Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de un CMS para el soporte de tiendas online

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Martin Martin, David

Tutor: Valderas Aranda, Pedro José

2020/2021

Resumen

El presente proyecto tiene como objetivo el desarrollo de una plataforma web para la creación y gestión de tiendas online, que facilite al usuario de forma sencilla y rápida la creación de todo el contenido, además permitirá al comercio electrónico crear una página web funcional y profesional en un tiempo muy reducido abaratando costes y aplicando funcionalidades que actualmente no se cubren de forma correcta en el mercado.

El proyecto se ha desarrollado siguiendo la metodología agile Scrum, realizando actualizaciones y marcando objetivos semanales. Esto ha provocado que se pueda gestionar de forma correcta el tiempo de entrega del proyecto y además se hayan podido corregir errores entre un *weekly* y otro.

Para la realización de la plataforma se ha usado Laravel como tecnología *core*, Blade, para la creación de las vistas y Bootstrap y Stylus para la creación de estilos. En cuanto a herramientas para la realización del proyecto se han utilizado MAMP para la gestión de servidores y base de datos en MySQL y Sublime como editor de texto.

Palabras clave: CMS, gestión de contenido, tienda online, Scrum, Laravel, Blade, Bootstrap, Stylus, MAMP, MySQL, WordPress, Drupal, Joomla, Magento, Shopify, configurador, afiliados.

Abstract

The objective of this project is to develop a web platform for the creation and management of online stores, which facilitates the user in a simple and fast way to create all the content and will also allow e-commerce to create a functional and professional web page in a very reduced time reducing costs and applying functionalities that are not currently covered correctly in the market.

The project has been developed following the agile Scrum methodology, making updates, and setting weekly objectives. This has meant that the delivery time of the project can be managed correctly, and errors have been corrected between one weekly and another.

To create the platform, Laravel has been used as core technology, Blade, for the creation of the views and Bootstrap and Stylus for the creation of styles. As for tools for the project, MAMP has been used to manage servers and databases in MySQL and Sublime as a text editor.

Keywords : CMS, gestión de contenido, tienda online, Scrum, Laravel, Blade, Bootstrap, Stylus, MAMP, MySQL, WordPress, Drupal, Joomla, Magento, Shopify, configurador, afiliados

Tabla de contenidos

Contenido

1. Introducción

El presente proyecto tiene como objetivo el desarrollo de una plataforma web para la creación y gestión de tiendas online, que facilite al usuario de forma sencilla y rápida la creación de todo el contenido, además permitirá a la empresa crear una página web funcional y profesional en un tiempo muy reducido abaratando costes y aplicando funcionalidades que actualmente no se cubren de forma correcta en el mercado.

1.1 Objetivos del proyecto

El objetivo principal de este trabajo es el de crear un CMS usable, intuitivo y capaz de optimizar todos los procesos que requiere una tienda online. Para ello se tendrá en cuenta la experiencia del usuario al realizar el proyecto. Uno de los principales problemas de las aplicaciones disponibles en el mercado es que muchas veces son difíciles de usar debido a que la mayoría se componen de otros elementos que ellos mismos no crean, por ello, nosotros destacaremos este apartado haciendo una aplicación usable, siguiendo la guía de estilos de Bootstrap y creando un diseño homogéneo durante todo el desarrollo.

Otro de los objetivos fundamentales es que nuestro proyecto, en un futuro, no dependa de terceros, creando nosotros mismos la funcionalidad que, por mi experiencia, he podido detectar que dan ciertos *plugins* ajenos a la plataforma y que al ser desarrollados por terceros no te garantizan una continuidad en cuanto a actualizaciones o soporte.

Además, nos vamos a fijar como objetivo poder integrar ciertas herramientas que hoy en día debe tener cualquier tienda online, ya sea para poder medir sus métricas, promocionarse en diferentes sitios de internet o poder aceptar pagos con nuestra propia pasarela de pago.

Otro de los objetivos principales será crear un sistema de afiliados con un portal específico para que ellos mismos puedan gestionar sus códigos de descuento y métricas, pudiendo hacer que se sientan parte de la tienda.

1.2 Estructura de la memoria

El presente proyecto se divide en los siguientes bloques:

- En primer lugar, se introduce el contexto del proyecto, los objetivos que nos hemos marcado y este propio apartado.
- En el segundo, se profundizará sobre cómo hemos detectado ciertas necesidades, lo que nos ha llevado a la creación del CMS, además hablaremos sobre los requisitos que nuestra solución debe cumplir.
- En el tercero, revisaremos las soluciones existentes similares a la nuestra que hay en el mercado.
- En el cuarto punto, explicaremos la metodología que se ha usado durante la realización del proyecto.
- En el quinto punto, detallaremos tanto los elementos que intervienen en nuestro CMS, como los lenguajes, *frameworks* y herramientas que usaremos para la realización del proyecto y mostraremos algunos ejemplos de código.
- En el sexto punto, realizaremos una prueba de concepto, explicando como usamos el *framework* para la creación de webs.
- Para finalizar, en el séptimo y último punto, incluiremos un resumen del trabajo realizado, una opinión personal y posibles mejoras que pudiera haber en la plataforma.

2. Definición del problema

Para este proyecto vamos a realizar la implementación de un sistema de gestión de contenidos o CMS, por sus siglas en inglés, para dar soporte a tiendas online. Partiendo de una experiencia previa desarrollando y gestionando comercios electrónicos, el objetivo principal de este trabajo es crear una herramienta intuitiva y fácil de usar que dé solución a alguno de los problemas que personalmente he podido encontrar en el sector.

La decisión de crear un CMS desde cero viene dada por la necesidad de implementar ciertas funcionalidades que o bien no se resuelven en las plataformas actuales o bien estas plataformas no nos aseguran que esta funcionalidad tenga el soporte correcto a lo largo del tiempo.

Al tener experiencia tanto en la creación como en la gestión de tiendas online a través de sistemas de gestión de contenido podemos profundizar en aspectos más avanzados e interesantes con el fin de optimizar lo máximo posible todo el proceso de venta y captación de clientes.

Principalmente, nuestro CMS será destinado a la creación de tiendas online de nicho, es decir, con artículos muy concretos y por lo general con pocos tipos de productos. En este tipo de tiendas es necesario cuidar mucho la estética, puesto que al haber pocos productos deben ser más vistosos.

Además, una de las principales funcionalidades que introduciremos y que cada vez cobra más sentido implementar está estrechamente relacionada con el contexto en el que nuestro CMS será de utilidad. Cada vez más clientes quieren poder personalizar sus productos, cambiando colores de ciertas partes o incluso materiales, es por ello que crearemos un configurador visual de producto, de forma que el cliente pueda ver en todo momento una vista final del producto tal y como él la quiere.

Por todo lo anteriormente comentado, nuestra plataforma debe ser lo más intuitiva posible, para facilitar el trabajo de la persona que gestionará la tienda y además nos debe dar la capacidad de crear una web práctica y lista para vender.

3. Estado del arte

Para tener una visión más acertada del trabajo que vamos a desarrollar debemos conocer con claridad las necesidades que nuestro público objetivo tendrá en cuanto a su forma de trabajar y de cara a sus clientes. También deberemos tener conciencia de las plataformas homólogas a la nuestra que están en el mercado, además, esto nos ayudará a saber los puntos fuertes de estas aplicaciones para fortalecernos en la nuestra y los puntos débiles para poder solventarlos.

Cabe destacar que actualmente el sector del comercio electrónico está en pleno auge, ya que la situación derivada por la crisis del *COVID-19* ha hecho que el uso de este tipo de tiendas crezca de forma vertiginosa. De hecho, según algunos estudios independientes, cerca del 84% de los internautas españoles asegura que la pandemia le ha impulsado a usar más internet para comprar frente al comercio tradicional.

3.1 Sistema de gestión de contenidos

Un gestor de contenido, o CMS, por sus siglas en inglés (*Content Management System*) es una plataforma informática usada para crear, editar y gestionar el contenido de páginas webs. El CMS genera la página web de forma dinámica para que el contenido que hemos creado o gestionado tenga sentido, por ejemplo, si creamos un producto en una tienda online, este se verá reflejado en la web de forma que sean legibles todos sus atributos (precio, descripción, stock si lo hubiera, fotos, etc.). Un buen gestor de contenido también debe organizar todos los apartados de la página web para que sean amigables con el posicionamiento en navegadores, entre otras funciones. Para conseguir todas estas funciones el gestor de contenidos hace de 'intermediario' entre la base de datos, donde alojaremos toda la información que vamos a mostrar en las diferentes secciones de la web y el servidor web, que será el encargado de mostrar en el cliente una vista ordenada de este contenido, obviamente con una correcta programación en el proceso.

En resumen, un gestor de contenidos es una plataforma web en la que los usuarios podrán crear, editar y gestionar contenido desde un *backend* sin necesidad de tener unos elevados conocimientos técnicos. Algunos de los ejemplos más utilizados de CMS son WordPress, Drupal, Joomla, Magento, Shopify…

3.2 Ventajas de usar un CMS

A la hora de plantear la creación de una tienda online, el uso de un CMS nos aporta una gran cantidad de ventajas, como son:

- Facilidad de uso una vez el usuario ha recibido formación para hacerlo. Una vez desarrollada la web, el usuario puede centrar su tiempo en crear, editar o gestionar el contenido de ella sin necesidad de hacer cambios en el código.
- El desarrollo de una página web desde un CMS es más rápido y limpio. Si hay una buena programación detrás de la plataforma de gestión el código generado será eficiente y seguirá unos estándares y por lo tanto a la larga será más fácil implementar cambios en la propia plataforma al partir de una base sólida.
- Permiten desarrollos escalables. Si deseamos introducir cambios en la web en un futuro no comprometeremos el resto de la página al introducir nuevas funcionalidades a través de módulos.
- En nuestro caso, al haber desarrollado en la totalidad el código del CMS podremos desarrollar prácticamente cualquier funcionalidad que se nos requiera en un futuro, además, al ser un sistema que no está abierto a todo el mundo será más difícil que aparezcan vulnerabilidades y problemas de dependencia entre módulos.

Con todas estas ventajas, hoy en día es realmente difícil encontrar una web conocida que no esté desarrollada a partir de un sistema de gestión de contenidos, ya sea uno de código libre o uno construido específicamente para sus necesidades.

3.3 Soluciones existentes

Para elegir correctamente un CMS para la gestión de una tienda online es importante saber el contexto en el que se va a desarrollar dicha web. Para empezar, nosotros realizaremos un estudio de las soluciones existentes que dan soporte a comercios electrónicos, dejando de lado blogs u otro tipo de páginas. Además, debemos tener claro que vamos a centrarnos en tiendas de nicho y que esto, por norma general, tiene algunas particularidades, como son, un número bajo de productos, algunos productos deben admitir un grado alto de personalización y actualmente, la mayoría se nutren de la promoción a partir de *influencers*, por lo que sería interesante contar con un portal para que ellos mismos o nosotros podamos ver sus estadísticas.

Teniendo en cuenta estas exigencias que nos presenta nuestro proyecto vamos a analizar de forma superflua algunos pros y contras de los CMS que más cuota de mercado tienen actualmente. Cabe destacar que la mayoría son

3.3.1 WordPress

Con más del 64% de los usuarios de gestores de contenido en su plataforma es de largo la referencia para los CMS. Gracias a su gran comunidad es muy fácil poder solventar problemas que nos puedan surgir en la instalación o incluso gestión de las herramientas que tiene, aunque este elevado número de usuarios también exige un esfuerzo extra en cuanto a seguridad pues, a menudo surgen vulnerabilidades que tienen que arreglar en su sistema.

Una de las principales ventajas que presenta es su fácil y rápida instalación, aunque para tener debidamente configurada una página web hay que invertir una gran cantidad de tiempo ya que para que WordPress funcione como tienda online hay que instalar ciertos *plugins* y al existir tantas extensiones se hace complicado elegir la adecuada y que, además, esté actualizada y funcione de forma correcta con nuestra versión de WordPress, aunque la mayoría se integran con facilidad.

Como principales contras debemos destacar las brechas de seguridad que revelan a menudo los *plugins*, la inestabilidad y poco rendimiento que podemos observar con un tráfico elevado y la dificultad de administrar con las continuas actualizaciones de *core* y *plugins*.

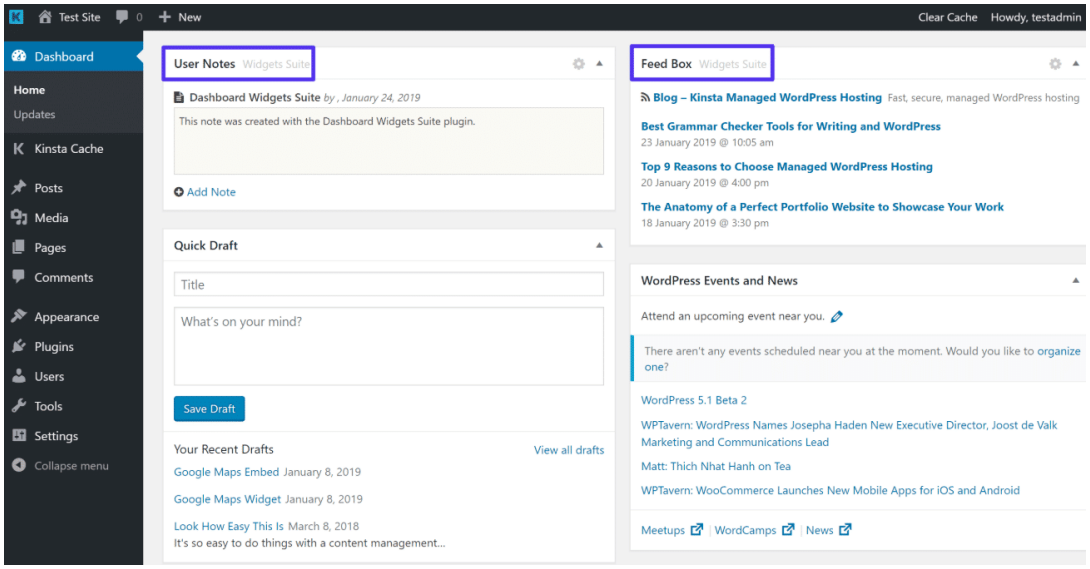

Figura 1 – Interfaz WordPress.

3.3.2 Drupal

Drupal, al igual que sus homólogos del mercado es de código abierto y, además, modular. Fue desarrollado originalmente como foro estudiantil, siendo hoy uno de los CMS más utilizados.

Al igual que la mayoría de los gestores de contenido que tienen éxito en el mercado, este se debe a su activa comunidad. La instalación más básica es realmente fácil de realizar y si lo requerimos se puede ampliar con una gran cantidad de módulos. El módulo básico ya contiene numerosas funcionalidades para crear una web social como *blog* o foro, pero tiene una gran cantidad de posibilidades ampliando estos módulos.

La arquitectura modular de Drupal permite un alto grado de personalización, aunque si dominamos su *framework* y sabemos que funcionalidades requerimos podemos omitir el ensamblado de los módulos y así ahorrar una gran cantidad de tiempo.

Por priorizar sobre todo el social publishing, Drupal es una excelente opción para comunidades pequeñas o medianas, aunque su amplio abanico de extensiones también permite crear portales corporativos con complejas estructuras multidominio. Es cierto que para administrar sitios complejos se requiere de un alto nivel técnico. Si las funciones básicas del software no bastasen, los módulos adicionales, además de tener complejas relaciones de dependencia se deberían instalar vía FTP y en ocasiones esto no es posible por parte del servidor en el que tenemos alojado el proyecto. Además, y esto nos complica realmente nuestros objetivos, es muy poco tolerante con versiones antiguas, por lo que si un módulo está desactualizado o se prevé que dejen de darle soporte nos dificultaría mucho la funcionalidad de nuestra plataforma.

El público principal de Drupal es la creación de comunidades sociales pequeñas o medianas a través del *social publising*. Por su flexibilidad, este es, quizás, el mejor CMS para plataformas que se nutren de contenido creado por los propios usuarios.

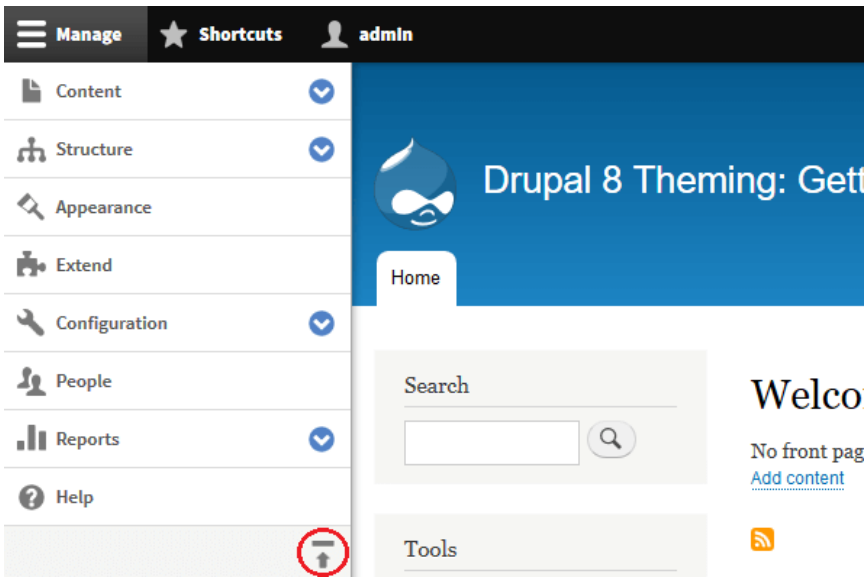

Figura 2 – Interfaz Drupal.

3.3.3 Joomla

Con un 3,5 % de cuota de mercado, Joomla es el tercer CMS más usado. Aunque está orientado por igual a personas con perfil técnico que a usuarios nuevos, es más exigente que WordPress. Cabe destacar que, a diferencia de este, trae de base muchas funciones para gestionar contenido y como aspecto positivo para usuarios que se inician en el sector, Joomla posee una excelente documentación en forma de manuales creados por la amplia y activa comunidad que poseen.

Se creó a partir del CMS *open source* Mambo y goza de especial popularidad en Estados Unidos. Además, se caracteriza por un diseño de software completamente orientado a objetos sobre la base de un *framework* MVC (modelo vista controlador) autónomo. Al igual que la mayoría de ya mencionados gestores de contenido, al ser de código abierto y estar basado en un *framework* asequible, es posible crear y publicar extensiones en el propio repositorio del CMS, con el cual también se pueden instalar las extensiones en el *backend*. Aunque si no disponemos de tanta destreza programando siempre se pueden combinar módulos para hacer funcionalidades elaboradas.

Joomla divide las extensiones en *plugins*, componentes, plantillas, módulos e idiomas y es posible usarlos tanto en el *frontend* como en el *backend*. La gestión de usuarios, accesos y autores se percibe, en ocasiones, incompleta, por lo que dificultaría en gran medida la implementación del portal para afiliado que queremos realizar.

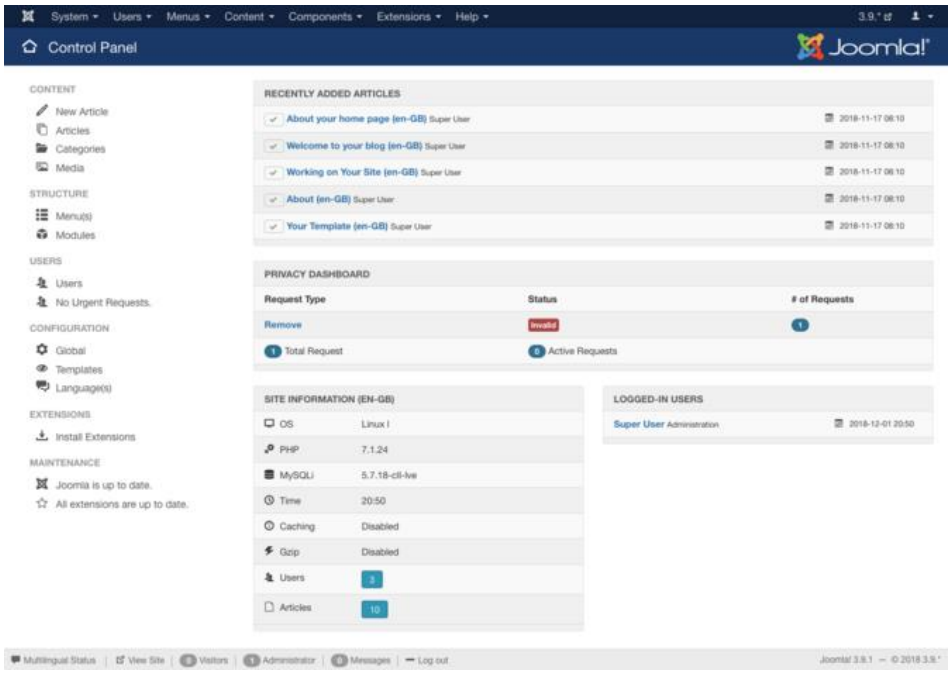

Figura 3 – Interfaz Joomla.

3.3.4 Magento

Magento en su antigua *Community Edition*, de ahora en adelante Magento Open Source, ofrece una gran cantidad de ventajas para los comercios electrónicos ya que está destinada íntegramente a ello.

Con un editor de páginas visual, permite a los administradores de la página ver en tiempo real los cambios que realizan en la web de forma fácil e intuitiva.

Como consecuencia de que la mayoría de las ventas online se realicen a través de un teléfono móvil o *tablet*, está orientado para webs '*mobile first*' y está preparado para optimizar las vistas en este tipo de terminales, poniendo especial énfasis, además de a los contenidos, a los procesos, puliendo especialmente la función de añadir al carrito o realizar el *checkout*, momento crítico en cualquier venta online.

Sobresale en cuanto al tratamiento del SEO ya que posee su propia herramienta de optimización. Además, está abierta para poder implementar cualquier retoque de posicionamiento que pueda pedir un experto SEO y, para aquellos que tienen el conocimiento y las habilidades suficientes, resulta realmente útil poder crear tu propia estrategia. El objetivo, como todas las herramientas destinadas a la misma función es posicionar de forma optima en buscadores como Google, Bing, etc.

Resulta de verdadero interés la posibilidad de poder integrar Google Analytics para medir las métricas del sitio, además de otros muchos *softwares* de terceros.

La plataforma no se queda en una simple tienda online para comercios pequeños o medianos, sino que puede llegar a albergar hasta 1.000.000 de productos con fichas sencillas o complejas, configurables en gran medida.

Magento, al igual que la mayoría de CMS orientados al comercio online, tiene la funcionalidad de crear ventas cruzadas o relacionadas al vincular productos, con el fin de aumentar el *ticket* medio, mejorar la ratio de conversión o ayudar en la experiencia del usuario. Además, estos tipos de relaciones se pueden llegar a personalizar para casos concretos. No solo es posible crear ventas cruzadas o relacionadas, sino que es posible anunciar que un producto es necesario para que otro funcione antes de finalizar la compra.

Además de ser altamente escalable, Magento es robusto y seguro. Permite administrar múltiples niveles de permisos de seguridad. Cumpliendo sobradamente con los PCI Data Security. Cabe destacar los múltiples detalles que posee a la hora de iniciar sesión, como son la posibilidad

de añadir un captcha o una contraseña secundaria para evitar ataques de fuerza bruta o automatizados.

Dispone de una herramienta de búsqueda realmente ágil, capaz de realizar un filtrado atendiendo a una gran cantidad de parámetros. Magento utiliza para sus filtrados o búsquedas una nube de términos únicos, *tags* de productos, productos recientemente comparados o visitados, y permite configurar los términos que aparecerán en la búsqueda desde un inicio y por tanto, también sobresaldría en este aspecto en comparación con sus competidores.

Cerca de un cuarto de millón de empresas usan Magento para gestionar sus comercios electrónicos. En 2012 se existían en la plataforma casi un millón de desarrolladores únicos, ese número ha crecido desde entonces, dando a los usuarios que empiezan en el CMS un sistema robusto y confiable. El hecho de que Magento sea uno de los líderes del sector ha hecho que la plataforma adquiera un carácter profesional y los desarrollos realizados dentro de ella sean de gran calidad.

Como ya hemos mencionado, al ser de código abierto y tener una amplia comunidad, Magento posee una gran cantidad de extensiones con funcionalidades muy concretas que nos podrían resultar de verdadera ayuda a la hora de implementar nuestro proyecto. Esto nos garantiza que la mayoría de las personalizaciones que haya que hacer en una web estarán cubiertas por estas extensiones, aunque si es verdad que en ocasiones hará falta un desarrollo a medida.

Como plus para las grandes empresas, Magento es capaz de gestionar varias tiendas online, con diferentes monedas o incluso idiomas, desde la misma plataforma.

Si comparamos los precios de Magento con la competencia, el CMS del que hablamos es notablemente más costoso que los demás, pero por razones como las anteriormente citadas hacen que sea un gestor a tener en cuenta.

Como contrapunto, a la hora de realizar desarrollos en Magento y permitir su conexión con ERPs, esto suele ocasionar que el gasto en tiempo sea elevado y por ello se recomienda hacer un *planning* concreto de desarrollo.

Como ya hemos comentado y debido a su complejidad, el nivel técnico requerido para desarrollar *software* a medida para Magento es muy elevado, dejando fuera de este a perfiles junior o seniors con poca experiencia en el CMS.

Por desgracia, Magento no cuenta con alojamiento propio por lo que habría que buscar una solución empresarial que nos de soporte para esta plataforma y cabe destacar que tendrá que ser un host potente ya que el CMS es bastante exigente en comparación con otros.

Si bien Magento Open Source es gratuito y de código abierto, es una opción perfecta para pequeñas empresas. Sin embargo, las medianas y grandes empresas que quieran adquirir la versión Enterprise tendrán que desembolsar una gran suma de dinero bien sea para adquirir el plan o bien para desarrollar las funcionalidades a partir de la versión Open Source.

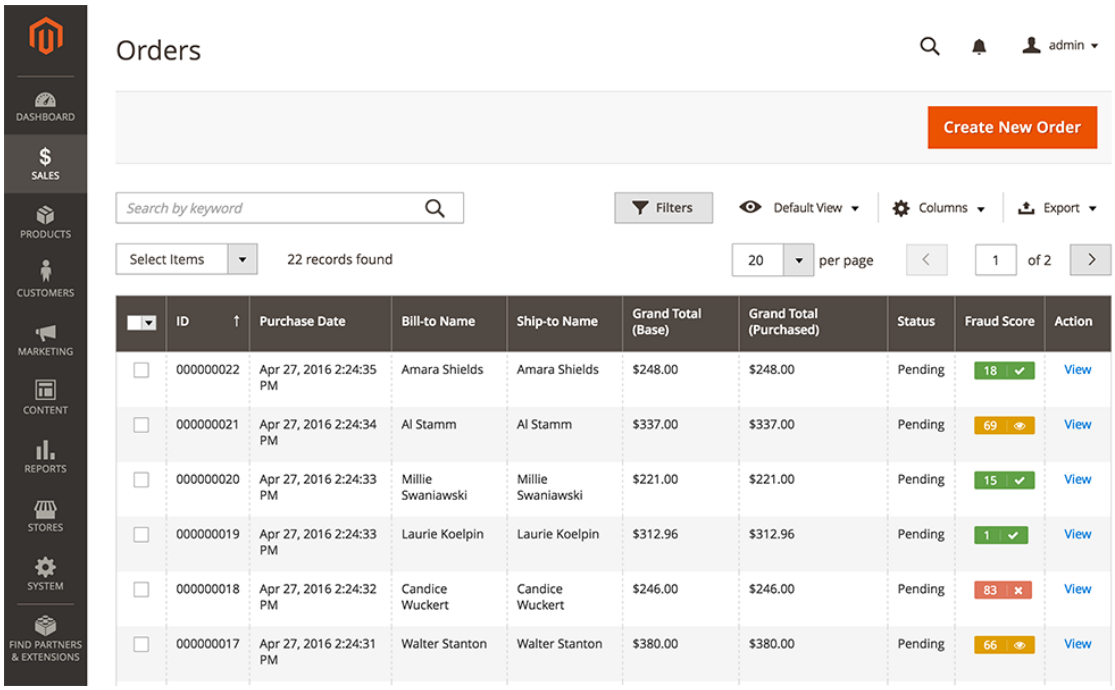

Figura 4 – Interfaz Magento.

3.3.5 Shopify

Shopify está orientada al emprendimiento, se ha enfocado más que otros CMS en crear una herramienta visual, en la que el usuario no tenga que aprender a usarla dedicando muchas horas. Por todo esto se centra más en pequeños comercios electrónicos que no quieren asimilar un desarrollo costoso de inicio.

Al igual que los demás CMS del mercado, Shopify tiene su propia tienda de extensiones o plugins para dar funcionalidad a la plataforma, además, posee bastantes plantillas que sin conocimientos de HTML o CSS se pueden personalizar de forma visual, aunque habría que destacar que al ser una plataforma más joven no tiene ni la comunidad ni la cantidad de desarrolladores que tienen otros gestores de contenido.

Además de todo lo comentado anteriormente, Shopify utiliza su propio alojamiento, dejando que el cliente no tenga que preocuparse por buscarlo fuera y así unificando los servicios.

Posee un gran sistema *responsive*, y puesto que, cada vez el comercio electrónico se utiliza a través de terminales móviles esto es una gran ventaja a la hora de llegar a todo tipo de clientes.

Negativamente hay que destacar la limitación que sufre durante el proceso de *checkout*. A pesar de estar altamente centrado en mejorar las conversiones y la usabilidad, la personalización de la página de finalizar compra es reducida, únicamente es posible modificar pequeños atributos, insuficientes para adaptar esta página al *branding* del comercio electrónico.

Al contrario que muchos otros CMS, Shopify basa su sistema de negocio en suscripciones mensuales, comisiones de venta y, además, venta de *addons* para la plataforma, esto hace que en ocasiones no sea tan recomendable su uso.

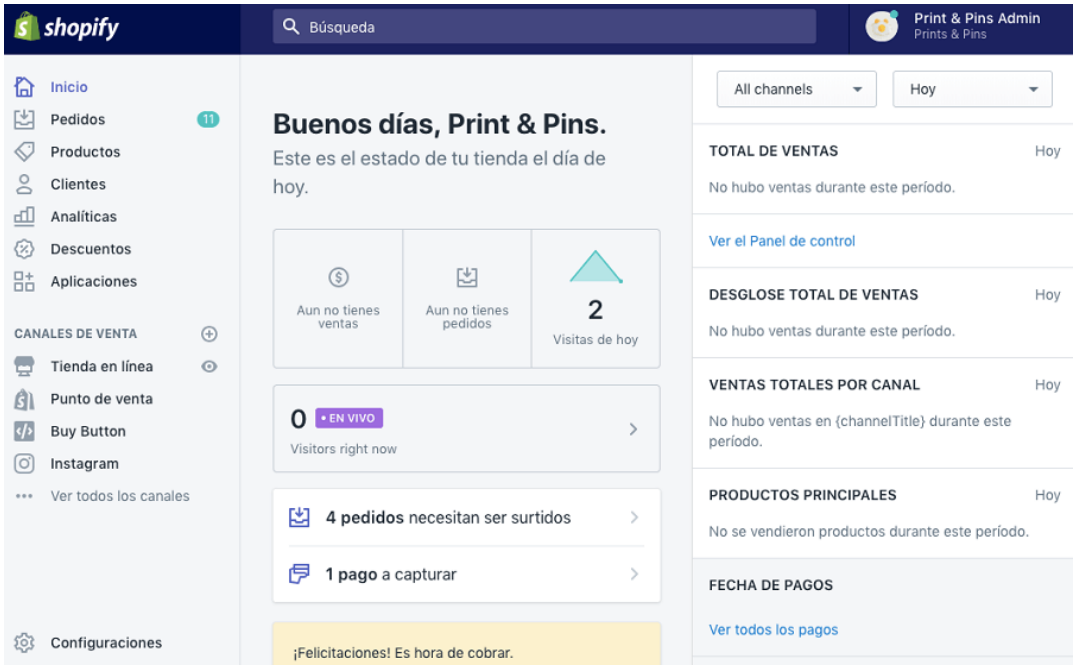

Figura 5 – Interfaz Shopify.

3.4 Soluciones existentes vs propuesta

Una vez que conocemos los CMS que hay en el mercado podemos centrarnos en la funcionalidad que queremos dar a nuestros sitios. En nuestro caso, buscamos, como hemos mencionado anteriormente, crear tiendas online de nicho, con productos muy concretos y con poca cantidad de artículos. Pero, a su vez, también deseamos que sea posible configurar de forma visual algún producto en especial. Además, necesitamos tener un sistema de afiliados o referidos que pueda llevar correctamente el conteo de las ventas que hacen nuestros *partners*, asignarles un link para que puedan promocionar nuestros productos y aparte, su propio portal para que ellos mismos puedan tener constancia de la evolución y de las métricas que están haciendo.

	Visual Configurator	Sistema de afiliados	Portal para afiliados
WordPress	✓	✓	✓
Drupal	✗	✓	✗
Joomla	✗	✓	✗
Magento	✓	✓	✓
Shopify	✗	✓	✓

Figura 6 – Tabla comparativa soluciones existentes.

De esta forma y tras estudiar en profundidad las diferentes opciones que nos dejaban los CMS existentes, analizando las extensiones que crean terceros, únicamente en el caso de Magento y WordPress hemos podido encontrar productos en los que todas nuestras necesidades se cubren, por lo menos de forma funcional.

En el caso de WordPress hemos podido encontrar un plugin realmente interesante para crear el sistema de afiliados y que incluye un portal bastante intuitivo y práctico para éstos. El creador de productos también reuniría la funcionalidad que deseamos, pero por experiencia personal este tipo de *plugins* se dejan abandonados por parte de los creadores y así es en este caso, ya que no ha recibido actualizaciones desde la versión anterior de WooCommerce, el gestor de ventas de WordPress. Además, en alguna ocasión, tras usarlo, he tenido que contactar con el soporte del desarrollador y no han podido darme solución a mis problemas.

Además, incluso si pudiera servirnos la opción del configurador desactualizado, nuestro cliente tendría que hacer frente a una factura de 1.293 $ anuales, precio que difícilmente podría pagar cualquier tienda online que esté empezando.

De igual manera nos ocurre en el caso de Magento, donde también tenemos las funcionalidades que necesitamos cubiertas, pero alguno de las extensiones ha dejado de dar soporte y tiene una versión de Magento antigua, en este, la necesaria para los afiliados y su portal. En cuanto al configurador si existe una extensión aparentemente estable. Aunque, de nuevo, el cliente tendría que hacer frente a 1.011 $ anuales de factura.

De forma adicional, habría que mencionar que, en ambas plataformas, las extensiones que necesitamos no son de fácil instalación ni configuración por lo que podría complicar en gran medida nuestro objetivo de obtener una plataforma sencilla de usar y gestionar. Además, realizando nosotros la programación del código necesario en nuestra plataforma conseguiremos

abaratar en gran medida el precio, que sea competitivo y nos aseguraríamos una continua actualización para que esté plenamente integrado en futuras versiones.

4. Metodología

Una vez hemos decidido que vamos a desarrollar nuestro propio CMS tenemos que dejar bien definidas tanto la forma de trabajar como los objetivos de tiempo que nos vamos a fijar. Al haber tenido experiencia previa en metodologías agile (Scrum), me siento más cómodo con esta forma de trabajar y será por tanto la forma en la que enfocaremos el proyecto.

En la metodología Scrum se realizan entregas parciales y regulares del producto final, en nuestro caso, serán semanales y definidas por nosotros mismos. Scrum está especialmente indicado para proyectos en entornos complejos o donde se necesita obtener resultados pronto. Además, al ser un proyecto con especificaciones cambiantes y donde la innovación y la productividad son fundamentales, este sistema se adapta perfectamente a nosotros. Por estos motivos creo que sería una forma de trabajar adecuada para el proyecto, ya que requiere crear de cero toda una plataforma en relativamente poco tiempo y, de hecho, entre la versión que esperamos realizar al principio y la final variarán bastantes elementos.

En primer lugar, al no disponer de un equipo, descartamos los *dailies* pero sí que realizaremos una pequeña revisión semanal de los objetivos definidos y alcanzados para poder reajustar la estrategia de cara a la siguiente semana y poder ser conscientes del ritmo real del proyecto.

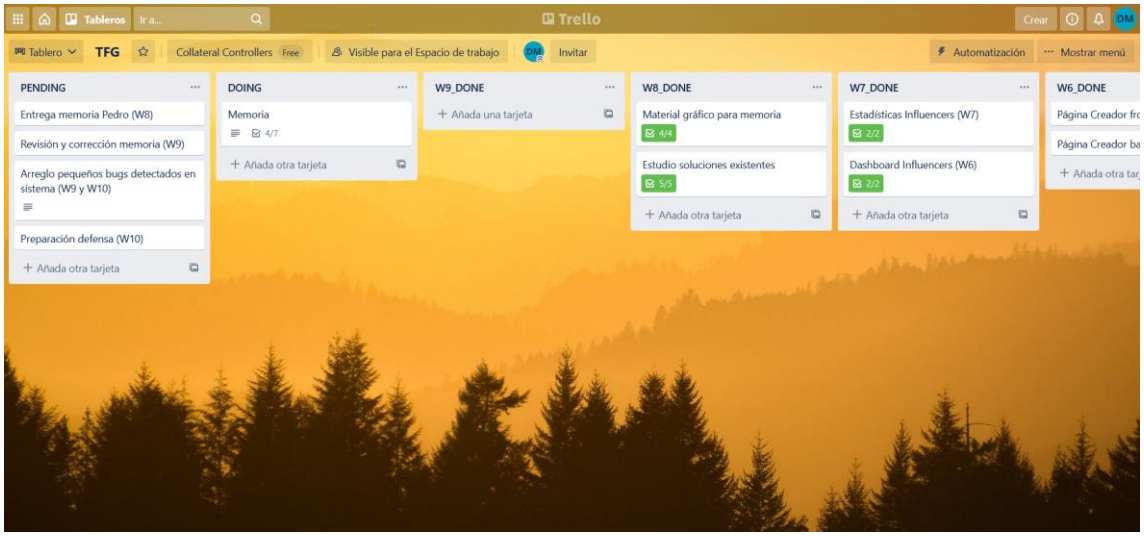

Figura 7 – Tablón de Trello en la parte final del proyecto.

Para organizar todas las tareas del proyecto hemos creado un tablero de Trello (ver figura X) en el que hemos creado todas las tarjetas necesarias para la creación de la plataforma, empezando por la parte más técnica, donde definiremos las tecnologías, *frameworks* y lenguajes a usar,

pasando por el diseño y maquetado de todas las pantallas necesarias tanto en *frontend* como en *backend* y finalizando en la parte funcional del proyecto, sin olvidarnos de la búsqueda de información y redacción de esta misma memoria.

5. Desarrollo de la solución

Antes de comenzar con el desarrollo de nuestro código vamos a definir de forma clara todas las dependencias que habrá en nuestro sistema, para ello debemos tener claros todos los elementos que intervendrán en nuestro CMS.

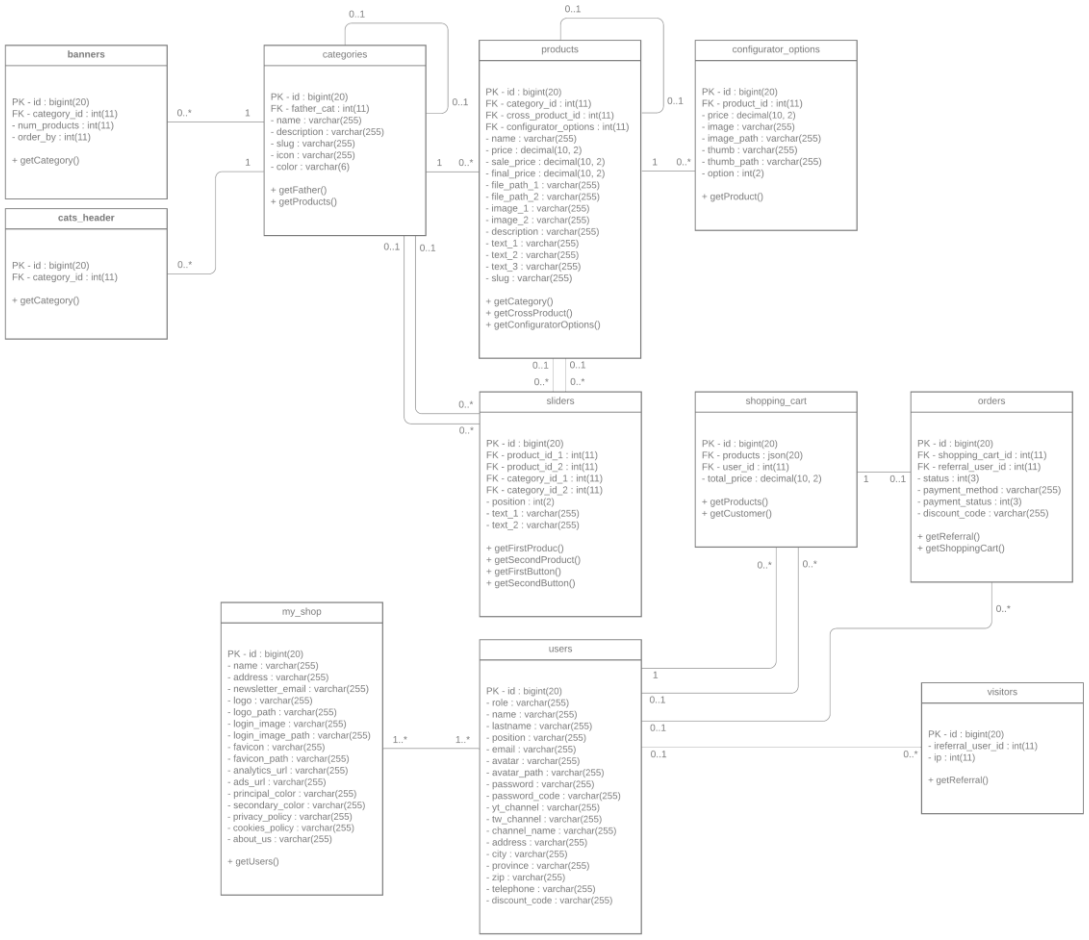

Figura 8 – Diagrama UML.

En primer lugar, debemos saber que en nuestra plataforma habrá tres roles diferenciados; administrador, cliente y afiliado, todos ellos serán usuarios, pero no tendrán ni las mismos funciones ni permisos. El administrador será el encargado de configurar toda la tienda online, configurando los parámetros de *my_shop* desde la página de gestión de la tienda y, además, introduciendo los *banners*, *cats_headers* y *sliders* que se mostrarán en ella. Estas tablas son necesarias para que en la vista del cliente se vea todo el contenido que deseamos de forma dinámica. También deberá introducir las categorías, productos y configuraciones de los artículos personalizados.

Los afiliados, por otro lado, y previa creación por parte del administrador, podrán elegir su código descuento, con un porcentaje y fecha de expiración ya asignados. Este código es el que usarán los clientes para comprar y obtener descuentos, a su vez, los *influencers* recibirán cierta recompensa por estas ventas. Esto podrán gestionarlo desde su propia página de gestión.

Los clientes, que a su vez son visitantes, para poder contabilizar esta estadística. Podrán añadir productos a su carrito y posteriormente comprarlos completando el pedido, podrán usar un código descuento como hemos mencionado anteriormente.

5.1 Arquitectura

Una vez que conocemos en detalle cada uno de los elementos que se verán involucrados en nuestro proyecto es momento de definir la arquitectura de este.

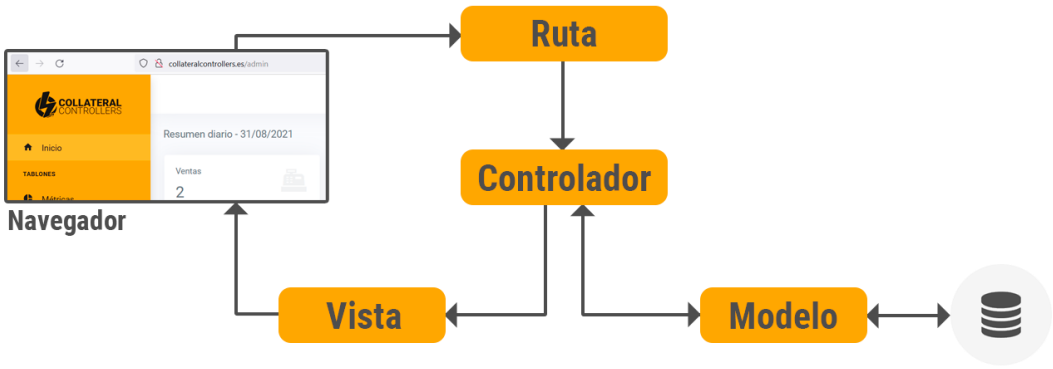

Figura 9 – Esquema de componentes de Laravel.

Vamos a desarrollar nuestra plataforma CMS con el *framework* Laravel y para ello debemos conocer como interactúan los diferentes elementos que éste nos proporciona.

En primer lugar, el cliente enviará una petición a través de su navegador y ésta será redirigida a través de nuestro archivo de *routing* al controlador adecuado, cabe destacar que Laravel basa todo su funcionamiento de cara al cliente con peticiones GET y POST, por lo que para una misma ruta podemos tener varios métodos en un mismo controlador.

Una vez que la petición ha llegado al controlador éste accederá al modelo de datos que se requiera para poder mostrar, crear o editar el contenido de la página en cuestión.

El modelo de datos está estrechamente relacionado con la base de datos y, de hecho, es el encargado de acceder a ella para poder atender las peticiones del controlador.

En el momento que el controlador ha obtenido todos los datos necesarios para la visualización cargará la vista, que, de nuevo, dependerá del método ejecutado en el controlador.

Durante todo el desarrollo del proyecto, organizaremos tanto las rutas, como los controladores, como las vistas en diferentes archivos que dependerán del destino de estos, por ejemplo, para la parte de *backend* usaremos los archivos y directorios de *admin* y/o *connect* y para la parte de *frontend* los ubicaremos en la raíz de las carpetas o usaremos los archivos *web*.

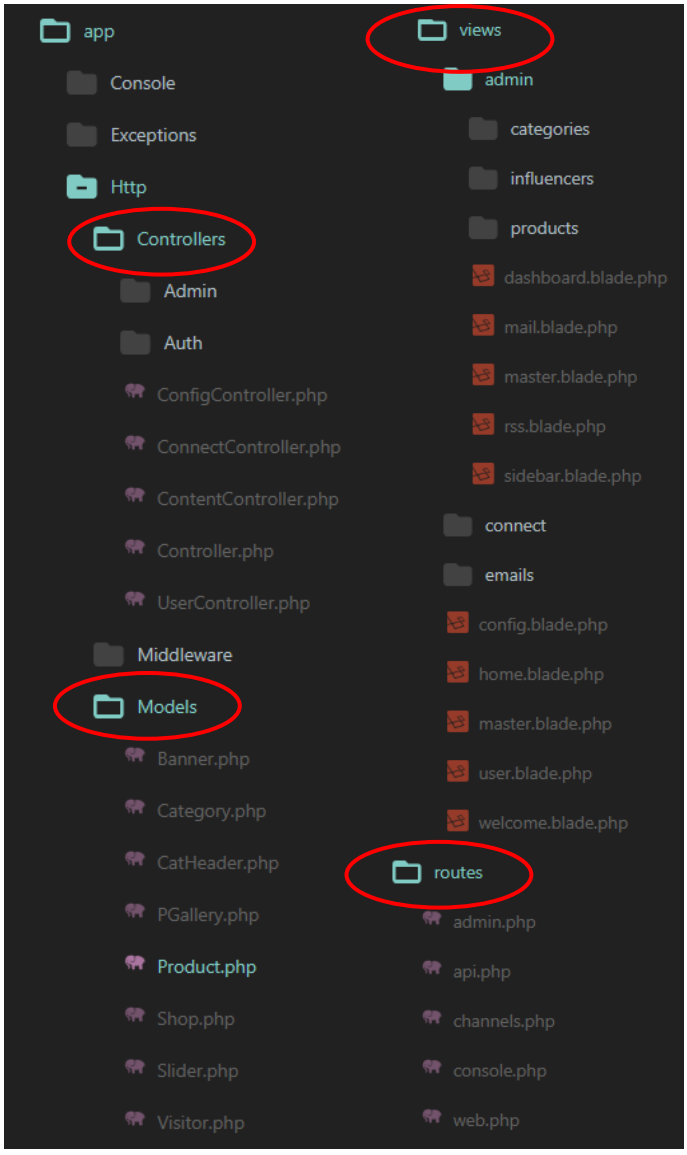

Figura 10 – Distribución de ficheros en el proyecto.

5.2 Contexto tecnológico

Vamos a detallar las herramientas, frameworks y lenguajes que hemos utilizado para la implementación de nuestro CMS de forma detallada, a continuación, mostraremos algunos ejemplos de código con diferentes funcionalidades del *frontend*, *backend* y base de datos.

5.2.1 MAMP

Para el desarrollo del proyecto hemos instalado el entorno de desarrollo local MAMP, ya que nos permite disponer de un servidor web Apache, una base de datos que funciona bajo MySQL y sobre todo, nos permite trabajar con PHP, en concreto trabajaremos con la versión 5.7.24 de MySQL y con la 7.4.1 de PHP.

5.2.2 Laravel

Laravel es un *framework* para aplicaciones web con una sintaxis elegante y expresiva. Este *framework* proporciona una estructura y un punto de partida para crear nuestra plataforma, lo que nos permite centrarnos en los detalles, dejando el trabajo 'duro' a él.

Laravel se esfuerza una experiencia de desarrollo agradable al mismo tiempo que proporciona unas funcionalidades realmente útiles como las dependencias fuertes, una capa de abstracción de base de dato, colas y trabajos programados, pruebas de integración y unidad, y más.

5.2.3 MySQL

MySQL es un sistema de administración de bases de datos (*Database Management System*, DBMS) para bases de datos relacionales. De esta forma, MySQL es un simple programa utilizado para gestionar objetos alojados en bases de datos.

Cabe destacar que MySQL es de código abierto, lo que permite descargar su código fuente y realizar modificaciones, esto ha sido uno de los motivos por los que su comunidad ha crecido tanto. Además de esto, ha permitido que sea continuamente actualizado, y por tanto ha hecho que

sea uno de las herramientas más utilizadas para gestionar las bases de datos de los sitios orientados a web.

5.2.4 HMTL5

En nuestro caso, no vamos a utilizar únicamente HTML5, sino que lo usaremos en conjunto con las plantillas Blade de Laravel, esto nos permite poder introducir código PHP en las vistas de HMTL de forma sencilla. Además, nos da la facilidad de crear secciones (ver figura 10) que puedan ser comunes a todos los HMTL y así poder reutilizarlas.

5.2.5 CSS3

Al igual que ocurre con HTML, a la hora de utilizar las hojas de estilos en cascada (CSS), lo haremos apoyándonos en la biblioteca Bootstrap. Siempre que no nos sea posible aplicar las clases de Bootstrap crearemos nuestros propios estilos, pero con la ayuda de Stylus.

5.2.6 Stylus

Es un lenguaje preprocesador de hojas de estilo dinámico que se compila en hojas de estilo en cascada. Su diseño está influenciado por Sass y Less. Además. está considerado como la cuarta sintaxis de preprocesador CSS más utilizada.

Stylus nos proporciona una gran cantidad de utilidades necesarias para nuestra plataforma, ya que, al estar pensada para poder ser personalizada en gran medida, nos resulta de verdadera ayuda el poder trabajar de forma sencilla con variables de colores y/o tamaños y fuentes.

5.2.7 Bootstrap

Bootstrap es una biblioteca multiplataforma o conjunto de herramientas de código abierto para diseño de sitios y aplicaciones web. Trabaja a partir de clases preestablecidas que dotan de un estilo limpio y sencillo a nuestros elementos visuales. Además, el sistema de *cards* nos ayudará en gran medida para la organización de nuestro *dashboard* y sistema de métricas del *backend*.

5.2.8 Otras herramientas

Además de las citadas herramientas y lenguajes de programación, también hemos dado funcionalidad a las páginas con JavaScript y hemos usado iconos previamente creados como son los proporcionados por FontAwesome y Material Design Icons.

5.3 Ejemplos de código

Una vez que conocemos las herramientas con las que vamos a trabajar vamos a mostrar algunos ejemplos de código del proyecto en los que podremos apreciar la sintaxis y forma de trabajar que nos aportan.

admin.php

```php
<?php

//Common routes

//Users module
Route::get('/user', 'UserController@getUser')->name('user');
Route::post('/user', 'UserController@postUser')->name('user');

//Config module
Route::get('/config', 'ConfigController@getConfig')->name('config');
Route::post('/config', 'ConfigController@postConfig')->name('config');

Route::prefix('/admin')->group(function(){
    // Dashboard module
    Route::get('/', 'Admin\DashboardController@getDashboard')->name('dashboard');
    Route::post('/', 'Admin\DashboardController@postDashboard')->name('dashboard');

    // Influencers module
    Route::get('/influencers', 'Admin\InfluencersController@getInfluencers')->name('influencers');
    Route::post('/influencers', 'Admin\InfluencersController@postInfluencers')->name('influencers');

    // Mail module
    Route::get('/mail', 'Admin\MailController@getMail')->name('mail');

    // Rss module
    Route::get('/rss', 'Admin\RssController@getRss')->name('rss');

    // Categories module
    Route::get('/categories', 'Admin\CategoriesController@getCategories')->name('categories');
    Route::post('/categories', 'Admin\CategoriesController@postCategory')->name('categories');
    Route::get('/category/{id}/edit', 'Admin\CategoriesController@getEdit')->name('category_edit');
    Route::post('/category/{id}/edit', 'Admin\CategoriesController@postEdit')->name('category_edit');
    Route::get('/category/{id}/remove', 'Admin\CategoriesController@getRemove')->name('category_remove');

    // Products module
    Route::get('/products', 'Admin\ProductsController@getProducts')->name('products');
    Route::post('/products', 'Admin\ProductsController@postProduct')->name('products');
    Route::get('/product/{id}/edit', 'Admin\ProductsController@getEdit')->name('product_edit');
    Route::post('/product/{id}/edit', 'Admin\ProductsController@postEdit')->name('product_edit');
    Route::get('/product/{id}/remove', 'Admin\ProductsController@getRemove')->name('product_remove');
    Route::post('/product/{id}/image/edit', 'Admin\ProductsController@postImageEdit')->name('product_image_edit');
    Route::post('/product/{id}/gallery/add', 'Admin\ProductsController@postGallery')->name('product_image_add');
    Route::get('/product/{id}/image/{gid}/remove', 'Admin\ProductsController@getGalleryRemove')->name('
        product_image_remove');
});
```

Figura 11 – Ejemplo de router.

Como podemos observar en la figura 11, el sistema de enrutamiento, creado en *php*, a través del *framework* Laravel, nos permite crear las rutas para los métodos *get* y *post* del navegador. Por

lo general usaremos el método *get* para únicamente mostrar información y el *post* para crear, editar o eliminar dicha información.

La estructura de un objeto Route en Laravel es muy sencilla, para su creación, basta con pasar por parámetro la ruta propiamente dicha y el controlador que la maneja junto con el método que ejecutará. Además, y para facilitarnos el trabajo a la hora de identificar el apartado en el que nos encontramos en todo momento, las nombraremos también por funcionalidad.

A modo de ejemplo, podemos observar que en la línea 31 de la figura 11 creamos la ruta para el método get y le pasamos por parámetros '/category/{id}/edit' que junto con el prefijo que hemos definido en la línea 13 de la misma figura nos daría la ruta relativa '/admin/category/{id}/edit' donde '{id}' será el id de una categoría que pasaremos por parámetro al método del controlador. Como segundo parámetro de la creación de la ruta, podemos observar que le asignamos 'Admin\CategoriesController@getEdit', esto significa que, para el controlador de categorías, al invocar el método *get* del navegador, se ejecutará el método *getEdit* del controlador. Y finalmente le asignamos el nombre 'categories', que, entre otras funciones nos puede servir para mantener en activa la sección del menú de gestión en la que estamos.

ProductsController.php

```
30      public function postProduct(Request $request){
31          $rules = [
32              'name' => 'required',
33              'price' => 'required',
34              'category_id' => 'required',
35              'description' => 'required',
36              'text_1' => 'required',
37              'text_2' => 'required',
38              'text_3' => 'required',
39              'main_img' => 'required'
40          ];
41
42          $messages = [
43              'name.required' => 'El nombre es obligatorio',
44              'price.required' => 'El precio es obligatorio',
45              'category_id.required' => 'La categoría es obligatoria',
46              'description.required' => 'La descripción es obligatoria',
47              'text_1.required' => 'El texto 1 es obligatorio',
48              'text_2.required' => 'El texto 2 es obligatorio',
49              'text_3.required' => 'El texto 3 es obligatorio',
50              'main_img.required' => 'La imagen principal es obligatoria'
51          ];
52
53          $validator = Validator::make($request->all(), $rules, $messages);
54          if($validator->fails()):
55              return back()->withErrors($validator)->with('message', 'Se ha producido un error.')->with('typealert', '
                    danger')->withInput();
56          else:
57              $path = '/'.date('d-m-Y');
58              $fileExt = trim($request->file('main_img')->getClientOriginalExtension());
59              $upload_path = Config::get('filesystems.disks.uploads.root');
60              $name = Str::slug(str_replace($fileExt, '', $request->file('main_img')->getClientOriginalName()));
61              $filename = rand(1,999).'-'.$name.'.'.$fileExt;
62              $file_file = $upload_path.'/'.$path.'/'.$filename;
63
64              $p = new Product;
```

Figura 12 – Ejemplo de controller.

En la figura 12 podemos observar como se realizan las llamadas a los métodos en el controlador, en este caso vemos el inicio de un método *post* en el que definimos unas reglas y mensajes de error para cuando no se cumplen estas en los *inputs* de la vista al pasar las reglas por el validador. Posteriormente y si todos los inputs están correctamente formados, vemos como creamos la ruta, directorio y fichero de imagen para la subida de un producto, cabe destacar que

almacenaremos una copia en el servidor además de crear una miniatura de la imagen para optimizar recursos. Si en algún momento borráramos la imagen esta se eliminaría también del servidor y de igual forma si se modificase.

Posteriormente creamos el objeto producto y asignamos todos los atributos de los inputs e imagen que hemos generado a este. Al trabajar con Laravel nos permite una gran flexibilidad a la hora de trabajar con objetos de nuestros modelos.

Product.php

```php
<?php

namespace App\Http\Models;

use Illuminate\Database\Eloquent\Model;
use Illuminate\Database\Eloquent\SoftDeletes;

class Product extends Model
{
    use SoftDeletes;

    protected $dates = ['deleted_at'];
    protected $table = 'products';
    protected $hidden = ['created_at', 'updated_at'];

    public function getCategory(){
        return $this->hasOne(Category::class, 'id', 'category_id');
    }

    public function getCrossProduct(){
        return $this->hasOne(Product::class, 'id', 'cross_product_id');
    }

    public function getConfiguratorOptions(){
        return $this->hasOne(ConfiguratorOptions::class, 'id', 'configurator_options');
    }
}

```

Figura 13 – Ejemplo de model.

Para la creación de los modelos que usaremos, en primer lugar, Laravel nos permite elaborar migraciones, con el comendo '*php artisan make:migration nombre_de_la_migración'*, para crear tablas asignando todos los elementos y tipos que tendrán. Una vez hemos ejecutado la migración, creando la tabla en base de dados y de igual forma, Laravel nos da la opción de crear un modelo, con el comando '*php artisan make:model nombre_del_modelo*'. En la figura 13 podemos observar un ejemplo de modelo, donde, haciendo uso de las librerías Model y SoftDeletes, creamos el modelo de Product, en las líneas 12, 13 y 14 asignamos a protected tanto los valores necesarios para hacer un borrado suave como la tabla con la que trabajaremos, 'products'. Además, también creamos los métodos que tendrá este modelo, lo que nos permitirá acceder a los objetos que estén relacionados con nuestro modelo en cualquier momento. Para retornar este objeto en cualquiera de los métodos únicamente asignamos el tipo de relación que existe, el modelo de dicho objeto y los campos de ambas bases de datos por los que se relacionarán.

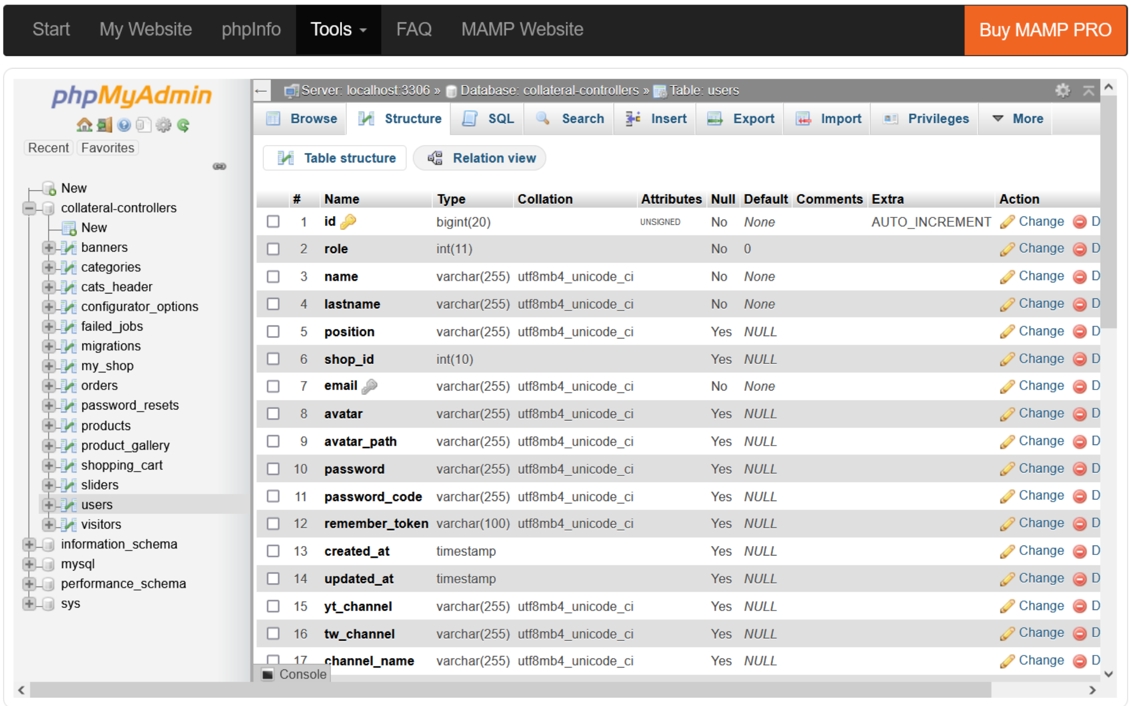

Figura 14 – Ejemplo de tabla en BBDD.

Como hemos mencionado anteriormente, al crear el proyecto en Laravel, podemos y debemos configurar la conexión a la base de datos y haciendo uso de las migraciones podemos crear las tablas de forma rápida con todos los valores, tipos y atributos necesarios. En la figura 14 podemos ver un pequeño ejemplo de la tabla *users* cuya clave primaria será el *id* que será de tipo entero autoincremental y el *email* será único en nuestro sistema.

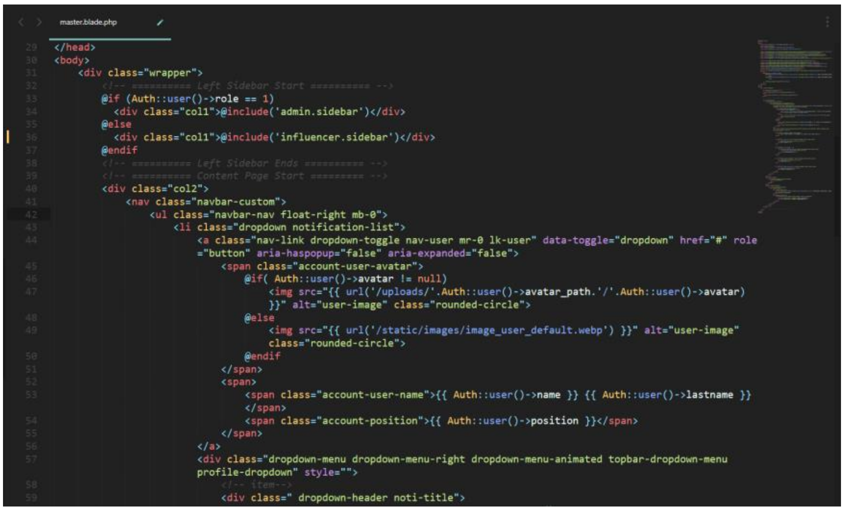

Figura 15 – Ejemplo de master view en admin.

Para la creación de las vistas también hacemos uso de funcionalidades de Laravel, puesto que podemos elaborar pequeños componentes e introducirlos en ciertas secciones del código con la etiqueta *@include*, por ejemplo, para el código de la figura 15 que es el relativo a el componente común a todo el *backend*, en la línea 46, comprobaremos si el usuario que ha hecho *login* tiene rol 1 y si es así mostraremos el menú de administrador, si no fuera el caso, mostraríamos el menú de afiliado. En ambos casos creamos un encabezado con la imagen y nombre del usuario logueado y acceso a su configuración a través del menú *dropdown*. En este fragmento de código también podemos apreciar el uso de las clases propias de Bootstrap.

6. Prueba de concepto

Para entender la prueba de concepto, debemos fijarnos de nuevo en la figura 8, donde definíamos las tablas de la base de datos y por tanto de donde podremos obtener los casos de uso que serán los siguientes:

Administrador

- Podrá logarse en el sistema.
- Podrá reiniciar su contraseña.
- Podrá seleccionar y cambiar el logo, colores e imagen de inicio de la entidad.
- Podrá editar su información personal y profesional dentro de la entidad y foto.
- Podrá cerrar sesión.
- Podrá consultar el resumen diario de la entidad.
- Podrá consultar las métricas de la entidad con comparativa con la semana pasada para las diarias y año anterior para las mensuales.
- Podrá consultar, crear, editar y eliminar afiliados.
- Podrá consultar el correo electrónico de la entidad.
- Podrá consultar y gestionar las redes sociales de la entidad.
- Podrá consultar, crear, editar y eliminar productos.
- Podrá consultar, crear, editar y eliminar categorías.
- Podrá consultar y editar pedidos.
- Podrá consultar, crear, editar y eliminar configuraciones.

Afiliado

- Podrá logarse en el sistema.
- Podrá reiniciar su contraseña.
- Podrá editar su información personal y profesional dentro de la entidad y foto.
- Podrá cerrar sesión.
- Podrá consultar su resumen diario.
- Podrá consultar sus métricas con comparativa con la semana pasada para las diarias y año anterior para las mensuales.

Cliente

- Podrá ver las categorías de la entidad.
- Podrá ver los productos de la entidad.
- Podrá ver la ficha de producto.
- Podrá ver la ficha de producto de configuración.
- Podrá añadir productos al carrito.
- Podrá añadir cupones descuento.
- Podrá realizar búsquedas de productos o categorías.
- Podrá suscribirse a la *newsletter*.

Nuestra plataforma tendrá tres tipos de usuario, el administrador o los administradores del sitio, los afiliados y los clientes. Los dos primeros tendrán acceso al *backend* a través de una página de *login* (ver figura 16). Los clientes únicamente tendrán acceso al *frontend* ya que no existirá una cuenta como tal para ellos. En primer lugar, vamos a realizar una prueba de concepto para administradores y afiliados, a través de la parte de gestión de nuestra plataforma.

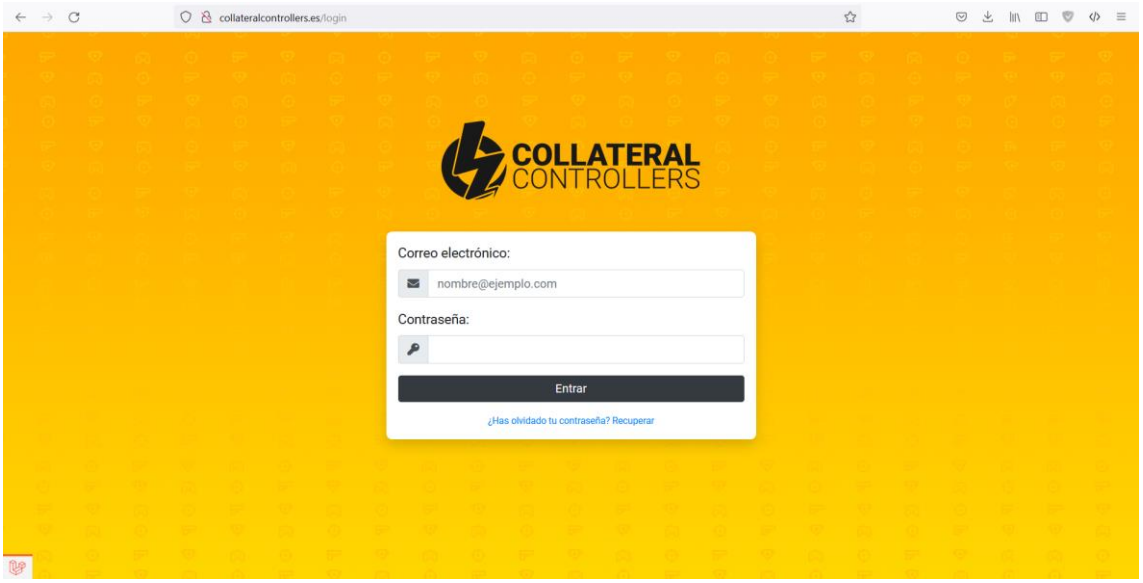

Figura 17 – Página de login.

A los administradores los daremos de alta nosotros mismos en el sistema, esto no nos permite ser un sistema escalable por el momento, pero si llevar un control inicial de los usuarios del sistema. Sin embargo, a los afiliados si les podrán dar de alta desde cada entidad uno de sus administradores. Para ambos casos hemos creado un sistema de contraseñas que se enviará al correo al crear la cuenta, además también es posible cambiarla, siguiendo una pequeña verificación en dos pasos (ver figuras 18-22), si alguno de los usuarios la ha olvidado.

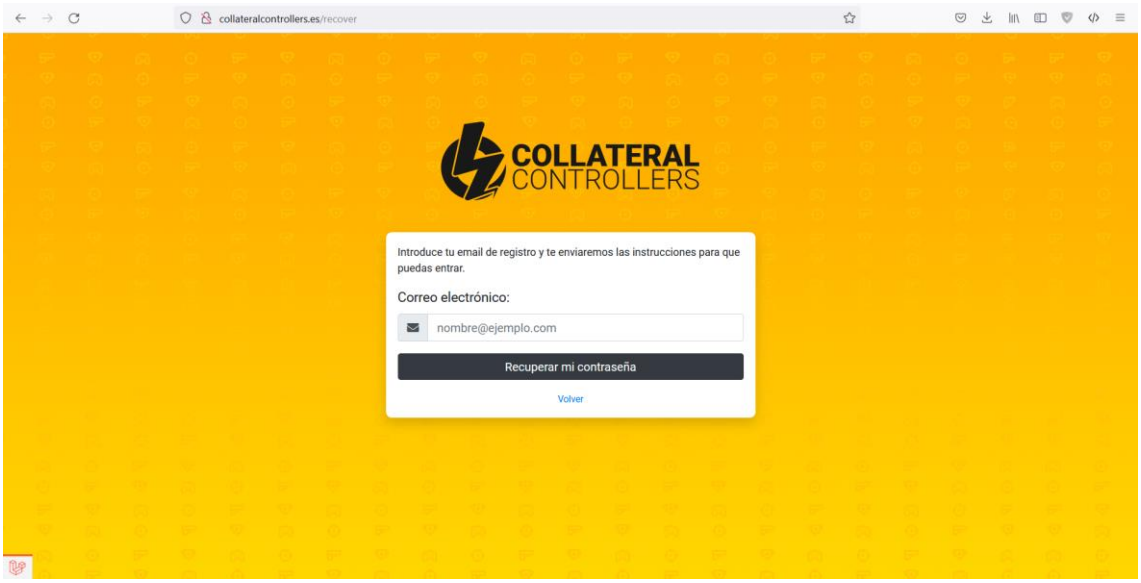

Figura 18 – Página de recover.

En la página de *recover*, el usuario, únicamente, debe introducir su email (ver figura 19), el sistema se encargará de comprobar que es un correo dado de alta y enviará un email con un código de verificación (ver figura 20).

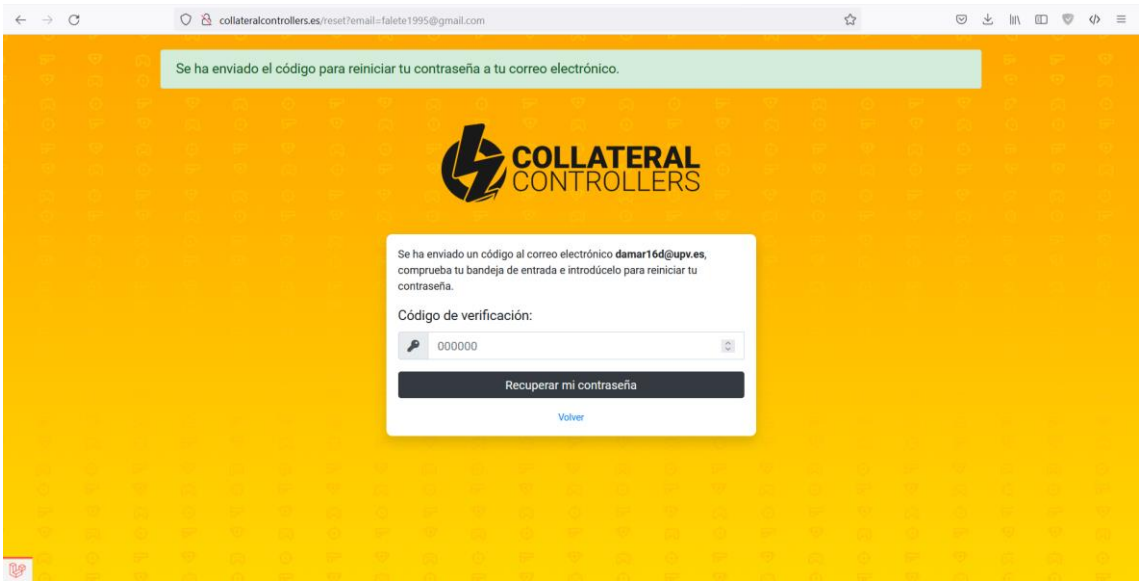

Figura 19 – Confirmación de email para reestablecer la contraseña.

Figura 20 – Email con código de verificación.

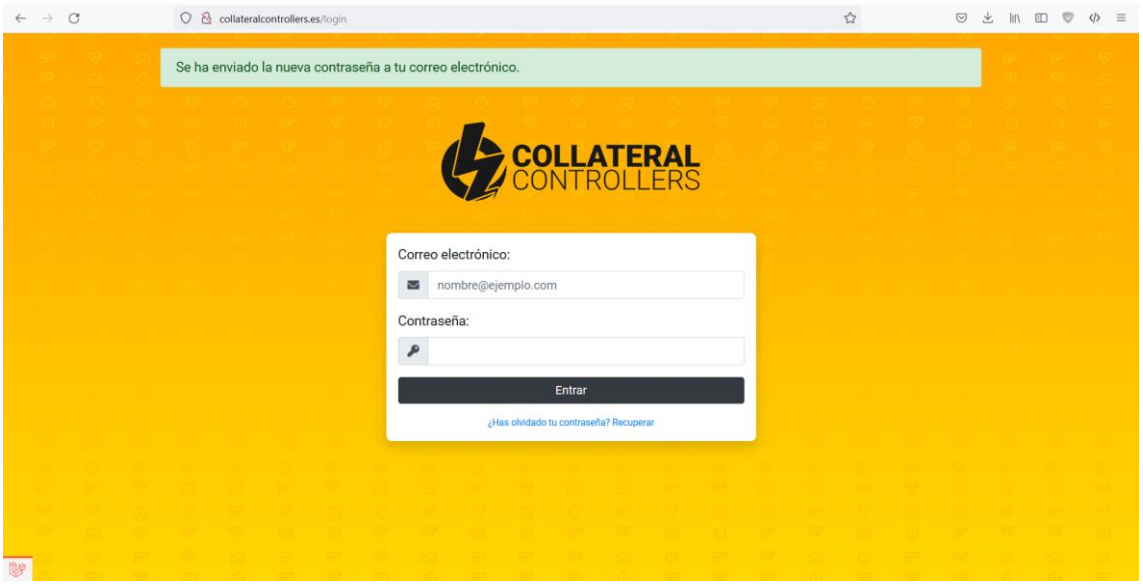

Figura 21 – Confirmación de email con cambio de contraseña.

Figura 22 – Email con contraseña autogenerada.

De esta forma garantizamos que en ningún momento nadie que no sea el propio usuario sabe la contraseña que ha introducido y, además, les permite poder acceder en caso de haberla olvidado.

Una vez hemos detallado el sistema de *login*, vamos a definir para cada uno de los usuarios que tiene acceso a la plataforma qué acciones podrán realizar dentro de ella.

6.1 Administrador

Al iniciar sesión la primera vez, el administrador irá directamente a la página de configuración de la tienda, donde podrá introducir los datos necesarios para empezar a configurarla (ver figura 23). Esta página estará siempre accesible desde el menú desplegable del usuario para que, en caso de querer cambiar colores, logo, analytics, ads o personalización de la parte *frontend* pueda hacerlo con total comodidad.

Debemos destacar que tanto nombre, como logo y colores acompañarán a la entidad en todas las páginas del *backend* y del *frontend* por lo que es aconsejable que estos sigan una guía de estilos y sean los que usaremos en todo momento.

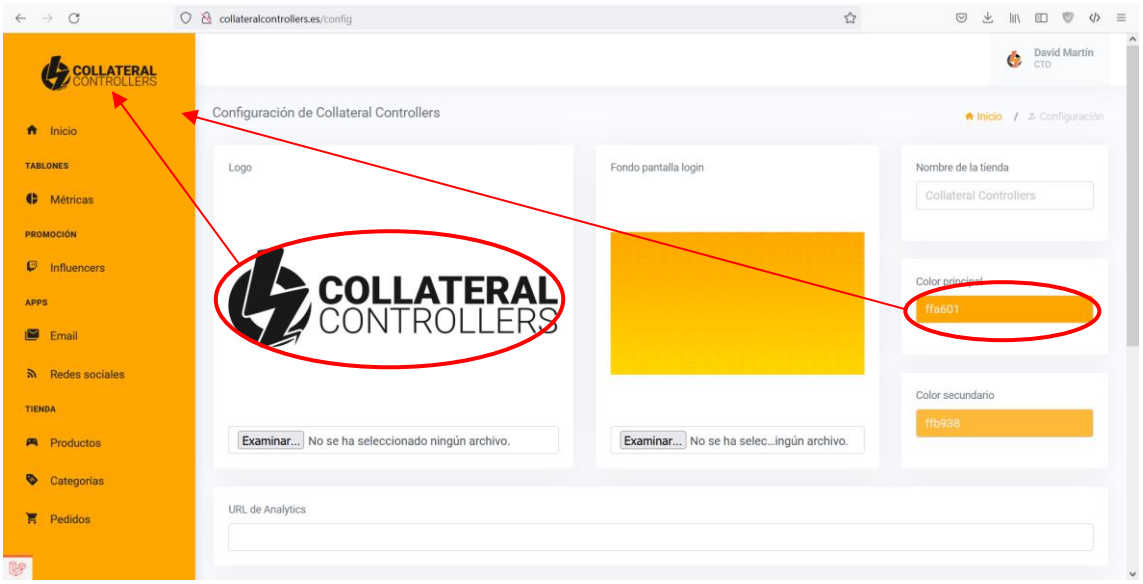

Figura 23 – Página de configuración de la entidad.

Desde el menú desplegable de la esquina superior derecha también podemos personalizar toda la información relativa al propio usuario (ver figura 24).

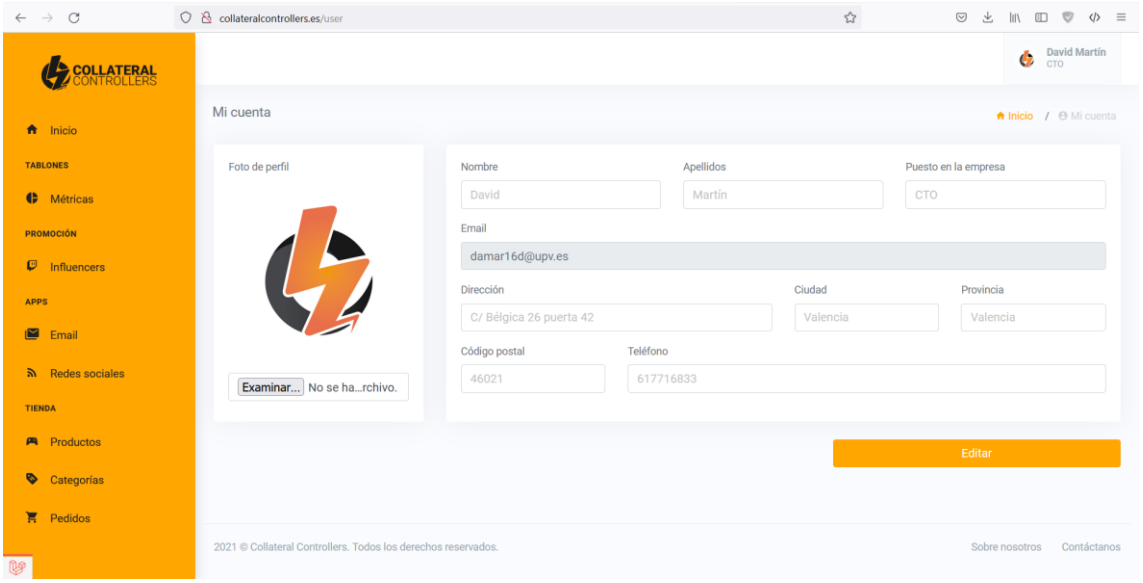

Figura 24 – Página de configuración de usuario.

Cabe destacar que desde el mismo menú desplegable también tenemos la opción de cerrar sesión para salir del sistema y un pequeño apartado de Soporte que nos reservaremos para en un futuro, introducir FAQs y/o datos de contacto para atención al usuario.

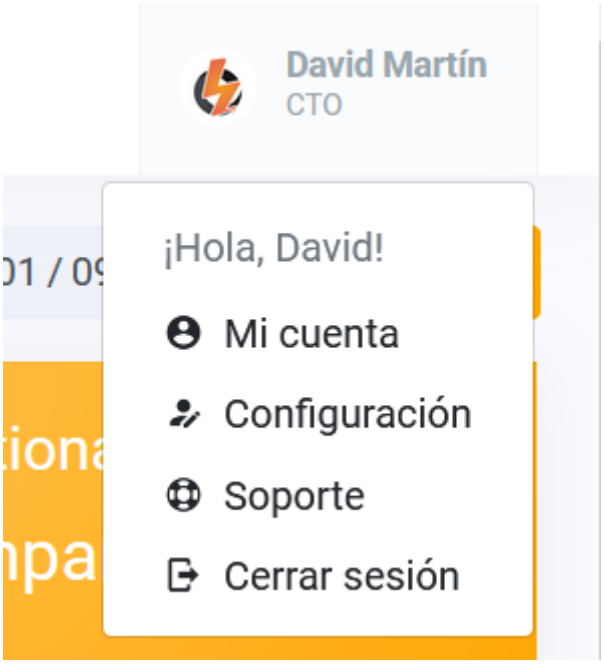

Figura 25 – Menú desplegable de usuario.

Una vez se han configurado tanto los aspectos estéticos de la entidad como la información de usuario, podemos centrarnos en la funcionalidad de la plataforma, empezando por el *dashboard*, donde a simple vista tendremos unas pinceladas rápidas de las estadísticas del día, pedidos pendientes más antiguos, en proceso y últimos pedidos completados y por último un listado con los afiliados que más beneficio nos han generado y los productos más vendidos (ver figura 25).

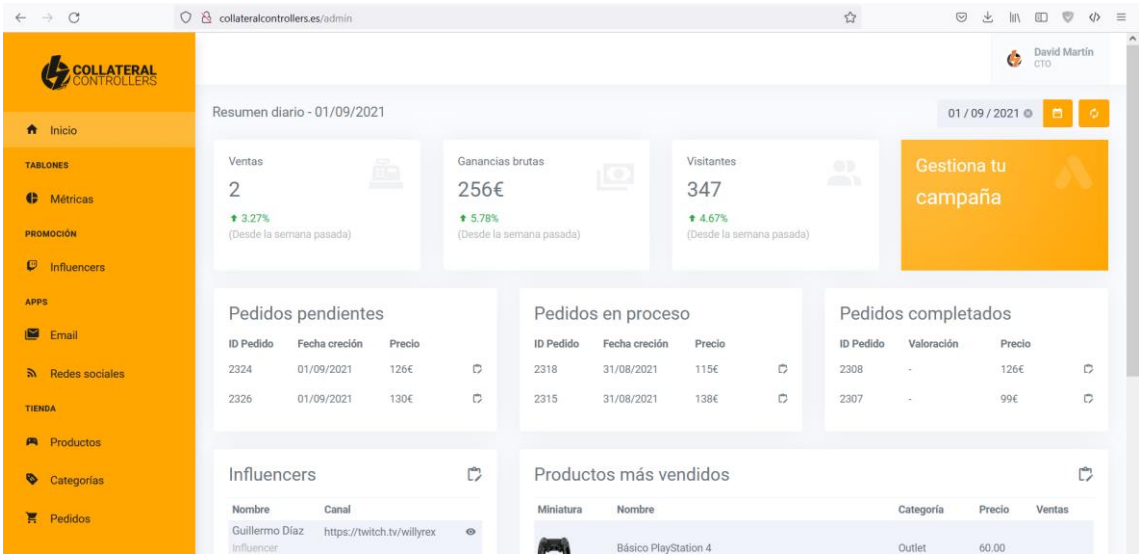

Figura 26 – Página de dashboard.

Debemos destacar también la tarjeta de visitantes, que recoge los visitantes del día en cuestión comparándolos con los obtenidos una semana atrás. En la esquina superior derecha podemos observar también un calendario desplegable que, al seleccionar otro día, nos mostrará las

estadísticas básicas de ese día en comparativa con las obtenidas una semana atrás de esa fecha. Por último, podemos ver que tanto el menú lateral como los botones y la tarjeta de Ads siguen los colores principal y secundario de la entidad, esto hace que la vista del *dashboard* sea agradable e intuitiva.

En la última tarjeta de la primera fila nos encontramos con un 'anuncio' de Google Ads, éste nos llevará a la página de configuración de la entidad si no hemos introducido la URL de la campaña publicitaria o a la propia campaña si ya hubiéramos completado este paso.

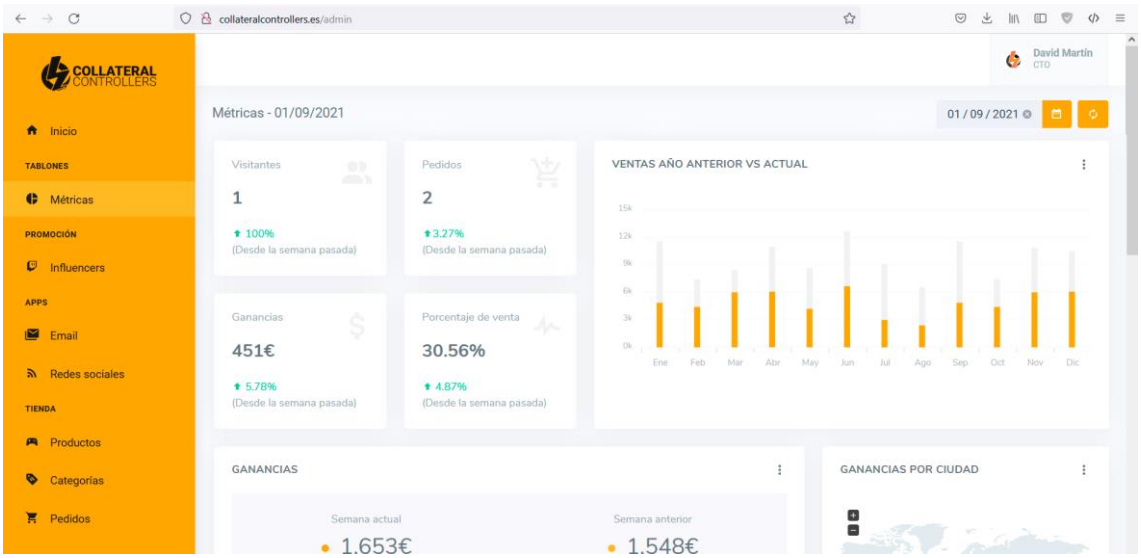

Figura 27 – Página de métricas.

En el tercer elemento del menú podemos encontrar el apartado de *influencers* o afiliados (ver figura 28), en esta página se permitirá editar y crear nuevos afiliados con todos los datos necesarios para que puedan iniciar sesión en el portal que crearemos para ellos y desde allí gestionar sus estadísticas, cupones y campañas.

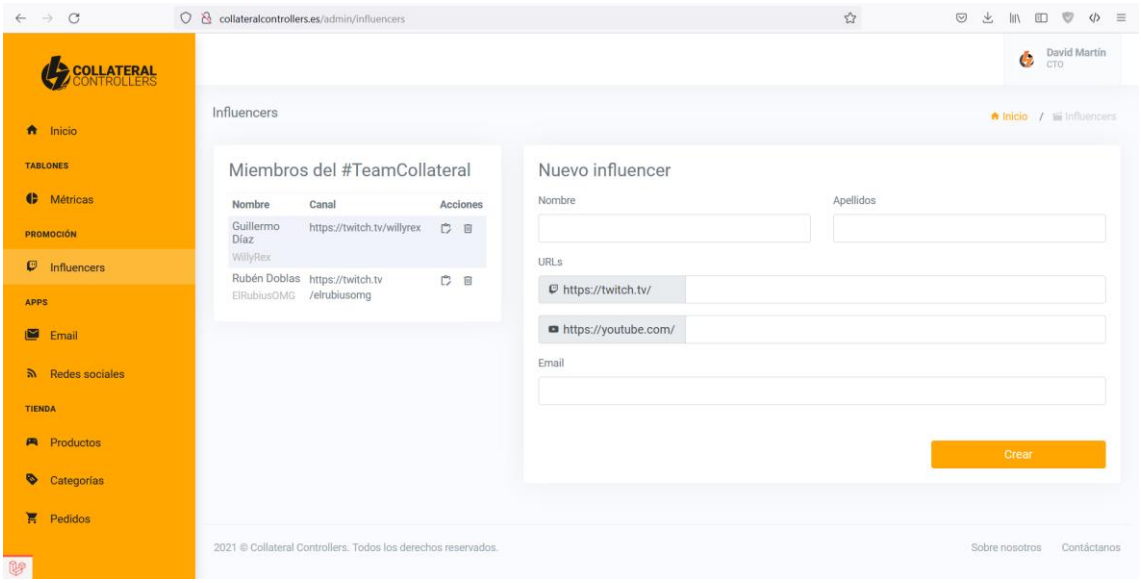

Figura 28 – Página de afiliados.

Dentro del apartado de APPS del menú lateral encontramos tanto el subapartado 'Email' (ver figura 29), que nos permitirá gestionar la cuenta de correo que hemos introducido como gestión de la entidad, como el subapartado 'Redes sociales' (ver figura 30) desde donde podremos ver las feeds de Twitter e Instagram de la entidad, esto nos permite tener más accesibles todas las redes y herramientas que normalmente un administrador tiene que tener abiertas para gestionar una tienda.

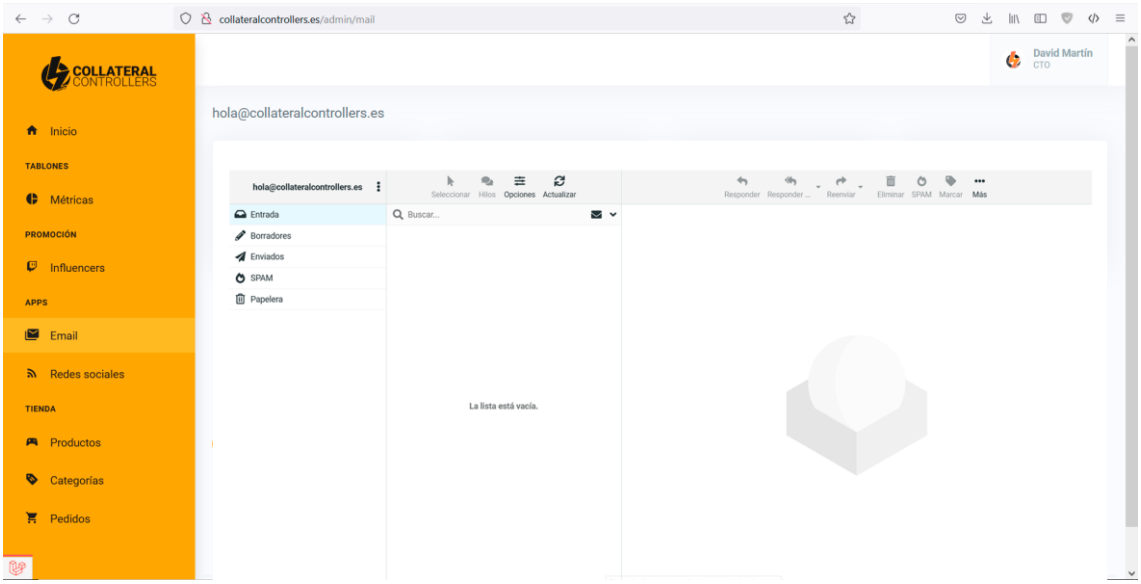

Figura 29 – Página de email.

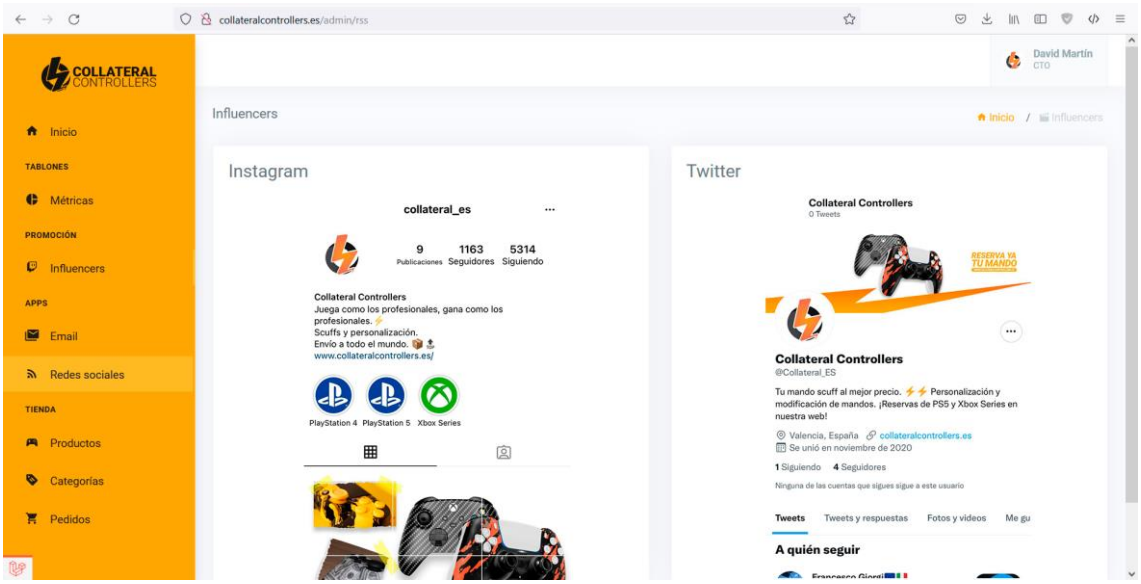

Figura 30 – Página de redes sociales.

Es momento de interactuar con los apartados de la tienda, en los que destacan ‘Productos’, ‘Categorías’, ‘Pedidos’ y ‘Creador’. Desde cada uno de ellos se podrá crear y editar su correspondiente elemento con todos los atributos necesarios para la correcta indexación en la parte frontal de la plataforma.

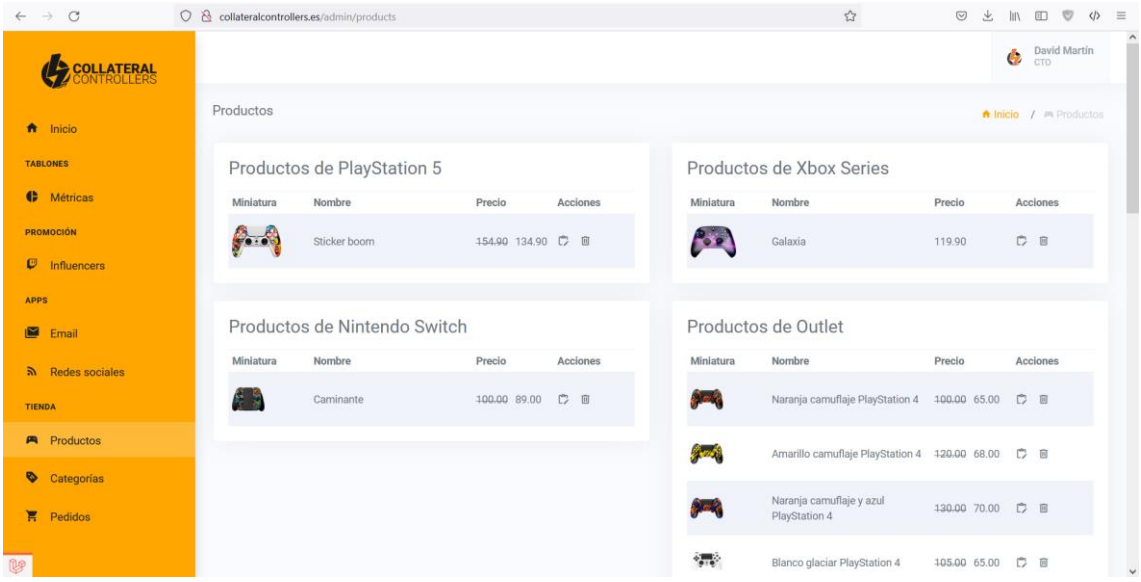

Figura 31 – Página de productos.

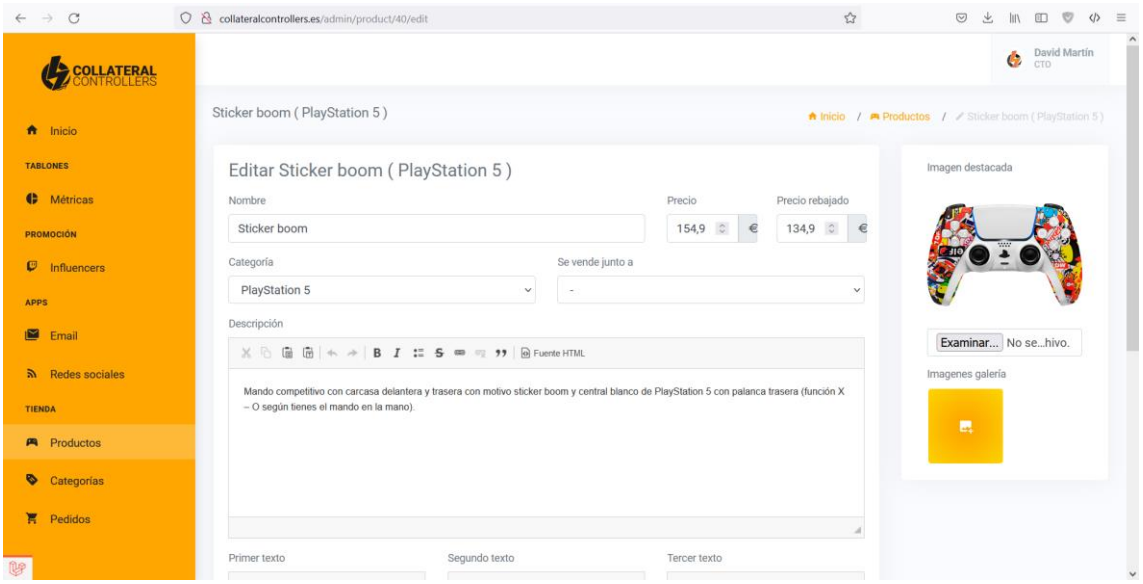

Figura 32 – Página de edición de producto.

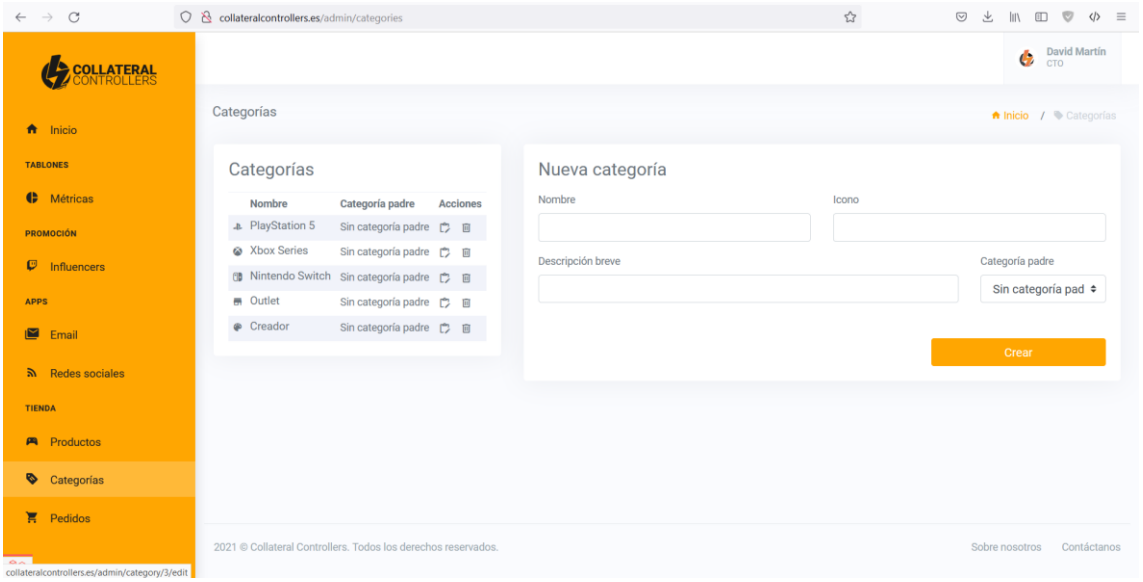

Figura 33 – Página de categorías.

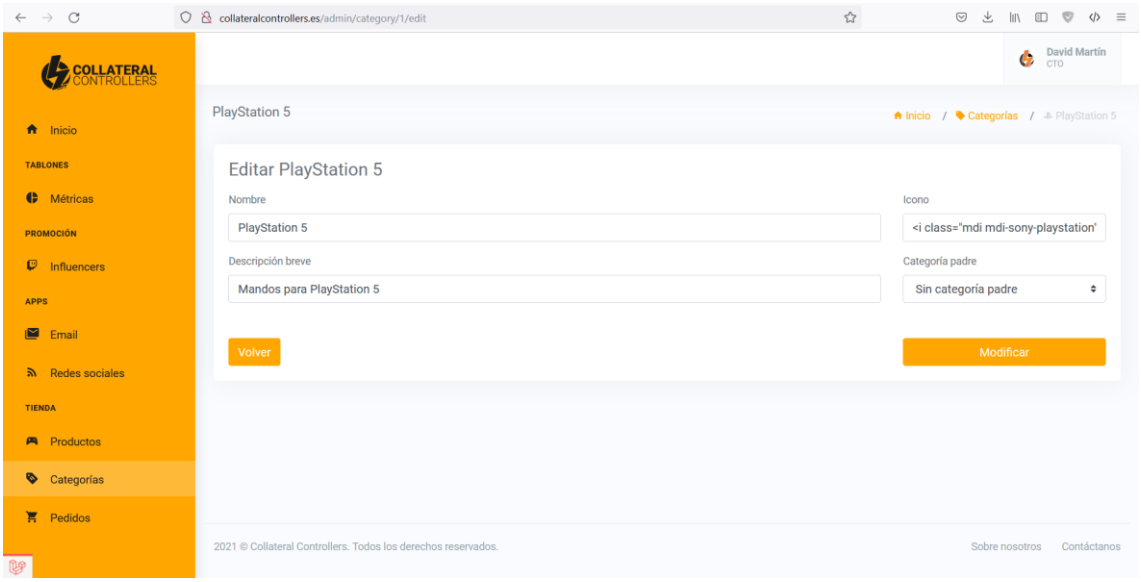

Figura 34 – Página de edición de categoría.

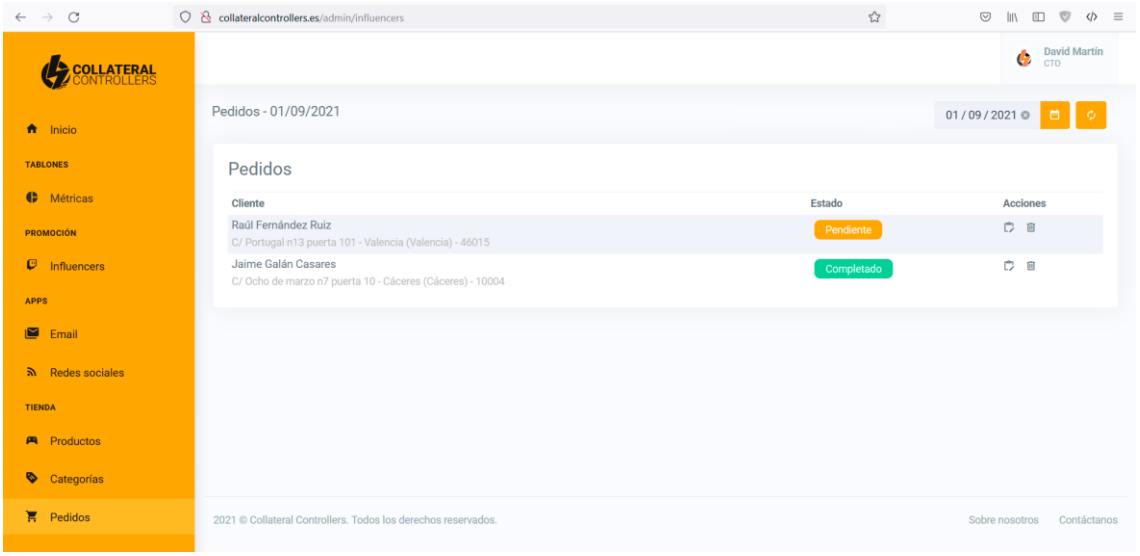

Figura 35 – Página de pedidos.

Podemos observar, además, en la figura 35, que el administrador podrá gestionar los pedidos, de forma que tenga visible en todo momento la dirección de envío y el estado del pedido, al entrar al detalle podrá ver el producto y más detalles correspondientes al pedido.

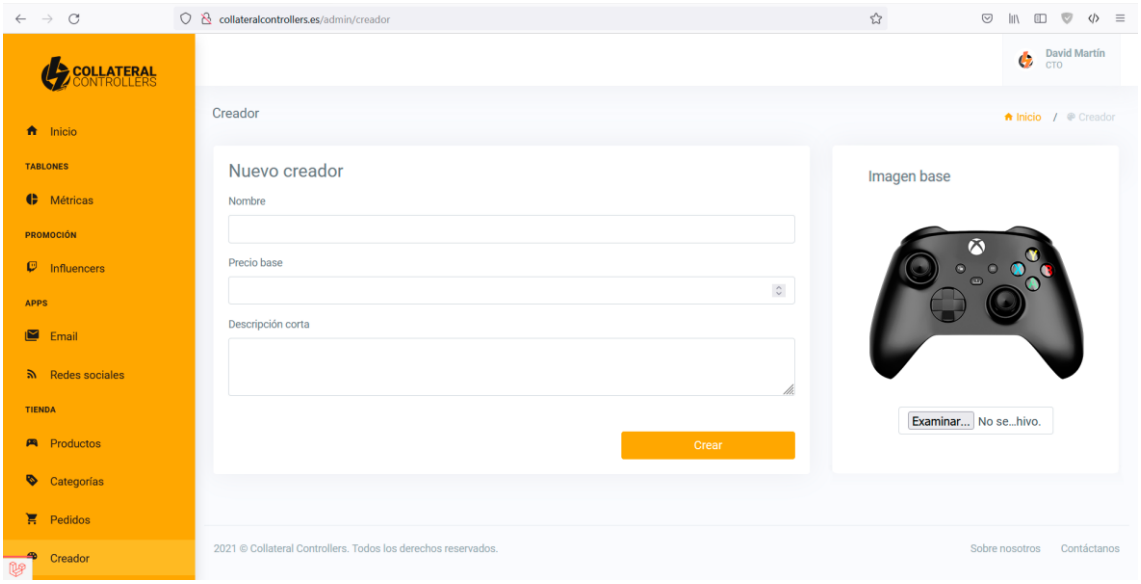

Figura 36 – Página de creador.

6.2 Afiliado

Hemos tratado que la vista de afiliado sea lo más sencilla posible ya que no querrán dedicar mucho tiempo a aprender a usar la plataforma. Su *dashboard* se basa en unas estadísticas rápidas de sus ventas, los códigos promocionales que tienen activos actualmente y un elemento destacado para que puedan configurar sus redes sociales en su perfil en caso de que no lo hayan hecho todavía.

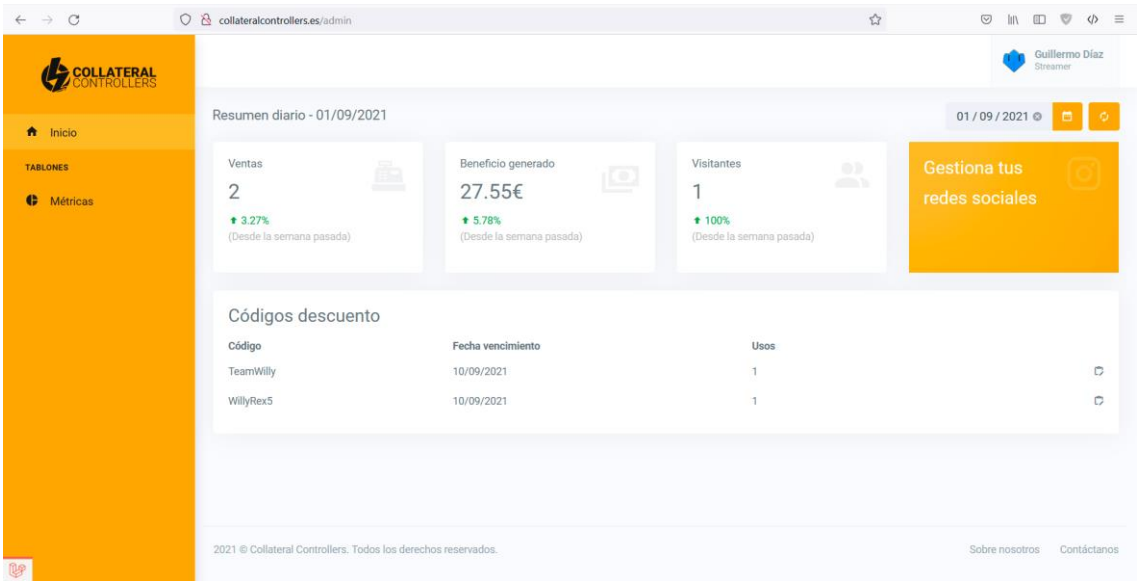

Figura 37 – Página de dashboard afiliado.

Si accedemos al apartado de métricas, ya veremos estadísticas mucho más precisas, donde podremos seleccionar la forma, periodo y métrica que nos interesa ver, todo siguiendo la línea gráfica de la entidad.

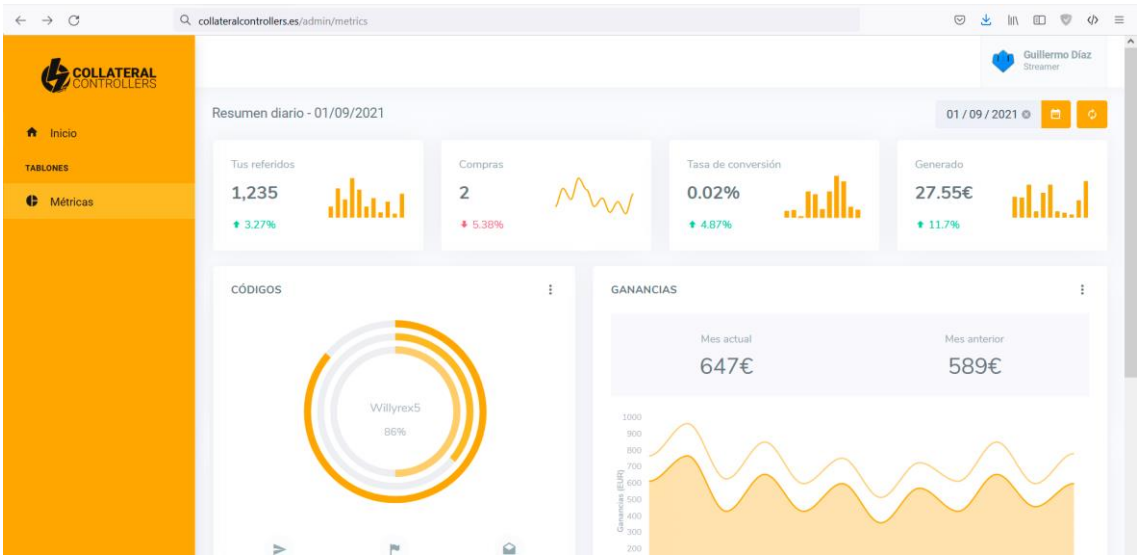

Figura 38 – Página de métricas afiliado.

6.3 Cliente

Nade más entrar a la página de inicio, el cliente podrá observar un vistoso slider en el que destacamos dos productos y dos categorías, esto se genera junto a todos los elementos de la página que el administrador ha indicado en la página de configuración de la entidad, incluyendo elementos del menú, categoría, texto e imagen del banner principal, categorías que aparecerán y número de productos por cada una de ellas… Además, en todas las páginas aparecerá el footer, también autogenerado con las categorías principales y las páginas de interés, incluyendo también, un formulario para suscribirse a la *newsletter*.

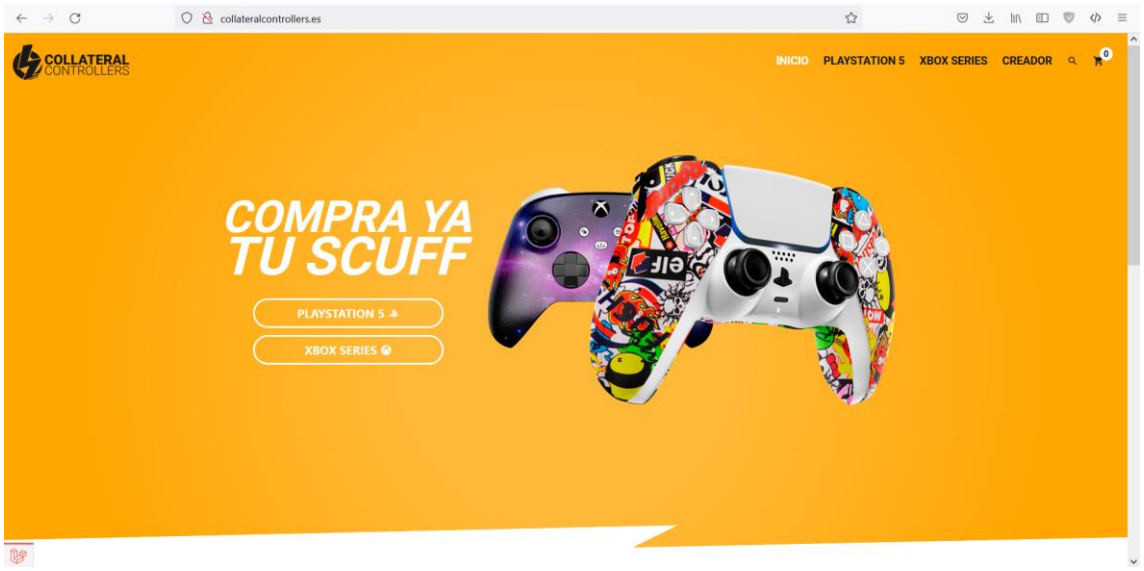

Figura 39 – Página de inicio.

Figura 40 – Banner principal en inicio.

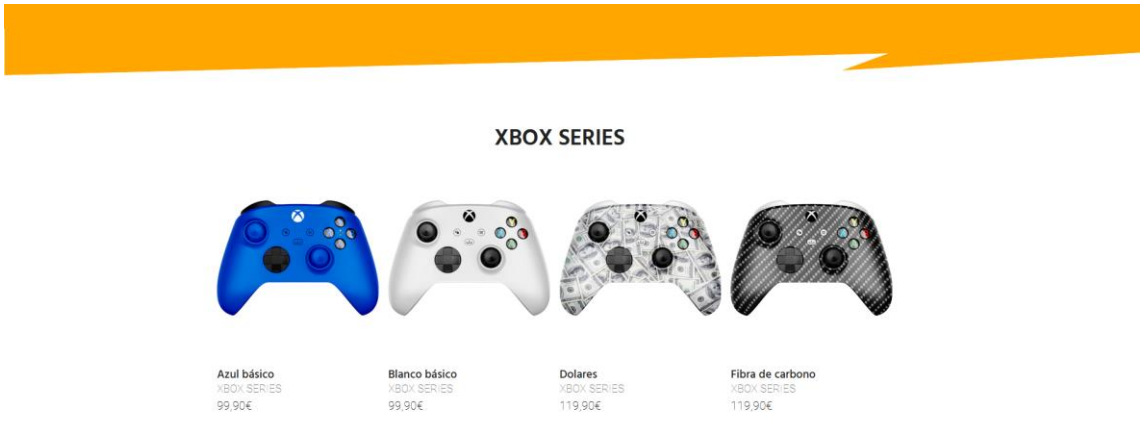

Figura 41 – Apartado categoría en inicio.

Figura 42 – Sticky header.

Figura 43 – Footer.

Una vez hemos navegado a cualquier categoría podemos ver un listado de los productos (ver figura 43), donde el administrador habrá configurado el orden en el que estos aparecerán.

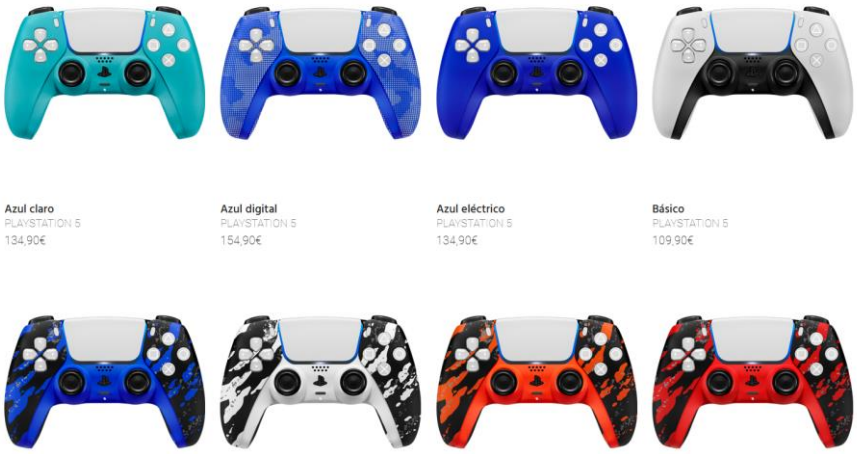

Figura 44 – Página de categoría.

De nuevo, si pinchamos en cualquiera de los productos, navegaremos a su página (ver figura 44), en la que veremos en una página muy visual todas las características del producto, además de poder agregarlo al carrito.

Figura 45 – Página de producto.

Viendo la figura 45 podemos destacar la parte más gráfica de la plataforma, queremos destacar el producto en un diseño minimalista y en el que a simple vista se vean sus características. Más abajo, en tres secciones, incluiremos información adicional sobre el producto.

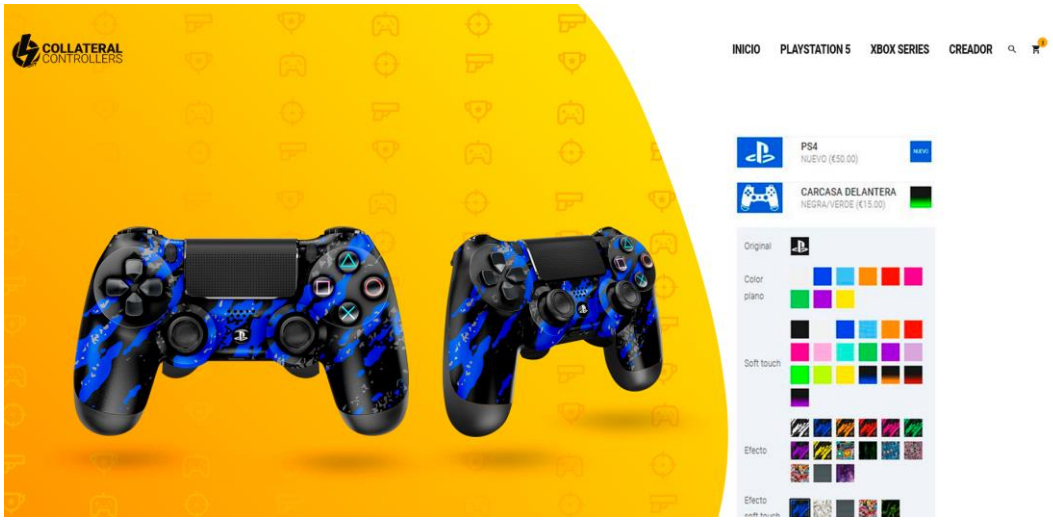

Figura 46 – Página de creador.

Al igual que en la página de producto, en el configurador, el cliente podrá ver de forma gráfica, rápida e intuitiva el aspecto que tomará su producto personalizado, haciendo que no tenga dudas de como quedará. De nuevo nos basamos en una página minimalista y sin distracciones para el cliente.

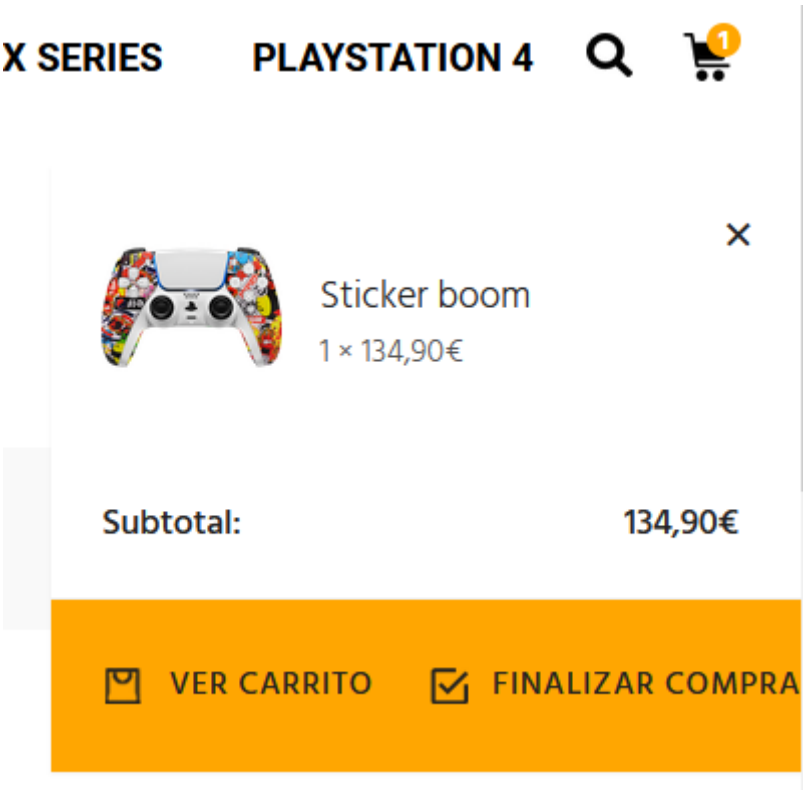

Figura 47 – Carrito desplegable de menú.

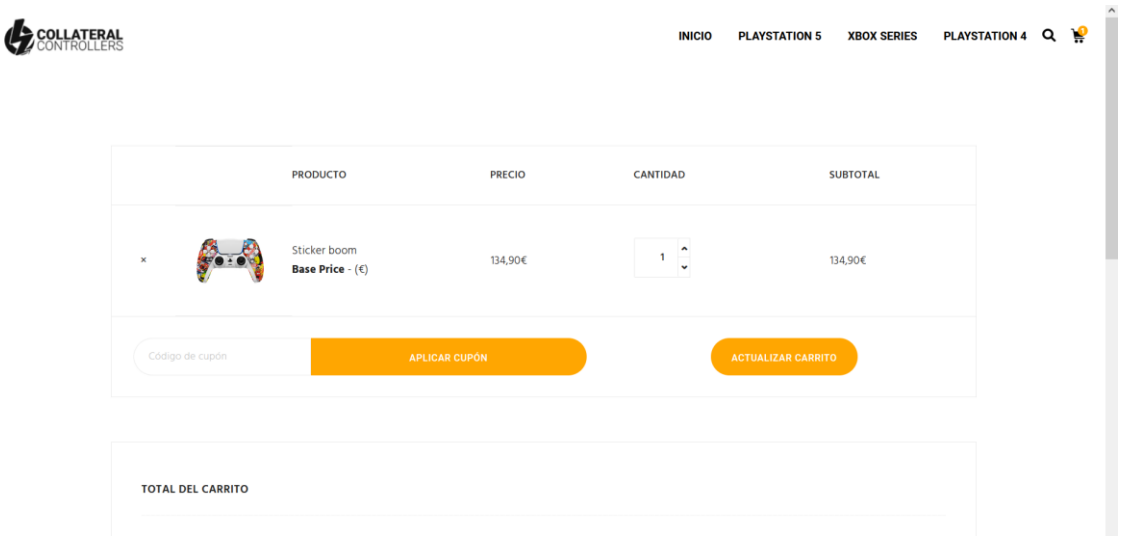

Figura 48 – Página de carrito.

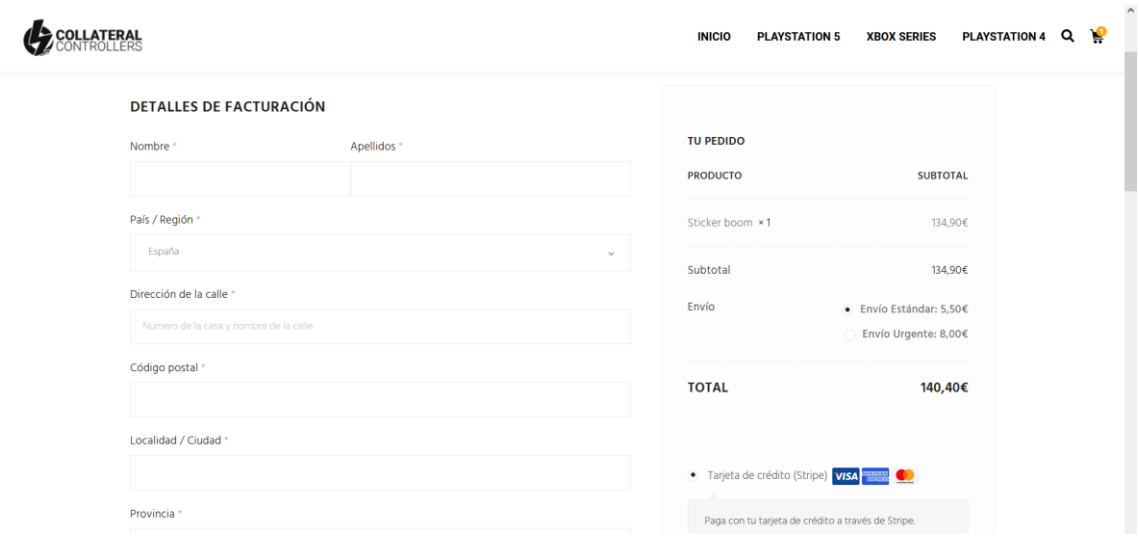

Figura 49 – Página de checkout.

Tanto en el carrito rápido, carrito, como en la página de *checkout* seguiremos la línea de diseño y mostraremos destacadas únicamente las opciones interesantes para realizar la conversión de forma sencilla. Además en la página de finalizar pedido y en el carrito el cliente podrá ingresar el código de descuento que los afiliados han podido publicitar.

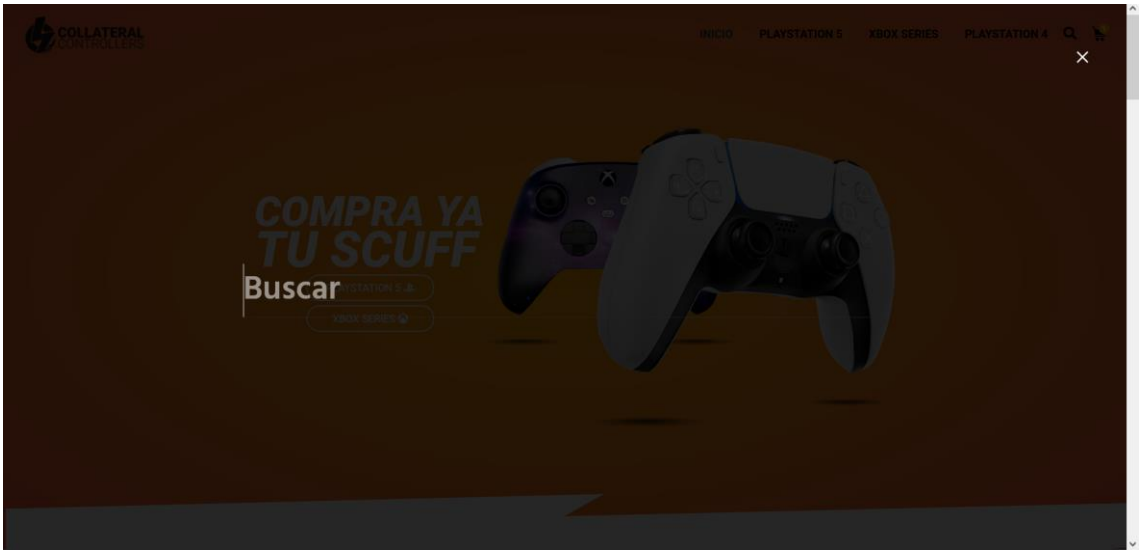

Figura 50 – Diálogo de buscar.

También, en cualquier momento durante la navegación por la web, el cliente puede realizar una simple búsqueda por producto o categoría.

7. Conclusiones y trabajos futuros

Como conclusión para este proyecto y tras haber logrado los objetivos marcados al inicio de él, considero que ha sido realmente importante mantener desde el principio un orden a la hora de desarrollar el código. Esto hace que podamos centrarnos en crear funcionalidades y no en arreglar problemas lógicos.

Como estrategia para atacar la problemática del proyecto me he basado en un primer momento en la experiencia aprendida desarrollando páginas a través de CMS y cómo los clientes necesitaban ciertas funcionalidades que estos no les daban o no podían hacer frente. Esto me hizo empezar a pensar en crear mi propio gestor de contenidos y realizando algunos estudios de herramientas y *frameworks* pude empezar a gestar la idea.

En primer lugar, desarrollé un *planning* para crear la plataforma, detallando en papel todas las dependencias que existirían a lo largo del desarrollo. Obviamente hay elementos que se escapan y se tienen que incluir a posteriori, pero en líneas generales esa planificación me ayudo bastante para poder mantener un orden y limpieza en el proyecto.

Una vez tenía decidida la forma de atacar el proyecto empecé a esbozar la parte gráfica y, de nuevo, basándome en mi experiencia personal decidí la temática de la parte de gestión y la parte frontal.

A la hora de elegir la tecnología y bajo la premisa de aprender lo máximo posible, no solo para crear un proyecto usable en un futuro sino también servirme como experiencia personal y profesional, decidí hacerlo bajo un *framework* que jamás había utilizado, aunque si el lenguaje sobre el que se basa, Laravel y PHP. Para la parte gráfica si decidí usar herramientas que había usado como son Bootstrap y Stylus ya que me resultaba más cómodo trabajar con ellas.

Sin duda creo que la elección de cada una de las herramientas que he utilizado para el proyecto ha sido un completo acierto puesto que me han facilitado en gran medida toda la funcionalidad que estaba en mi cabeza y que hasta este momento no había podido programar en un solo proyecto completo y con valor real.

En cuanto a las implementaciones futuras, aunque se han conseguido los objetivos marcados en un principio, durante el desarrollo de la plataforma he encontrado nuevas funciones que serían realmente interesantes para aumentar el valor de ella. Además, puesto que todo el proyecto se ha organizado correctamente y está bastante modularizado, no será difícil realizar estas mejoras en un futuro.

En mi trayectoria profesional y siempre que me sea posible, usaré la propia plataforma para realizar las páginas webs que como *freelance* me puedan encargar. Por lo tanto yo mismo seré el encargado de detectar mejoras que puedan servir de ayuda en la creación de las webs como en la gestión de ellas.

8. Bibliografía

Estudio ecommerce. Consultado en https://www.iebschool.com/blog/estudio-anual-2021-e-commerce/. Fecha de consulta: junio de 2021.

WordPress. Consultado en https://wordpress.com/es/. Fecha de consulta: junio de 2021.

Drupal. Consultado en https://www.drupal.org/. Fecha de consulta: junio de 2021.

Joomla. Consultado en https://www.joomla.org/. Fecha de consulta: junio de 2021.

Magento. Consultado en https://magento.com/. Fecha de consulta: junio de 2021.

Shopify. Consultado en https://www.shopify.es/. Fecha de consulta: junio de 2021.

Plugins. Consultado en https://themeforest.net/ y stores únicas de cada CMS. Fecha de consulta: octubre de 2020.

Laravel. Consultado en https://laravel.com/. Fecha de consulta: octubre de 2020.

Stylus. Consultado en https://stylus-lang.com/. Fecha de consulta: octubre de 2020.

Bootstrap. Consultado en https://getbootstrap.com/. Fecha de consulta: octubre de 2020.

MAMP. Consultado en https://www.mamp.info/en/windows/. Fecha de consulta: octubre de 2020.

MySQL. Consultado en https://www.mysql.com/. Fecha de consulta: octubre de 2020.