DISEÑO DE UN PLUGIN PERSONALIZADO PARA WORDPRESS

Juan Luis Talavera Peláez
Desarrollador web

ÍNDICE

1. Introducción

Wordpress es hoy en día la herramienta de creación de webs más utilizada, con más del 60% del total que existe en la web.
Esto es debido a la facilidad que las personas pueden crear su sitio web sin necesidad de saber programar con código y lo intuitivo que es su panel. Wordpress cuenta además con una gran comunidad, la cual trabaja desde el núcleo de este CMS hasta los temas y los plugins que se utilizan para ofrecer al usuario una buena experiencia.

Aún así, tenemos que tener en cuenta que Wordpress tiene límites. La mayoría de plugins que se pueden instalar de forma gratuita cubre la mayoría de la demanda de los desarrolladores. Aún así hay funcionalidades que como desarrolladores necesitamos que nuestra web tenga pero nadie ha programado un plugin o tema que pueda satisfacer esa necesidad. En estos casos necesitamos crear y desarrollar un plugin desde cero por nosotros mismos.

El caso que se presenta a continuación consiste en la creación de un formulario que nos permite recoger cierta información de los clientes antes de realizar la compra o reserva de un producto o servicio. Una vez relleno y pulsado el botón de *submit*, automáticamente se nos añadirá el correspondiente producto al carrito de Woocommerce y se nos llevará a la pantalla del checkout, donde terminaremos la compra del producto.

A simple vista parece ser un proceso sencillo, pero veremos más adelante que esto no es así.

Durante el desarrollo se usará PHP y las funciones y características propias que nos ofrece Wordpress en su entorno de trabajo, como por ejemplo los *hooks*, los cuales tendrán un papel fundamental durante todo el proceso.

2. Conceptos fundamentales

Wordpress es un CMS (Sistema de gestión de contenidos) realizado con el lenguaje de programación PHP. Wordpress se nutre de la comunidad a través de temas y plugins, éstos últimos son los que son relevantes para este caso real.

Un plugin no es más que un trozo de código que añade funcionalidad a Wordpress. Estos plugins se pueden descargar e instalar desde nuestro panel de Wordpress en la sección 'Plugins'.

Un plugin muy importante y que utilizaremos en este caso es Woocommerce, el cual es el plugin más utilizado en Wordpress para la creación de un E-commerce o una tienda online, el cual tendrá una relación muy estrecha con el plugin que creemos.

Para esta relación utilizaremos los denominados *hooks*, los cuales nos sirven para llamar a una función cuando se realice una determinada acción dentro de Wordpress, en este caso, comprar un producto.

En este caso también se han usado los librerías de Javascript: Jquery y Ajax: Jquery nos permite añadir funciones cuando se producen acciones en el DOM. Por ejemplo, pinchar en un botón determinado. Por otro lado, Ajax nos permite comunicar PHP con Javascript, por lo que podremos comunicar el front-end con el back-end.

Además, se ha usado la librería Bootstrap para añadir estilos tanto al formulario en el front-end como al panel de administración del plugin.

Luego, para poder incluir nuestro formulario personalizado en nuestra web, necesitamos los *shortcodes*, los cuales contituyen una abreviación del código con el que podremos indicar exactamente en qué parte de nuestra web queremos el formulario.

3. Finalidad del proyecto

La finalidad del presente proyecto es la de crear un plugin que pueda satisfacer la demanda de un cliente del sector de la animación infantil, ya que los plugins de los que se dispone no añade la funcionalidad que necesitamos para el sitio web del cliente.

El cliente necesita una web en la que sus usuarios finales rellenen un formulario con un archivo PDF con los datos de los participantes en las actividades (antes del checkout), para que acto seguido se añada el producto en el carrito y se proceda al pago del importe. Para esta finalidad, se necesita un plugin personalizado con el que poder incluir el formulario con los campos que el cliente necesita y se pueda conectar con el pedido realizado.

4. Documentación para el usuario

A continuación se explica cómo el usuario (o el cliente en este caso) debe utilizar el plugin:

Al instalarse en Wordpress y activarlo, el plugin crea automáticamente la tabla Wp_formularios (**en caso de que el prefijo sea 'Wp_'**)en la base de datos.

Debe tenerse en cuenta que es necesario que tengamos instalado en nuestro Wordpress el plugin **Woocommerce** y **Woocommerce Checkout Manager**, ambos se pueden descargar e instalar desde la sección 'Plugins' en nuestro panel de administración de Wordpress.

En **Woocommerce** es necesario crear el producto que va a necesitar el formulario y luego indicar en el plugin el id del producto en el archivo del plugin **includes/form.php** en la variable **$product_id**.

Acto seguido, es necesario crear con **Woocommerce Checkout Manager, en la opción de Facturación,** un campo para el DNI, el cual conectará con el campo DNI que pondremos en el front-end con el shortocode. Si la ID del campo de facturación no es '*billing_wooccm11*', tendremos que modificar el nombre del campo en el plugin para que apunte al campo que creemos en el checkout. El archivo donde debemos incluir el id del campo de woocommerce está en el archivo principal del plugin, **extraform.php**, en la variable **$dni** dentro de la función **id_pedido_form(…)**.

NOTA: El id del campo que tenemos que indicar en el plugin no tiene el nombre exacto que en el panel de woocommerce. Para saberlo con seguridad, podemos hacer un pedido de prueba y verificar los campos en la tabla wp_postmeta, donde veremos todos los datos que se añaden a la base de datos al realizar un pedido.

Una vez hecho todo esto, sólo debemos incluir el formulario en una página con el shortocode **'formulario_cumpleaños'**. En elementor, por ejemplo, podemos usar un texto con **[formulario_cumpleaños].**

Con todo esto listo, debemos entender el proceso de compra: el usuario final rellenará el formulario, y acto seguido se le llevará a la página del checkout. Hasta aquí hemos introducido todos los datos a excepción del Id del pedido. Este parámetro se va a añadir cuando completemos el proceso, el cual funciona gracias al campo del DNI, ya que conecta tanto el campo del DNI del formulario del shortcode como el del checkout para incluir el id del pedido al registro correspondiente. Ahora sólo tendremos que ver el id del pedido en el panel del plugin y buscar ese id en la tabla de los pedidos en Woocommerce.

Se pensó anteriormente en lugar del DNI de que se haga consulta directamente al último registro de la tabla del formulario, pero esto puede dar errores ya que en producción puede ocurrir el caso en que más de una persona estén rellenando el formulario.

5. Documentación técnica

En este apartado se va a explicar el código del plugin.

Como hemos dicho antes, se va a utilizar los siguientes lenguajes de programación y librerías:

- PHP
- JavaScript
- Jquery
- Ajax
- HTML y CSS

El arhivo **extraform.php** es el archivo principal de nuestro plugin, el cual conecta el resto de los archivos entre sí.

Dentro de este archivo tenemos la información del mismo entre '/*' y '*/'. Esta información vendrá reflejada en el panel de administración de todos los plugins que tengamos instalados.

La línea de código **require_once plugin_dir_path(__FILE__) . 'includes/extra-functions.php';** nos indica el archivo donde indicaremos el menú lateral de nuestro plugin: Con el *hook* **add action('admin_menu', 'extra_Add_my_Admin_Link')** indicamos que la función extra_Add_my_Admin_Link se usará para el menú de administración, y con las funciones de Wordpress **add_menu_page** y **add_submenu_page** añadimos los elementos correspondientes al menú principal y al menú secundario que va por debajo. Cada una de estas funciones renderizará un panel llamando a otro archivo, en el caso del menú principal a **registro-form.php** y en el caso del menú secundario (el cual contiene información relacionada al plugin como el shortcode) llamará a **informacion.php**.

Volviendo a nuestro archivo principal, la función activar() se activará gracias al *hook* **register_activation_hook.** Lo mismo aplica para las funciones de desactivar y borrar plugin.

La función **bootstrap($hook)** nos permite incluir bootstrap en el panel del plugin, y con el if que se encuentra dentro, indicamos donde queremos que se ejecute.

La función **script_propio($hook)** nos permite cargar nuestros propios archivos JS y CSS, además de poder usar Ajax indicando que vamos a hacer todas las llamadas Ajax desde **admin-ajax.php** y un *nonce* o *token* de seguridad.

Las funciones **EliminarRegistroForm** y **MostrarRegistro** se utilizan junto con Ajax y que explicaremos más adelante.

La función **form_shortcode** aquella que nos permite incluir el shortcode en nuestra web. Ésta hace referencia al archivo **form.php** el cual incluye el formulario y cómo se tratan los datos que se rellenan en él.

La función **id_pedido_form($order_id)** es aquella que nos permite conectar los datos rellenos en el formulario del shortcode con los del checkout. Básicamente, gracias al *hook* '**woocommerce_new_order**' indicamos que esta función debe ejecutarse al hacerse un pedido, entonces buscará en la tabla del plugin, wp_formularios, un registro donde coincida el DNI del checkout con el DNI del formulario del plugin, y cuando lo encuentre añadirá el id del pedido en la tabla del plugin.

El archivo **registro-form.php** es el panel que se muestra al pinchar en el menú principal del plugin. En esta parte del plugin, se nos muestra todos los registros de la tabla wp_formularios, con la posibilidad de añadir nuevos registros, ver, modificarlos y borrarlos. A la hora de guardar los archivos enviados desde el formulario es necesario tener en cuenta algunas cosas:

- Se deben utilizar estas líneas para poder iniciar el guardado de los archivos:

    *require_once ABSPATH . 'wp-admin/includes/file.php es una dependencia necesaria.

    *$wp_filesystem y Wp_filesystem() inician lo necesario.

*En el form enctype=”multipart/form-data” para poder tratar los archivos enviados.

```
$name_file = $_FILES['modificar_archivo_cliente']['name']; //Se obtiene el nombre del archivo
    $fecha_actual = date('d-m-Y');
    $new_name_file = $modificar_dni . '-' . $fecha_actual . '-' . $name_file;
    $tmp_name = $_FILES['modificar_archivo_cliente']['tmp_name']; //Se obtiene la ubicación temporal del archivo
    $allow_extensions = ['pdf', 'png', 'jpg']; //Array con las extensiones de archivo permitidas

    // Validación de los archivos
    $path_parts = pathinfo($new_name_file); //se obtiene la ubicación del archivo
    $ext = $path_parts['extension']; //se obtiene la extensión del archivo desde la ubicación

    if ( ! in_array($ext, $allow_extensions) ) { //Se comprueba la extensión del archivo
        echo "Error: La extensión del archivo no está admitida";
        return;
    }

    //Se establece la ruta donde se guardan los archivos
    $content_directory = $wp_filesystem->wp_content_dir() . 'uploads/solicitudes/'; //Concatenamos wp_content_dir para obtener la ruta de wp-content con el directorio que elijamos
    $wp_filesystem->mkdir( $content_directory ); //Se crea el directorio si no existe

    if( move_uploaded_file( $tmp_name, $content_directory . $new_name_file ) ) { //Mueve el archivo desde la ubicación temporal a la final
        echo "El archivo se ha subido sin problemas";
    } else {
        echo "El archivo no se ha subido";
    }
```

Con las líneas anteriores obtenemos el nombre del archivo y su extensión para luego validarlos, subirlos a la carpeta final, en este caso ‘wp-content/uploads/solicitudes’. Tener en cuenta que al nombre del archivo se le añade tanto el DNI como la fecha por si ocurre el caso en el que hay dos archivos de igual nombre, pues en este caso los archivos se sobreescriben. Acto seguido obtenemos la ruta final del archivo y lo guardamos en la base de datos para verlo en el panel del plugin con el $wpdb→ update más adelante. Con wp_redirect recargamos la página una vez finalizado el proceso. Tener en cuenta que al usar update debemos

indicar primero el array de valores a actualizar y luego la condición o parámetro a través del cual obtenemos el registro a actualizar.

Más adelante, tenemos las ventanas modales con las que podemos añadir los nuevos registros y luego otro para ver y modificarlos cuando se pincha en 'Ver'.

Para poder hacer todo el proceso con las ventanas modales necesitamos usar Jquery (para mostrar los modales) y Ajax (para manejar las solicitudes entre JS y PHP). Primero debemos cargar Ajax y nuestro archivo de scripts en el archivo principal tal y como hablamos anteriormente. Una vez hecho esto, en el archivo **js/extra-form.js** usamos el Jquery y el Ajax.

```
$("#btn-nuevo-form").click(function(){

    $('#modal-form').modal("show");
  })
```

Aquí indicamos que se muestre el modal cuando pulsemos en el botón con id 'btn-nuevo-form' con Jquery.

```
$(document).on('click', 'a[data-ver]', function(){

    var id = this.dataset.ver; // dataset accede a los atributos de un elemento HTML del tipo data-...
    var url = solicitudesAjax.url; // le pasamos la url del array solicitudesAjax que indicamos en wp_localize_script.

    $.ajax({
      type:"POST",
      url: url,
      data: { //ARRAY DE DATOS
        action: "mostraregistro", // viene del alias del hook de extraform.php, en la función MostrarRegistro()
        nonce: solicitudesAjax.seguridad, // Wordpress necesita este token de seguridad que lo obtenemos de wp_localize_script
        id: id,
      },

      success: function(response){ // Función que se ejecuta si la solicitud es correcta
        jsonparse = JSON.parse(response); //Lo que se recibe es una cadena JSON y hay que convertirlo en un objeto

        /*Insertamos los valores en los input de la ventana modal*/
        $("#ver-cliente").val(jsonparse.Nombre_cliente);
```

```
            $("#ver-dni").val(jsonparse.DNI);
            $("#ver-fecha").val(jsonparse.Fecha_registro);
            $("#ver-pedido").val(jsonparse.Id_pedido);
            $("#ver_id_registro").val(jsonparse.Id_formulario);

            $('#modal-form-ver').modal("show");
        },
        error: function(xhr, status, error) {
            console.error(xhr.responseText);
            // Envía el error si lo hay
        }
    });
  })
```

Aquí se indica que cuando pulsamos en un botón con la propiedad data-ver dentro de una etiqueta ‘a’ pasamos dicha información al Ajax en la variable ‘id’. Entonces toda esta información se procesa junto con el **nonce**, que es un token de seguridad que necesita Wordpress. Esta información va la función MostrarRegistro() del archivo principal:

```
function MostrarRegistro(){
  $nonce = $_POST['nonce']; // este valor viene de los datos que se envían a través de la solicitud Ajax
  if(!wp_verify_nonce($nonce, 'seg')){ // verificamos si el nonce es correcto
    die('No tiene permisos para ejecutar esta petición.');
  }

  $id_ajax = $_POST['id'];
  global $wpdb;
  $tabla = "{$wpdb->prefix}formularios";
  // Realizar la consulta SQL para obtener los datos del registro según el ID
  $query = $wpdb->prepare("SELECT * FROM $tabla WHERE Id_formulario = %d", $id_ajax);

  // Ejecutar la consulta
  $registro = $wpdb->get_row($query); //get_row se usa cuando sólo se espera una sola fila y get_results siempre devuelve un array

  echo json_encode($registro);
  die();
}


add_action('wp_ajax_mostraregistro','MostrarRegistro');
```

El action del Ajax se corresponde con el alias del *hook* de add_action wp_ajax_mostraregistro. En esta función se coge la id que nos llega desde Ajax y se realiza la consulta a la base de datos, obteniendo el resultado en formato JSON y cerrando la función con un die() (si esto último no lo hacemo, la consulta no sale bien).

Esto que obtenemos del PHP vuelve a Ajax en el *success*, donde con $ ("#ver-cliente").val(jsonparse.Nombre_cliente), rellenamos los campos del form de la ventana modal con lo obtenido de PHP, y finalmente mostramos el modal con los datos rellenos. Esto mismo aplica para la función de borrado de registros.

Si queremos modificar la información de los registros se usará el código de **registro-form.php**, utilizando lo visto antes de las ventanas modales.

En el archivo **form.php** es donde tenemos el formulario que se va a incluir al añadir el shortcode en nuestra web. Este archivo cuenta también con la consulta a base de datos necesaria para subir la información, además de añadir al carrito el producto deseado y la cantidad una vez finalizado el proceso con la función WC()→cart→add_to_cart(), para luego llevarnos al checkout con el script:

```
<script>window.location.href = '<?php echo wc_get_checkout_url(); ?>';</script>
```

Esto se ha puesto así ya que dependiendo del tema nos puede salir un error si sólo usamos las funciones de Wordpress **Wp_redirect()** o **Wp_safe_redirect()**.

En **información.php** tenemos un breve texto sobre el uso del plugin.

6. Posibles mejoras

Como posibles mejoras, podríamos destacar el uso de una paginación en **registro-form.php**, con el que reducir el scroll para ver todos los datos de los registros del formulario.

También podríamos añadir más opciones para personalización del cliente como una función que nos permita cómodamente editar el formulario del front-end sin necesidad de que el cliente tenga que escribir código.

7. Referencias

- Documentación de Wordpress: *https://developer.wordpress.org/*

- Tutoriales para crear plugins desde cero en Youtube: *https://www.youtube.com/*

- Para subir archivos desde un formulario en Wordpress: *https://decodecms.com/subir-archivos-en-wordpress-a-traves-de-codigo/*

- Hooks en Wordpress:

https://leondesarrollo.es/desarrollo-web/programacion-wordpress/hook-de-creacion-de-un-pedido-en-woocommerce/