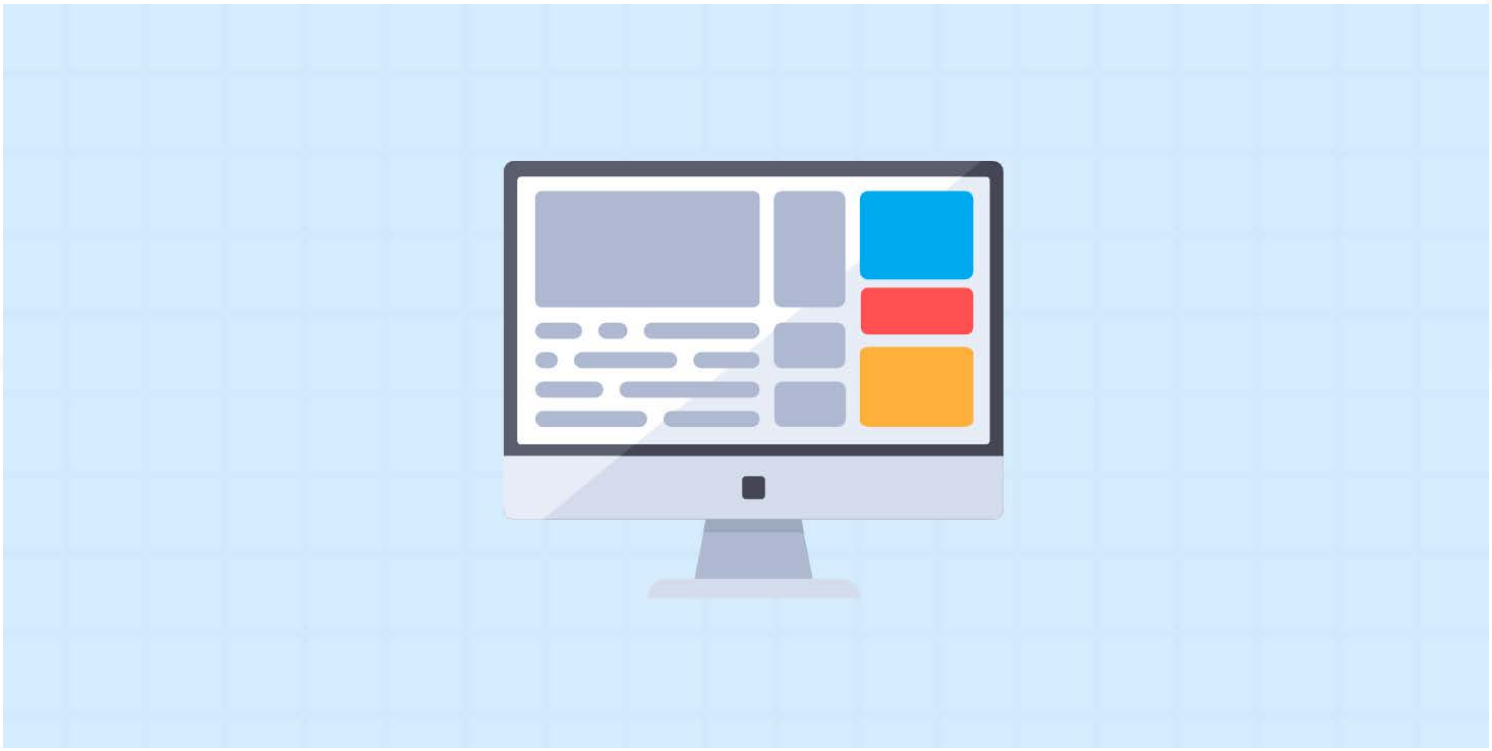
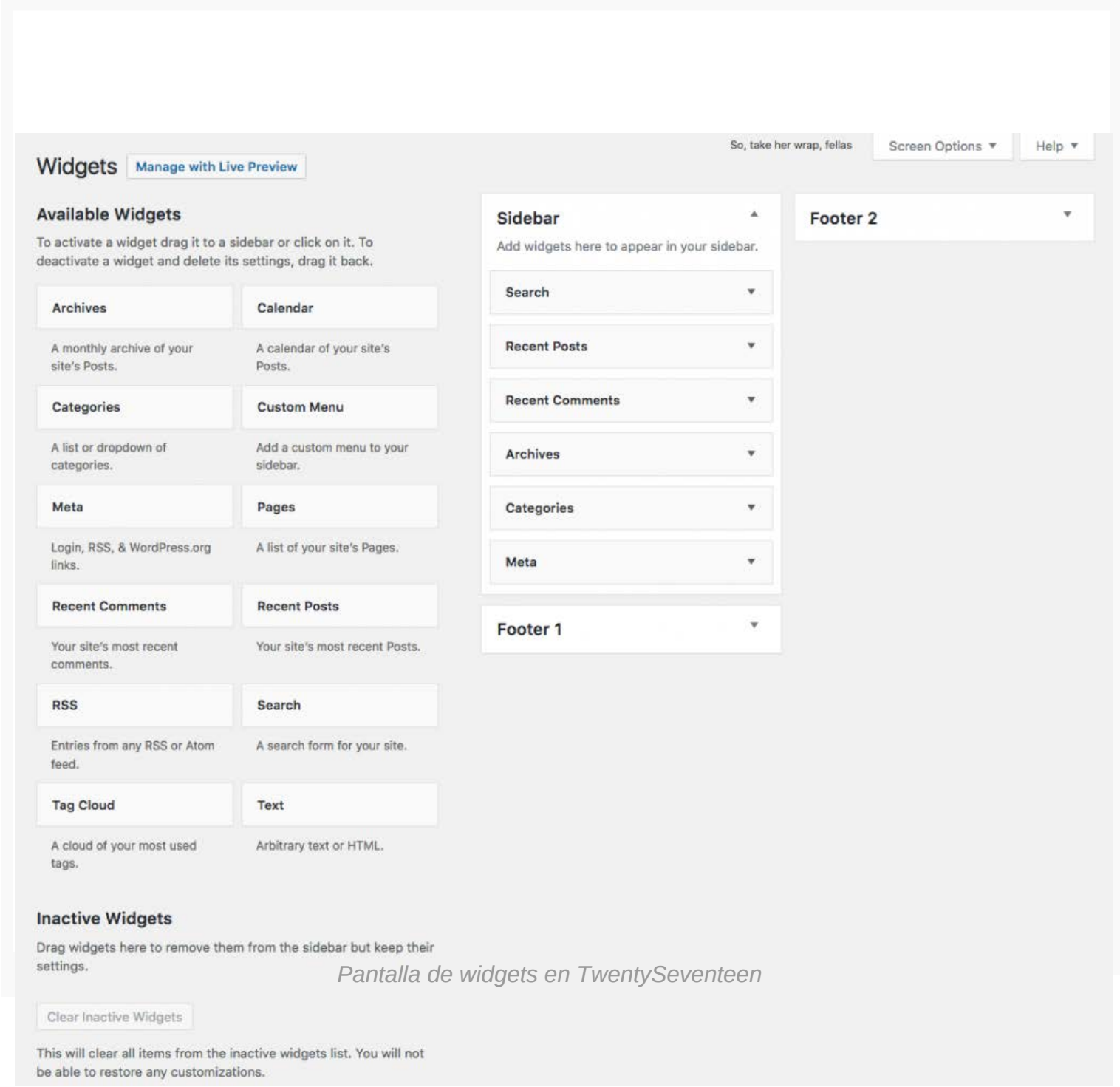
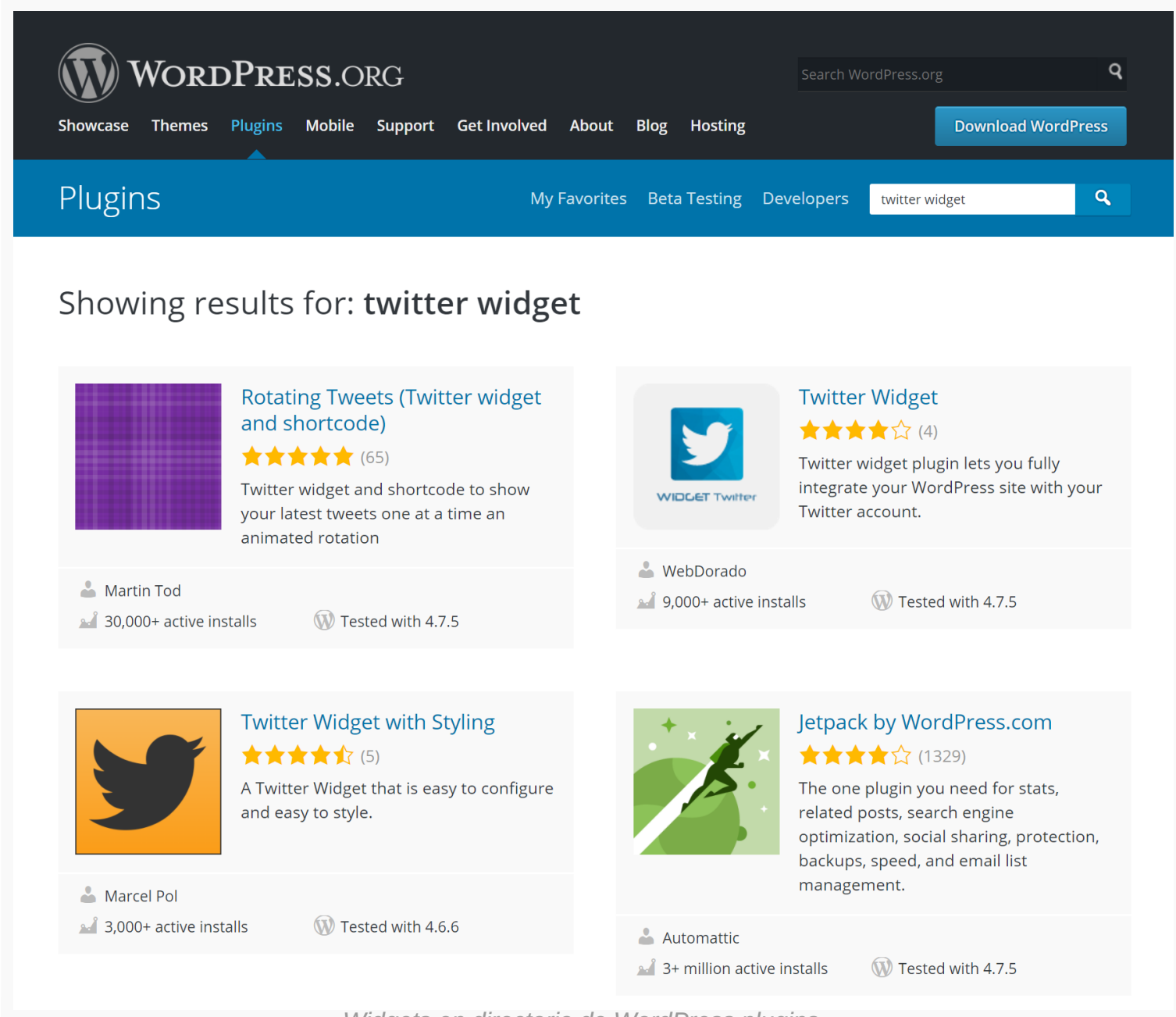


Los Widgets de WordPress son bloques de HTML estático o dinámico, contenido que puede ser añadido a zonas específicas de las páginas front-end (zonas widget o barras laterales) WordPress proporciona un buen número de widgets integrados, como Archivos, Categorías, Nube de Etiquetar, Buscar, Publicaciones Recientes, Calendario, y más. Además, como voy a explicar en esta publicación, los desarrolladores de plugins pueden fácilmente crear un WordPress widget desde cero, añadir características personalizadas y especificar contenido a cualquier tema que sea compatible con esta característica increíble.



Pantalla de widgets en TwentySeventeen

Se puede encontrar toneladas de widgets de WordPress en el Directorio de Plugins (actualmente más de 50,000), en los mercados de WordPress y sitios web de proveedores y es probable que encuentre cualquier widget que pueda necesitar. De todas formas, de vez en cuando no encontrará el widget que está buscando, y necesitará construir su propio.



Widgets en directorio de WordPress plugins

Cómo Crear un WordPress Widget

La buena noticia es que usted puede crear un widget de WordPress incluso si no es un desarrollador profesional. Solo necesita un conocimiento muy básico de OOP en PHP,y un entendimiento general del desarrollo de plugin de WordPress . Esta publicación le guiará a través del desarrollo de widgets de WordPress, explicando cómo crear un widget que permita a los administradores del sitio seleccionar una lista de publicaciones que se mostrarán en una barra lateral.

Seguiremos estos pasos:

- Construyendo e instalando un plugin que proporciona un widget simple
- Configurar el widget
- Construyendo el formulario de admin
- Actualizando opciones de widget
- Produciendo e imprimiendo la salida de widget

La Estructura Básica de WordPress Widgets

Desde la perspectiva de un desarrollador, un widget es una instancia de la clase WP_Widget. Por lo tanto, para crear un widget personalizado, necesitamos extender esta clase desde un plugin. Nuestra primera tarea es crear un nuevo archivo .php en la carpeta/wp-content/plugins/my-widget/ que tenga el siguiente encabezado:

```
<?php

/*
Plugin Name: My Widget
Plugin URI: http://wordpress.org/extend/plugins/#
Description: This is an example plugin
Author: Your Name
Version: 1.0
Author URI: http://example.com/
*/
```

Este es un simple encabezado de plugin, pero es suficiente para nuestros propósitos (encontrará información adicional acerca de requisitos de encabezado en el Codex). Guarda el script como my-widget.php, vaya a la Pantalla de Plugins, y active el plugin. Ahora tenemos que extender la clase WP_Widget y algunas de sus funciones en un momento preciso de la ejecución de WordPress. Agreguemos el siguiente código a nuestro plugin:

```
// register My_Widget

add_action( 'widgets_init', function(){
register_widget( 'My_Widget' );
});
```

La función añadir acción engancha una devolución de llamada personalizada al action hook widgets_init, que se activa después de que los widgets por defecto han sido registrados.

> Nota: un hook de acción proporciona una forma para ejecutar una función en un punto específico en la ejecución de WordPress y sus extensiones (ver WordPress Codex para más información)

La función registrar widget registra el widget especificado, el cual es una extensión de la clase WP_Widget:

```
class My_Widget extends WP_Widget {
// class constructor
public function __construct() {}

// output the widget content on the front-end
public function widget( $args, $instance ) {}

// output the option form field in admin Widgets screen
public function form( $instance ) {}

// save options
public function update( $new_instance, $old_instance ) {}
}
```

In order to build our simple widget we just need to use four class methods:

- __construct es el constructor de clase y permite establecer parámetros de widget personalizados.
- widget() repite el contenido del widget en el front-end.
- form() tiene como salida los elementos de formulario del widget admin.
- update() actualiza el objeto de widget actual.

Usted puede tomar este código desde Gist. Una vez que hemos configurado una plantilla de plugin básica, podemos avanzar un paso adelante y definir cada método de subclase.

Configurando el Widget: Constructor de Clase

El constructor de clase registra la ID del widget, título y opciones adicionales, como classname, description, base_id, anchura y altura. En nuestro widget de ejemplo asignamos ID, nombre y dos opciones de widget:

```
public function __construct() {
```

```
$widget_ops = array(
'classname' => 'my_widget',
'description' => 'A plugin for Kinsta blog readers',
);
parent::__construct( 'my_widget', 'My Widget', $widget_ops );
}
```

my_widget es la Base ID para el widget
'My Widget' es el título del widget
$widget_ops es una matriz de opciones

Vea la Referencia del Código para la lista completa de opciones disponibles.

Construyendo el Formulario de Admin

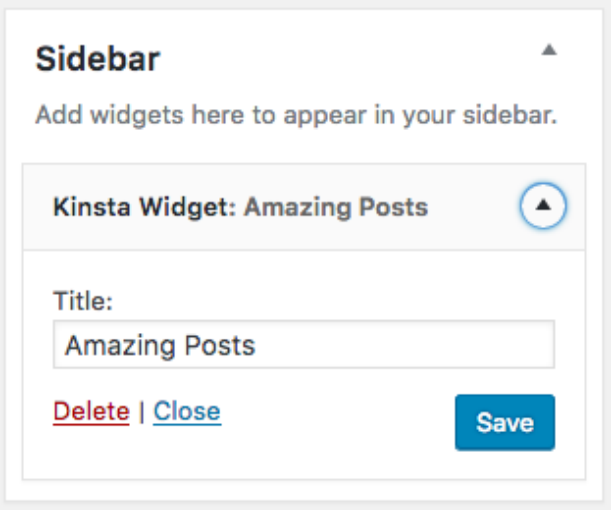
A continuación, tenemos que crear un formulario de administración que permita a los usuarios establecer opciones personalizadas para el widget. Para lograr esta tarea definimos el método form() como sigue:

```
public function form( $instance ) {
```

```
$title = ! empty( $instance['title'] ) ? $instance['title'] : esc_html__( 'Title',
'text_domain' );
?>
<p>
<label for="<?php echo esc_attr( $this->get_field_id( 'title' ) ); ?>">
<?php esc_attr_e( 'Title:', 'text_domain' ); ?>
</label>

<input
class="widefat"
id="<?php echo esc_attr( $this->get_field_id( 'title' ) ); ?>"
name="<?php echo esc_attr( $this->get_field_name( 'title' ) ); ?>"
```

```
type="text"
value="<?php echo esc_attr( $title ); ?>">
</p>
<?php
}
```

La función form() function mantiene como argumento una matriz de las opciones del widget actual. El elemento 'title' de $instance proporciona la opción de título, mientras que get_field_id y get_field_name establece la ID y valores de nombre al campo de texto.



Un formulario de widget muy básico con un campo de entrada singular

Este sencillo ejemplo demuestra cómo obtener valores de opciones de la base de datos y usarlos para crear un formulario de administración para configurar el comportamiento del widget. Obviamente, no estamos limitados a campos de texto. El siguiente código proporciona una lista de verificación de las publicaciones más recientes:

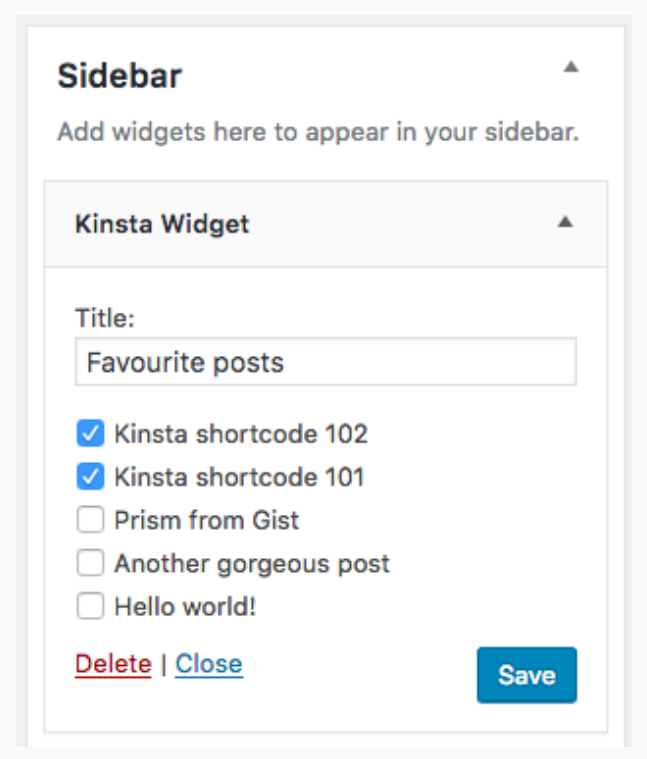
```
public function form( $instance ) {
```

```
$posts = get_posts( array(
'posts_per_page' => 20,
'offset' => 0
) );
$selected_posts = ! empty( $instance['selected_posts'] ) ?
$instance['selected_posts'] : array();
?>
<div style="max-height: 120px; overflow: auto;">
<ul>
<?php foreach ( $posts as $post ) { ?>

<li><input
type="checkbox"
name="<?php echo esc_attr( $this->get_field_name( 'selected_posts' ) ); ?>[]"
value="<?php echo $post->ID; ?>"
<?php checked( ( in_array( $post->ID, $selected_posts ) ) ? $post->ID : '', $post-
>ID ); ?> />
<?php echo get_the_title( $post->ID ); ?></li>

<?php } ?>
</ul>
</div>
<?php
}
```

La función get_posts() proporciona una matriz de mensajes basados en los parámetros especificados. En nuestro ejemplo, sólo recibimos las últimas veinte publicaciones en orden cronológico, pero podemos crear consultas más avanzadas estableciendo cualquier combinación de las variables de consulta disponibles (consulta el Codex para obtener una lista completa de query vars).

A continuación, un operador ternario comprueba si existe un valor válido de $instance['selected_posts'] de lo contrario, el valor de $selected_posts se establece a una matriz vacía. El ciclo foreach itera sobre los elementos de $posts, e imprime una casilla de verificación para cada uno. La función checked compara dos argumentos e imprime el atributo checked si son idénticos.

La siguiente imagen muestra el formulario de administración del widget actual.



Actualizando las Opciones de Widget

Las actualizaciones de método update() actualiza una instancia de un widget. Definimos la función como:

```
public function update( $new_instance, $old_instance ) {
```

```
$instance = array();
$instance['title'] = ( ! empty( $new_instance['title'] ) ) ? strip_tags(
$new_instance['title'] ) : '';

$selected_posts = ( ! empty ( $new_instance['selected_posts'] ) ) ? (array)
$new_instance['selected_posts'] : array();
$instance['selected_posts'] = array_map( 'sanitize_text_field', $selected_posts );

return $instance;
}
```

La función mantiene dos argumentos:

$new_instance es una matriz de configuraciones tal como fue presentada por el usuario
$old_instance es una matriz de configuraciones almacenadas en la base de datos

Esto es lo que sucede:

$instance[‘title’] almacena el nuevo valor del título del widget o una cadena vacía si no está disponible
$selected_posts almacena las IDs de las publicaciones seleccionadas por el usuario o una matriz vacía
$instance[‘selected_posts’] almacena una versión desinfectada de $selected_posts
la función devuelve $instance

Ahora el formulario del widget está en acción y las opciones del usuario se pueden guardar en la base de datos. Nuestra tarea final es mostrar la emisión en el sitio front-end.

La Salida de Widget

El método de widget() imprime el contenido del widget en el sitio front-end. La función obtiene las publicaciones seleccionadas y ejecuta un ciclo foreach que produce un elemento de lista para cada publicación. La función widget se define como sigue:

```
public function widget( $args, $instance ) {
```

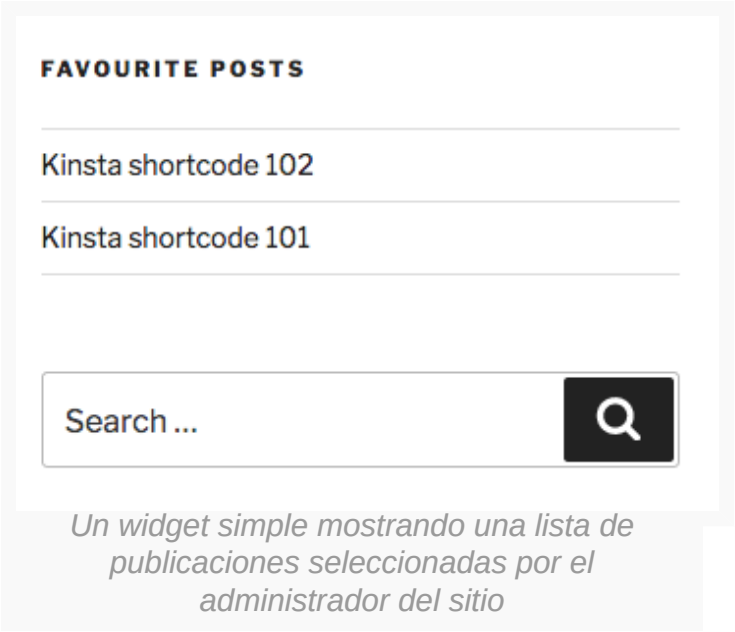
```
echo $args['before_widget'];
if ( ! empty( $instance['title'] ) ) {
echo $args['before_title'] . apply_filters( 'widget_title', $instance['title'] ) .
$args['after_title'];
}

if( ! empty( $instance['selected_posts'] ) && is_array( $instance['selected_posts']
) ){

$selected_posts = get_posts( array( 'post__in' => $instance['selected_posts'] ) );
?>
<ul>
<?php foreach ( $selected_posts as $post ) { ?>
<li><a href="<?php echo get_permalink( $post->ID ); ?>">
<?php echo $post->post_title; ?>
</a></li>
<?php } ?>
</ul>
<?php

}else{
echo esc_html__( 'No posts selected!', 'text_domain' );
}

echo $args['after_widget'];
}
```

La función widget() mantiene dos argumentos:

$args es una matriz de argumentos incluyendo ‘before_title’, ‘after_title’, ‘before_widget’, y ‘after_widget’
$instance es una matriz de configuraciones de widget

Si se encuentra un título válido, se desinfecta a través del filtro widget_title . Luego, si se ha seleccionado al menos una publicación, get_posts devuelve una matriz de objetos de publicación. Finalmente, el ciclo foreach construye un elemento de lista para cada publicación seleccionada. El código completo de este

plugin está disponible en Gist.



Un widget simple mostrando una lista de publicaciones seleccionadas por el administrador del sitio

Conclusión

Cualquier persona con un conocimiento básico de PHP y desarrollo WordPress puede aprender rápidamente cómo construir sus propios widgets WordPress, e incluso si usted no es un desarrollador de WordPress aún, la construcción de un widget podría darle la ocasión de un primer acercamiento para el desarrollo de plugins. ¿Está preparado para crear un widget de WordPress?