Wordpress Theming

Introducción

WordPress Theming se refiere al proceso de diseñar y desarrollar temas para sitios web basados en WordPress. Un tema de WordPress define la apariencia visual y el diseño de un sitio, incluyendo elementos como colores, tipografía, diseño de página y estilos. Los temas son esenciales para personalizar la apariencia de un sitio de WordPress y proporcionar una experiencia única para los usuarios.

El Wordpress Theming no sólo tiene que ver con el aspecto gráfico de la web, sino que además puede incorporar nuevas herramientas dentro de la administración de Wordpress, como puede ser la creación de custom post types.

Estructura de carpetas

Partiremos de la base de que ya tenemos instalado Wordpress en nuestro servidor local. Para ello, ya sabemos que debemos tener preparada una base de datos con el nombre que le queríamos dar a, para que Wordpress la use en la instalación de todas las tablas necesarias para su funcionamiento.

Una vez instalado Wp, podemos ver cual es la estructura de archivos y carpetas. Básicamente, tendremos un montón de archivos en el directorio raíz, del cual nos interesa uno de ellos: wp-config.php. En él vamos a encontrar un montón de información de nuestro site, así como algunas opciones que nos pueden resultar útiles.

El archivo wp-config.php es uno de los archivos más importantes en una instalación de WordPress. Contiene información de configuración crucial que conecta la base de datos con WordPress y define varios parámetros esenciales para el funcionamiento del sitio. Aquí hay información clave sobre el archivo wp-config.php:

Ubicación: El archivo wp-config.php se encuentra en el directorio raíz de tu instalación de WordPress. Si instalas WordPress manualmente, copiarás el archivo wp-config-sample.php y lo renombrarás a wp-config.php.

Base de Datos: La información de conexión a la base de datos es fundamental y se encuentra en este archivo. Incluye detalles como nombre de la base de datos, nombre de usuario de la base de datos, contraseña y servidor de la base de datos.

```
define('DB_NAME', 'nombre_de_la_base_de_datos');
define('DB_USER', 'nombre_de_usuario');
define('DB_PASSWORD', 'contraseña');
define('DB_HOST', 'localhost');
```

Salts y Claves de Autenticación: WordPress utiliza claves y sales para mejorar la seguridad de las contraseñas almacenadas en la base de datos. Puedes generar estas claves en el Generador de Claves de WordPress.

```
define('AUTH_KEY', 'pon_tu_frase_aqui');
define('SECURE_AUTH_KEY', 'otra_frase_aqui');
define('LOGGED_IN_KEY', 'otra_frase_aqui');
define('NONCE_KEY', 'otra_frase_aqui');
define('AUTH_SALT', 'otra_frase_aqui');
define('SECURE_AUTH_SALT', 'otra_frase_aqui');
define('LOGGED_IN_SALT', 'otra_frase_aqui');
```

define('NONCE_SALT', 'otra_frase_aqui');

Prefijo de Tabla de la Base de Datos: Por motivos de seguridad, es recomendable cambiar el prefijo de las tablas de la base de datos. Por defecto, WordPress utiliza "wp_" como prefijo. También podemos usar el prefijo de la base de datos de Wordpress para tener distintos Wp en una misma base de datos, con tablas que se van a diferenciar únicamente por el prefijo.

$table_prefix = 'wp_';

Modo de Depuración: El modo de depuración es útil durante el desarrollo para detectar errores. No se debe dejar activado en un entorno de producción. Si lo ponemos a true, podremos ver los errores que tengamos en nuestro código mientras estemos en desarrollo, antes de pasar la web a producción.

define('WP_DEBUG', false);

De todas las carpetas que crea Wp, la única que nos interesa es wp-content. Las demás contienen todos los archivos encargados de hacer que este CMS funcione correctamente, por lo que no debemos tocarlos.

Dentro de la carpeta wp-content, tenemos tres subcarpetas:
- **languages**: La carpeta "languages" en WordPress se utiliza para almacenar archivos de traducción que permiten localizar el contenido del sitio a diferentes idiomas. La internacionalización (i18n) y la localización (l10n) son procesos importantes para garantizar que un sitio web sea accesible y comprensible para usuarios de todo el mundo.

Dentro de la carpeta "languages", los archivos de traducción suelen tener extensiones .mo (Machine Object) y .po (Portable Object). Estos archivos contienen las traducciones de las cadenas de texto.

Los archivos .po son archivos de texto plano que contienen las cadenas de texto originales y sus traducciones. Los archivos .mo son versiones compiladas y optimizadas de los archivos .po que se utilizan realmente para la traducción en tiempo de ejecución.

Para habilitar la internacionalización en WordPress, debes configurar la constante WPLANG en el archivo wp-config.php. Esta constante apunta al archivo de traducción que se utilizará.

define('WPLANG', 'es_ES'); // Por ejemplo, para español (España).

Si no se establece WPLANG, WordPress utilizará el idioma predeterminado.

Los temas y complementos de WordPress pueden incluir sus propios archivos de traducción en sus directorios respectivos. Los archivos de traducción del núcleo de WordPress se almacenan en la carpeta "wp-content/languages".

- **plugins**: La carpeta "plugins" en WordPress es un directorio crucial que alberga todos los archivos de los plugins instalados en tu sitio. Los plugins son extensiones de software que añaden funcionalidades adicionales a tu sitio de WordPress.

Cada plugin tiene un archivo principal que WordPress utiliza para cargar el plugin. Este archivo suele tener el mismo nombre que el directorio del plugin con la extensión ".php". Por ejemplo, para un plugin llamado "mi-plugin", el archivo principal sería “mi-plugin.php".

Los plugins se activan y desactivan desde el panel de administración de WordPress. Cuando activas un plugin, WordPress carga el archivo principal del plugin y habilita su funcionalidad. Si desactivas un plugin, su funcionalidad se detiene, pero los archivos del plugin aún permanecen en la carpeta “plugins".

Los plugins pueden recibir actualizaciones periódicas, ya sea para añadir nuevas funciones, mejorar la seguridad o corregir errores. Puedes actualizar un plugin desde el panel de administración de WordPress o, en algunos casos, descargando la última versión desde el repositorio de plugins de WordPress.

El repositorio oficial de plugins de WordPress, accesible desde el panel de administración de WordPress o desde el sitio web de WordPress.org, es el lugar principal para buscar, descargar e instalar plugins. Aquí puedes encontrar miles de plugins gratuitos y premium.

Dado que los plugins pueden ejecutar código en tu sitio, es crucial instalar plugins solo desde fuentes confiables y mantenerlos actualizados. Los plugins obsoletos o inseguros pueden representar riesgos de seguridad.

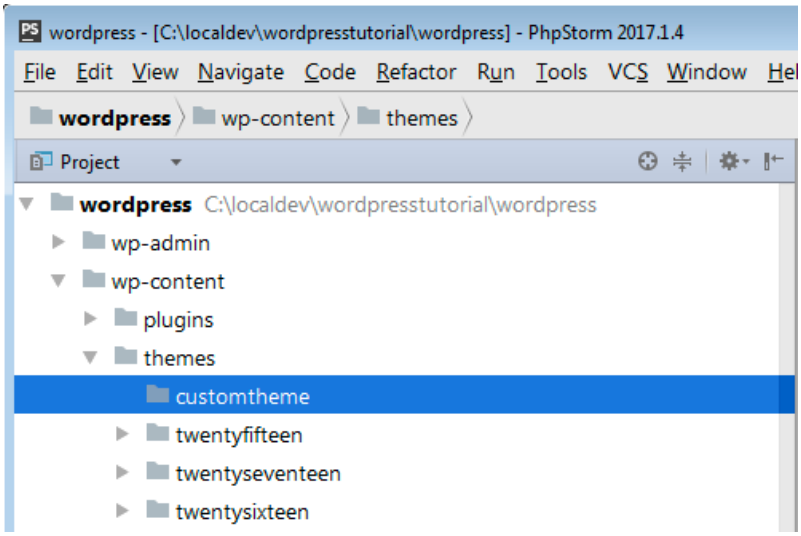
- **themes**: La carpeta "themes" en WordPress es un directorio central que alberga todos los temas instalados en tu sitio. Los temas son esenciales para la apariencia y el diseño de tu sitio, determinando cómo se presentan los contenidos y cómo interactúan los visitantes con la interfaz.

Cada tema debe tener su propio directorio dentro de la carpeta "themes". Dentro de este directorio, encontrarás archivos y subdirectorios que controlan la apariencia y funcionalidad del tema.

```
wp-content
└── themes
    ├── tema-1
    │   ├── index.php
    │   ├── style.css
    │   ├── screenshot.png
    │   └── ...
    ├── tema-2
    │   ├── index.php
    │   └── ...
    └── …
```

Ahora veremos en más detalle qué archivos debe contener la carpeta de nuestro tema.

Comencemos
Dentro de la carpeta wp-content, crearemos la carpeta de nuestro tema., por ejemplo customtheme:



En ella es donde vamos a almacenar todos los archivos referentes a nuestro tema personalizado. Lo primero que vamos a crear es el archivo style.css. Este archivo, además de contener lenguaje CSS, va a darle información a la administración sobre nuestro tema. Ninguno de los campos es obligatorio, pero si queremos tener una buena apariencia dentro de la administración, es altamente recomendable usarlos. También son necesarios si vamos a distribuir nuestro tema para terceros.

```
/*
Theme Name: My Custom Theme
Theme URI: https://yourwebsite.com/theme
Author: Your Name
Author URI: https://yourwebsite.com
Description: This is my first custom theme!
Version: 1.0.0
License: GNU General Public License v2 or later
License URI: <https://www.gnu.org/licenses/gpl-2.0.html>
Text Domain: my-custom-theme
Tags: custom-background
*/
```

Theme Name – Debes proporcionar un nombre de tema. Si no lo das, cogerá por defecto el nombre de la carpeta.

Theme URI – Si lo usas, la URI proporciona un link donde el usuario puede aprender más sobre el tema.

Author – Tu nombre.

Author URI – Un enlace a tu web personal o a la web de tu empresa.

Description – La descripción se muestra en la administración de Wordpress como una ventana modal, así como en el listado de WordPress theme.

Version – El número de versión ayuda a los desarrolladores para mantener de una forma Clara los cambios y permite a los usuarios saber si disponen de la última versión.

License – La forma de obtener la licencia de su tema depende de tí, pero si eliges una licencia no compatible con GPL, no podrás distribuir tu tema en WordPress.

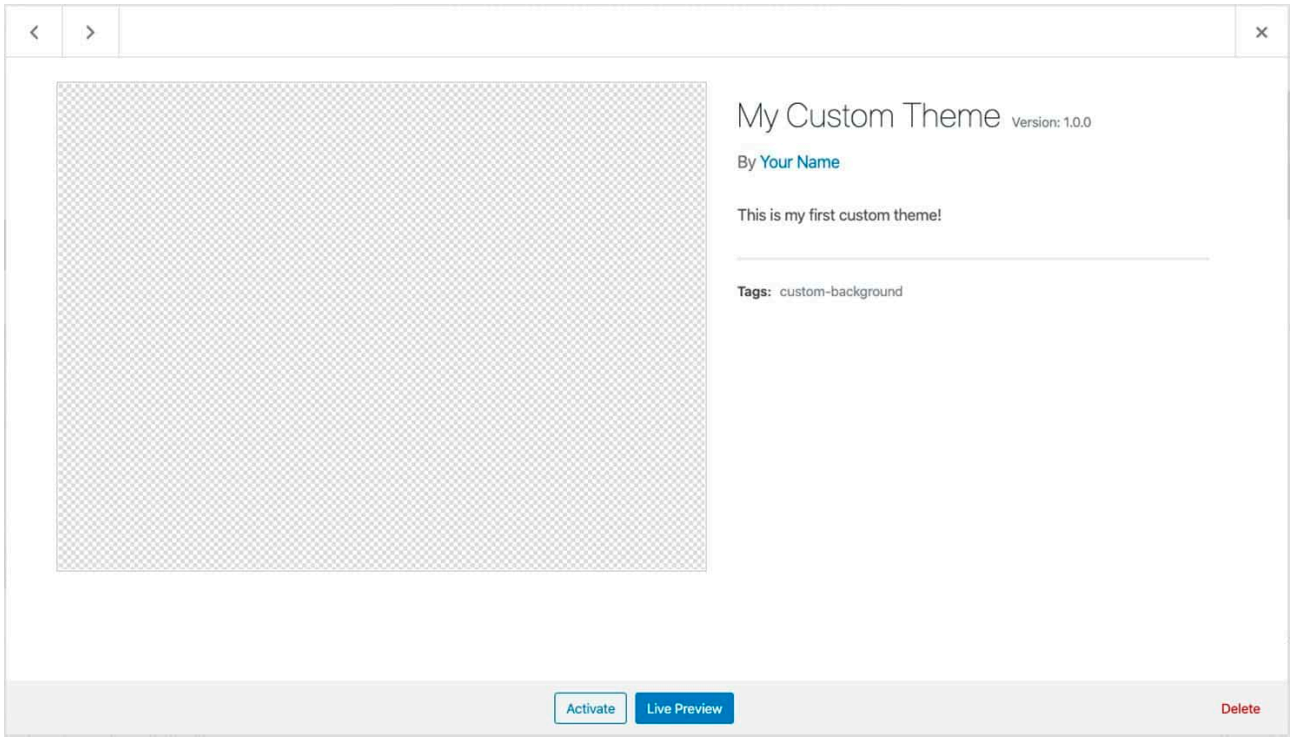
License URI – Este es simplemente un enlace a la licencia mencionada anteriormente.

Text Domain – El dominio de texto se utiliza al traducir tu tema a otros idiomas. No te preocupes, exploraremos esto en profundidad más adelante. Por ahora, basta con saber que es una buena práctica que la carpeta del tema y el dominio de texto sean el nombre del tema separado por guiones en lugar de espacios.

Tags – Las etiquetas solo se utilizan si carga su tema en el directorio de temas de WordPress.org. Son la base del mecanismo de "Filtro de funciones”.

Si creas el archivo style.css y activas el tema, podrás ver algo así en la administración:

Carpetas



Ahora crearemos un nuevo archivo, el índex.php. Éste es el único otro archivo estrictamente requerido. Su trabajo es renderizar toda la parte públical de nuestro tema. Dado que index.php presentará todas nuestras páginas (inicio, publicaciones, categorías, archivos), hará mucho trabajo. Para comenzar necesitamos una sección de encabezado que cubra los conceptos básicos de HTML.

```
<!DOCTYPE html>
<html <?php language_attributes(); ?>>
  <head>
    <meta charset="<?php bloginfo( 'charset' ); ?>">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="profile" href="https://gmpg.org/xfn/11">
    <?php wp_head(); ?>
  </head>
```

Este es HTML estándar con una excepción, [wp_head()](<https://developer.wordpress.org/reference/hooks/wp_head/>). wp_head es una función principal que permite a WordPress y a complementos de terceros insertar código en el encabezado sin modificar los archivos de plantilla. Esto se llama hook de acción. Imaginemos que vamos instalando en nuestro Wordpress plugins de terceros. Usualmente, estos plugins poseen su propia hoja de estilos y sus propios scripts en javascript, que serán necesarios para su buen funcionamiento. Pues wp-head es quien se encarga de cargar esos archivos sin que tengamos que preocuparnos por ellos.

language_attributes() es una función en WordPress que se utiliza para imprimir los atributos del idioma en el elemento <html> de una página. Estos atributos del idioma son esenciales para la internacionalización y la localización de un sitio, ya que ayudan a los navegadores y motores de búsqueda a identificar el idioma principal de la página. Esto es particularmente importante para la accesibilidad y la optimización de motores de búsqueda (SEO).

La función language_attributes() generalmente se encuentra en el archivo header.php del tema de WordPress y se utiliza dentro de la etiqueta <html> para imprimir los atributos del idioma. Aquí hay un ejemplo típico de cómo se usa:

```
<!DOCTYPE html>
<html <?php language_attributes(); ?>>
<head>
   <meta charset="<?php bloginfo( 'charset' ); ?>">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title><?php bloginfo( 'name' ); ?> | <?php bloginfo( 'description' ); ?></title>
   <?php wp_head(); ?>
</head>
<body <?php body_class(); ?>>
```

En este ejemplo:

<?php language_attributes(); ?>: Imprime los atributos del idioma en la etiqueta <html>. Esto incluirá el atributo lang con el código de idioma correspondiente, como "en" para inglés, "es" para español, etc.

<?php bloginfo('charset'); ?>: Imprime la codificación de caracteres utilizada en el sitio web

<?php bloginfo('name'); ?>: Imprime el nombre del sitio web

<?php bloginfo('description'); ?>: Imprime la descripción del sitio web

<?php wp_head(); ?>: Imprime una serie de etiquetas <head> adicionales que WordPress puede necesitar para incluir scripts, estilos y otros elementos dinámicos.

<?php body_class(); ?>: Imprime clases adicionales en el elemento <body>, que pueden ser útiles para estilos y scripts específicos. Wordpress añade clases al body en función del tipo de contenido que va a mostrar. Por ejemplo:

class="page page-id-2 page-parent page-template-default logged-in”

Nosotros también podemos añadir estilos personalizados al body, de esta manera:

<body <?php body_class('wide-template blue-bg'); ?>>

La función language_attributes() ayuda a asegurar que la información del idioma esté correctamente configurada en el HTML de tu sitio, lo que es crucial para garantizar una experiencia de usuario adecuada y facilitar la indexación por parte de los motores de búsqueda en diferentes idiomas.

El header
Lo siguiente será crear el header de nuestro site:

```
<header class="site-header">
        <p class="site-title">
                <a href="<?php echo esc_url( home_url( '/' ) ); ?>">
                        <?php bloginfo( 'name' ); ?>
                </a>
        </p>
        <p class="site-description"><?php bloginfo( 'description' ); ?></p>
</header>
<!-- .site-header —>
```

Aquí estamos utilizando las funciones de plantilla integradas de WordPress para generar el título y la descripción del sitio. También hemos utilizado una función auxiliar, home_url(), para vincular el título del sitio a la página de inicio.

La función home_url() nos devuelve el enlace a nuestro site, mientras que bloginfo('name') nos da el nombre que hayamos proporcionado al sistema para nuestra web.

La función bloginfo nos puede proporcionar mucha información sobre el sistema. Además del título del blog, bloginfo() se puede utilizar con diferentes argumentos para recuperar y mostrar otras piezas de información. Algunos ejemplos de argumentos comunes incluyen:

- **name**: El nombre del sitio.

- **description**: La descripción del sitio

- **url**: La URL del sitio.

- **admin_email**: La dirección de correo electrónico del administrador del sitio.

Nosotros podemos incluir esta cabecera en todos nuestros archivos o, sin embargo, podemos usar la cabecera que, por defecto, trae Wordpress. Esta cabecera se incluye usando get_header(). Esta función incluye una cabecera por defecto en nuestra página, o la que se haya codificado en el archivo header.php de nuestro theme. Por lo tanto, podemos ahorrarnos mucho código si creamos el archivo header.php, incluimos nuestro código en él y lo llamamos tras la apertura de la etiqueta <body>.

El loop
Ahora podemos hablar un poco sobre el WordPress Loop. WordPress Loop es realmente el motor que hace que WordPress funcione. Es a través de este bucle que los desarrolladores de temas buscan publicaciones y las muestran en la página según sea necesario.

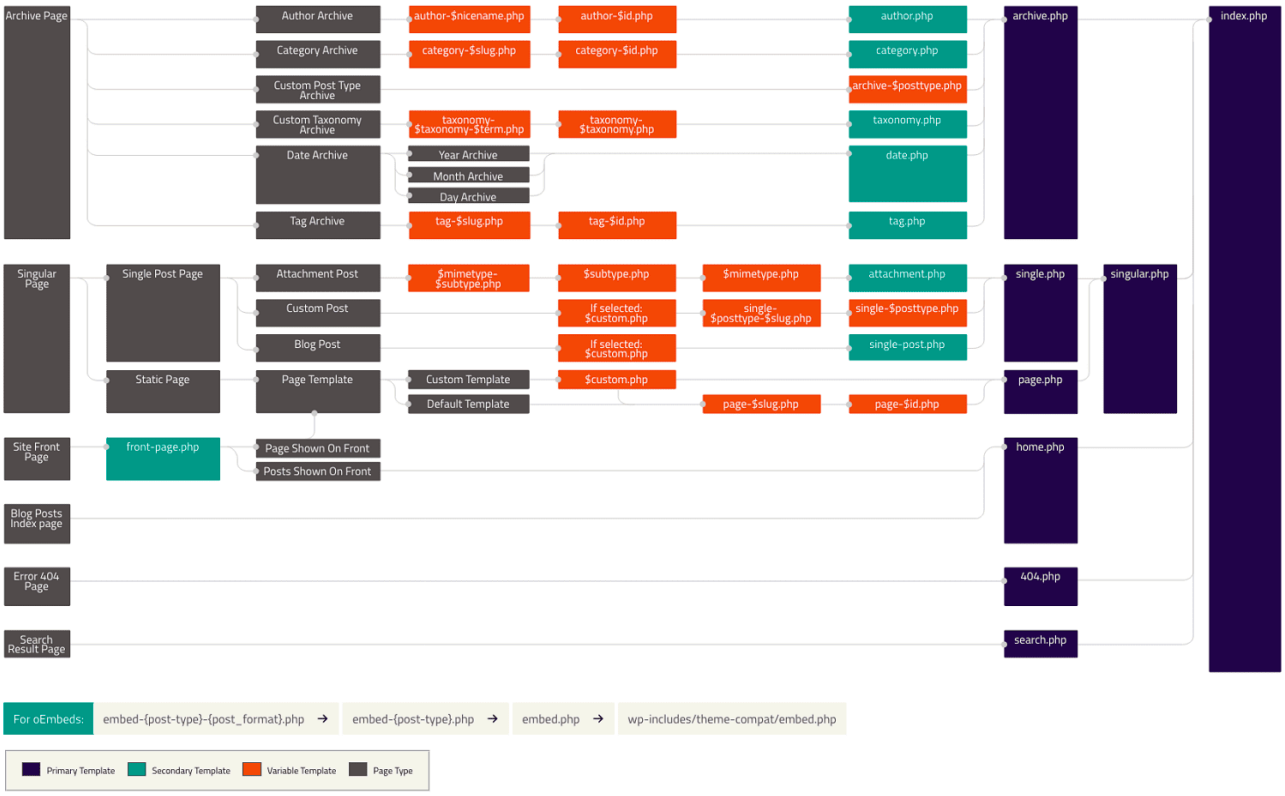
Si la base de datos tiene publicaciones, recorrámoslas mientras todavía haya publicaciones; de lo contrario, le informaremos al usuario que no hay publicaciones. Se ve así en código PHP.

```
<?php

if ( have_posts() ) :
	while ( have_posts() ) : the_post();

	php endwhile;

else :
	echo '<p>There are no posts!</p>';

endif;

?>
```

Tengamos en cuenta que The Loop utiliza dos funciones en su forma más básica. Esos son have_posts() y the_post(). La función have_posts() hace solo una cosa. Le indica si hay publicaciones en la base de datos para recorrer. Esta función devolverá verdadero o falso y eso es todo. Si devuelve verdadero, entonces hay publicaciones disponibles para recorrer. Si devuelve falso, entonces no hay publicaciones para recorrer.

La otra función, the_post() no devuelve nada. Su trabajo es preparar WordPress para publicar publicaciones. Específicamente, recupera la siguiente publicación, configura la publicación y establece la propiedad in_the_loop en verdadero. Hasta ahora, nuestra página todavía no mostrará ninguna información sobre las publicaciones de nuestro blog, pero podemos actualizarla ahora en nuestro archivo index.php.

La jerarquía de archivos en Wordpress



WordPress es una plataforma dinámica basada en Php. Cada sitio de WordPress carga múltiples archivos *.php*, cada uno de los cuales controla el aspecto de una sección o componente específico.
Cada vez que cargas un determinado tipo de página, el CMS busca los archivos de plantilla que le corresponden.
Por ejemplo, si utilizas la búsqueda y cargas una página de resultados de búsqueda, el CMS buscará dos archivos de plantilla:

1. *search.php*, que controla el aspecto de las páginas de resultados de búsqueda

2. *index.php*, que es el archivo de plantilla predeterminado que WordPress utiliza cuando no puede encontrar la opción superior dentro de cada jerarquía

Los archivos de plantilla a los que tienes acceso dependerán del tema que utilices. Todos los temas de Wordpress son una colección de plantillas, hojas de estilo y otros elementos, como imágenes. Así que en el ejemplo anterior, si el tema que utilizas incluye una plantilla *search.php*, WordPress la encontrará y la cargará.

En algunos casos, puedes usar un tema que no incluya archivos de plantilla para los tipos de páginas que quieres cargar. Ahí es donde entra la jerarquía de plantillas de WordPress. Es un sistema incorporado que le dice a WordPress qué archivos de plantilla debe cargar y en qué orden.

Para una página de búsqueda, si WordPress no puede encontrar el archivo *search.php*, pasará al siguiente archivo de la jerarquía, que es *index.php*. Este archivo es el último recurso para cada rama dentro de la jerarquía de plantillas.

En teoría, puedes tener un tema completamente funcional que solo incluye un archivo de plantilla, y que es *index.php*.

En la práctica, sin embargo, un tema con un solo archivo de plantilla apenas incluiría personalizaciones de estilo, y cada tipo de página se vería prácticamente igual. A menos que eso sea lo que busques, entender la jerarquía de plantillas de WordPress es uno de los pasos más importantes que puedes dar como desarrollador de temas.

Como ya sabrás, WordPress te permite usar múltiples tipos de páginas dependiendo de lo que quieras publicar. Hay siete categorías principales que puedes usar:

1. Página principal
2. Publicaciones individuales
3. Páginas individuales
4. Tipos de publicaciones personalizados
5. Páginas de resultados de la búsqueda
6. Páginas de categorías y etiquetas
7. Páginas de error 404

Cada una de esas páginas tiene su propia jerarquía personalizada, lo que significa que utiliza un conjunto específico de archivos de plantilla.

Si echas un vistazo rápido a cualquiera de las carpetas de tu tema, normalmente encontrarás una colección de archivos de plantilla.

Es importante comprender que, aunque cada tipo de página tiene su propia jerarquía, también suelen compartir archivos de plantilla comunes, como *header.php* y *footer.php*.

Si estás creando tu propio tema, eso significa que puedes crear estilos personalizados para cada tipo de página, a la vez que construyes archivos de plantillas para volver a utilizarlos.

También puedes crear archivos de plantillas personalizadas para elementos como barras laterales y pies de página que *solo* se aplican a ciertos tipos de páginas. Toda esta flexibilidad se debe al enfoque modular de las plantillas de WordPress.

Un Desglose de la Jerarquía de la Plantilla de WordPress por Tipo de Página

WordPress utiliza siete tipos principales de páginas. En esta sección, hablaremos de cada uno de ellos, y te proporcionaremos un desglose de los archivos de plantilla que utilizan. Empecemos con la página principal.

- Archivos de Plantillas de Página Principal
- Posts Individuales
- Páginas Individuales
- Tipos de Posts Personalizados
- Páginas de Resultados de Búsqueda
- Páginas de Categorías y Etiquetas
- Páginas de Error 404

Archivos de Plantillas de Página Principal

Tu página de inicio es la primera parada que la mayoría de los usuarios hacen cuando visitan tu sitio web. En la práctica, los diseños de la página principal pueden variar dramáticamente de un sitio a otro.

Sin embargo, para una portada básica, WordPress buscará estos tres archivos de plantillas en orden:

1. front-page.php
2. home.php
3. index.php

Si el tema no incluye un archivo de plantilla *front-page.php*, WordPress utilizará la segunda opción de forma predeterminada, y así sucesivamente. Como siempre, el archivo *index.php* es la última parada que hace WordPress en el árbol de decisión de plantillas.

Posts Individuales

Los artículos individuales de WordPress utilizan la jerarquía de plantillas de publicaciones individuales. Para el contenido principal de cada entrada del blog, WordPress buscará los siguientes archivos:

1. single.php
2. singular.php
3. index.php

Sin embargo, si miras la hoja de trucos de la plantilla, notarás que en algunos casos surge una jerarquía más compleja.

Esto se debe a que WordPress te permite designar archivos de plantillas para publicaciones individuales y categorías individuales, y luego se establece como predeterminado *single.php* si no puede encontrar ninguna de esas opciones.

Más allá de los archivos de plantilla primarios, tienes elementos como el encabezado, el pie de página, las barras laterales y las secciones de comentarios. Como mencionamos antes, cada uno de esos elementos puede tener su propio archivo de plantilla.

Páginas Individuales

Después de las publicaciones, las páginas individuales son el pan de cada día para la mayoría de lossitios web. En la mayoría de los casos, utilizan una plantilla diferente a la de tu página de inicio, a menos que ambos tipos de páginas estén predeterminadas en index.*php*.

Así es como se ve la jerarquía de las plantillas de páginas individuales:

1. page.php
2. singular.php
3. index.php

Aunque la jerarquía de la plantilla para las publicaciones y páginas individuales es similar, hay algunas diferencias clave.

En primer lugar, cuando se trata de páginas, WordPress utiliza la ruta get_page_templates() para los archivos personalizados, que puedes utilizar para cambiar o redirigir la plantilla de página predeterminada.

Además, puedes crear plantillas para ID y fichas específicas. Si tienes un archivo *page-{slug}.php* o *page-{id}.php*, WordPress tratará de cargar cada archivo en orden, antes de predeterminar el archivo page.*php*.

Tipos de Posts Personalizados

Puedes crear tipos de posts personalizados para varios tipos de contenido que tal vez no quieras combinar con páginas o posts. Los tipos de publicaciones personalizadas te proporcionan un mayor grado de organización para tu contenido y también tienen su propia jerarquía de plantillas:

1. archive-{post_type}.php
2. archive.php
3. index.php

La jerarquía de plantillas para los tipos de posts personalizados no es tan compleja como lo es para las páginas o posts completas. Sin embargo, WordPress te permite crear archivos de plantilla para cada tipo de post personalizado, de modo que no tengas que compartir exactamente los mismos diseños.

Páginas de Resultados de Búsqueda

Ya te hemos presentado la jerarquía de plantillas que WordPress utiliza para las páginas de resultados de búsqueda, así que recapitulemos brevemente cómo es:

1. search.php
2. index.php

A medida que nos alejamos de los tipos de páginas «complejas» como las publicaciones o la página principal, la jerarquía de plantillas de WordPress se vuelve mucho más sencilla.

En una página de búsqueda, normalmente no es necesario incluir demasiados elementos más allá de los resultados en sí. Cuanto más simple es la estructura, más corta suele ser la jerarquía.

Páginas de Categorías y Etiquetas

Aunque muchos sitios web no enlazan directamente con ellos, WordPress genera páginas colectivas para tus etiquetas y categorías. También hay sub páginas específicas para cada elemento dentro de esa taxonomía.

Considerando lo compleja que puede llegar a ser la taxonomía de los sitios web con grandes bibliotecas de contenido, esta jerarquía implica más «pasos» de lo habitual:

1. category–{slug}.php
2. category-{id}.php
3. category.php
4. archive.php
5. index.php

Nota que se utiliza la misma jerarquía para las etiquetas, excepto que «etiqueta» sustituye a «categoría» en todos los casos.

En teoría, puedes crear archivos de plantilla individuales para cada categoría o etiqueta en tu sitio web de WordPress, e identificarlos ya sea a través de fichas o ID. Sin embargo, pocos sitios web se toman esa molestia.

Si no planeas dejar que los visitantes naveguen por tu página de categorías, no dudes en usar la plantilla *archive.php* por defecto.

Páginas de Error 404

A veces, los visitantes intentarán acceder a una página que no existe. Cuando esto sucede, WordPress mostrará una página de error 404.

De forma predeterminada, WordPress no ofrece opciones para personalizar el aspecto de esta página. Sin embargo, puedes ajustar la apariencia de la misma tú mismo a través del archivo de la plantilla. La jerarquía de la plantilla es bastante corta:

1. 404.php
2. index.php

A medida que tu sitio web crezca, también lo harán las instancias en las que los usuarios puedan encontrarse con errores 404. Tener una página de errores personalizada para esas situaciones puede ayudarte a informar a los visitantes de por qué no se carga la página, orientarlos en otra dirección y reducir la frustración.

Al cargar un archivo de plantilla 404, WordPress buscará y cargará tu archivo personalizado antes de usar el predeterminado.

Volvamos al Loop

Ahora vamos a utilizar dos funciones adicionales de WordPress, the_title() y the_content(). La mayoría de las veces, estas funciones se usan dentro del bucle y lo que hacen es buscar el título y el contenido de cada publicación a medida que el bucle recorre cada una en la base de datos.

Entonces, a medida que se ejecuta el bucle, se encontrará con la primera publicación. En ese momento, la función the_title() enviará el título de la publicación a la página y the_content() enviará el cuerpo de esa publicación a la página. En el siguiente ciclo, estas funciones volverán a buscar el siguiente título y contenido y los enviarán a la página. Entonces, con esto en su lugar, ahora deberíamos ver información sobre el envío de nuestras publicaciones a la pantalla.

De esa manera, haríamos el siguiente cambio en el archivo índex.php

```
<?php

if ( have_posts() ) :
        while ( have_posts() ) : the_post(); ?>

      <h2><?php the_title() ?></h2>
              <?php the_content() ?>

        <?php endwhile;

else :
        echo '<p>There are no posts!</p>';

endif;

?>
```

¿Qué tal estarí vincular a cada publicación individual para que podamos ver una publicación por sí sola en lugar de solo como parte de la página de inicio? Podemos hacerlo con bastante facilidad, nuevamente con funciones especiales que proporciona WordPress. Para esta tarea, podemos hacer uso de la función the_permalink(). Podemos actualizar nuestro código así:

```
<?php

if ( have_posts() ) :
        while ( have_posts() ) : the_post(); ?>

      <h2><a href="<?php the_permalink() ?>"><?php the_title() ?></a></h2>
              <?php the_content() ?>

        <?php endwhile;
```

```
else :
        echo '<p>There are no posts!</p>';

endif;
?>
```

Ahora podemos hacer click en el título de cada una de las entradas y de esa manera poder verla individualmente.

Añadir header y footer

El título y el contenido de la publicación son fundamentales para crear un buen tema. Casi tan importante es tener una sección de encabezado y pie de página de tu tema. Estas secciones contendrían el contenido que siempre está visible en todas las páginas del sitio web.

Estas secciones están encima y debajo del contenido de la publicación. Para hacer esto, lo has adivinado, haremos uso de funciones especiales proporcionadas por WordPress para obtener el encabezado y el pie de página.

¿Ves un patrón que comienza a desarrollarse todavía? Casi todo lo que puede hacer como desarrollador de temas en WordPress ya se ha hecho mediante estas funciones personalizadas. Por lo tanto, descubrirás que vale la pena memorizar estas funciones comúnmente utilizadas en el desarrollo de temas de WordPress. Sigamos adelante y agreguemos las funciones get_header() y get_footer() a nuestro archivo de tema ahora.

```
<?php

get_header();

if ( have_posts() ) :
        while ( have_posts() ) : the_post(); ?>

      <h2><a href="<?php the_permalink() ?>"><?php the_title() ?></a></h2>
                <?php the_content() ?>

        <?php endwhile;

else :
        echo '<p>There are no posts!</p>';

endif;

get_footer();

?>
```

Podemos ver que nuestro tema personalizado ahora tiene un área de encabezado y un área de pie de página. En el encabezado está el nombre y el eslogan del sitio, mientras que en el pie de página vemos el texto familiar: Tutorial de WordPress funciona con WordPress. Estas son las opciones predeterminadas de encabezado y pie de página cuando se utilizan estas funciones. ¿Qué pasa cuando queremos tener un encabezado y pie de página personalizados?

De 2 a 4 archivos

Hasta ahora en este tutorial, tenemos dos archivos que se encuentran en nuestra carpeta de temas personalizados (que a su vez está en la carpeta de temas). Esos archivos son style.css e index.php. En este punto, necesitaremos agregar más archivos para avanzar. Continuemos y crea dos archivos nuevos en la carpeta de tema personalizado. Estos archivos se llamarán convenientemente header.php y footer.php.

Ahora, lo que harán estos archivos es sobrescribir los diseños de encabezado y pie de página predeterminados proporcionados de forma predeterminada cuando llamas a las funciones get_header() o get_footer(). De hecho, si actualizamos nuestro sitio web, parece que el encabezado y el pie de página han desaparecido. Esto se debe a que aún no hemos agregado ningún marcado a esos archivos. Solo para sonreír, configure los archivos así para probar esto.

header.php

```
<h2>The Header!</h2>
<hr>
```

footer.php

```
<hr>
<h2>The Footer!</h2>
```

Trabajando con el header

Nuestro ejemplo anterior funcionó muy bien y nos muestra cómo funciona este archivo en su nivel más básico. Sin embargo, el archivo header.php es bastante importante, ¡así que no pasemos por alto sus detalles demasiado rápido! Aquí es donde incluye el código al que todas las páginas de su sitio necesitarán acceder de una forma u otra.

Para empezar, todas las páginas HTML tendrán un tipo de documento. Lo especificarías en este archivo. Además, todas las páginas tendrán una etiqueta html de apertura, una sección de encabezado y una etiqueta de cuerpo de apertura. Todo esto puede ir en el archivo header.php.

Agreguemos rápidamente algunas de estas cosas que todas las páginas web utilizarían. Aquí también utilizaremos algunas funciones nuevas de WordPress. Esos serán language_attributes(), bloginfo() y body_class().

header.php

```
<!DOCTYPE html>
<html <?php language_attributes(); ?>>

<head>
    <meta charset="<?php bloginfo( 'charset' ); ?>">
    <title><?php bloginfo( 'name' ); ?></title>
</head>

<body <?php body_class(); ?>>

<header class="site-header">
    <h1><?php bloginfo( 'name' ); ?></h1>
    <h4><?php bloginfo( 'description' ); ?></h4>
</header>
```

Si recargamos la página y luego vemos el código fuente de la página en nuestro navegador, podemos tener una idea de qué están haciendo estas funciones.

Nuevamente, podemos ver el uso muy liberal de las funciones de WordPress al desarrollar sus propios temas en WordPress. De hecho, todavía no hemos escrito ningún código personalizado. Simplemente estamos aprendiendo lo que nos pueden ofrecer las diversas funciones de WordPress y luego las ponemos a trabajar en nuestro tema personalizado.

Incluyendo wp_head()

wp_head() es una especie de función especial cuando trabajas con temas de WordPress. No es tan simple como todos los demás que hemos visto hasta ahora. El propósito de esta función es

finalizar la salida en la sección <head> de su archivo header.php. De hecho, está destinado a ir justo antes de la etiqueta </head> de cierre.

Esto se vuelve importante cuando comienzas a agregar varios complementos a tu sitio. Imprime scripts o datos en la etiqueta principal en la parte frontal. Es una buena práctica incluir siempre wp_head() en tus temas, ya que muchos otros complementos pueden depender de este enlace

para agregar estilos, scripts o metaelementos en el área <head> del sitio. Lo agregaremos como tal aquí:

header.php

```
<!DOCTYPE html>
<html <?php language_attributes(); ?>>

<head>
   <meta charset="<?php bloginfo( 'charset' ); ?>">
   <title><?php bloginfo( 'name' ); ?></title>
         <?php wp_head() ?>
</head>

<body <?php body_class(); ?>>

<header class="site-header">
   <h1><?php bloginfo( 'name' ); ?></h1>
   <h4><?php bloginfo( 'description' ); ?></h4>
</header>
```

Completando el footer

Hemos terminado de agregar los conceptos básicos de lo que necesitaremos en el archivo header.php. Sigamos ahora y completemos el archivo footer.php. Son algunas cosas que debemos hacer. Recuerda que en nuestro archivo header.php tenemos etiquetas de apertura html y body. Es necesario cerrarlos en algún momento. El archivo footer.php es un lugar perfecto para hacerlo. Entonces agregaremos etiquetas de cierre </html> y </body> además de realizar una llamada a la función wp_footer().

footer.php

```
<footer class="site-footer">
   <p><?php bloginfo( 'name' ) ?></p>
</footer>

<?php wp_footer() ?>
</body>
</html>
```

Quizás te preguntes por qué tuvimos que utilizar todas estas funciones sofisticadas para desarrollar nuestro tema. Por ejemplo, cuando queríamos enumerar el nombre y el eslogan de nuestro sitio, utilizamos la función bloginfo() pasando parámetros de nombre y descripción. La razón de esto es que, en general, nunca desea codificar estos valores en su sitio. Esta es información que podría cambiar. Además, si pone su tema a disposición del público, tendrán su propio nombre y eslogan para su sitio web. Deberían poder simplemente visitar el panel de administración de WordPress y actualizar su Configuración general para ver que estos datos se completan automáticamente.

Hacer que el título del sitio enlace a la página de inicio

La mayoría de los temas ofrecerán la posibilidad de hacer clic en el texto del título del sitio web y dirigir al usuario a la página de inicio del sitio. De esta manera, no importa dónde se encuentre el

usuario en el sitio, siempre puede hacer clic en el texto del título y volver directamente a la página principal del sitio web. Agreguemos ese enlace ahora en header.php.

```
<!DOCTYPE html>
<html <?php language_attributes(); ?>>

<head>
   <meta charset="<?php bloginfo( 'charset' ); ?>">
   <title><?php bloginfo( 'name' ); ?></title>
        <?php wp_head() ?>
</head>

<body <?php body_class(); ?>>
<header class="site-header">
   <h1><a href="<?php echo home_url(); ?>"><?php bloginfo( 'name' ); ?></a></h1>
   <h4><?php bloginfo( 'description' ); ?></h4>
</header>
```

Agregar un archivo functions.php al tema

En este punto, tenemos cuatro archivos en nuestro tema personalizado. Esos son index.php, style.css, header.php y footer.php. Probablemente el siguiente archivo más importante que debemos tener es el archivo functions.php.

El archivo functions.php en WordPress hace muchas cosas por tu tema. Es el archivo donde colocas el código para modificar el comportamiento predeterminado de WordPress. Casi puedes pensar en functions.php como una forma de complemento para WordPress con algunos puntos clave para recordar:

- No requiere texto de encabezado único

- Almacenado en la carpeta que contiene los archivos de su tema.

- Se ejecuta solo cuando está en el directorio del tema actualmente activado.

- Se aplica sólo al tema actual.

Puede llamar a funciones PHP, funciones de WordPress o funciones personalizadas ¡Una cosa que necesitamos urgentemente en nuestro tema es un mejor estilo!

Creemos una función en nuestro archivo funciones.php para incluir el archivo style.css en nuestro tema. Así es como podemos lograr ese objetivo.

```
<?php

function custom_theme_assets() {
        wp_enqueue_style( 'style', get_stylesheet_uri() );
}

add_action( 'wp_enqueue_scripts', 'custom_theme_assets' );
```

Este fragmento de código incluirá, o activará, la hoja de estilo de nuestro tema personalizado. Ahora quizás te preguntes por qué estamos usando una función personalizada, cuando parece que podríamos vincularnos manualmente a la hoja de estilo con la misma facilidad en el archivo header.php. Bueno, esto se reduce a trabajar un poco más desde el principio para obtener un mayor retorno de tu esfuerzo más adelante. A medida que los temas se vuelven más complejos y se agregan más recursos, estarás feliz de tener esta función que puede manejar todo el trabajo pesado por ti.

Ahora es el momento de hacer que las cosas luzcan un poco más bonitas. Primero, agreguemos un envoltorio con una clase de contenedor. El de apertura estará en header.php, mientras que el de cierre estará en footer.php. También envolveremos la salida de la

publicación en index.php con una etiqueta <article> que tiene una clase de publicación. Esto nos dará clases a las que apuntar en nuestro archivo style.css para que podamos establecer el ancho de la página, entre otras cosas. También agregaremos un estilo mejor a style.css en este paso.

header.php
Añadimos el div en header

```
<!DOCTYPE html>
<html <?php language_attributes(); ?>>

<head>
   <meta charset="<?php bloginfo( 'charset' ); ?>">
   <title><?php bloginfo( 'name' ); ?></title>
         <?php wp_head() ?>
</head>

<body <?php body_class(); ?>>
<div class="container">
   <header class="site-header">
      <h1><a href="<?php echo home_url(); ?>"><?php bloginfo( 'name' ); ?></a></h1>
      <h4><?php bloginfo( 'description' ); ?></h4>
   </header>
```

footer.php
Añadimos el cierre del div en el footer

```
<footer class="site-footer">
   <p><?php bloginfo( 'name' ) ?></p>
</footer>
</div> <!-- closes <div class=container"> -->

<?php wp_footer() ?>
</body>
</html>
```

index.php
Encerramos cada por en un article

```
<?php

get_header();

if ( have_posts() ) :
         while ( have_posts() ) : the_post(); ?>

      <article class="post">
         <h2><a href="<?php the_permalink() ?>"><?php the_title() ?></a></h2>
                        <?php the_content() ?>
      </article>

         <?php endwhile;

else :
         echo '<p>There are no posts!</p>';
```

```
endif;

get_footer();
?>
```

style.css

```
/*
Theme Name: customtheme
Author: Vegibit
Author URI: https://vegibit.com
Version: 1.0
 */

body {
    font-family: Arial, sans-serif;
    font-size: 16px;
    color: #545454;
}

a:link, a:visited {
    color: #4285f4;
}

p {
    line-height: 1.7em;
}

div.container {
    max-width: 960px;
    margin: 0 auto;
}

article.post {
    border-bottom: 4px dashed #ecf0f1;
}

article.post:last-of-type {
    border-bottom: none;
}

.site-header {
    border-bottom: 3px solid #ecf0f1;
}

.site-footer {
    border-top: 3px solid #ecf0f1;
}
```

Añadir estilo o javascript personalizado en un tema

Ya que tenemos nuestro archivo functions.php creado, vamos a usarlo para importar hojas de estilo o archivos javascript para usarlos en nuestro tema. Obviamente, se pueden incluir dentro de nuestra índex.php, pero conforme vayamos aumentando el número de páginas en nuestro proyecto, tendremos que añadirlo en más archivos. Usar el archivo de funciones será la mejor manera de hacerlo.

```
<?php
/* Enqueues the external CSS file */
add_action( 'wp_enqueue_scripts', 'load_external_styles' );

function load_external_styles() {
   wp_register_style( 'widget-css',     get_stylesheet_directory_uri().'/assets/widget.css' );
         wp_enqueue_style( 'widget-css' );
}
```

En primer lugar, tenemos la función load_external_styles. En ella, se usa la función ese Wordpress wp_register_style(). wp_register_style() es una función de WordPress que se utiliza para registrar un estilo (CSS) en el sistema, lo que permite que pueda ser encolado (enqueue) en las páginas del sitio. Esto es útil cuando deseas organizar y gestionar estilos de manera más controlada en lugar de incluirlos directamente en el encabezado de la página.

Su sintaxis es
wp_register_style('estilo-personalizado', get_template_directory_uri() . '/estilo-personalizado.css', array(), '1.0', ‘all')

- 'estilo-personalizado' es el nombre identificativo del estilo. Puedes elegir un nombre significativo para tu estilo.

- get_template_directory_uri() . '/estilo-personalizado.css' especifica la URL del archivo CSS. Asegúrate de crear ese archivo y guardarlo en la carpeta de tu tema.

- array() es un array de dependencias. Puedes especificar otros estilos que deben cargarse antes de este.

- '1.0' es la versión del estilo.

- 'all' especifica en qué tipo de medios se aplicará el estilo. En este caso, "all" significa que se aplicará en todos los tipos de medios.

Después usamos wp_enqueue_style(), que lo que hace que es que lanza esa hoja de estilo para que sea usada.

Una vez definida la función load_external_styles(), usaremos un hook o gancho para llamarlo:

add_action('wp_enqueue_scripts', 'load_external_styles');

Esta línea de código utiliza la función add_action() para enganchar la acción wp_enqueue_scripts y ejecutar la función load_external_styles cuando esa acción se dispare. Vamos a desglosar lo que hace esta línea de código:

- **add_action**: Esta función de WordPress se utiliza para "enganchar" (hook) una función o método específico a una acción específica. En este caso, la acción es wp_enqueue_scripts, que se dispara durante la carga de scripts y estilos en el frontend del sitio.

- '**wp_enqueue_scripts**': Es el nombre de la acción a la que estás enganchando tu función. En este caso, wp_enqueue_scripts se utiliza cuando se están encolando scripts y estilos en el frontend del sitio.

- '**load_external_styles**': Es el nombre de la función que se ejecutará cuando se dispare la acción. En este caso, se espera que haya una función llamada load_external_styles definida en tu código.

En resumen, esta línea de código establece que cuando WordPress esté en la etapa de encolar scripts y estilos en el frontend, se debe ejecutar la función load_external_styles. Esto es

típicamente utilizado para cargar estilos externos (por ejemplo, hojas de estilo de fuentes o bibliotecas) en tu sitio WordPress.

En el caso de que queramos añadir un archivo js a nuestro proyecto, lo haremos de la siguiente manera:

```
function load_external_scripts() {
   // Enqueue scripts externos
        wp_enqueue_script('nombre_script',  'URL_del_script_externo',  array('dependencia_script'), 'version_del_script', true);
}

add_action('wp_enqueue_scripts', ‘load_external_scripts’);
```

El último parámetro puede ser true o false, y significa si queremos que el archivo js sea incluido o no en el footer (por defecto es false). Es una buena práctica incluir los archivos js en el footer, de manera que sea lo último que se lea del contenido de la página.

Ahora, un sidebar

En WordPress, una "sidebar" (barra lateral) es un área en el diseño de tu sitio donde puedes colocar widgets, que son bloques funcionales que realizan tareas específicas. Las barras laterales son comunes en muchos temas de WordPress y proporcionan una manera de mostrar contenido adicional o funciones específicas en páginas o entradas.

Veamos como se incluye una sidebar en nuestro tema:

```
// Registrar la barra lateral
function registrar_sidebar() {
   register_sidebar(array(
      'name'          => __('Barra lateral', 'textodominio'),
      'id'            => 'barra-lateral',
      'description'   => __('Esta es la barra lateral principal.', 'textodominio'),
      'before_widget' => '<div id="%1$s" class="widget %2$s">',
      'after_widget'  => '</div>',
      'before_title'  => '<h2 class="widget-title">',
      'after_title'   => '</h2>',
   ));
}

// Enganchar la función al hook 'widgets_init'
add_action('widgets_init', ‘registrar_sidebar');
```

Los parámetros se pasan como un array asociativo ($args). Aquí están los parámetros más comunes:

- **name** (cadena): Especifica el nombre de la barra lateral que será mostrado en el área de administración. Por ejemplo: 'name' => __('Barra lateral principal', ‘textodominio').

- **id** (cadena): Especifica un identificador único para la barra lateral. Este identificador se utilizará para referirse a la barra lateral en otras partes del código. Por ejemplo: 'id' => ‘barra-lateral'.

- **description** (cadena): Proporciona una descripción opcional de la barra lateral. Esto puede ser útil para proporcionar información adicional sobre el propósito de la barra lateral. Por ejemplo: 'description' => __('Esta es la barra lateral principal.', ‘textodominio').

- **before_widget** (cadena): Define el HTML que se imprimirá antes de cada widget en la barra lateral. Puedes utilizar %1$s para el ID del widget y %2$s para las clases del widget. Por ejemplo: 'before_widget' => '<div id="%1$s" class="widget %2$s”>'.

- **after_widget** (cadena): Define el HTML que se imprimirá después de cada widget en la barra lateral. Por ejemplo: 'after_widget' => ‘ '.

- **before_title** (cadena): Define el HTML que se imprimirá antes del título de cada widget en la barra lateral. Por ejemplo: 'before_title' => '<h2 class=“widget-title">'.

- **after_title** (cadena): Define el HTML que se imprimirá después del título de cada widget en la barra lateral. Por ejemplo: 'after_title' => ‘</h2>'.

- **class** (cadena): Agrega clases adicionales a la barra lateral. Esto puede ser útil para aplicar estilos específicos. Por ejemplo: 'class' => ‘mi-clase-adicional'.

- **before_sidebar** (cadena): HTML que se imprimirá antes de la barra lateral. Útil para envolver la barra lateral en algún contenedor adicional. Por ejemplo: 'before_sidebar' => '<div class=“mi-contenedor">'.

- **after_sidebar** (cadena): HTML que se imprimirá después de la barra lateral. Útil para envolver la barra lateral en algún contenedor adicional. Por ejemplo: 'after_sidebar' => ‘ ’.

Una vez creada la función que va a registrar la sidebar, la llamamos mediante un hook:

add_action('widgets_init', ‘registrar_sidebar’);

Ese hook se lanza cuando se inicializan los widgets de nuestro site. Posteriormente incluimos el sidebar en nuestro archivo:

```
<?php if ( is_active_sidebar( "barra-lateral' ) ) : ?>
        <div id="widget-area" class="widget-area">
          <?php dynamic_sidebar( ‘'barra-lateral' ); ?>
        </div>
<?php endif; ?>
```

Otra manera de incluir la sidebar en nuestras páginas es crear un archivo sidebar.php en el que pegaremos el código anterior, y lo llamaremos en nuestros archivos usando:

```
<?php get_sidebar(); ?>
```

¿Por qué no un menú?

Evidentemente, podemos incluir un menù, programando directamente en HTML en el archivo header.php. Pero de esa manera estaríamos limitando en gran medida la interactividad y la experiencia de usuario, que puede crear de forma dinámica todos los menús que desee dentro de la administración de Wordpress. Por ello, vamos a usar las funciones que nos proporciona el sistema para incluir un menú en nuestro site.

En realidad WordPress nos permite añadir todos los menús que queramos, lo que podemos echar en falta de un tema no es realmente la posibilidad de añadir un menú, sino la posibilidad de añadirlo en una posición concreta. Es por ello que lo correcto no sería decir que vamos a añadir un menú a nuestro tema, sino que lo que en realidad vamos a añadir es una posición donde colocar un menú.

Esta operación es más sencilla de lo que pudierais pensar, por lo que si realmente os gusta un tema al que le falta una posición de menú, no deberíais de descartarlo y deberíais de intentar añadirsela, Obviamente, antes de hacer nada de lo que no estemos muy seguros conviene sacar copia de seguridad por si acaso.

Lo primero que hemos de hacer es registrar la posición del menú: esto lo haremos en el archivo **functions.php** de nuestro tema.

```
add_action( 'after_setup_theme', 'register_new_menus' );
function register_new_menus() {
  register_nav_menu( 'menu-footer', __('Menú pie de página'));
}
```

A esta nueva posición le tenemos que dar un nombre, en este caso he usado ‘menu-footer’ pero podéis ponerle otro nombre. Tenéis que tener cuidado al elegir el nombre; lo ideal es usar una única palabra (si queréis varias podéis usar un guión – para separarlas). Después, le damos una descripción, en mi caso le he puesto ‘Menú pie de página’, pero podéis ponerle la que queráis.

Una vez registrada la posición del menú, debemos de indicar en que lugar o lugares de nuestro tema se debe de mostrar. En mi caso vamos a colocar el menú en el pie de página, por lo que el archivo que debo de modificar es el archivo footer.php.

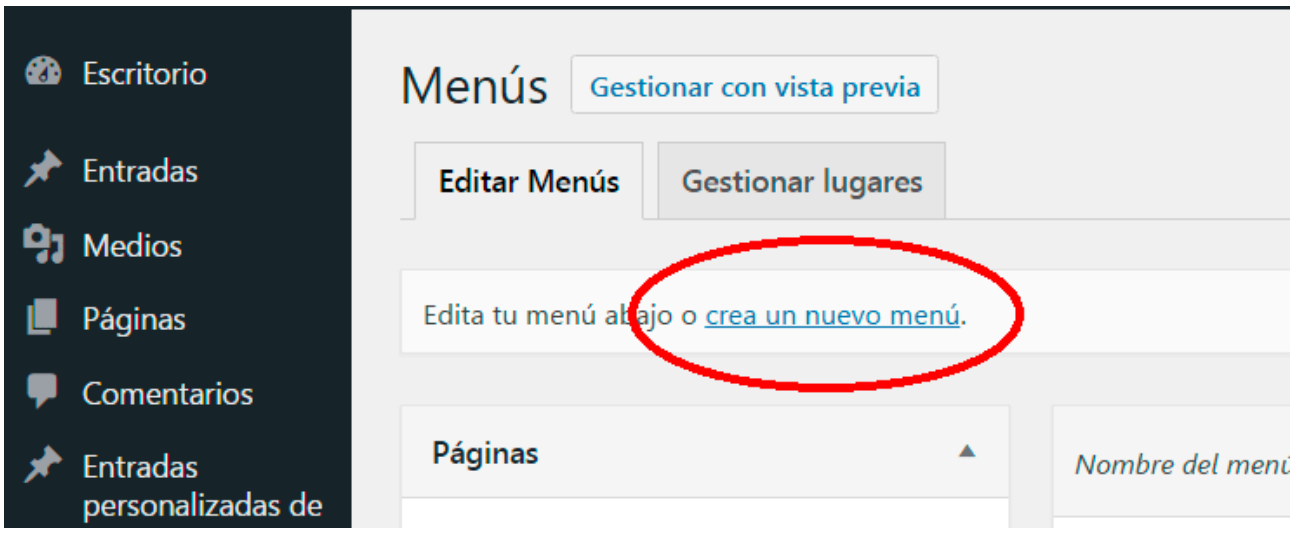
Cada tema es diferente, por lo que tendréis que familiarizaros con la estructura de archivos de vuestro tema para poder determinar el sitio correcto donde colocar vuestro menú. Para más pistas sobre que archivo debéis de modificar, podéis consultar la jerarquía de archivos de WordPress. Este paso es probablemente el mas complicado de todo el proceso, por lo que tendréis que tomaros vuestro tiempo comprendiendo los diferentes archivos que componen vuestro tema.

Una vez localizado el sitio de nuestro tema donde queremos colocar nuestro menú, tan solo tenemos que añadir el siguiente código

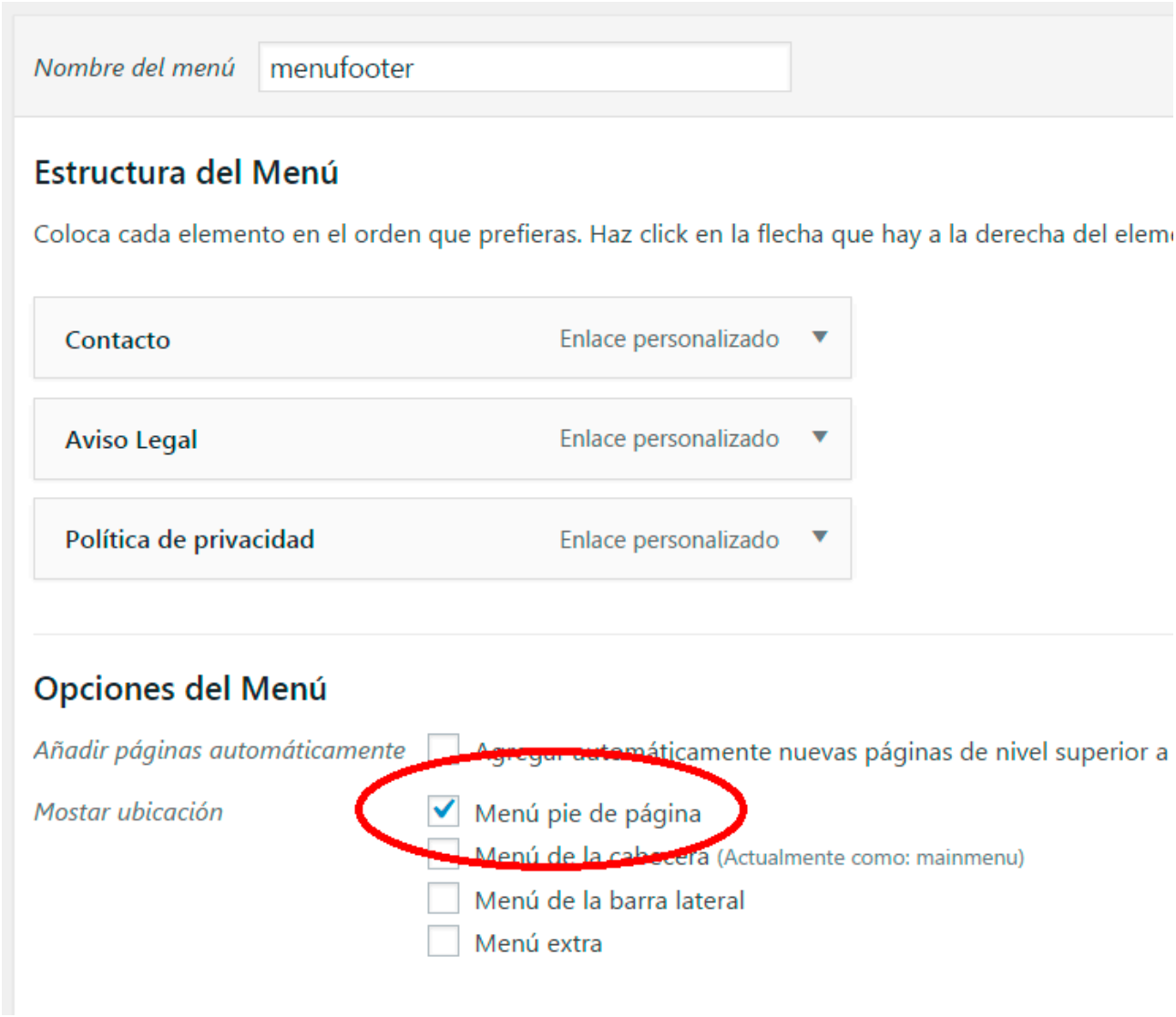
```
<?php
wp_nav_menu(
        array(
                'theme_location' => ‘menu-footer',
                 'menu_class' => 'menu_footer_class'
        )
);
?>
```

De este forma, mediante la función de WordPress wp_nav_menu, indicamos el sitio donde queremos que WordPress «pinte» nuestro menú. Hemos de asegurarnos de que indicamos el mismo nombre del menú que hemos registrado antes (menu-footer en mi caso). Además, le estoy indicando el nombre de una clase de forma que después pueda usar dicha clase CSS para indicarle al navegador el aspecto visual del menú.

Una vez hechos todos estos pasos ya podemos crear un nuevo menú para añadirlo a dicha posición.



Una vez creado el nuevo menú, podemos indicar en que posición (o posiciones) queremos mostrarlo. Como ya hemos creado la nueva posición para el pie de página, tan solo tenemos que seleccionarla para que el menú aparezca en ella.



Formulario de búsqueda y página de resultados

Es totalmente habitual que en la web tengamos un formulario de búsqueda, en el que escribiremos la palabra o palabras a buscar, y que nos llevará a una página con los resultados de nuestra búsqueda. Puedes seguir estos sencillos pasos:

Modificar el archivo header.php o el archivo donde desees mostrar el formulario de búsqueda:
Abre el archivo header.php o el archivo donde desees que aparezca el formulario de búsqueda en tu tema. Agrega el siguiente código donde quieras que aparezca el formulario:

```
<div class="search-form-container">
    <?php get_search_form(); ?>
</div>
```

El código utiliza get_search_form() para obtener y mostrar el formulario de búsqueda.

Opcional: Personalizar el estilo del formulario de búsqueda: Puedes agregar estilos personalizados para el formulario de búsqueda. A continuación, un ejemplo básico de CSS:

```
.search-form-container {
    margin: 10px;
}

.search-form {
    display: flex;
}

.search-form input[type="text"] {
    padding: 5px;
    width: 200px;
}

.search-form input[type="submit"] {
    padding: 5px;
    background-color: #0073e5;
    color: #fff;
    cursor: pointer;
}
```

También podemos hacerlo creando el archivo searchform.php, que será el que se cargue al llamar a get_search_form (por defecto se usa el del sistema).

```
<form id="searchform" method="get" action="<?php echo esc_url( home_url( '/' ) ); ?>">
        <input type="text" class="search-field" name="s" placeholder="Search" value="<?php echo
get_search_query(); ?>">
        <input type="submit" value="Search">
</form>
```

Crear archivo search.php
Lo primero que debes comprobar es que archivo utiliza tu tema para mostrar los resultados de búsqueda. Abre la carpeta de tu tema: wp-content/themes/tema-activo y comprueba si existe el archivo search.php. Si no lo encuentras tu tema está utilizando index.php o page.php, en este post te explico un método rápido para saber que archivo se está ejecutando en cada página.

Si el archivo existe puedes abrirlo y pasar al apartado de edición de la página search.php
En caso de que tu tema no cuente con este archivo, vamos a crearlo. Para hacerlo fácil abre el archivo index o page.php (el que utilice tu tema para mostrar resultados), copia el código en un nuevo archivo php y lo guardas dentro de la carpeta del tema con el nombre de: search.php

Edición de la página search.php

Antes de editar y modificar el archivo comprueba que tenga una estructura ***similar** a la siguiente:
*depende de tu tema pero en general debes tener; llamada al head, cuerpo de página con el loop, llamada a título y contenido del post y llamada al footer o pie de página.

```
<!--llamada a la cabecera; logo, menú...-->
<?php
get_header();
?>
<!--contenido principal-->
<div id=“main-content">
        <div class=“container">
                <div id="content-area" class="clearfix">
                        <div id="left-area">
                        <!--el loop-->
                        <?php while ( have_posts() ) : the_post(); ?>
```

```
                        <!-- recopilando info de cada post -->
                                <article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>
                                        <!-- título del post -->
                                        <h1 class="entry-title main_title"><?php the_title(); ?></h1>
                                        <!-- contenido del post -->
                                        <div class="entry-content">
                                        <?php
                                                the_content();
                                        ?>
                                        </div> <!-- fin contenido post -->
                                </article> <!-- fin info de cada post -->
                        <?php endwhile; ?><!-- fin del loop -->
                        </div> <!-- termina contenido derecha -->

                        <?php get_sidebar(); ?><!-- barra lateral -->
                </div> <!-- fin de div contentarea -->
        </div> <!-- fin de div container -->
</div> <!-- fin de div contenido principal -->

<?php get_footer(); ?><!-- llamada al pie de página —>
```

1ª modificación: que muestre un extracto del post en lugar de todo el contenido

La primera variación que vamos a realizar en la página de resultados es cambiar la llamada al contenido del post para que **muestre sólo un extracto de cada artículo**. No nos interesa que se muestre todo el contenido de cada post, piensa como sería una página con muchos resultados de búsqueda, creo que terrible para nuestro usuario.

Si tu página ya llama a la función the_excerpt(), no necesitas hacer nada.

Vamos al código de nuestra página, y donde veamos la función:

```
<?php
  the_content();
?>
```

la cambiamos por:

```
<?php
  the_excerpt();
?>
<a href="<?php the_permalink();?>">Leer más…</a>
```

Los extractos pueden recoger el contenido del principio del post o del campo extracto de la ventana de edición del post.

2ª modificación: cabecera de página con todas las categorías de nuestro blog

Vamos a añadir a nuestra página de resultados de búsqueda un título general tipo: «Resultados de su búsqueda:» y una lista horizontal de todas las categorías del blog con enlaces a los post según categorías.

Nos situamos **antes** del comienzo del loop o bucle:

```
<!--el loop-->
<?php while ( have_posts() ) : the_post(); ?>
```

y copiamos el siguiente código:

```
<h1>Resultados de su B&uacute;squeda:</h1>
        <span class="et_pb_fullwidth_header_subhead"> Encontrará más información en:
            <?php $categories = get_categories();
                  foreach ( $categories as $category ) {
                    if (!each($categories)){?>
```

```
                        <a href="<?php echo esc_url( get_category_link(get_cat_ID($category->name)))?>"> <?php echo esc_html( $category->name )?> </a>
                    <?php }
                else {?>
                    <a href="<?php echo esc_url( get_category_link(get_cat_ID($category->name)))?>"> <?php echo esc_html( $category->name )?> >></a>
                    <?php }
            } ?>
        </span>
```

3ª modificación: añadir buscador

En algunos casos, los resultados de búsqueda encontrados pueden no satisfacer las necesidades del cliente. Para solucionar este inconveniente podemos agregar un buscador a la página.

Para agregar esta opción, debemos añadir una llamada la buscador antes del bucle. Copia y pega el siguiente código:

```
<div class="buscador center"><?php echo get_search_form(); ?></div>
<!--debajo comienza el loop—>
```

4ª modificación: ofrecer distintas opciones si no hay ningún resultado de búsqueda

En ocasiones el usuario introduce un término de búsqueda que no coincide con información de nuestros artículos. En algunos temas, cuando no hay resultados se muestra un aviso del estilo: «No hemos encontrado ningún resultado«… en otros ni eso, simplemente aparece una página en blanco sin más información.

Debes fijarte si tu tema tiene un aviso del tipo que indico. Si es así sustituye el aviso por otro tipo de información. Por ejemplo, un enlace a la página categorías de tu blog, un texto de invitación para que realice una nueva búsqueda, una lista con los post más leídos o comentados…

Si tu tema no contempla la posibilidad de falta de resultados, tenemos que modificar el archivo search.php

En primer lugar, antes del bucle, vamos a añadir una condición «if» que compruebe si hay entradas de búsqueda. Si la condición se cumple ejecutará el bucle:

```
<?php if ( have_posts() ) : ?>
<!--debajo comienza el loop—>
```

Si la condición no se cumple, saltamos al «else» y ofrecemos otras posibilidades al usuario:

```
<?php endwhile; ?>
<!--Termina el bucle y añadimos el nuevo código else-->
<?php else:
    echo '<p>si no hay entradas de resultados de búsqueda hacer algo</p>';
endif;?>
<!--Fin del if else—>
```

el «else» debes copiarlo después de cerrar el bucle (endwhile) y vaciar la consulta (wp_reset_query).

Además del aviso, puedes añadir alguna otra opción, por ejemplo un enlace a todos los artículos del blog.

```
<?php else: ?>
     <p>No hemos encontrado resultados para su b&uacute;squeda. Encontrar&aacute; un &iacute;ndice con todas las entradas de este blog en:</p>
  <div id="enlace-todas-las-entradas">
```

```
        <h1> <a href="https://laprogramaciondehoy.com/indice-entradas-blog-lph/">>> Ver todas las entradas del Blog Lph</a></h1>
  </div>
<?php endif;?>
```

Código completo

Código completo para modificar la página de resultados de búsqueda en WordPress:

```
<!--llamada a la cabecera; logo, menú...-->
<?php
get_header();
?>
<!--contenido principal-->
<div id="main-content">
<div class="container">
<div id="content-area" class="clearfix">
<div id="left-area">
<!--título general-->
 <h1>Resultados de su Búsqueda:</h1>
<span class="et_pb_fullwidth_header_subhead"> Encontrarás más información en:
<!--las categorías del blog-->
<?php $categories = get_categories();
foreach ( $categories as $category ) {
if (!each($categories)){?>
<a href="<?php echo esc_url( get_category_link(get_cat_ID($category->name)))?>">
<?php echo esc_html( $category->name )?>
</a>
<?php }
else {?>
<a href="<?php echo esc_url( get_category_link(get_cat_ID($category->name)))?>">
<?php echo esc_html( $category->name )?> >>
</a>
<?php }
 } ?>
</span>
<!—buscador-->
<div class="buscador center"><?php echo get_search_form(); ?></div>
<!--si hay post, entra en el bucle-->
<?php if ( have_posts() ) : ?>
<!--el loop-->
<?php while ( have_posts() ) : the_post(); ?>
<!-- recopilando info de cada post -->
<article id="post-<?php the_ID(); ?>" <?php post_class(); ?>>
<!-- título del post -->
<h1 class="entry-title main_title"><?php the_title(); ?></h1>
<!-- contenido del post -->
<div class="entry-content">
<?php the_excerpt(); ?>
<a href="<?php the_permalink();?>">Leer más...</a>
</div> <!-- fin contenido post -->
</article> <!-- fin info de cada post -->
<?php endwhile; ?><!-- fin del loop -->
<!-- si no hay post de búsqueda -->
<?php else: ?>
<p>No hemos encontrado resultados para su búsqueda. Encontrará un índice con todas las entradas de este blog en:</p>
<div id="enlace-todas-las-entradas">
<h1> <a href="https://laprogramaciondehoy.com/indice-entradas-blog-lph/">>> Ver todas las entradas del Blog Lph</a></h1>
</div>
```

```
<?php endif;?>
</div> <!-- termina contenido derecha -->
<?php get_sidebar(); ?><!-- barra lateral -->
</div> <!-- fin de div contentarea -->
</div> <!-- fin de div container -->
</div> <!-- fin de div contenido principal -->

<?php get_footer(); ?><!-- llamada al pie de página —>
```

Página 404
Cuando un usuario encuentra nuestro sitio web (en Google, por ejemplo) y la URL indexada ya no existe, se genera un error. Suele ver la página 404 genérica, que viene por defecto con el tema, indicándole que la información solicitada no se encontró en nuestro sitio web.

Al crear un sitio web, no solemos prestar mucha atención a la página 404 pero, sabías que: ¿cada página 404 que es visitada en nuestro sitio web es una oportunidad que podríamos aprovechar?

¿Qué es un error 404?
Un error 404 es uno de los códigos de estado HTTP que devuelve el servidor para indicar que no se encontró la página solicitada.
Este error se genera por las URLs de un sitio web.
Algunas de las causas más comunes son:

- El usuario hizo clic en un enlace roto.
- La página se ha eliminado o se ha modificado la URL sin hacer la correspondiente redirección.
- El usuario escribió mal la URL.

Sea cual sea el caso, el usuario ha llegado a un callejón sin salida ya que no ha encontrado lo que estaba buscando y lo más seguro es que se vaya y no regrese más a nuestro sitio web.

Por estas razones, es muy importante ofrecer una página 404 personalizada, con un diseño atractivo, para que el usuario sienta que puede quedarse y encontrar lo que busca.

Aunque los enlaces rotos y las redirecciones podemos corregirlas nosotros mismos, si el usuario escribe mal la URL que quiere visitar, no podríamos hacer absolutamente nada ya que no es una URL que exista en nuestro sitio web.

¿Por qué crear una página 404 personalizada?
La mayoría de temas incluyen una página 404 genérica que se mostrará al usuario. Puede que no parezca algo importante y, sabiendo que el tema ya ofrece una página 404, no nos tomemos la molestia de crear una página 404 personalizada.

A continuación te indico 5 razones por las cuales deberíamos hacerlo, tanto para nuestros usuarios como para el SEO:

1 Incrementa el número de páginas indexadas en Google
Cuando tenemos un sitio web con muchas páginas internas es posible que existan errores 404. Una de las formas más efectivas para aprovechar estos errores es poner enlaces "aleatorios" a páginas internas desde la página 404.

De esta manera, cuando los usuarios llegan a nuestra página 404, pueden hacer clic en cualquier enlace y continuar navegando en nuestro sitio web. Es posible que terminen encontrando lo que estaban buscando.

2 Genera confianza construyendo una imagen de marca consistente
A las personas les gusta las cosas consistentes. Cuando nos vamos familiarizando con un sitio web comenzamos a confiar en el. Es importante ofrecer una imagen consistente manteniendo la esencia y estructura del sitio original.

Una página 404 "genérica" no generará confianza en el usuario y, el diseño de esta página, no debe confundirlo haciéndole pensar que está en un sitio web distinto.

Por esta razón, deberíamos crear una página 404 personalizada con una imagen similar a la imagen que ofrecemos en todo nuestro sitio web: logo, menú, colores, fuente entre otros.

3 Ofrece contenido "atractivo" a los usuarios insatisfechos
No es agradable buscar información en Internet y encontrarse con una página que muestre un error 404 "genérico". Lo más seguro es que nos vayamos y busquemos la información en otro sitio.

Por esta razón es importante ofrecer información relevante sobre nuestro sitio web en la página 404 de manera que el usuario mantenga la atención en el sitio y continúe navegando.

4 Muestra más personalidad en tu diseño
Cuando personalices una página 404 en WordPress, intenta no ser tan formal. Es importante que muestres algo de "personalidad" de manera que conectes con tus usuarios.

Tus usuarios no son robots y tienen sentimientos. Si muestras algo de simpatía, alegría o entretenimiento, es posible que tus usuarios te recuerden por eso y pasen por alto el hecho de que llegaron a una página de error.

5 Saca provecho de tu error
Al personalizar tu página 404 en WordPress, no solo asumes el error (que deberías intentar resolverlo cuanto antes) sino que también puedes lograr que el usuario pueda interesarse por tus servicios.

Por ejemplo, podemos ofrecer la descarga de una guía que el usuario ni siquiera estaba buscando o podemos ofrecer productos con descuentos en el caso que tengamos una tienda online.
Sea cual sea el caso, la idea principal es que el usuario se sienta como si no hubiera pasado nada y logre encontrar lo que estaba buscando en nuestro sitio web.

Crear una página 404 en WordPress
Antes de crear y personalizar una página 404 en WordPress, es importante que tengas en cuenta el tipo de contenido que deberá tener esta página:

- Informa al usuario a qué tipo de página ha llegado.

- Añade un contenido atractivo para el usuario.

- Integra un enlace a la página de inicio.

Vamos a crear un archivo 404.php en la misma carpeta de nuestro tema,. Es un archivo PHP que contiene el código de la **página 404** manteniendo el estilo y diseño del sitio web.

```
<?php get_header(); // Agrega la función para incluir el encabezado de tu tema ?>

<div id="primary" class=“content-area">
        <main id="main" class=“site-main">
                <section class="error-404 not-found”>
                        <header class=“page-header">
                                <h1 class=“page-title”>
                                        <?php esc_html_e('Página no encontrada', 'textodominio'); ?>
                                </h1>
                        </header><!-- .page-header -->
```

```
                        <div class="page-content">
                            <p><?php esc_html_e('Lo siento, pero la página que estás buscando no existe.', 'textodominio'); ?></p>
                            <p><?php esc_html_e('Puedes intentar realizar una búsqueda o regresar a la página de inicio.', 'textodominio'); ?></p>
                            <?php get_search_form(); // Muestra el formulario de búsqueda ?>
                        </div><!-- .page-content —>
                </section><!-- .error-404 —>
        </main><!-- #main -->
</div><!-- #primary -->

<?php get_footer(); // Agrega la función para incluir el pie de página de tu tema ?>
```

Al seguir estos pasos, has creado una página de error 404 personalizada sin necesidad de utilizar plugins. La plantilla `404.php` se utiliza automáticamente cuando WordPress no puede encontrar una página, lo que garantiza que tu diseño personalizado se muestre en lugar del mensaje de error predeterminado.

Plantilla de página personalizada

Esa posible que nuestro site necesite una página con un diseño diferente al resto, con una estructura diferente. Wordpress nos ofrece la posibilidad de crear plantillas de páginas personalizadas. Para ello crearemos un archivo pagina_personalizada.php:

```
<?php
/*
Template Name: Plantilla Personalizada
*/
get_header(); // Agrega la función para incluir el encabezado de tu tema
?>
```

En este código, `Template Name` define el nombre de la plantilla tal como aparecerá en el área de administración de WordPress cuando estés editando una página. En este caso, se llama "Plantilla Personalizada”.

Después del código mencionado, puedes agregar el contenido HTML y PHP que deseas que aparezca en tu página personalizada. Puedes incluir bucles de WordPress, llamadas a funciones, estilos, scripts, etc.

```
<div id="primary" class="content-area">
    <main id="main" class="site-main">

        <!-- Contenido de tu plantilla personalizada aquí -->

    </main><!-- #main -->
</div><!-- #primary —>
```

Al final de tu plantilla, agrega el siguiente código para incluir el pie de página de tu tema:

```
<?php
get_footer(); // Agrega la función para incluir el pie de página de tu tema
?>
```

Guarda los cambios en tu nuevo archivo PHP y sube el archivo al directorio de tu tema. En el área de administración de WordPress, edita o crea una nueva página. En el editor de páginas, busca la sección de "Atributos de la página" y selecciona tu plantilla personalizada del menú desplegable llamado “Plantilla". Guarda y visualiza:

Ahora, tendrás una plantilla de página personalizada en WordPress que puedes asignar a páginas específicas según tus necesidades. Puedes repetir estos pasos para crear varias plantillas personalizadas si lo deseas.

Temas Hijo
La plataforma WordPress es un imán para aquellos que quieren tomar el asunto en sus propias manos, quieren un control total sobre sus sitios web y quieren ser independientes al administrarlos. WordPress hace que sea muy fácil personalizar completamente un sitio web. Si tienes un poco de conocimiento de HTML, CSS y/o PHP, no hay nada que no puedas cambiar.

Quiero decir, simplemente compara los temas predeterminados, Twenty Fifteen y Twenty Fourteen. Es difícil creer que estén funcionando en la misma plataforma, ¿no? Por lo tanto, es natural que desees adaptar el aspecto de su sitio web para que se ajuste a tu visión. Dudo que haya muchos usuarios de WordPress que no piensen constantemente en qué implementar a continuación. Sin embargo, surge un problema.

La mayor desventaja es que cualquier modificación realizada en el tema de esta manera se perderá una vez que el desarrollador actualice el tema. Como consecuencia, los usuarios no podrán mantener su tema actualizado (lo cual es malo para la seguridad) o descubrirán que todas sus personalizaciones desaparecerán cuando lo hagan.

De cualquier manera, la situación dista mucho de ser ideal.

Una idea mucho mejor es utilizar un tema secundario. Esto le permite realizar cualquier cantidad de cambios en un sitio web sin tocar ninguno de los archivos del tema original.

¿Suena bien? Genial, porque analizaremos detalladamente qué son los temas secundarios de WordPress, cómo crearlos y cómo usarlos para personalizar su sitio web, de la manera correcta.

¿Qué son los temas hijo y por qué utilizarlos?

Cuando hablamos de temas secundarios, primero tenemos que hablar de temas principales. Un tema solo se convierte en tema principal cuando alguien crea un tema secundario para él. Hasta entonces, es sólo un tema, como los que encuentras en el directorio de WordPress. Cada tema que incluye todos los archivos necesarios para ser considerado completo puede ser un tema principal.

Sin embargo, si bien cualquiera de estos temas puede ser un tema principal, algunos son más adecuados para este propósito que otros.

¿Qué es entonces un tema hijo? Bueno, desde el back-end de WordPress, un tema hijo no se comporta de manera diferente. Puede encontrarlo y activarlo en "Apariencia" → "Temas", tal como lo haría con cualquier otro tema.

La gran diferencia es que un tema hijo depende completamente de su padre para funcionar. Sin su tema principal presente, no hará nada y ni siquiera podrá activarse.

Esto se debe a que un tema secundario no es una entidad independiente, sino que modifica o agrega archivos a un tema existente. Utiliza todo lo presente en el tema principal y cambia solo aquellas partes que desea que sean diferentes.

Esto le permite modificar estilos, funciones, diseño, plantillas y más. De hecho, puedes personalizar el tema principal más allá del reconocimiento. Sin embargo, sin que esté presente, nada de esto funcionará.

Ventajas de los temas hijo

Existen numerosas ventajas al seguir la ruta del tema hijo:

En lugar de tener que crear un tema completo desde cero, puedes construir sobre algo que ya existe, acelerando así el tiempo de desarrollo.

Puede aprovechar la funcionalidad de marcos sofisticados y temas principales, mientras personaliza el diseño según sus necesidades.

Puede actualizar el tema principal sin perder sus personalizaciones.

Si no está satisfecho con sus personalizaciones, simplemente desactive el tema secundario y todo volverá a ser como antes.

Es una excelente manera de comenzar a aprender cómo funcionan los temas.

Un tema hijo puede contener carpetas de imágenes, JavaScript, CSS, archivos de plantilla y muchas otras cosas. Sin embargo, lo hermoso es que no es necesario. Puede incluir tanta personalización como desee.

De hecho, un tema hijo realmente sólo necesita tres cosas: una carpeta, una hoja de estilo y un archivo funciones.php. Eso es todo. Y los dos archivos pueden incluso estar prácticamente vacíos.

Cuándo usar un tema hijo

Entonces, ¿debería crear siempre un tema secundario cada vez que desee realizar cambios en un sitio web de WordPress? No, realmente depende.

Si planea realizar solo modificaciones menores, como cambios de color o una fuente diferente, entonces un complemento CSS personalizado puede ser todo lo que necesita. Muchos temas hoy en día también ofrecen la opción de agregar código personalizado de forma nativa.

Sin embargo, si planeas introducir cambios más importantes, como una revisión completa del diseño, múltiples cambios de plantilla o cualquier otra cosa de esa magnitud, entonces un tema hijo es definitivamente el camino a seguir.

Configurar un tema hijo básico

Muy bien, ahora que sabemos lo fantásticos que son los temas hijo y lo que pueden hacer por nosotros, repasemos cómo crear uno paso a paso. Para nuestro ejemplo, usaremos Twenty Fifteen, el último tema predeterminado para WordPress. No te preocupes, es realmente fácil y lo conseguirás en poco tiempo.

Nota al margen: Los pasos siguientes se pueden realizar directamente en su servidor a través de un cliente FTP. Sin embargo, le recomiendo que primero configure todo localmente, luego comprima la carpeta de temas secundarios y la instale como un tema normal a través del menú "Tema". Esto hará que todo sea mucho más fácil.

Creamos una carpeta en wp-content/themes. Como se mencionó, un tema hijo necesita tres cosas: su propia carpeta, una hoja de estilo y un archivo funciones.php. Empezaremos con la carpeta.

Como cualquier tema, los temas secundarios se encuentran en wp-content/themes en su instalación de WordPress. Entonces, navega allí ahora y crea una nueva carpeta para tu tema hijo.

Una mejor práctica es darle a la carpeta de su tema el mismo nombre que el tema principal y agregarle -child. Como estamos usando el tema Twenty Fifteen, llamaremos a nuestra carpeta veinticinco-child.

Crea una hoja de estilo. Ahora que tenemos nuestra carpeta, necesitaremos una hoja de estilo. Por si no lo sabes, una hoja de estilo contiene el código que determina el diseño de un sitio web. Los temas pueden tener varias hojas de estilo, pero por el momento nos contentaremos con una.

Crear una hoja de estilo es fácil: simplemente cree un nuevo archivo de texto y llámelo style.css. ¡Hecho! Sin embargo, para que realmente funcione, tendremos que pegar el siguiente código, el llamado “encabezado de hoja de estilo”, justo al principio del archivo:

```
/*
Theme Name: Twenty Fifteen Child
Theme URI: https://example.com/twenty-fifteen-child/
Description: >- Twenty Fifteen Child Theme
Author: John Doe
Author URI: https://example.com
Template: twentyfifteen
Version: 1.0.0
License: GNU General Public License v2 or later
License URI: https://www.gnu.org/licenses/gpl-2.0.html
Tags: light, dark, two-columns, right-sidebar, responsive-layout, accessibility-ready
Text Domain: twenty-fifteen-child
*/
```

Esto es lo que significa cada línea:

Theme Name. Este es el nombre que aparecerá para su tema en el back-end de WordPress.

Theme URI. Esto apunta al sitio web o página de demostración del tema en cuestión. Este o el URI del autor deben estar presentes para que el tema sea aceptado en el directorio de WordPress.

Description. Esta descripción de su tema aparecerá en el menú del tema cuando haga clic en "Detalles del tema”.

Author. Este es el nombre del autor; ese eres tú, en este caso.

Author URI. Puede poner la dirección de su sitio web aquí si lo desea.

Template. Esta parte es crucial. Aquí va el nombre del tema principal, es decir, el nombre de su carpeta. Tenga en cuenta que distingue entre mayúsculas y minúsculas y, si no ingresa la información correcta, recibirá un mensaje de error, así que verifique nuevamente.

Version. Esto muestra la versión de su tema hijo. Normalmente, comenzarías con 1.0.

License. Esta es la licencia de su tema hijo. Los temas de WordPress del directorio suelen publicarse bajo una licencia GPL; debes seguir con la misma licencia que tu tema principal.

License URI. Esta es la dirección donde se explica la licencia de su tema. Nuevamente, quédate con lo que dice tu tema principal.

Tags. Las etiquetas ayudan a otros a encontrar su tema en el directorio de WordPress. Por lo tanto, si incluyes alguno, asegúrate de que sea relevante.

Text Domain. Esta parte se utiliza para la internacionalización y para hacer que los temas sean traducibles.

Si te sientes un poco abrumado (¿ya?), le alegrará saber que no toda esta información es realmente necesaria. De hecho, todo lo que realmente necesitas es el nombre del tema y la plantilla.

El resto es importante sólo si planeas publicar tu tema, cosa que yo no hago. Por esta razón, el encabezado de mi tema hijo se parece a lo que se muestra a continuación. No dudes en copiarlo y hacer tus propios ajustes.

```
/*
 Theme Name:   Twenty Fifteen Child Theme
 Description: >-
  A child theme of the Twenty Fifteen default WordPress theme
 Author:       Nick Schäferhoff
 Template:     twentyfifteen
 Version:      1.0.0
*/
```

Una vez que su carpeta y hoja de estilo estén presentes, vaya a "Apariencia" → "Temas" en el back-end de WordPress y busque su tema secundario allí. Cuando haga clic en "Detalles del tema" ahora, verá el contenido del encabezado de la hoja de estilo. Para eso es esa información.

No te preocupes, todo está bien. No lo has cagado. Saca tu cara de la bolsa de papel. La razón por la que su sitio web está vacío es porque aún no tiene ningún estilo. Sin estilos, disfrutarás de una experiencia basada exclusivamente en texto.

Sólo quería mostrarles que, en teoría, tener una hoja de estilo y una carpeta es suficiente para crear un tema hijo. Y si te funcionó, ¡entonces ya lo has hecho! Sin embargo, seré el primero en admitir que podría verse un poco mejor. Vayamos a eso ahora.

El siguiente es el archivo functions.php. Probablemente hayas oído hablar de este archivo antes, pero repasemos rápidamente para qué sirve.

El archivo functions.php le permite cambiar y agregar funcionalidades y características a un sitio web de WordPress. Puede contener funciones PHP y nativas de WordPress. Además, eres libre de crear tus propias funciones.

En resumen, functions.php contiene código que cambia fundamentalmente la apariencia y el comportamiento de un sitio web. ¿Entiendo? Genial, sabía que podía contar contigo.

Crear el archivo es tan fácil como crear una hoja de estilo, si no más. Todo lo que necesitas es un archivo de texto llamado functions.php y luego pegar el siguiente código:

```
<?php
//* Code goes here
```

En serio, eso es todo. Simplemente agregue esa etiqueta php de apertura y listo. Por supuesto, puedes ponerte elegante y escribir algo de información en el encabezado (no olvides comentarlo para que WordPress no intente ejecutarlo), pero esto servirá para nuestro propósito. Agrégualo también a la carpeta de su tema.

Ahora, déjame decirte esto: no necesitas functions.php. Si no planeas usar PHP para modificar tu tema, entonces puedes prescindir de él por completo. Una hoja de estilo y otros archivos pueden ser suficientes para ti.

Sin embargo, quería incluir esta parte, primero, para que conocieras este importante archivo y, segundo, para conocer el siguiente paso.

Ahora, lo que debemos hacer es importar el estilo del tema padre, para que, de entrada, nuestro tema hijo tenga la misma apariencia que el tema padre. Esto podemos hacerlo de dos formas:

Una es mediante CSS y la regla @import. Al copiar el código siguiente en su archivo style.css, le está indicando a su tema secundario que use la información contenida en la hoja de estilo de su tema principal para presentar su contenido.

@import url("../twentyfifteen/style.css");

Sin embargo, ten en cuenta que esta es la forma antigua de heredar estilos principales y ya no se recomienda. La razón de esto es el rendimiento.

Si necesita importar varias hojas de estilo (lo cual no es inaudito), usar @import hará que se descarguen consecutivamente. Esto puede ralentizar el tiempo de carga de la página varios segundos (lo cual, probablemente no hace falta que te lo diga, no es algo bueno).

La segunda forma recomendada de cargar la hoja de estilo principal (y la razón por la que creamos functions.php anteriormente) es usar wp_enqueue_style(). Esta función de WordPress agrega de forma segura archivos de hojas de estilo a un tema de WordPress.

En nuestro caso, el código correspondiente se parece a esto:

```
add_action( 'wp_enqueue_scripts', 'enqueue_parent_styles' );

function enqueue_parent_styles() {
   wp_enqueue_style( 'parent-style', get_template_directory_uri().'/style.css' );
}
```

Una vez hayamos leído el estilo del tema padre, podremos incorporar nuestra propia hoja de estilo para el tema hijo, además de todas las funciones que queramos añadir a nuestro archivo functions.php.

También podemos sobreescribir cualquier archivo que deseemos. Es decir, si el tema padre tiene aun archivo header.php y creamos otro en el tema hijo, este segundo sobreescribirá al primero, siempre que estemos activado el tema hijo, evidentemente.

Custom Post Types

Cuando WordPress empezó, sólo existían los "posts". Nada más. No había páginas, sólo posts, porque WordPress sólo servía para hacer blogs.

Pero con el tiempo sus desarrolladores vieron clarísimo que necesitábamos ir más allá. Y crearon las "Páginas", que son una "especie de post", o por decirlo de otra forma, un "tipo de post". De ahí le viene lo de "Post Type". Seguramente hubiera tenido más sentido llamarle "Content Type", o sea, "Tipo de contenido", pero así quedó.

Las páginas pues, son como posts, con algunas diferencias. Por ejemplo, no tienen fecha, ni categorías ni etiquetas.

Así pues, ya teníamos Posts y Páginas (tipo de post). Pero la verdadera revolución llegó con WordPress 3.5, que trajo los "Custom Post Types". ¿Y de que se trata? Pues ni más ni menos que de la posibilidad de crear tus propios tipos de post. Con otras palabras, "Tipos de Posts Personalizados".

Y eso es lo que son los CPT: Nuevos contenidos que podemos agregar a WordPress, además de las Entradas (posts) y las Páginas (post types). Un ejemplo de CPT serían los "**Productos**" de un eCommerce. Si usáis WooCommerce, veréis el menú de "Productos" en el panel de control. Eso es un CPT. Pero podríamos poner cualquier otro tipo de contenido: Servicios, personas, películas, contactos, animales…

¿Cuando vale la pena, hacer un CPT?

Si en alguna ocasión os habéis planteado la estructura de una web, seguramente en algún momento os habréis planteado qué categorías y etiquetas usar. O incluso si queréis usar categorías y etiquetas. O incluso si usar CPTs. ¿Cuándo debemos usar una cosa u otra?

La respuesta está en la similitud de esos CPTs con las páginas y entradas. Si comparten el mismo tipo de información o sistema de organización, no tiene sentido hacer CPTs.

Así pues, yo aconsejo usar CPTs en lugar de categorías cuando la entidad del contenido sea suficientemente significativa como para independizarse.

Bien, supongamos pues, que habéis analizado vuestro caso y que queréis usar CPTs para algo en concreto. Ahora la pregunta clave es... ¿Y cómo creo un CPT?

Cómo crear un CPT

Puedes registrar un Custom Post Type utilizando la función `register_post_type()` en el archivo `functions.php` de tu tema o en un plugin. Aquí hay un ejemplo:

```
function registrar_tipo_personalizado() {
   $args = array(
      'public' => true,
      'label'  => 'Ejemplo', // Nombre del Custom Post Type
      // Otros parámetros...
   );
   register_post_type('tipo_personalizado', $args);
}

add_action('init', ‘registrar_tipo_personalizado');
```

Los Custom Post Types (Tipos de Contenido Personalizado) son una característica poderosa de WordPress que te permite definir y crear tipos de contenido adicionales además de las publicaciones y páginas predeterminadas. Los Custom Post Types te permiten organizar y mostrar diferentes tipos de contenido de manera estructurada en tu sitio.

Aquí hay algunas cosas clave que debes saber sobre los Custom Post Types:

1. Definición de un Custom Post Type: Un Custom Post Type es esencialmente un tipo de contenido específico que defines. Puedes crear Custom Post Types para manejar diferentes tipos de información, como "Productos", "Eventos", "Portafolio", etc.

2. Cómo Registrar un Custom Post Type: Puedes registrar un Custom Post Type utilizando la función register_post_type() en el archivo functions.php de tu tema o en un plugin. Aquí hay un ejemplo:

```
function registrar_tipo_personalizado() {
        $args = array(
                        'public' => true,
                       'label' => 'Ejemplo', // Nombre del Custom Post Type // Otros parámetros...
                       );
        register_post_type('tipo_personalizado', $args);
}

add_action('init', 'registrar_tipo_personalizado');
```

3. Parámetros Comunes al Registrar un Custom Post Type: Al registrar un Custom Post Type, puedes especificar una variedad de parámetros. Algunos comunes incluyen:

- 'public': Define si el Custom Post Type es público y visible en el frontend.
- 'label': El nombre del Custom Post Type.
- 'supports': Qué características admitirá (títulos, editores, miniaturas, etc.).
- 'rewrite': Permite personalizar la estructura de las URLs de los Custom Post Types.

4. Cómo Usar un Custom Post Type: Una vez que has registrado un Custom Post Type, puedes crear, editar y gestionar entradas de ese tipo desde el área de administración de WordPress, similar a las publicaciones y páginas predeterminadas.

5. Cómo Mostrar Entradas de un Custom Post Type: Puedes mostrar entradas de un Custom Post Type utilizando consultas personalizadas en tus plantillas de tema. Por ejemplo:

```
$args = array(
        'post_type' => 'tipo_personalizado', ‘
        posts_per_page' => 10,
);
 $query = new WP_Query($args);
```

Veamos una versión de creación de un CPT con todos sus parámetros:

```
/* Register a custom post type called "book". */

function wpdocs_codex_book_init() {
        $labels = array(
                'name'               => _x( 'Books', 'Post type general name', 'textdomain' ),
                'singular_name'      => _x( 'Book', 'Post type singular name', 'textdomain' ),
                'menu_name'          => _x( 'Books', 'Admin Menu text', 'textdomain' ),
                'name_admin_bar'     => _x( 'Book', 'Add New on Toolbar', 'textdomain' ),
                'add_new'            => __( 'Add New', 'textdomain' ),
                'add_new_item'       => __( 'Add New Book', 'textdomain' ),
                'new_item'           => __( 'New Book', 'textdomain' ),
                'edit_item'          => __( 'Edit Book', 'textdomain' ),
                'view_item'          => __( 'View Book', 'textdomain' ),
                'all_items'          => __( 'All Books', 'textdomain' ),
                'search_items'       => __( 'Search Books', 'textdomain' ),
                'parent_item_colon'  => __( 'Parent Books:', 'textdomain' ),
                'not_found'          => __( 'No books found.', 'textdomain' ),
                'not_found_in_trash' => __( 'No books found in Trash.', 'textdomain' ),
                'featured_image'     => _x( 'Book Cover Image', 'Overrides the “Featured Image” phrase for this post type. Added in 4.3', 'textdomain' ),
                'set_featured_image' => _x( 'Set cover image', 'Overrides the “Set featured image” phrase for this post type. Added in 4.3', 'textdomain' ),
                'remove_featured_image' => _x( 'Remove cover image', 'Overrides the “Remove featured image” phrase for this post type. Added in 4.3', 'textdomain' ),
                'use_featured_image' => _x( 'Use as cover image', 'Overrides the “Use as featured image” phrase for this post type. Added in 4.3', 'textdomain' ),
                'archives'           => _x( 'Book archives', 'The post type archive label used in nav menus. Default “Post Archives”. Added in 4.4', 'textdomain' ),
                'insert_into_item'   => _x( 'Insert into book', 'Overrides the “Insert into post”/”Insert into page” phrase (used when inserting media into a post). Added in 4.4', 'textdomain' ),
                'uploaded_to_this_item' => _x( 'Uploaded to this book', 'Overrides the “Uploaded to this post”/”Uploaded to this page” phrase (used when viewing media attached to a post). Added in 4.4', 'textdomain' ),
                'filter_items_list'  => _x( 'Filter books list', 'Screen reader text for the filter links heading on the post type listing screen. Default “Filter posts list”/”Filter pages list”. Added in 4.4', 'textdomain' ),
```

```
		'items_list_navigation' => _x( 'Books list navigation', 'Screen reader text for the pagination heading on the post type listing screen. Default “Posts list navigation”/”Pages list navigation”. Added in 4.4', 'textdomain' ),
		'items_list'            => _x( 'Books list', 'Screen reader text for the items list heading on the post type listing screen. Default “Posts list”/”Pages list”. Added in 4.4', 'textdomain' ),
	);

	$args = array(
		'labels'             => $labels,
		'public'             => true,
		'publicly_queryable' => true,
		'show_ui'            => true,
		'show_in_menu'       => true,
		'query_var'          => true,
		'rewrite'            => array( 'slug' => 'book' ),
		'capability_type'    => 'post',
		'has_archive'        => true,
		'hierarchical'       => false,
		'menu_position'      => null,
		'supports'           => array( 'title', 'editor', 'author', 'thumbnail', 'excerpt', 'comments' ),
	);

	register_post_type( 'book', $args );
}

add_action( 'init', 'wpdocs_codex_book_init' );
```

Un último consejo acerca de los CPT. Muchas veces podemos estar tentados de usar custom post types dentro de un tema. Y no es una mala opción si ese tema NO SE VA A CAMBIAR EN EL FUTURO. Es decir, si ese es el tema que se va a usar siempre en el site, no hay problema. Pero si el cliente o el usuario tiene la posibilidad de cambiar tema, los CPT se perderán, porque van anexadas al tema, dentro del archivo functions.php. Lo adecuado es que se añadan en un plugin.

Los Custom Post Types son una herramienta poderosa que permite una gran flexibilidad en la estructuración de contenidos en WordPress. Pueden ser utilizados para construir diferentes secciones de tu sitio, proporcionando una mejor organización y presentación del contenido.

Taxonomías

Las taxonomías en WordPress son una forma de organizar y clasificar contenido. Pueden ser especialmente útiles cuando trabajas con grandes cantidades de información y necesitas una estructura de categorización más compleja que las categorías y etiquetas predeterminadas. Las dos principales taxonomías en WordPress son "Categorías" y "Etiquetas", pero también puedes crear taxonomías personalizadas según tus necesidades.

Taxonomías Predeterminadas:

Categorías: Las categorías son una taxonomía jerárquica. Puedes organizarlas en una estructura de árbol, lo que permite subcategorías. Por ejemplo, puedes tener una categoría principal como "Tecnología" y subcategorías como "Software" y “Hardware".

Etiquetas: Las etiquetas son una taxonomía no jerárquica. Son útiles para proporcionar palabras clave adicionales o descripciones más detalladas. Por ejemplo, puedes tener una entrada sobre un tutorial de WordPress y usar las etiquetas "WordPress", "Tutorial" y "Desarrollo Web”.

Crear Taxonomías Personalizadas:
Puedes crear taxonomías personalizadas usando la función register_taxonomy(). Aquí hay un ejemplo básico:

```
function registrar_taxonomia_personalizada() {
    $args = array(
        'hierarchical' => true, // true para una taxonomía jerárquica, false para no jerárquica
        'labels' => array(
            'name' => 'Categorías Personalizadas',
            'singular_name' => 'Categoría Personalizada',
        ),
        'show_ui' => true,
        'show_admin_column' => true,
        'query_var' => true,
        'rewrite' => array('slug' => 'categoria-personalizada'),
    );
    register_taxonomy('categoria_personalizada', array('post'), $args);
}

add_action('init', ‘registrar_taxonomia_personalizada');
```

Este código crea una taxonomía personalizada llamada "Categorías Personalizadas" que se aplica a las entradas ('post'). Ajusta los parámetros según tus necesidades.

Asignar Términos de Taxonomía a las Entradas:
Una vez que has creado una taxonomía, puedes asignar términos de esa taxonomía a las entradas de WordPress. Esto se hace en la pantalla de edición de la entrada en el área de administración.

Mostrar Entradas según Términos de Taxonomía:
Puedes mostrar entradas según los términos de una taxonomía en tus plantillas utilizando consultas personalizadas. Aquí hay un ejemplo para mostrar entradas de una categoría personalizada:

```
$args = array(
    'post_type' => 'post',
    'tax_query' => array(
        array(
            'taxonomy' => 'categoria_personalizada',
            'field' => 'slug',
            'terms' => 'termino-slug',
        ),
    ),
);
$query = new WP_Query($args);
```

Este código mostrará entradas que tienen el término "termino-slug" en la taxonomía “categoria_personalizada".

Las taxonomías en WordPress son herramientas poderosas para organizar y clasificar tu contenido de una manera más estructurada y significativa. Puedes personalizarlas según tus necesidades y utilizarlas para mejorar la navegación y la presentación de tu sitio.

Metaboxes

Si no lo sabes, un metabox te permite añadir campos extra a la página de edición de las entradas (como dije antes, sean del tipo que sean). Con esto podrás guardar más información sobre un mismo tipo de contenido.

Te refresco la memoria: habíamos creado un custom post type “Libros” con el que añadir fichas de libros a nuestro WordPress; en el siempre hipotético caso de que nuestra web fuese a tratar de libros. Esto nos permitiría tener el blog original de WordPress intacto y las fichas de los libros almacenadas en otro tipo de entrada.

Para añadir un metabox a un post type, usaremos este código en el archivo functions.php:

```
// Función para añadir un metabox a las entradas
function agregar_metabox_personalizado() {
    add_meta_box(
        'id_metabox_personalizado',
        'Información Adicional',
        'mostrar_contenido_metabox',
        'post', // Tipo de contenido al que se aplicará (en este caso, 'post' para las entradas)
        'normal', // Contexto donde se mostrará (puede ser 'normal', 'advanced', o 'side')
        'high' // Prioridad en la que se mostrará ('high', 'core', 'default' o 'low')
    );
}
add_action('add_meta_boxes', 'agregar_metabox_personalizado');

// Función para mostrar el contenido del metabox
function mostrar_contenido_metabox($post) {
    // Recupera el valor almacenado (si existe)
    $valor_metabox = get_post_meta($post->ID, '_clave_metabox', true);

    // Campo de entrada para recoger la información
    echo '<label for="campo_metabox">Información Adicional:</label>';
            echo '<input type="text" id="campo_metabox" name="campo_metabox" value="' .
esc_attr($valor_metabox) . '" size="25">';
}

// Función para guardar el valor del metabox cuando se guarda la entrada
function guardar_valor_metabox($post_id) {
    if (isset($_POST['campo_metabox'])) {
                        update_post_meta($post_id, '_clave_metabox',
sanitize_text_field($_POST['campo_metabox']));
    }
}
add_action('save_post', ‘guardar_valor_metabox');
```

Para añadir un metabox, tenemos que utilizar la función add_meta_box, que según la documentación oficial recibe la siguiente información:

add_meta_box($id, $title, $callback, $page, $context, $priority, $callback_args);

Al desglosar los parámetros que recibe, entendemos que:

- $id el el ID del metabox. Es útil, por ejemplo, si vas a utilizar un CSS custom para darle estilos a este metabox o incluso vas a hacer algo con Javascript; sino, no tiene mucha importancia.

- $title es el título que será mostrado en la parte superior del metabox, como veremos próximamente.

- $callback es la función que dará uso a nuestro metabox, lo veremos en el siguiente punto.

- $page es donde queremos que se muestre nuestro metabox; podemos decidir que se muestre en los post, las páginas o un custom post type (nuestro caso).

- $context es dónde queremos que se muestre nuestro metabox. “normal” significará que se muestre bajo el editor de la entrada o página, “side” colocará el metabox a la barra lateral de la página de edición y “advanced” lo colocará en la misma columna que el editor, pero con algún extra que no viene al caso.

- $priority le dice a WordPress dónde cargar el metabox en el contenido. “high”, “default” o “low” coloca la caja del metabox arriba, en su posición natural o abajo del todo

respectivamente. Aunque todos los metaboxes de WordPress funcionan con *Drag and drop*, $priority no tiene demasiado sentido.

- $callback_args es de otra batalla, no lo utilizaremos aquí y no quiero complicarte demasiado con cosas que no utilizarás.

Tras la breve explicación, nuestra llamada en el functions.php quedará así:

```
function libro_metabox() {
	add_meta_box(
		'informacion-libro',
		'Datos adicionales del libro’,
		 'dariobf_meta_box_content',
		'libro',
		'normal',
		'high'
	);
}

add_action( 'add_meta_boxes', 'libro_metabox' );
```

Algo a resaltar es que, he metido la llamada a *add_meta_box* dentro de una función (libro_*metabox*) y la he enganchado al *hook add_meta_boxes*.

Esto del metabox está muy bien, pero no hemos añadido nada útil hasta el momento, ¿Qué tal si añado unos cuantos campos que me permitan crear una ficha de producto?

¿Recuerdas que prometí explicarte el uso de $callback en el primer punto de este tutorial? Ahora voy a crear una función con ese mismo nombre, que será la encargada de mostrar el contenido de nuestro metabox.

```
function libro_meta_box_content( $post ) {
   ?>
   <p>Aquí pondremos todo el contenido de nuestro metabox<p>
   <?php
}
```

Añadir campos de texto (Input text) al metabox

Partiendo de la función anterior, puedo añadir campos de texto al metabox, adaptándola de la siguiente forma para, por ejemplo, recoger el ISBN del libro en cuestión:

```
function libro_meta_box_content( $post ) {
   ?>
   <p>
      <label for="info_libro_isbn">ISBN</label>
      <input type="text" name="info_libro_isbn" id="info_libro_isbn" value="" />
   </p>
   <?php
}
```

Ya tenemos un campo ISBN, pero… ¿Cómo y dónde guarda la información que introduzco en el input? Fácil, en la base de datos, este campo será un custom field de WordPress.

Cuando edite un libro (recuerda que nuestro ejemplo va sobre fichas de libros) quiero recuperar esta información, así que tendré que utilizar la función get_post_meta.

Voy a añadir unas variables al ejemplo anterior que recuperarán esta información de la base de datos cada vez que carguemos la página de edición del libro.

```
function libro_meta_box_content( $post ) {
   $values    = get_post_custom( $post->ID );
   $isbn      = isset( $values['info_libro_isbn'] ) ? esc_attr( $values['info_libro_isbn'][0] ) : '';
   ?>
   <p>
      <label for="info_libro_isbn">ISBN</label>
                  <input type="text" name="info_libro_isbn" id="info_libro_isbn" value="<?php echo
esc_html( $isbn ); ?>" />
   </p>
   <?php
}
```

Con este añadido lo que hago es recoger la información del libro actual en la variable *$values* y, valiéndome de esta, el valor del input ISBN (en este ejemplo) en la variable *$isbn*. Como ves, también insertamos ese contenido recogido en *$isbn* dentro del value del input, lo que mostrará el contenido recogido anteriormente en dicho input.

Añadir un selector desplegable (Select) al metabox

Por razones lógicas, en ocasiones necesitarás darle al editor de la página la posibilidad de seleccionar una opción entre varias, por lo que vamos a añadir a nuestra ficha de libro dos campo select para, por ejemplo, el tipo de tapa de encuadernación de libro (Rústica, pegada o cosida) y para el tipo de tapa (dura o blanda).

```
function libro_meta_box_content( $post ) {
         $values         = get_post_custom( $post->ID );
         $isbn           = isset( $values['info_libro_isbn'] ) ? esc_attr( $values['info_libro_isbn'][0] ) : '';
         $encuadernacion = isset( $values['info_libro_encuadernacion'] ) ?
esc_attr( $values['info_libro_encuadernacion'][0] ) : '';
         $tapa           = isset( $values['info_libro_tapa'] ) ? esc_attr( $values['info_libro_tapa'][0] ) : '';
   ?>
   <p>
      <label for="info_libro_isbn">ISBN</label>
                  <input type="text" name="info_libro_isbn" id="info_libro_isbn" value="<?php echo
esc_html( $isbn ); ?>" />
   </p>
   <p>
      <label for="info_libro_encuadernacion">Tipo de encuadernación</label>
      <select name="info_libro_encuadernacion" id="info_libro_encuadernacion">
                  <option value="rustica" <?php selected( $encuadernacion, 'rustica' ); ?>>Rústica</
option>
                  <option value="pegada" <?php selected( $encuadernacion, 'pegada' ); ?
>>Pegada</option>
                  <option value="cosida" <?php selected( $encuadernacion, 'cosida' ); ?>>Cosida</
option>
      </select>
   </p>
   <p>
      <label for="info_libro_tapa">Tipo de Tapa</label>
      <select name="info_libro_tapa" id="info_libro_tapa">
                  <option value="dura" <?php selected( $tapa, 'dura' ); ?>>Dura</option>
                  <option value="blanda" <?php selected( $tapa, 'blanda' ); ?>>Blanda</option>
      </select>
   </p>
   <?php
}
```

La función va cogiendo volumen, vamos por partes.

- He añadido dos select, como hemos mencionado arriba.
- He añadido, también, dos variables más (*$encuadernacion* y *$tapa*) donde recogo los datos de ambos select. Tal y como hicimos con el input del ISBN.

¿Vas cogiendo la idea? Con este sistema podemos añadir tantos campos como queramos, del tipo que queramos… Ahí va otro ejemplo más:

Añadiendo checkbox al metabox

A continuación añadiré un campo checkbox, que me permitirá marcar si hay stock o no del libro en cuestión.

```
function libro_meta_box_content( $post ) {
        $values      = get_post_custom( $post->ID );
        $isbn        = isset( $values['info_libro_isbn'] ) ? esc_attr( $values['info_libro_isbn'][0] ) : '';
        $encuadernacion = isset( $values['info_libro_encuadernacion'] ) ? esc_attr( $values['info_libro_encuadernacion'][0] ) : '';
        $tapa        = isset( $values['info_libro_tapa'] ) ? esc_attr( $values['info_libro_tapa'][0] ) : '';
        $check       = isset( $values['info_libro_stock'] ) ? esc_attr( $values['info_libro_stock'][0] ) : '';
    ?>
    <p>
        <label for="info_libro_isbn">ISBN</label>
                <input type="text" name="info_libro_isbn" id="info_libro_isbn" value="<?php echo esc_html( $isbn ); ?>" />
    </p>
    <p>
        <label for="info_libro_encuadernacion">Tipo de encuadernación</label>
        <select name="info_libro_encuadernacion" id="info_libro_encuadernacion">
            <option value="rustica" <?php selected( $encuadernacion, 'rustica' ); ?>>Rústica</option>
          <option value="pegada" <?php selected( $encuadernacion, 'pegada' ); ?>>Pegada</option>
            <option value="cosida" <?php selected( $encuadernacion, 'cosida' ); ?>>Cosida</option>
        </select>
    </p>
    <p>
        <label for="info_libro_tapa">Tipo de Tapa</label>
        <select name="info_libro_tapa" id="info_libro_tapa">
            <option value="dura" <?php selected( $tapa, 'dura' ); ?>>Dura</option>
            <option value="blanda" <?php selected( $tapa, 'blanda' ); ?>>Blanda</option>
        </select>
    </p>
    <p>
                    <input type="checkbox" id="info_libro_stock" name="info_libro_stock" <?php checked( $check, 'on' ); ?> />
        <label for="info_libro_stock">Libro disponible en Stock</label>
    </p>
    <?php
}
```

Al igual que con el input y el select, hemos agregado una variable que recoge la información de la base de datos (llamada *$check*) y el propio checkbox.

Creo que la idea se ha entendido; si no es así vuelve al punto donde explico cómo introducir el input text y el select.

Guardando los datos

Hasta ahora, hemos creado la interfaz de usuario: un metabox con varios campos que añaden información adicional a la ficha de los libros.

A ese metabox le hemos dado forma con una función donde, además, hemos dejado listas unas cuantas variables que recogen esa información de la base de datos cada vez que decidamos editar la ficha de un libro.

Pero… ¿En qué momento hemos guardado los datos? ¿Funciona por arte de magia? La respuesta, tras leer el título de esta sección, es obviamente que no; no hemos guardado los datos en ningún sitio, por lo que nuestro metabox todavía no es funcional.

Para guardar los datos necesitamos una función, que vamos a enganchar con el hook de WordPress *save_post*.

No te voy a pedir que entiendas cómo funciona este proceso, pero sí quiero que sepas que eso se hace con la función *add_action* de WordPress:

```
add_action( 'save_post', 'dariobf_metabox_save' );
```

Aún no he programado la función *libro_metabox_save*, que será la encargada de guardar nuestros datos.

Esta función recibirá un argumento, el id del post que estamos editando, y se encargará, como digo, de guardar todos los datos de nuestro metabox en la base de datos.

El hook *save_post* ejecutará nuestra función libro_*metabox_save* siempre pulsemos el botón “Guardar borrador” o “Actualizar” (también “Publicar”) en la página de edición.

Antes de guardar la información, necesito verificar tres cosas:

1. Si la entrada se está autoguardando
2. Verificar el valor nonce que hemos creado en la función *dariobf_metabox_content*
3. Comprobar que el usuario actual puede realmente modificar este contenido.

```
function libro_metabox_save( $post_id ) {
    // Ignoramos los auto guardados.
    if ( defined( 'DOING_AUTOSAVE' ) && DOING_AUTOSAVE ) {
        return;
    }

    // Si no está el nonce declarado antes o no podemos verificarlo no seguimos.
     if ( ! isset( $_POST['bf_metabox_nonce'] ) || ! wp_verify_nonce( $_POST['bf_metabox_nonce'],
'dariobf_metabox_nonce' ) ) {
        return;
    }

    // Si el usuario actual no puede editar entradas no debería estar aquí.
    if ( ! current_user_can( 'edit_post' ) ) {
        return;
    }
}
```

Y, por fin, viene la última parte: guardar los datos.

Como primera regla a la hora de poner cualquier información en una base de datos diría que jamás te fíes del usuario.

Incluso si ese usuario es tu hermano, hermana o padre. Es más, incluso si ese usuario eres tú mismo, nunca te fíes de lo que intentan introducir en tu base de datos.

Por esto, antes de guardar los datos vamos a comprobar que no hay nada malicioso en los datos. Afortunadamente WordPress cuenta con sistemas que controlan esto por nosotros.

Ya hemos utilizado antes la función *esc_attr()*. Esta función codifica las comillas simples y dobles así como los símbolos mayor qué y menor qué en HTML.

Vamos a utilizar la función *update_post_meta* para guardar nuestros datos.

Esta función recibe tres argumentos: el ID del post, la “llave” (key) del meta y el valor.

```
function libro_metabox_save( $post_id ) {
    // Ignoramos los auto guardados.
    if ( defined( 'DOING_AUTOSAVE' ) && DOING_AUTOSAVE ) {
        return;
    }

    // Si no está el nonce declarado antes o no podemos verificarlo no seguimos.
     if ( ! isset( $_POST['bf_metabox_nonce'] ) || ! wp_verify_nonce( $_POST['bf_metabox_nonce'],
'dariobf_metabox_nonce' ) ) {
        return;
    }

    // Si el usuario actual no puede editar entradas no debería estar aquí.
    if ( ! current_user_can( 'edit_post' ) ) {
        return;
    }

    // AHORA es cuando podemos guardar la información.

    // Nos aseguramos de que hay información que guardar.
    if ( isset( $_POST['info_libro_isbn'] ) ) {
        update_post_meta( $post_id, 'info_libro_isbn', wp_kses( $_POST['info_libro_isbn'], $allowed )
);
    }

    if ( isset( $_POST['info_libro_encuadernacion'] ) ) {
                        update_post_meta( $post_id, 'info_libro_encuadernacion',
esc_attr( $_POST['info_libro_encuadernacion'] ) );
    }

    if ( isset( $_POST['info_libro_tapa'] ) ) {
        update_post_meta( $post_id, 'info_libro_tapa', esc_attr( $_POST['info_libro_tapa'] ) );
    }

    // Así es como guardo yo los checkboxes; hay otros métodos, pero este es el mío.
    $check = isset( $_POST['info_libro_stock'] ) && $_POST['info_libro_stock'] ? 'on' : 'off';
    update_post_meta( $post_id, 'info_libro_stock', $check );
}
```

Como antes, los meta boxes deben aparecer en plugins y no en temas, a no ser que el tema sea el definitivo para el site.