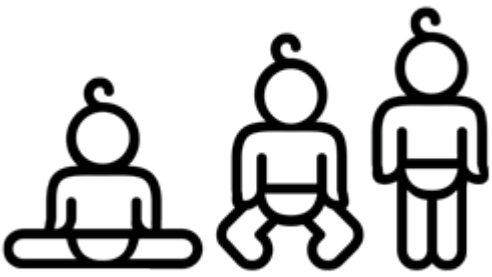
¿Qué es un tema de WordPress?

Otro inciso, ya sabéis lo que tenéis que saber para hacer un tema. Pero… ¿sabéis qué es exactamente un tema? Oh, cielos. Sí, seguro que lo sabéis, pero por si acaso, un tema de WordPress **es lo que nos permite cambiar la apariencia de nuestra página web**. También nos permite cambiar alguna funcionalidad, pero lo principal es el aspecto visual. Es todo esto, pero claro, al final son archivos .php, .css o .js. Eso es lo que es.

La evolución del desarrollador de temas de WordPress

Si eres de los que acabas de aprender los básicos de HTML que os decía hace unos párrafos, es probable que os asuste esta historia de crear un tema de WordPress. "*No puede ser tan fácil pasar de cero a cien*", pensaréis alguno. Efectivamente. Que seguro que alguno sí que termine este post (no creo que nadie lo termine, pero necesito soltarlo todo) aprendiendo un montón, pero yo os voy a decir las fases de la evolución del desarrollador de temas que yo viví.



La **primera vez** que un diseñador te pase un diseño de una web, **dedicarás dos horas a buscar en Themeforest un tema que se parezca lo máximo posible** a lo que tienes entre manos. Te gastarás los 60 dólares correspondientes y después pasarás unas cuantas horas quitando historias y dejando los colores, las imágenes, las columnas, etc., lo más parecido a lo que te han enseñado.

El problema llegará cuando tengas que meter una funcionalidad absurda y no seas capaz de encontrar en qué archivo tienes que hacer la modificación. Y te desesperes porque va lento y no eres capaz de aligerarlo sin romper otra cosa que aparentemente no tenía relación con lo que acabas de hacer.

Cuando te cansas de esto, decides coger uno de los temas que vienen por defecto, como el Twenty Seventeen, o el Twenty Sixteen o incluso el Twenty Thirteen y te planteas **crear un tema hijo**. Has oído que los temas hijos son perfectos para ir modificando cosas y piensas que es la manera de ir aprendiendo. Y, efectivamente, así es. No voy a entrar en detalles

sobre cómo hacer un tema hijo, pero os recomiendo esta charla de **Carla Saiz** en la WordCamp Madrid 2017 sobre el tema (patapan pshh) porque merece mucho la pena y cubre todo lo que se puede hacer y cómo hacerlo.

Claro que llega un momento que ves a tu tema hijo y a su padre y parece que es de otro. Has hecho tantas modificaciones que al final piensas, ¿no habría sido mejor empezarlo de cero? Probablemente. Pero el archivo *untitled* en blanco da mucho miedo. Por eso existen esqueletos, ***starter themes*** **como Underscores** para solventar este problema. ¿Qué es lo que hacen? Te dan una estructura de archivos básica para tu tema, sin ningún estilo -solo los básicos de reset y normalización- para que puedas modificarlo a tu gusto y crear tu propio tema.

Existe también una versión de Underscores que se llama **Components**, donde puedes bajarte ese mismo esqueleto pero con algún estilo de más (para montar una web tipo blog, tipo corporativa, tipo revista, tipo portfolio…) por si también te asusta el tener que maquetarlo todo desde cero. ¡Deja de asustarte!

Finalmente, cuando ya has metido tantas modificaciones a tu Underscores que lo has dejado a tu gusto y prácticamente has creado tu propio framework para empezar proyectos… ya podemos decir que **estás listo para crear tus propios temas desde cero.** Yo considero que hacerlo con _s ya equivale a hacerlo desde cero, pero hay puristas para todo.

Oiga, yo he venido aquí a por el código

Esto pensó más de uno en la charla, pero tampoco era plan de ponerme a meter línea a línea con todas las cosas que quería contar, ¿no?

Está bien, está bien. **Para hacer un tema estaría bien tener un diseño en el que basarse**, ¿no? No sé si llegaremos a conseguir el código exacto para que funcione, pero… digamos que queremos hacer la guía para disfrutar de una WordCamp como debe ser, tanto para novatos como para veteranos. Digamos que queremos que cuatro expertos WordCamperos sean los protagonistas de esta página. Digamos que tenemos un diseño tal que así:

Ahora sí que ya podemos ir trabajando…

Si habéis escuchado la charla de Darío que enlazaba más arriba, tendréis claro que un tema lo que necesita sí o sí, exclusivamente para funcionar es **un index.php y un style.css**. Así que con este código, técnicamente ya tenemos un tema, ¿verdad? Si supongo que tenéis instalado WordPress con una entrada de *¡Hola, mundo!* y habéis activado el tema con estos dos archivos… ¡técnicamente ya lo tenéis! Así que a partir de ahora sólo es ir mejorando cosillas…

```
<?php
        if ( have_posts() ) :
                while ( have_posts() ) : the_post();
                        the_content();
```

```
                endwhile;
            endif;
        ?>
```

index.php hosted with by GitHub view raw

```
/*
Theme Name: WordCamp Guide 2017
Theme URI: http://laguiawordcamp.local
Author: ciudadanob
Author URI: https://ciudadanob.com
Description: El tema de las mejores WordCamps, claro.
Version: 0.0.1
License: GNU General Public License v2 or later
License URI: LICENSE
Text Domain: wordcampguide2017
*/
```

style.css hosted with by GitHub view raw

Además, podemos añadir un **screenshot.png**, una imagen de máximo 1200x900px que es la que sale luego como miniatura en la pantalla de Apariencia > Temas.

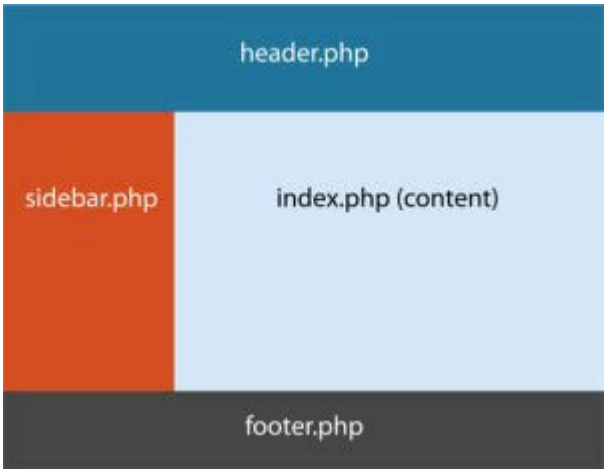
El loop y otros básicos de un tema

Lo que veis en el código anterior tiene fácil explicación. En el index.php tenemos el loop. **El famoso loop de WordPress**. Aquí podéis profundizar sobre el funcionamiento del bucle de WordPress. Hay que comprender cómo funciona un poco WordPress por dentro… por un lado tenemos los archivos que forman parte de la instalación, del core, los plugins, nuestro fantástico tema… y por otro lado hay una base de datos donde se guardan todos los contenidos. El bucle lo que hace es preguntar a la base de datos si hay contenidos que mostrar y si hay algo, lo pinta y vuelve a preguntar por si hay más cosas que pintar.

En la hoja de estilos metemos la información que tiene que estar sí o sí sobre nuestro tema, su nombre, el creador, la licencia, el textdomain para las traducciones…

Pero claro, la web tal y como está ahora no es lo que buscamos, y tampoco vamos a meterlo todo en el mismo archivo index.php, ¿verdad? Efectivamente. Hay otras plantillas básicas en un tema que tenemos que contar con ellas, como son **header.php, footer.php o sidebar.php**.

El concepto es sencillo, todo el código que tenga



que ir en el cabecero de la página y la parte superior -el encabezado-, irá al header.php. El pie de página con su contenido y scripts correspondientes, al footer.php. ¿Hay barra lateral? Su sitio es el sidebar.php. Y el contenido puede seguir estando en el index.php o…

O utilizar cualquiera del **resto de plantillas** que existen para cada caso, como son page.php (para páginas), single.php (para entradas individuales), home.php (para la portada si muestra las últimas entradas), front-page.php (para la portada si es una página estática), archive.php (para los archivos de categorías, etiquetas, etc.…).

¿Cómo sabe WordPress que plantilla tiene que utilizar para cada caso cuando varias valdrían para la misma página? Aquí entra la jerarquía de plantillas, cuanto más específica sea, mayor prioridad tendrá para ser la utilizada. Aquí tenéis toda la información de la jerarquía de plantillas en el Codex, y aquí el famoso gráfico que todo desarrollador acaba consultando para saber qué manda sobre qué. Como podéis ver, si no existe ninguna, al final siempre acaba apareciendo al rescate el index.php, por eso es la única obligatoria.

¿Cómo trocear entonces un diseño?

Una vez que hemos decidido cuál va a ser la estructura de nuestra web y dónde vamos a ir colocando cada trozo de nuestro código, hay que pasar al troceado literal, ¿no? Si estáis utilizando una herramienta como Zeplin que os recomendé antes, tenéis gran parte del trabajo hecho. Si no, simplemente tendréis que **exportar las imágenes de la manera más optimizada posible** para web y más ajustada para que el diseño vaya cuadrando píxel a píxel.

Aquí ya podemos entrar en diversas situaciones, como querer utilizar un logo o unos iconos en **formato SVG**, un formato vectorial que hace que no se pierda calidad al modificar el tamaño del archivo, o si vamos a tener en cuenta las **pantallas Retina** que nos harán guardar todas nuestras imágenes a @2x, el doble del tamaño del esperado, para que se ajusten bien a pantallas con el doble de resolución. Esto (como casi todo en esta entrada) da para un post individual, así que no me enrollaré, pero sabed que hay que tenerlo en cuenta.

Además de las imágenes, tendremos que tener en cuenta la **paleta de colores** a utilizar (en

formato HEX y RGB, por si necesitamos hacer transparencias con alpha), las **fuentes con sus tamaños y grosores** para cada apartado, y la estructura de la página, **si se utiliza o no un grid**, una retícula predefinida (x columnas de x píxeles con tanto margen) o si simplemente adecuaremos el contenido que tengamos al ancho máximo mediante CSS con flexbox.

Una vez que lo tenemos todo listo, las imágenes en una carpeta /img y todo bien apuntadito para ir maquetando vía CSS, podemos seguir adelante…

Mobile first y media queries

Uno de los mayores cambios de estos últimos años es el *mobile first*. Es algo que se tiene que tener en cuenta desde el momento del diseño, a ser posible, pero si no, es fundamental a la hora del desarrollo. ¿Qué quiere decir? Que en vez de crear la web en modo escritorio y luego ir haciendo el navegador pequeño y reorganizando todo el contenido hasta tenerlo todo en una única columna, **hay que hacerlo al revés**.

¿Por qué es esto? **¿No da lo mismo?** Es un concepto interesante, técnicamente es similar, pero requiere algo más de pensamiento en la parte móvil, cosa que no se suele hacer. Y por eso terminamos con menús de hamburguesa que no funcionan bien, con elementos que apenas se pueden pulsar sin pinchar en varios, con sliders diminutos que no aportan nada y no se ve su contenido, con barras laterales que en el escritorio tienen un call to action fundamental para la web y en móvil lo hemos mandado debajo de todo el contenido, a kilómetros de scroll para nuestro visitante…

¿Y cómo sería? Veamos un simple ejemplo de cómo quedaría nuestra hoja de estilos si quisiéramos que en pantallas pequeñas (hasta 640px) el cuerpo de la página tuviera un padding lateral de 16px, y a partir de que crezca, ya no haga falta -porque tendremos capas contenedoras centradas con su propio padding, por ejemplo):

```
/*
Theme Name: WordCamp Guide 2017
Theme URI: http://laguiawordcamp.local
Author: ciudadanob
Author URI: https://ciudadanob.com
Description: El tema de las mejores WordCamps, claro.
Version: 0.0.1
License: GNU General Public License v2 or later
License URI: LICENSE
Text Domain: wordcampguide2017
```

```
*/

body {
  /* Se aplica de 0px a 640px de ancho */
  background: #FFF;
  color: #495057;
  font-family: sans-serif;
  font-size: 16px;
  font-size: 1rem;
  line-height: 1.5;
  padding: 0 1rem;
}

@media (min-width: 40rem) {
  /* Se aplica para un ancho mayor a 640px */
  body {
    padding: 0;
  }
}
```

style.css hosted with by GitHub view raw

Organización y template-parts

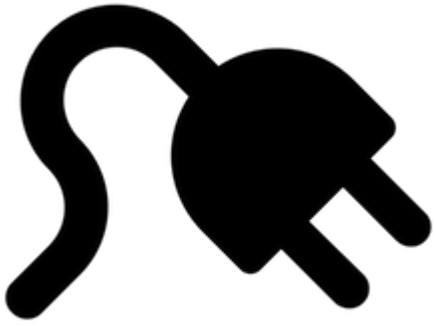
Vale, ya tenemos claro cómo vamos a separarlo todo, tenemos los archivos, tenemos la motivación… Así que os voy a interrumpir para contar una anécdota sobre por qué hacen falta **template-parts**. Que para los que vienen del PHP o de cualquier otro lenguaje de programación, vienen a ser un ***include*** de toda la vida, pero versión WordPress.

Hace no mucho tuve que desarrollar a partir de un diseño una web de un local de conciertos. En la portada hacía falta meter al final un listado con los próximos eventos. Lo programé y lo añadí en *page-portada.php*. Todo correcto. Después me dijeron que había que meter esa parte también en la página de eventos. Ni corto ni perezoso, copy, paste, solucionado. Oye, también en ésta sobre el local. Paste. Estupendo. Una semana después había que hacer modificaciones en el código, porque no habíamos tenido en cuenta que no todos los conciertos tenían entrada, sino que algunos podían ser gratuitos. Modificar código. Copiar y pegar en los otros dos lados. Un mes después surgió un evento que no era sólo de un día, sino que duraba varios. Modificar código. Esto… ¿dónde estaba esto también? Creo que aquí… Pegar…

Creo que entendéis todos lo poco óptimo que esto y los problemas que conlleva cuando dejas de acordarte (porque para qué lo iba a apuntar) en qué página estaba el código. **Para esto existen los template-parts**. Creas un archivo aparte con el trozo de código necesario y lo invocas desde donde necesites mediante la función get_template_part(). Ahora sólo tienes

que modificarlo una vez y… magia.

Functions.php y plugins



Tal vez os extrañe que aún no hayamos hablado del **archivo functions.php**, cuando es la solución a todos los males y el lugar donde te llevan todas las respuestas que encontréis en Google sobre cómo arreglar lo que sea.

Este archivo contiene funciones que permiten modificar funcionalidades o aspectos de nuestro tema, por eso es tan importante y hay tantos códigos que nos permiten modificarlo todo desde aquí. Ya sea mediante funciones creadas a propósito y luego enganchadas en la cabecera o pie de nuestra web, o mediante ganchos, ya sean de acciones o de filtros. Esto último es muy interesante para ir haciendo cambios, pero os recomiendo esos dos artículos de Pablo López sobre ellos muy bien explicados.

¿Y qué hay que tener en cuenta para esto? Pues que no hay que abusar del functions.php. Por norma general metemos todo ahí y funciona sin problemas, claro. Pero ¿qué pasa si el día de mañana decidimos cambiar de tema a otro más bonito o más útil que hemos creado o que hemos encontrado por ahí? Pues que perderemos toda esa funcionalidad al desactivar el tema actual. ¡Oh cielos! Por eso hay que tener en cuenta que lo que metamos en este archivo tiene que estar relacionado con el tema que vamos a utilizar, y si pensamos que debiera estar ahí (un tipo de contenido personalizado, por ejemplo) aunque no usemos ese tema, lo más correcto es añadirlo en un plugin y activarlo. De esta manera estará ahí, sea cual sea el tema que usemos.

¿Pero hacer un plugin no es muy complicado? Oh, no, nada de eso… Igual que los temas, con muy poco podemos ir arrancando y según vayamos aprendiendo y mejorando, iremos modificando nuestro plugin para usar más opciones, modularizarlo, etc, etc. Por ejemplo, si queremos crear aparte de las entradas y las páginas un tipo de contenido que sea Recomendaciones, podemos hacerlo de esta manera:

```php
<?php
/**
* Plugin Name: WordCamp Guide Recommendations
* Description: Un plugin para crear un custom post type y no meterlo en el functions.php
* Author: Ciudadanob
* Version: 0.1
*/
function crear_post_type() {
```

```
        register_post_type( 'recomendaciones',
                array(
                        'labels' => array(
                                'name' => __( 'Recomendaciones' ),
                                'singular_name' => __( 'Recomendación' )
                        ),
                        'public' => true,
                        'has_archive' => true,
                        'rewrite' => array('slug' => 'recomendaciones'),
                )
        );
}
add_action( 'init', 'crear_post_type' );
?>
```

wc-guide-recommendations.php hosted with by GitHub view raw

Enqueue scripts & styles

Al hacer una web en HTML solemos llamar a pelo desde el head de nuestro código a las hojas de estilo CSS y a los archivos JavaScript que necesitamos. Eso en WordPress es una mala práctica. ¿Por qué? Porque tenemos dos funciones que nos permiten hacerlo de manera correcta y, sobre todo, que permiten a otros desarrolladores que utilicen nuestros temas o plugins, desenganchar nuestros archivos y cambiarlos por otros o por los suyos.

Aquí tenéis toda la información del Codex sobre cómo incluir archivos CSS y JS en nuestros desarrollos. Y aquí un par de ejemplos si, por ejemplo, hacemos lo que recomienda WordPress para cargar nuestra hoja de estilos y cargar sólo el script de comentarios donde se pueden hacer comentarios, y también si quisiéramos cargar una biblioteca externa como puede ser prettyPhoto de JavaScript para hacer un efecto de lightbox en las imágenes de la página de multimedia. Tenemos un archivo para inicializar el script y luego cómo quedaría nuestro functions.php.

```
jQuery(document).ready(function(){
  jQuery("a[rel^='prettyPhoto']").prettyPhoto({
    social_tools: false,
  });
});
```

custom-prettyPhoto.js hosted with by GitHub view raw

```
<?php
function incluir_scripts() {
  // Cargamos nuestra hoja de estilos con esta función que da directamente la URI del style.css
  wp_enqueue_style( 'style', get_stylesheet_uri() );
```

```
  // Cargamos el script para los comentarios sólo en las páginas o entradas que tengan abiertos los come
  if ( is_singular() && comments_open() && get_option( 'thread_comments' ) ) {
    wp_enqueue_script( 'comment-reply' );
  }

  // Cargamos el JS y el CSS para abrir un lightbox de prettyPhoto en nuestra página de multimedia.
  // Subimos los archivos correspondientes a la carpeta /js/prettyPhoto/
  if ( is_page('multimedia') ) {
          wp_enqueue_script( 'wcguide-prettyphoto', get_stylesheet_directory_uri() . '/js/prettyPhoto/jc
);
          wp_enqueue_script( 'wcguide-prettyphoto-init', get_stylesheet_directory_uri() . '/js/prettyPho
          wp_enqueue_style( 'wcguide-prettyphoto', get_stylesheet_directory_uri() . '/js/prettyPhoto/pre
        }
}
add_action( 'wp_enqueue_scripts', 'incluir_scripts' );
```

functions.php hosted with by GitHub view raw

Durante la charla hablé de utilizar esto para cargar los scripts del plugin Contact Form 7, muy usuado a nivel global, sólo en la página de contacto. Revisando la documentación del plugin, ahora hay una manera diferente de hacerlo, por si queréis echarle un ojo, mediante un par de filtros y un par de funciones.

Imágenes y thumbnails

Uno de los problemas clásicos al pasar de un diseño a un tema es que nos volvamos locos con los tamaños de las imágenes. A veces se terminan cargando imágenes gigantes (en píxeles y en megabytes) para necesidades muy pequeñas del diseño como pueden ser miniaturas. Si tenemos bien seleccionados los tamaños de las miniaturas que vamos a necesitar, los podremos usar más adelante en nuestras plantillas y nos ayudará a que todo esté más optimizado.

Gracias a la función add_image_size() podemos generar las imágenes que justo queramos. Toda la info sobre esta función, claro, en su sección del Codex.

```
<?php
// Añadimos soporte para thumbnails y creamos dos miniaturas que luego podremos utilizar vía
the_post_thumbnail()
add_theme_support( 'post-thumbnails' );
// Miniatura de 1200 píxels de ancho por 300 de alto
add_image_size( 'imagen-hero', 1200, 300 );
// Miniatura de 250px cuadrada que es recortada desde arriba a la izquierda
```

```
add_image_size( 'imagen-secciones', 250, 250, array( 'left', 'top' ) );
```

functions.php hosted with by GitHub view raw

El menú

No caigamos nunca en la tentación de generar un menú a mano aunque sean sólo un par de elementos, porque a la larga pueden ir creciendo y, además, si WordPress te ofrece una función para hacerlo automáticamente, es buena idea utilizarlo. En este caso tenemos wp_nav_menu() que nos permite utilizar después el elemento de Apariencia > Menú del administrador para crear nuevos menús (que hemos tenido que registrar previamente) y gestionarlos.

Si por ejemplo queremos que en nuestra web haya un menú superior para luego irle añadiendo páginas que vayamos creando, tendríamos que utilizar este código.

```
<?php
// Registramos un menú para el tema
register_nav_menus( array(
  'menu-1' => esc_html__( 'Main Menu', 'wordcampguide2017' ),
) );
```

functions.php hosted with by GitHub view raw

Usar el personalizador: el logo

Hasta hace no mucho, la mayoría de los temas premium tenían una página de opciones desde la que hacer todo tipo de modificaciones al tema: colores, logos, imágenes, fondos, código personalizado, etc… Desde hace un tiempo y cada vez más este 2017 **se está potenciando el uso del personalizador integrado en WordPress** para todas estas cosas. Se puede acceder desde el menú Apariencia > Personalizar y ahí se abren una serie de paneles para modificar nuestro tema. Podéis probar con cualquiera de los que vienen por defecto.

Pero nosotros también podemos utilizarlo sin ningún problema para nuestro tema. Por ejemplo, si queremos utilizar el logo que desde ahí subamos para ponerlo en la cabecera de nuestra web, simplemente tenemos que añadir este código al functions.php. De esta manera hemos añadido la compatibilidad de nuestro tema con la imagen del sitio que viene por defecto en el personalizador.

```
<?php
```

```php
function logo_personalizado() {
	add_theme_support( 'custom-logo', array(
		'height'      => 48,
		'width'       => 134,
		'flex-width' => true,
	) );
}
add_action( 'after_setup_theme', 'logo_personalizado' );
```

functions.php hosted with by GitHub view raw

Si queréis añadir nuevos paneles, nuevas opciones o lo que se os ocurra, **existe la API del personalizador de temas** con toda la información en la documentación sobre ella.

Añadir sidebars: los widgets

Las barras laterales en nuestra cabeza está claro lo que son, barras laterales a la derecha o a la izquierda del contenido principal donde metemos el contenido… menos importante casi siempre. Pero en WordPress no tiene por qué ser así, pueden ser lo que nosotros queramos que sean. Una de las ventajas que tienen es que en ellas **podemos meter widgets** (desde Apariencia > Widgets) y sacarlos mucho partido, ya sean los widgets que vienen por defecto en WordPress, los que aparecen al instalar ciertos plugins o incluso widgets personalizados que podemos crear para la ocasión.

Y podemos colocarlo **allá donde queramos** nosotros en nuestro código y que **tengan la apariencia que nosotros queramos** con nuestros estilos CSS. Por ejemplo, en muchos temas la zona del footer tiene diferentes sidebars que equivalen a diferentes columnas donde podemos poner un cuadro de búsqueda, un texto plano o un widget que muestra las entradas más recientes o los últimos comentarios publicados. ¿Y qué necesitamos para tener una de estas sidebars en nuestro tema? Simplemente declararlo en nuestro functions.php y aparecerá mágicamente en Apariencia > Widgets para que lo gestionemos como queramos.

```php
<?php
// Añadimos una barra lateral
function wordcampguide2017_widgets_init() {
	register_sidebar( array(
		'name'          => esc_html__( 'Sidebar', 'wordcampguide2017' ),
		'id'            => 'sidebar-1',
		'description'   => esc_html__( 'Add widgets here.', 'wordcampguide2017' ),
		'before_widget' => '<section id="%1$s" class="widget %2$s">',
		'after_widget'  => '</section>',
		'before_title'  => '<h2 class="widget-title">',
		'after_title'   => '</h2>',
	) );
```

```
}
add_action( 'widgets_init', 'wordcampguide2017_widgets_init' );
```

functions.php hosted with by GitHub view raw

No olvidemos los tags condicionales

En este repaso a todo lo que podemos hacer con nuestro tema, hay un apartado que nos va a salvar la vida en muchas ocasiones: los tags condicionales. Uno de los ejemplos más claros ocurren con el título de nuestra web. Casi todos estaremos de acuerdo en que el título debería ir entre etiquetas <h1> en la portada de la web, ya que es el término más importante de la misma (hola amigüitos del SEO). Pero claro, sí en nuestro header.php definimos nuestro título entre etiquetas <h1>… ¡sería así en todas las páginas!

Por eso existen los tags condicionales que nos permiten decir… si es la portada, escríbelo entre h1, y si es cualquier otra página, mételo dentro de un div, o un span, o un lo que sea… ¿Y a nivel de código cómo se hace esto? ¿Y qué tipos de condicionales hay? Prácticamente todos los que se os ocurran, aquí tenéis el listado del Codex. Destacan algunos como is_home(), is_front_page(), is_admin(), is_single(), is_page(), is_category(), in_category()…

```
<?php
if ( is_front_page() && is_home() ) : ?>
  <h1 class="site-title"><a href="<?php echo esc_url( home_url( '/' ) ); ?>"><?php bloginfo( 'name' ); ?>
</h1>
<?php else : ?>
  <p class="site-title"><a href="<?php echo esc_url( home_url( '/' ) ); ?>"><?php bloginfo( 'name' ); ?><
</p>
<?php endif;
```

header.php hosted with by GitHub view raw

Los campos personalizados

A veces no todos los contenidos que tenemos para introducir encajan en el recuadro de escribir el contenido –el the_content()– y conviene utilizar los campos personalizados. Yo soy declarado fan de ACF Pro (Advanced Custom Fields), pero si queréis conocer todas las opciones que ofrece WordPress para esto, os recomiendo ver la charla que dio Mauricio Gelves en la WordCamp Madrid 2017 sobre "**Boxeo de campos personalizados**" para ver quién gana la batalla.

A la hora de trabajar con clientes está genial que ellos sólo tengan que ir rellenando campos y nosotros poder mostrarlo en nuestro tema personalizado **utilizando la función the_field()**.

Una maravilla.

Internacionalización y localización

Esta última parte de la presentación ya son las típicas cosas que **tienes que tener en cuenta desde el principio… pero una vez que ya controlas todo lo demás**. Es decir, son muy importantes, pero si te centras en ellas sin aprender la base, te vas a volver loco y es probable que te pierdas y acabes cayendo de nuevo en las redes de ThemeForest…

La **internacionalización (I18N) y la localización (L10N)** tienen sus páginas en el theme handbook y se refieren tanto a la capacidad de permitir que nuestro tema se traduzca a cualquier idioma que se quiera traducir, como al hecho de hacer una traducción acorde al idioma y a la cultura de las personas que hablen ese lenguaje.

Este fantástico post de Pascal Birchler, que dio una charla precisamente sobre internacionalización en la WordPress Bilbao el mismo día que yo, explica con pelos y señales todo lo que tenéis que saber sobre este tema y por qué es importante. Desde las diferencias entre culturas que muchas veces damos por supuesto que no serán, a herramientas para que nuestro código esté correctamente preparado para ser traducido.

Accesibilidad

Si hay algo que muchas veces pasamos por encima por desconocimiento o vagancia desarrollando un tema es la accesibilidad. Que por supuesto tiene su apartado en el theme handbook. Pero que no es sólo importante "*porque ayuda a mejorar el SEO*", sino porque gente con algún tipo de discapacidad, incluso temporal (¿te has roto un brazo y has tenido que navegar en alguna web móvil? ¿has tenido vista cansada e intentado leer una web con una fuente de 12 píxeles?) necesita poder utilizar bien tu web.

Yo reconozco que hasta hace no mucho lo que sabía de accesibilidad lo sabía de oídas, los típicos consejos sobre poner texto alternativo a las imágenes, utilizar los headings de h1 a h6 y no hacer *trapalladas* como dicen aquí en Galicia. Pero **después de ver esta charla de Juanjo Montiel en el Front Fest 2017 cambió mi percepción de la accesibilidad un montón**. Os lo recomiendo por encima de todo lo que podáis ver y aprender en este post. Vedlo. Y alucinad.

Seguridad

¿La seguridad aquí abajo? ¡Madre mía! Lo sé, lo sé, pero como ya he dicho antes, éstas son las cosas que hay que tener en cuenta desde el principio una vez tenemos claro todo lo que vamos a hacer para crear nuestro tema, si no es probable que lo dejemos a mitad de camino. ¿Cuándo hablamos de seguridad en un tema de WordPress de qué hablamos exactamente? Esto no es como los bugs del core o los problemas de algún plugin… Estamos hablando de utilizar buenas prácticas de desarrollo de temas, y eso trae consigo la validación, sanación y escape de todos los datos que integremos en el tema. **Validating, sanitizing and escaping**, en inglés que es como lo veréis más a menudo.

Esto quiere decir que si tenemos por ejemplo un formulario que va a guardar los datos que se envíen en nuestra base de datos, nos aseguremos primero de que si lo que tienen que poner es una dirección de correo electrónico, lo que el usuario introduzca en el campo sea una dirección de correo electrónico. Y no una orden para ejecutar un script que nos borre la base de datos de nuestra web, por ejemplo. O cualquier otra maldad que se nos ocurra. En serio, es algo a tener muy en cuenta.

¿Y si tiene WooCommerce?

A estas alturas ya habréis visto que en la documentación oficial de WordPress está todo muy bien explicadito para hacer un tema con un montón de buenas prácticas. Pero ¿qué pasa si queremos que nuestro tema quede integrado con algún plugin de los más utilizados? ¿Es posible? Por supuesto que sí. Y no es nada difícil, ya que **la mayoría de plugins tienen una gran documentación** que nos facilita la tarea (hay que leer un poco, pero si has llegado hasta aquí no pongo en duda tu capacidad).

En el caso de WooCommerce tenemos la explicación de cómo hacerlo en la página Third party / custom / non-WC theme compatibility, y también podemos aprender sobre la gestión de plantillas de WooCommerce y su estructura en Template Structure + Overriding Templates via a Theme.

Si, por ejemplo, en nuestro tema queremos tener preparada una sección de tienda, no tendríamos más que utilizar el siguiente código para tener un tema *WooCommerce ready*.

```
<?php
// Desenganchamos los hooks que marcan el inicio estándar de la zona de WooCommerce en un tema
remove_action( 'woocommerce_before_main_content', 'woocommerce_output_content_wrapper', 10);
remove_action( 'woocommerce_after_main_content', 'woocommerce_output_content_wrapper_end', 10);
// Enganchamos nuestros propios hooks con el código entre el que irá la tienda y sus páginas
```

```php
add_action('woocommerce_before_main_content', 'wordcampguide2017_wrapper_start', 10);
add_action('woocommerce_after_main_content', 'wordcampguide2017_wrapper_end', 10);
function wordcampguide2017_wrapper_start() {
  echo '<div id="main">';
}
function wordcampguide2017_wrapper_end() {
  echo '</div>';
}
// Declaramos que nuestro tema soporta WooCommerce mediante esta función
add_action( 'after_setup_theme', 'woocommerce_support' );
function woocommerce_support() {
    add_theme_support( 'woocommerce' );
}
```

functions.php hosted with by GitHub view raw

Conclusión final y presentación

Como podéis ver, el tema de la creación de temas **da para horas y horas de aprendizaje, y siempre quedan cosas por seguir investigando y mejorando**. Cada uno de los apartados por los que he pasado por encima da para una charla de una hora tranquilamente, pero como no podía estar 20 horas hablando sin parar, he decidido hacer este post con un montón de enlaces para profundizar sobre todo lo comentado.

También sé que podría haber hecho quince posts diferentes menos largos y entrando un poquito más en cada asunto, pero ya lo siento, nunca he tenido visión de aprovecharme del SEO o de ir manteniendo a mis lectores aquí pillados leyendo las cosas cachito a cachito. Aquí está todo y **si creéis que faltan cosas, no dudéis de añadirlas** en los comentarios.

Para finalizar, os dejo más o menos **cómo quedaría el código de la página de portada del tema para la web** de la guía de disfrutar las WordCamps, quién sabe, igual algún día me animo y lo creo como web (más o menos) seria… Hay que tener en cuenta que desde el administrador de WordPress he creado las páginas correspondientes para el menú, he subido un logo desde el personalizador, he puesto el título y el subtítulo, he añadido una imagen destacada a la página de Portada, he añadido un widget con una frase al sidebar y he creado una serie de campos personalizados con ACF para la página de portada para meter el texto principal y el de cierre, así como un repeater para generar las columnas con imagen, título y subtítulo.

```php
<div class="hero">
        <?php the_post_thumbnail('imagen-hero');?>
        <div class="texto-hero">
                <?php the_field('texto_principal');?>
        </div>
```

```
</div>

<?php
if( have_rows('bloques') ): ?>
<div class="bloques">
  <?php while ( have_rows('bloques') ) : the_row(); ?>

        <div class="columna">

        <?php
        $image = get_sub_field('imagen');
        if( !empty($image) ): ?>
                <img src="<?php echo $image['url']; ?>" alt="<?php echo $image['alt']; ?>" />
        <?php endif; ?>

    <h3><?php the_sub_field('titulo');?></h3>
    <p><?php the_sub_field('subtitulo');?></p>

   </div>

  <?php endwhile;?>
</div>
<?php endif;
?>
```

content-portada.php hosted with by GitHub view raw

```
<?php
// El pie de nuestro tema, donde simplemente pondremos un sidebar con un widget y un campo personalizado
frase
?>

        </div><!-- #content -->

        <footer class="site-footer">
                <?php if ( ! is_active_sidebar( 'sidebar-1' ) ) {
      return;
    }
    ?>

    <div class="widget-area" role="complementary">
      <?php dynamic_sidebar( 'sidebar-1' ); ?>
    </div>

    <div class="site-end">
      <?php the_field('frase_footer', 19); // Utilizamos el ID de la página de portada que hemos creado
    </div>
```

```
        </footer>

<?php wp_footer(); ?>

</body>
</html>
```

footer.php hosted with by GitHub view raw

```
<?php
// Añadimos soporte para un menú
register_nav_menus( array(
        'menu-1' => esc_html__( 'Main Menu', 'wordcampguide2017' ),
) );
// Añadimos soporte para thumbnails y creamos dos miniaturas
add_theme_support( 'post-thumbnails' );
// Miniatura de 1200 píxels de ancho por 300 de alto
add_image_size( 'imagen-hero', 1000, 300, true );
// Miniatura de 250px cuadrada que es recortada desde arriba a la izquierda
add_image_size( 'imagen-secciones', 250, 250, array( 'left', 'top' ) );
// Añadimos una barra lateral para widgets
function wordcampguide2017_widgets_init() {
        register_sidebar( array(
                'name'          => esc_html__( 'Sidebar', 'wordcampguide2017' ),
                'id'            => 'sidebar-1',
                'description'   => esc_html__( 'Add widgets here.', 'wordcampguide2017' ),
                'before_widget' => '<section id="%1$s" class="widget %2$s">',
                'after_widget'  => '</section>',
                'before_title'  => '<h2 class="widget-title">',
                'after_title'   => '</h2>',
        ) );
}
add_action( 'widgets_init', 'wordcampguide2017_widgets_init' );
// Utilizamos la función del logo personalizado desde el personalizador
function logo_personalizado() {
        add_theme_support( 'custom-logo', array(
                'height'      => 48,
                'width'       => 134,
                'flex-width' => true,
        ) );
}
add_action( 'after_setup_theme', 'logo_personalizado' );
// Cargamos una fuente de Google Fonts
function wordcampguide2017_load_fonts() {
        wp_register_style('open-sans-font', 'http://fonts.googleapis.com/css?
family=Open+Sans:300,400,700');
```

```
        wp_enqueue_style( 'open-sans-font');
}
add_action('wp_enqueue_styles', 'wordcampguide2017_load_fonts');
```

functions.php hosted with by GitHub view raw

```
<?php
// La cabecera de nuestro tema
?>

<!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="http://gmpg.org/xfn/11">

<?php wp_head(); ?>
</head>

<body <?php body_class(); ?>> <!-- https://developer.wordpress.org/reference/functions/body_class/ -->

        <header class="site-header">
                <?php the_custom_logo(); ?> <!-- Añadimos nuestro logo desde el personalizador -->
                <?php
    if ( is_front_page() && is_home() ) : ?>
      <h1 class="site-title"><a href="<?php echo esc_url( home_url( '/' ) ); ?>"><?php bloginfo( 'name'
</h1>
      <h2 class="site-description"><?php bloginfo( 'description' ); ?></h2>
    <?php else : ?>
      <p class="site-title"><a href="<?php echo esc_url( home_url( '/' ) ); ?>"><?php bloginfo( 'name' )
</p>
      <p class="site-description"><?php bloginfo( 'description' ); ?></p>
    <?php endif; ?>

                <nav class="main-navigation">
                        <?php wp_nav_menu( array( 'theme_location' => 'menu-1', 'menu_id' => 'main-menu'
                </nav>
        </header>

        <div id="content" class="site-content">
```

header.php hosted with by GitHub view raw

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

```php
<?php
	if ( have_posts() ) :
		while ( have_posts() ) : the_post();
			the_content();
		endwhile;
		the_posts_navigation();
	endif;
?>

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

index.php hosted with by GitHub view raw

```php
<?php
/**
 * Template Name: Página de portada
 * Funciona automáticamente para la página creada con título y slug Portada
 */
get_header();
get_template_part( 'content-portada' ); // Hacemos una llamada a otro archivo por si queremos modularizar
get_footer();?>
```

page-portada.php hosted with by GitHub view raw

```css
/*
Theme Name: wordcampguide2017
Theme URI: http://laguiawordcamp.local
Author: ciudadanob
Author URI: https://ciudadanob.com
Description: El tema de las mejores WordCamps, claro.
Version: 1.0.0
License: GNU General Public License v2 or later
License URI: LICENSE
Text Domain: wordcampguide2017
*/

body {
  background: #FFF;
  color: #495057;
  font-family: 'Open Sans', sans-serif;
  font-size: 16px;
  font-size: 1rem;
  line-height: 1.5;
  padding: 0 1rem;
}
```

```
a,
a:visited {
	color: #0C8599;
	text-decoration: none;
}

img {
	height: auto;
	max-width: 100%;
}

.custom-logo {
	float: none;
	margin: 0 1.25rem;
}

.site-title {
	font-size: 2rem;
	font-weight: 700;
	line-height: 1em;
	margin: 1.25rem 0 0;
}

.site-description {
	font-size: 1.25rem;
	line-height: 1em;
	margin: 0.25rem 0 0;
}

.main-navigation ul {
	display: flex;
	flex-flow: row wrap;
	justify-content: flex-start;
	list-style: none;
	padding: 0;
}

.main-navigation li {
	margin: 0 1rem 0 0;
	text-transform: uppercase;
}

.site-content {
	margin: 0 auto;
	max-width: 1000px;
	width: 100%;
}
```

```
.hero {
        position: relative;
}

.texto-hero {
        margin: 1rem 0 3rem;
        text-align: center;
}

.bloques {
        display: flex;
        flex-flow: row wrap;
        justify-content: space-between;
        margin: 5rem 0;
}

.columna {
        text-align: center;
        width: 150px;
}

.widget-area {
        background-color: #0C8599;
        color: #FFF;
        padding: 4rem 0;
        text-align: center;
}

.site-end {
        padding: 2rem 0;
        text-align: center;
}

@media (min-width: 40rem) {
  body {
    padding: 0;
  }

  .custom-logo {
                float: left;
        }

        .main-navigation {
                float: right;
        }
```

```
        .texto-hero {
                background-color: rgba(255,255,255,0.8);
                left: 0;
                margin: 3rem auto;
                padding: 1rem;
                position: absolute;
                right: 0;
                top: 3rem;
                width: 550px;
        }
}
```

style.css hosted with by GitHub view raw