrir a servicios de comparación de características, como el de: http://www. cmsmatrix.org/ o a nuestra propia experiencia luego de probar decenas de estas aplicaciones (algo que no debería llevarnos mucho tiempo, en pocas horas, se pueden instalar y probar varias aplicaciones distintas).

Aquí ofrecemos un breve listado de las aplicaciones más populares (no necesariamente son por eso las "mejores", solo las más utilizadas), y las hemos agrupado en las categorías más comunes:

Categoría	Sistema	Sitio Web
Portales	Joomla!	http://www.joomla.org/
Weblogs	Drupal	http://drupal.org/
	WordPress	http://es.wordpress.org/
	b2evolution	http://b2evolution.net/
	Typo3	http://www.typo3.org
Comercio electrónico	osCommerce	http://www.oscommerce.com/
	Magento	http://www.magento.com
	PrestaShop	http://www.prestashop.com/

Campus virtuales	Moodle	http://www.moodle.org
	Atutor	http://atutor.ca/
	.LRN	http://www.dotlrn.org/
Intranet, gestión de proyectos	EGroupware	http://www.egroupware.org/
	phProjekt	http://www.phprojekt.com/
	dotProject	http://www.dotproject.net/
	todoyu	http://www.todoyu.com
Wikis	MediaWiki	http://www.mediawiki.org
Foros	phpBB	http://www.phpbb.com/
Listas de correo	phpList	http://www.phplist.com
Inmobiliarias	Open-Realty	http://www.open-realty.org/
Portal de Empleos	joobsbox	http://www.joobsbox.com/
Hoteles	MRBS	http://mrbs.sourceforge.net/
	PHP Residence (gestión hotelera)	http://www.hoteldruid.com/es
Anuncios clasificados	Open Classifieds	http://open-classifieds.com/
Redes sociales	PeoplePods	http://peoplepods.net/
	Family	http://www.familycms.com/
Microblogging	Sharetronix (red social privada)	http://sharetronix.com/
Mostrar tweets en una Web	twitster	http://plasticmind.com/twitster/
Venta de automóviles	Open Auto Classifieds	http://openautoclassifieds.com/
Soporte técnico, tickets	OTRS	http://otrs.com/?lang=es
	osTicket	http://www.osticket.com/
Consultorios médicos	Open Clinic	http://openclinic.sourceforge.net/
Galerías de imágenes	Piwigo	http://piwigo.org/
	Plogger	http://www.plogger.org/
Galerías de arte, museos, bibliotecas	Omeka	http://omeka.org/

Estadísticas de visitas (estilo Google Analytics)	Piwik	http://piwik.org/
Votación de ideas	IdeaTorrent	http://ideatorrent.org/
Gestión de ONG	CiviCRM	http://civicrm.org/
Videoteca	AJAX-FilmDB	http://sourceforge.net/projects/ajaxfilmdb/
Subastas (estilo Mercado Libre)	PHPauction	http://sorceforge.net/projects/phpnggpl/

Tabla 2. Los sistemas más populares de cada rubro.

Podemos encontrar estas y muchísimas otras aplicaciones web libres, simplemente buscando en Google el rubro al que apuntamos (si es en inglés, tendremos mayores resultados) junto con la sigla GPL, o con las palabras *open source* o *free software.*

También, podemos recurrir a sitios que mantienen **listados** de software libre, clasificado por categoría, como los siguientes:

Dirección	Comentarios
http://www.opensourcescripts.com/dir/PHP/	Casi 40 categorías de scripts PHP.
http://osliving.com/	Más de 60 categorías, pero muchas no son aplicaciones Web hechas con PHP.
http://php.opensourcecms.com/	Pulsando la opción "All CMS Demos" en la columna izquierda, nos muestra más de 320 scripts.
http://www.scriptscenter.com.ar/category/scripts-php/open-source-scripts-php	Decenas de aplicaciones.

Tabla 3. Sitios con listados de software libre para descargar.

Como vemos, contamos con una variedad de proyectos para todos los gustos, listos para ser descargados.

Criterios para elegir entre proyectos similares

El criterio clave para decidirnos a implementar un proyecto es el **grado de actividad** de su **comunidad** de desarrolladores: cuánto tiempo transcurre entre versiones nuevas del sistema, y cuál es el tiempo de respuesta ante un error detectado. Es el punto crítico.

Podemos darnos cuenta de qué tan "viva" está la comunidad de un proyecto, por la actividad de sus foros o listas de correo oficiales, por la cantidad de nuevas contribuciones o agregados al sistema original que se publican e, incluso, por la cantidad de libros que se publican dedicados al sistema (existen **decenas** de libros sobre distintos aspectos de Joomla, decenas de libros sobre Magento, decenas sobre Drupal, WordPress, etc. y lo mismo sucede con la mayoría de los demás proyectos exitosos).

Otro punto importante, ya que puede ahorrarnos mucho tiempo de investigación, es que el proyecto posea una **documentación** disponible para entender cómo se puede modificar cada cosa. Si está en nuestro idioma, mucho mejor.

Por último (y es totalmente intencional haberlo dejado para el final), el criterio que paradójicamente menor importancia tiene a la hora de elegir este tipo de sistemas es la **calidad** de la codificación (uso de versiones nuevas de PHP, uso de OOP). Ya que de nada sirve la mejor calidad inicial, si el proyecto no logró convocar una comunidad de programadores que hagan avanzar el proyecto y publiquen contribuciones, parches de seguridad, etc. Por supuesto, lo ideal es que se unan una gran comunidad y un buen código.

Instalando un sistema para blogs

Para comprender cómo se pueden realizar modificaciones en una aplicación programada por otros, vamos a realizar una modificación paso a paso, que nos sirva de ejemplo de cómo enfrentar esta tarea.

Supongamos que nuestro cliente nos está solicitando un *blog*. Tenemos a nuestra disposición muchos sistemas libres de este rubro, listos para usar: WordPress, Typo3, b2evolution, y muchos más.

Luego de comparar comunidades, observaremos que notoriamente la comunidad más grande y activa es la que mantiene el proyecto **WordPress**, con decenas de miles de plugins o módulos agregados por la comunidad de usuarios y miles de temas (plantillas, diseños) al momento

de escribir estas líneas: solo por eso, merece que nos inclinemos por este sistema de *blogs,* ya que encontraremos plugins que realizarán casi cualquier tarea que se nos ocurra. Únicamente, tendremos que aprender a instalar esos agregados.

Comenzaremos descargando WordPress en español desde: http://es.wordpress.org/

Figura 1. Página de descarga de WordPress en español.

Una vez descomprimido en una carpeta de nuestro servidor (nosotros lo hemos colocado en una carpeta llamada **weblog**), abriremos con nuestro editor el archivo wp-configsample.php, y escribiremos los datos de conexión a nuestra base.

Atención: debemos tener **ya creada** de antemano una base de datos con ese nombre, para que, al continuar la instalación, se puedan crear las tablas necesarias. Si estamos trabajando localmente, iremos al phpMyAdmin y crearemos una nueva base; si estamos en un *hosting,* usaremos las herramientas que ofrezca para crear una base (o utilizaremos una base existente).

```
18 /** El nombre de tu base de datos de WordPress */
19 define('DB_NAME', 'mibase');
20
21 /** Tu nombre de usuario de MySQL */
22 define('DB_USER', 'root');
23
24 /** Tu contraseña de MySQL */
25 define('DB_PASSWORD', 'clave');
```

Figura 2. Editando el archivo de configuración.

Una vez ingresados los datos de conexión, deberemos escribir frases diferentes en toda esta zona:

```
44 define('AUTH_KEY', 'pon aquí tu frase aleatoria'); // Cambia esto por tu frase
    aleatoria.
45 define('SECURE_AUTH_KEY', 'pon aquí tu frase aleatoria'); // Cambia esto
    por tu frase aleatoria.
46 define('LOGGED_IN_KEY', 'pon aquí tu frase aleatoria'); // Cambia esto por
    tu frase aleatoria.
47 define('NONCE_KEY', 'pon aquí tu frase aleatoria'); // Cambia esto por tu
    frase aleatoria.
48 define('AUTH_SALT', 'pon aquí tu frase aleatoria'); // Cambia esto por tu
    frase aleatoria.
49 define('SECURE_AUTH_SALT', 'pon aquí tu frase aleatoria'); // Cambia esto
    por tu frase aleatoria.
50 define('LOGGED_IN_SALT', 'pon aquí tu frase aleatoria'); // Cambia esto por
    tu frase aleatoria.
51 define('NONCE_SALT', 'pon aquí tu frase aleatoria'); // Cambia esto por tu
    frase aleatoria.
```

Figura 3. Frases aleatorias.

El sistema las usará para encriptar datos.

Ahora, guardaremos una copia de este archivo con el nombre de **wp-config.php** en la misma carpeta en la que estábamos. Una vez hecho esto, entraremos con nuestro navegador a (reemplazar *localhost* por el dominio en caso de estar trabajando *on-line,* y, en lugar de *weblog,* coloquemos el nombre de la carpeta donde lo hayamos instalado):

```
http://localhost/weblog/wp-admin/install.php
```

Allí, luego del texto de bienvenida, veremos la siguiente pantalla, donde deberemos cargar los datos que nos solicita:

Figura 4. Instalación de WordPress.

Una vez ingresados, nos dirá que ya tenemos instalado nuestro *blog*. Ahora, nos pide que nos identifiquemos para comenzar a administrar el contenido de nuestro nuevo sitio:

Figura 5. Entramos con nuestro usuario y clave.

Una vez identificados, quedaremos frente al panel de administración, desde donde puede configurarse casi todo.

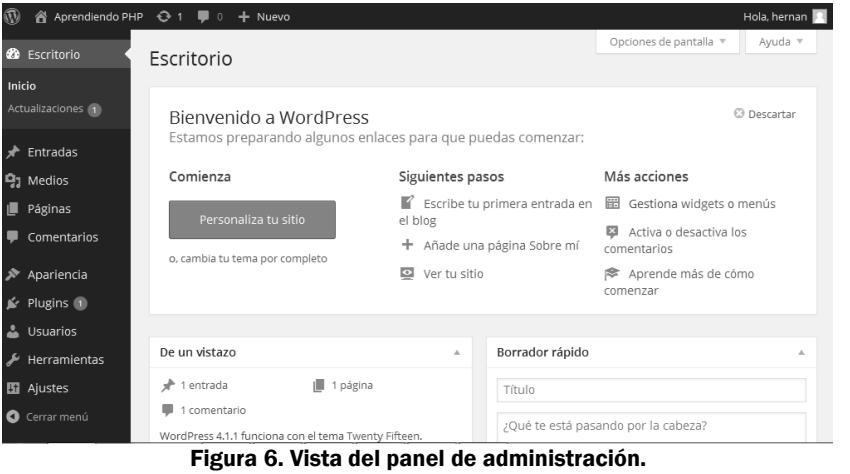

Figura 6. Vista del panel de administración.

Si entramos a http://localhost/weblog/ veremos nuestro sitio ya listo para funcionar:

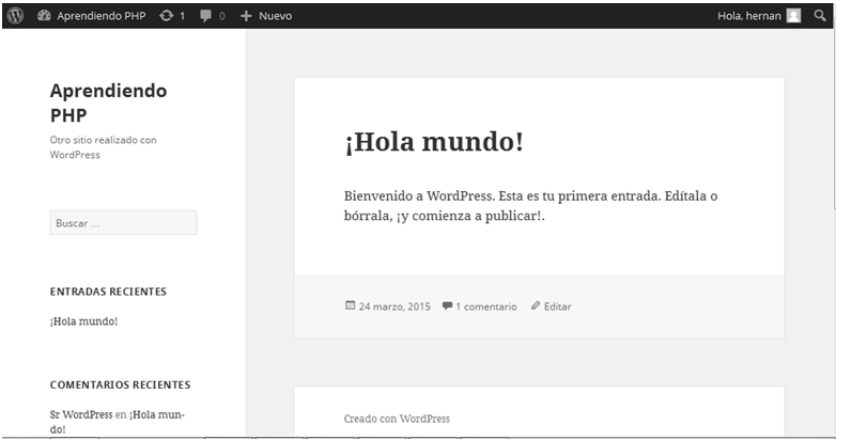

Figura 7. Apariencia de nuestro sitio.

Con estos simples pasos, ya tenemos un sistema de *weblog* funcionando. Ahora, vamos a comenzar con las modificaciones.

Modificaciones básicas

Una vez que nuestro sistema esté funcionando, la lista de modificaciones que podemos realizar es muy extensa, pero debería incluir al menos estos aspectos:

1. Modificar el diseño: elegir un *theme* (plantilla), cambiando la imagen, los colores y estilos CSS, y agregarle *widgets.*
2. Configurar las categorías en las que clasificaremos los textos que publicaremos y agregar los primeros textos.
3. Por último, agregar "contribuciones" *(plugins)*, o programar a mano modificaciones a las funcionalidades existentes.

Veamos rápidamente los cambios más fáciles de realizar:

1. Para elegir otra plantilla de diseño, en el panel de administración, iremos al menú izquierdo, pulsaremos donde dice **Apariencia** → **Temas** y, luego, en el botón **Añadir nuevo**.

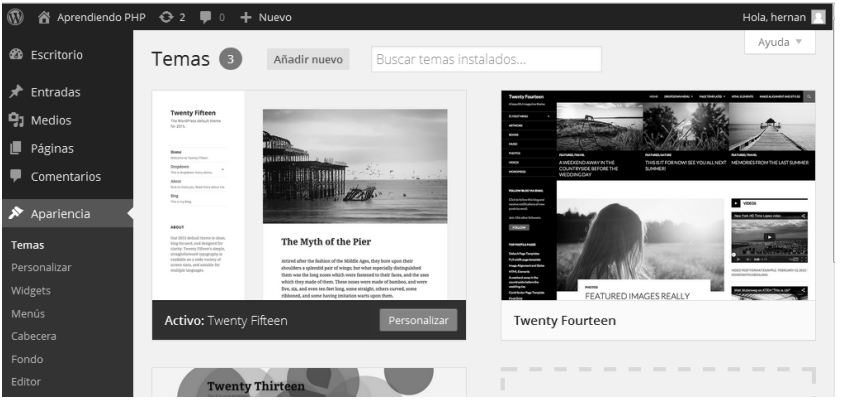

Figura 8. Eligiendo un tema.

Allí, podremos seleccionar temas por varios criterios: color, cantidad de columnas, tipo de ancho (fijo o flexible), características especiales, y asunto del sitio (para qué tipo de contenidos es). También, podemos navegar por los más **Destacados**, o por los **Recientes**.

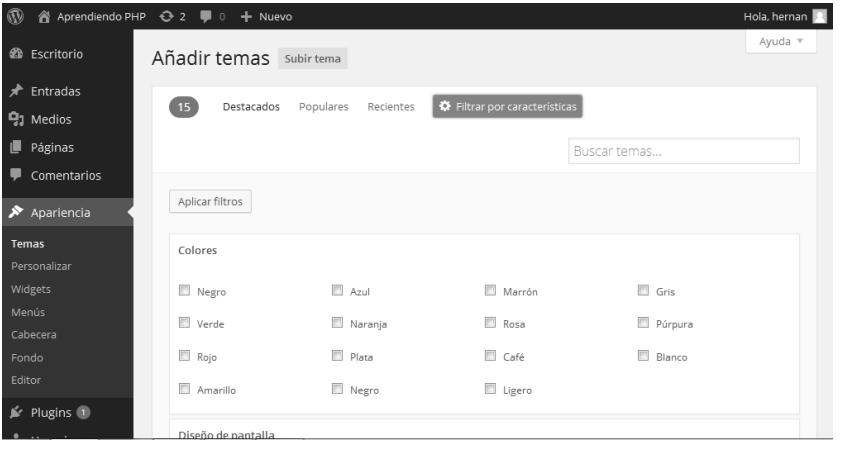

Figura 9. Instalando el tema elegido.

Una vez elegida la plantilla que nos guste, podemos previsualizarla pulsando en **Vista previa**, y también instalarla en nuestro *weblog* con el enlace **Activar** que cada miniatura de diseño tiene debajo:

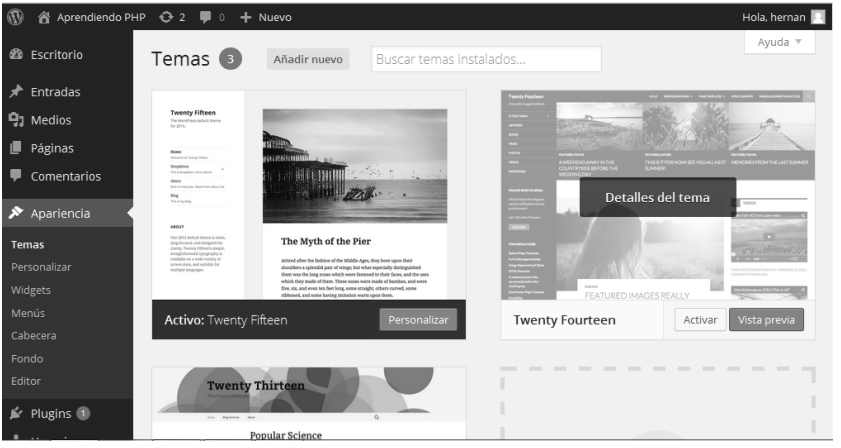

Figura 10. Instalando el tema elegido.

Podemos completar la parte estética y a la vez funcional, arrastrando a la columna derecha los *widgets* que queremos poner a disposición de nuestros usuarios:

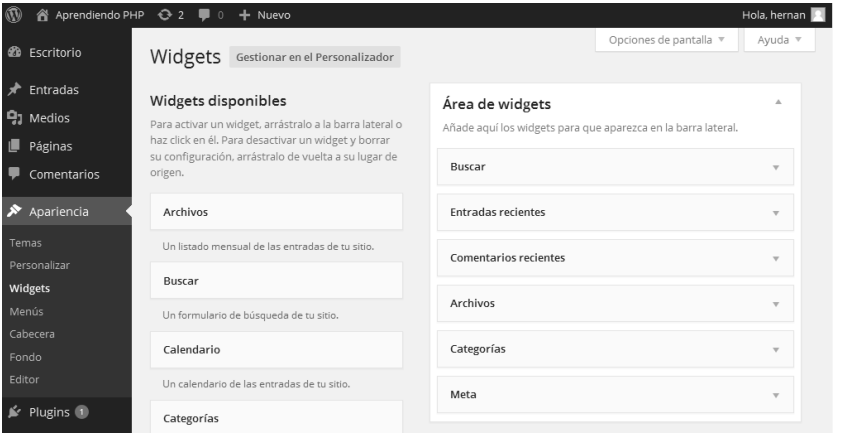

Figura 11. Activando *widgets*.

2. Pasemos al segundo punto: para configurar las **Categorías** en las que agruparemos nuestros contenidos, dentro del panel de administración vamos a **Entradas** → **Categorías**:

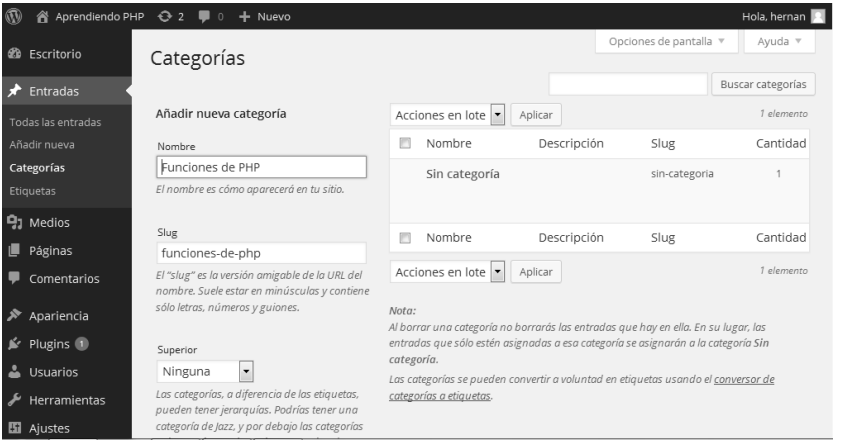

Figura 12. Creando categorías.

Si volvemos a *Widgets* y agregamos categorías en la barra lateral, nuestro sitio irá tomando forma:

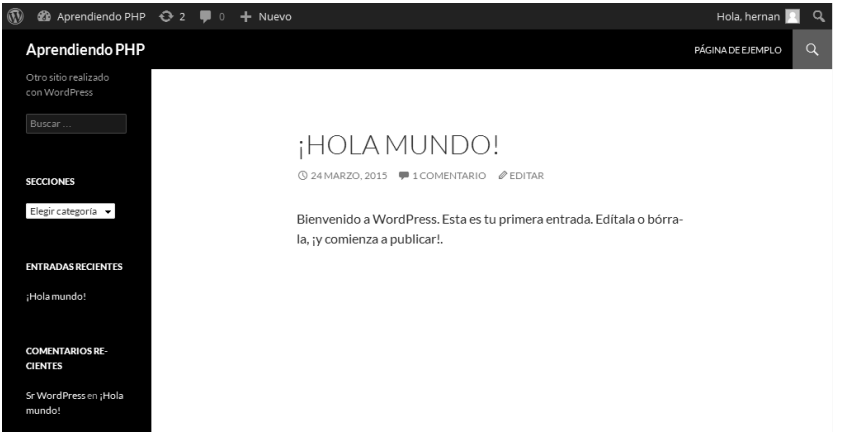

Figura 13. Nuestro nuevo diseño.

3. Por último, vamos a trabajar con *plugins* (agregados). En el panel de administración de WordPress, iremos a la sección de **Plugins → Añadir nuevo**, y escribiremos dentro del campo de búsqueda la palabra "Weather" (estamos

buscando un *plugin* para mostrar el estado del tiempo) y en lugar de **Palabra clave**, elegiremos **Etiqueta**:

Figura 14. Navegaremos por los más de 36000 plugins.

Una vez elegido alguno de los candidatos (en este caso, elegiremos "POWr Weather"), pulsamos en **Instalar ahora**:

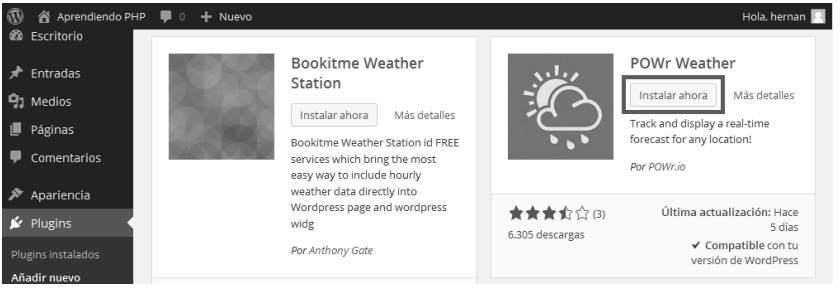

Figura 15. Instalando el *plugin* "POWr Weather".

Una vez instalado, vamos a activarlo:

Figura 16. Activando el *plugin* "POWr Weather".

Luego de activarlo, vamos a agregar el *widget* en nuestra columna derecha:

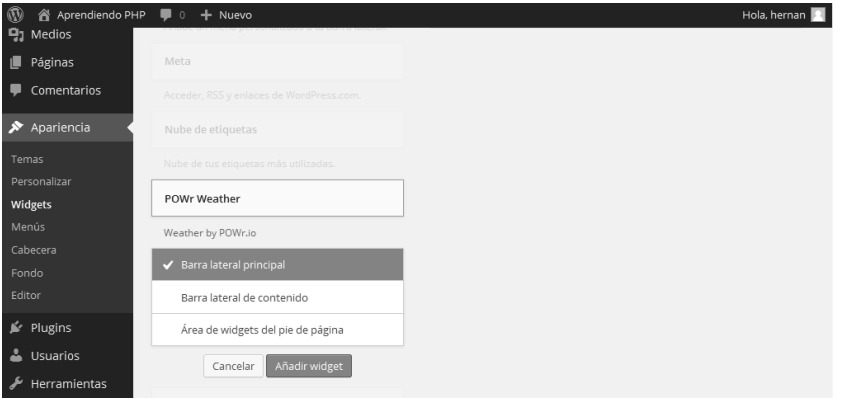

Figura 17. Agregando el *widget* a la barra lateral principal.

Si entramos a navegar por nuestro sitio, veremos el bloque a la derecha de la página, y si lo pulsamos, podemos editar la ciudad de la que mostrará su clima:

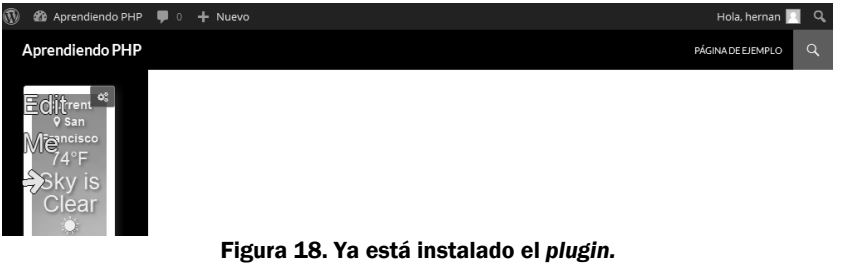

Figura 18. Ya está instalado el *plugin*.

Podemos aprender en detalle estas modificaciones básicas mediante la documentación oficial del sitio Web de WordPress: http://developer.wordpress.org/reference

Ya podemos pasar, entonces, a analizar el funcionamiento interno de este sistema, para realizar agregados o modificaciones más artesanales, que impliquen codificar en PHP.

Estructura para modificar un plugin

Todos los agregados que deseemos realizar a nuestro WordPress, debemos plantearlos en forma de *plugin,* sin modificar el código a mano, sino utilizando la API que ofrece WordPress. Si antes de crear nuestro primer *plugin* queremos analizar uno existente, podemos examinar la estructura de archivos, *includes* y dependencias de un *plugin* similar al que recientemente instalamos.

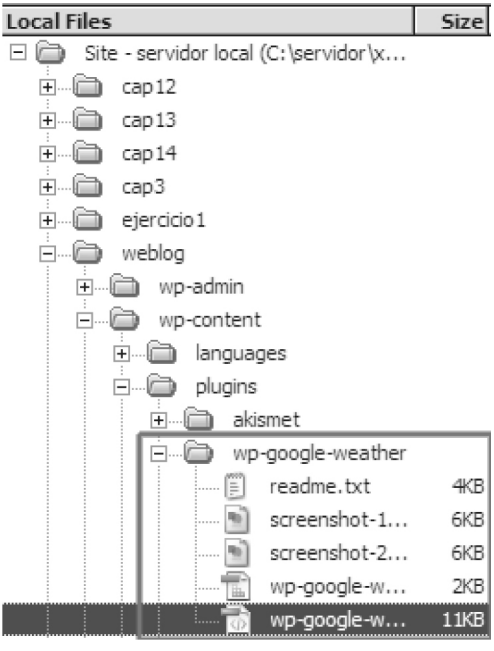

Figura 19. Estructura de archivos del *plugin* "WP Google Weather".

Notemos que por cada *plugin* instalado, existe una carpeta dentro de /wp-content/plugins/

En este caso, la carpeta de nuestro *plugin* se llama **wp-google-weather**. Dentro de la misma, encontraremos una hoja de estilos (desde donde podremos modificar los colores, tipos de letra, tamaños, etc. de nuestro bloque), y un archivo **wp-google-weather.php** que es el que contiene todo lo necesario para que el *plugin* funcione. Allí, están definidas las clases y las funciones necesarias.

Cómo crear un plugin

Pero en lugar de modificar algo, vamos a crear desde cero nuestro primer *plugin.* Comenzaremos definiendo su objetivo: crearemos un *plugin* para mostrar **textos aleatorios** (frases, citas, publicidades en formato texto, etc.).

Lo primero que debemos hacer es crear una **carpeta** dentro de /wp-content/plugins/ y le pondremos un nombre único: esto es importante, ya que si el *plugin* lo compartimos, otros usuarios podrían llegar a tener instalado uno con el mismo nombre, y habrá un conflicto, ya que dentro de la carpeta *"plugins"* no puede haber dos con el mismo nombre.

Así que, ante todo, buscaremos dentro del listado de *plugins* el nombre que estamos deseando utilizar, para confirmar que no esté siendo utilizado por otro *plugin.* En nuestro caso, lo llamaremos "muestrafrases".

Dentro de esa carpeta, crearemos un archivo llamado muestrafrases.php que contendrá un **comentario** estandarizado en un formato que WordPress puede leer, con datos acer- ca del autor y la licencia utilizada (la podemos ver en los archivos de ejemplo que pueden descargarse de la Web de este libro).

A continuación, crearemos nuestro propio código, teniendo en cuenta que debemos "enlazarlo" al núcleo de WordPress utilizando uno de los dos *hooks* (ganchos) que WordPress proporciona: acciones y filtros.

1. Las **acciones** sirven para que nuestro código se ejecute ante determinado evento (al cargarse la página, por ejemplo, o al publicarse un nuevo *post,* al enviar un mensaje por correo electrónico, etc.).
2. Los **filtros** sirven para ir modificando sucesivamente un elemento antes de dejarlo completo para que sea mostrado. Por ejemplo, si un *plugin* modifica la manera en que se muestra la fecha y la hora, y nuestro *plugin* también

modifica esos mismos datos, éstos se irán pasando "de función en función" por todas las funciones "enganchadas", para que cada una lo vaya modificando sucesivamente hasta su estado final.

En este caso, no vamos a usar filtros, pero sí vamos a utilizar una acción para engancharnos al hilo de ejecución de WordPress.

Para que nuestra función se enganche con el núcleo, debemos seguir estos pasos:

1. Crear nuestra función, declarándola dentro del archivo de nuestro *plugin.* Ésta es la función que se ejecutará cuando suceda el evento que especifiquemos como "enganche".
2. Luego, crearemos un *hook* de acción, para relacionar nuestra función con el hilo de ejecución de WordPress. Para eso, se utiliza la función add_action().
3. Por último, si ya hemos colocado nuestra función y nuestro "gancho", solo resta activar el *plugin* y ubicarlo en la página desde el panel de administración.

Vamos a hacerlo, paso a paso:

Empecemos creando la **función** que leerá las frases desde un archivo de texto (para simplificar, no hemos usado una base de datos), elegirá una al azar y la mostrará:

```
<?php
function muestrafrases(){


      $archivo = ABSPATH. "/wp-content/plugins/
                  muestrafrases/frases.txt";
      // La ruta al archivo. ABSPATH es una constante
definida en WordPress.
      $frases = file($archivo);
      shuffle($frases);
      echo ("<p>Nuestra frase del día es
".$frases[0]."</p>");
```

```
}
?>
```

Ahora, el segundo paso: crear un **enganche** *(hook)* entre nuestra función y alguno de los eventos que se van sucediendo en WordPress.

Por ejemplo, el evento podría ser el instante en el que se inicializan todos los *widgets:*

```
add_action('widgets_init', 'muestrafrases');
```

Podemos consultar la lista completa de *hooks* posibles (en inglés) en: http://adambrown.info/p/wp_hooks

Esa llamada a add_action debe estar colocada en el mismo archivo que la declaración de nuestra función, pero fuera de ella.

Por último, debemos entrar al panel de administración y **activar** el *plugin.*

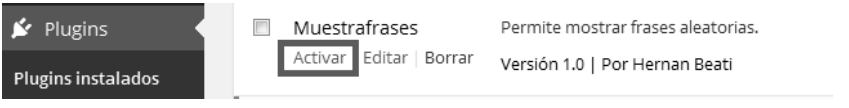

Figura 20. Activamos nuestro *plugin.*

La frase aún no aparece, porque falta arrastrar el widget a una zona de la plantilla:

Figura 21. Definamos en qué zona aparecerá el widget.

Además, aparece inclusive en el panel de administración.

Es necesario, entonces, agregar algo que nos permita ubicar los datos producidos por nuestro *plugin,* en algún **lugar específico** de la página, dentro del diseño.

Para eso, vamos a añadir a nuestro *plugin* (que por ahora es solo una funcionalidad que manipula datos) un bloque visible dentro de la columna derecha: un *widget.*

Para crear un *widget,* tendremos que producir una clase que "extienda" la clase *WP_ Widget* (que herede sus métodos y propiedades), y nuestra función, en lugar de estar suelta, será el método **constructor** de esa clase que se ejecutará automáticamente cuando se cree una instancia con el operador *new*.

```
<?php
add_action('widgets_init', 'cargar_frases');


function cargar_frases(){
      register_widget('muestrafrases');
}


class muestrafrases extends WP_Widget {


      // Método constructor:


      function muestrafrases() {
         // Opciones del Widget
         $widget_ops = array('description' => __('Un widget
         que muestra frases.', 'muestrafrases'));


      // Controles del Widget
         $control_ops = array('width' => 200, 'height' =>
```

```
        350, 'id_base' => 'muestrafrases');

        // Creación del Widget
        $this->WP_Widget('muestrafrases', __('Muestra
        Frases', 'muestrafrases'), $widget_ops,
        $control_ops);
    }// fin de muestrafrases

    function update( $new_instance, $old_instance ) {
      $instance = $old_instance;
      $instance['title'] = strip_tags( $new_
                    instance['title'] );
      $instance['lang'] = strip_tags( $new_
                    instance['lang'] );
      $instance['alignment']=$new_instance['alignment'];

      return $instance;
    }// fin de update

    function form($instance) {
    // Configuración de valores por defecto para elwidget
    $defaults = array( 'title' => __('Ejemplo',
   'example'),  'lang' => 'es', 'alignment' => false );
   $instance = wp_parse_args( (array) $instance,
   $defaults );
?>
```

```
<p>

<label for="<?php echo $this->get_field_id('title' );
?>"><?php _e('Titulo:', 'hybrid'); ?></label>
<input id="<?php echo $this->get_field_id('title' ); ?>"
name="<?php echo $this->get_field_name( 'title'); ?>"
value="<?php echo $instance['title']; ?>" size="30" /></p>

<p>
<label for="<?php echo $this->get_field_id('lang' );
?>"><?php _e('Idioma:', 'lang'); ?></label>
<input id="<?php echo $this->get_field_id('lang' ); ?>"
name="<?php echo $this->get_field_name( 'lang' );?>"
value="<?php echo $instance['lang']; ?>" size="2"></p>
<p>

<input class="checkbox" type="checkbox"
<?php if($instance['alignment'] == true) echo 'checked';?>
id="<?php echo $this->get_field_id( 'alignment' ); ?>"
name="<?php echo $this->get_field_name( 'alignment' );
?>">
<label for="<?php echo $this->get_field_id( 'alignment');
?>"><?php _e('Alineación centrada?','alignment');
?></label></p>

<?php
}// fin de form
```

```
/**
* Cómo mostrar el widget en la pantalla.
*/
function widget( $args, $instance ) {
      extract( $args );

      // Nuestras variables de configuración del widget
      $title = apply_filters('widget_title',
      $instance['title'] );

      $lang = $instance['lang'];
      $alignment = $instance['alignment'];

      /* Parte previa al widget (definida en las plantillas
o themes) */
      echo $before_widget;

      // Muestra el título del widget si fue definido
      if ( $title )
      echo $before_title . $title . $after_title;

      // Muestra el nombre del widget, si fue definido
      $this->buildWidget($lang, $alignment);
}
// Parte siguiente al widget (definida en los themes)
```

```
echo $after_widget;


function buildWidget($lang, $alignment) {


      if($alignment == true) {

         $alignmentclass = 'centered';

      }


      echo '<div class="muestrafrases"><dl
class="'.$alignmentclass.'">';


      $archivo = ABSPATH . "/wp-
content/plugins/muestrafrases/frases.txt";

      // La ruta al archivo


      $frases = file($archivo);

      /* A $frases lo convertimos en una matriz con tantas

celdas como líneas -renglones- tenía el archivo de texto.
*/


      shuffle($frases);

      /* Mezclamos el contenido de esa matriz con la
función shuffle, por lo cual, no sabemos cuál frase quedó
en el primer lugar, y la mostramos: */


      echo ("<p>".$frases[0]."</p>");
```

```
      echo '</dl></div><div style="clear: both;"></div>';
}

function replaceChars($data){
      $search = array('Ä', 'Ö', 'Ü', 'ä', 'ö', 'ü', ' ');
      $replace = array('Ae', 'Oe', 'Ue', 'ae', 'oe','ue',
'%20');
      $output = str_replace($search, $replace, $data);

      return $output;
}
}

/* Esta es la función que llamará a mostrar, que a su vez
es quien crea una instancia de "muestrafrases" y ejecuta
su método buildWidget que realmente escribe el código del
widget. */
add_shortcode('muestrafrases', 'mostrar');

function mostrar($atts, $content = null) {

      $muestrafrases = new muestrafrases();
      ob_start();
      $muestrafrases->buildWidget($language, $alignment);
      $output = ob_get_contents();
      ob_end_clean();
      return $output;
```

```
}
```

Por supuesto, nos queda por delante seguir investigando y probando, agregando cada vez un grado mayor de dificultad a nuestros *plugins* y *widgets* (haciendo que interactúen con la base de datos, por ejemplo).

Figura 22. Ahora sí, nuestro plugin genera los datos y un widget los muestra.

Existen libros enteros dedicados a conocer a fondo la API de WordPress, que recomendamos leer en caso de querer dedicarnos a adaptar este sistema tan popular de *blog*s. Y aunque no consigamos ningún libro, en la Web oficial, contamos con la información suficiente como para desarrollar:

- Podemos revisar la lista oficial de todas las **funciones** que utiliza WordPress en:

  http://codex.wordpress.org/es:Referencia_de_Funciones

- Podemos analizar las **tablas** de su base de datos:

  http://codex.wordpress.org/es:Database_Description

- Podemos guiarnos para crear un *plugin* con este material: http://codex.wordpress.org/Escribiendo_un_Plugin

- Y podemos conocer las funciones de la API de WordPress: http://codex.wordpress.org/es:Plugin_API

Trasladando esta experiencia a otros sistemas de código abierto, podemos concluir que la clave es **aprender el uso de la API** (*Application Programming Interface,* o Interfaz para Programación de Aplicaciones) que cada uno de estos sistemas ofrece. Entender poco a poco sus clases, sus métodos, y de qué forma se agregan nuevas funciones (como hicimos recién con WordPress).

Seguramente, puede costarnos un par de días de investigación ante cada nuevo sistema que aprendamos a modificar pero, a cambio, contaremos con nada menos que un **sistema completo** listo para usar, testeado y mantenido al día por una comunidad enorme de programadores. Ganaremos tiempo, ganaremos dinero, ganaremos calidad para ofrecer a nuestros clientes.

Y eso es muy bueno.