TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

UNIVERSIDAD DE ALMERIA

ESCUELA SUPERIOR DE INGENIERÍA

Desarrollo de módulos en Magento

Curso 2020/2021

Alumno/a:
Paola Alicia Díaz Aliaga

Director/es:
José Antonio Torres Arriaza

Resumen

El comercio electrónico ha estado creciendo considerablemente durante la última década y los consumidores han optado para realizar sus compras cada vez más por las tiendas online.

Durante el periodo pleno de la pandemia del COVID-19, la digitalización se ha impulsado y el sector eCommerce ha sido el más beneficiado. La previsión de crecimiento digital que había para los próximos años se ha visto acelerada y ha impactado a los usuarios de todo el mundo cambiando sus preferencias de consumo durante la pandemia. Además, ha permitido que negocios más pequeños puedan dar el paso al sector online. [1] [43]

El rol de un CMS es ayudar a gestionar la página de una manera sencilla. El desarrollo con Magento permite ampliar las funcionalidades del CMS con la creación de módulos bajo las necesidades de la tienda online (que en este proyecto ha sido mejorar el motor de búsqueda) e intentando que el gestor pueda modificar algunos ajustes bajo su demanda. [44]

El objetivo principal de este trabajo es mejorar el buscador y se han creado dos módulos que satisfacen esa finalidad. El objetivo secundario fue crear un dashboard que analice a los usuarios y los términos de búsqueda, decisión que se tomó tras estudiar los buscadores más usados. Para cumplir con las necesidades de un CMS, se desarrolló para cada módulo en el panel de administración, las opciones de habilitar/deshabilitar las funcionalidades.

Debido al impacto del eCommerce actualmente y la capacidad de extensión mediante módulos de Magento, se considera que es un proyecto innovador por mejorar el canal de compras para usuarios de manera más simple y cómoda. También lo es por el uso de la tecnología utilizada ya que las más antiguas rondan los 30 años y las más recientes la mitad.

El trabajo se ha desarrollado para la empresa Hiberus Tecnologías como propuesta para mejorar las funcionalidades del buscador al tener pocos desarrollos centrados en él. La finalidad es poder aplicar las extensiones que se han desarrollado a proyectos que lo requieran.

Palabras clave: comercio electrónico, eCommerce, módulo, CMS, framework

Abstract

E-commerce has been growing considerably over the last decade and consumers have increasingly opted for online stores to make their purchases. During the full period of the COVID-19 pandemic, digitization has been improved and eCommerce sector has benefited the most. The digital growth forecast for the coming years has accelerated and had an impact in users around the world by changing their consumption preferences during the pandemic. In addition, it has allowed smaller businesses to take the step to the online sector.

The CMS role is to help to manage the page in a simple way. The development with Magento allows to extend the functionalities of the CMS with the creation of modules under the needs of the online store (which in this project has been to improve the search engine) and givinf the chance for the manager to modify some settings on demand.

The main objective of this work is to improve the search engine and two modules have been created that satisfy that purpose. The secondary objective was to create a user analysis, a decision that was made after studying the most used search engines. To meet the needs of a CMS, it was developed for each module in the administration panel, the options to enable / disable the functionalities.

Due to the current impact of eCommerce and the ability to extend through Magento modules, it is considered an innovative project to improve the purchasing channel for users in a simpler and more comfortable way. It is also due to the use of the technology used since the oldest ones are around 30 years old and the most recent half.

This proyect has been developed for the company Hiberus Technologies as a proposal to improve the functionalities of the search engine by having few developments focused on it. The purpose is to be able to apply the extensions that have been developed to projects that require it.

Key words: comercio electrónico, eCommerce, módulo

Gracias a la empresa Hiberus por darme la oportunidad de poder realizar mi trabajo de fin de grado con ellos y espero que las funcionalidades que he desarrollado, sirvan en el futuro para ser aplicadas en un cliente real si requiere de estas necesidades.

UNIVERSIDAD DE ALMERÍA

ESCUELA SUPERIOR DE INGENIERIA

GRADO EN INGENIERÍA INFORMÁTICA

TRABAJO DE FIN DE GRADO

DESARROLLO DE MODULOS EN MAGENTO

AUTOR:

PAOLA ALICIA DIAZ ALIAGA

DIRECTOR:

JOSÉ ANTONIO TORRES ARRIAZA

Índice de contenidos

Índice de figuras

Índice de tablas

Listado de acrónimos

Abreviatura	Denominación original	Significado en castellano
CMS	Content Management System	Software de gestión de contenidos
eCommerce	Electronic Commerce	Comercio electrónico
SQL	Structured Query Language	Lenguaje de consulta estructurado
SEO	Search Engine Optimization	Optimización de motores de búsqueda
UI	User Interface	Interfaz de usuario
ERP	Enterprise Resource Planning	Sistema de planificación de recursos empresariales
IDE	Integral Development Environment	Entorno de desarrollo integrado
MVC	Model-View-Controller	Modelo vista controlador
SO	Operating System (OS)	Sistema Operativo
SAAS	Software as a service	Software como servicio
API	Application Programming Interface	Interfaz de programación de aplicaciones
DAO	Data Access Object	Objeto de acceso a datos
DTO	Data Transfer Object	Objeto de transferencia de datos
CRUD	Create-Read-Update-Delete	Crear-leer-actualizar-borrar
ORM	Object Relational Mapping	Mapeo objeto relacional
REST	Representational State Transfer	Transferencia de estado representacional
ACL	Access Control List	Lista de control de acceso
DOM	Document Object Model	Modelo de objeto del documento

Tabla Abreviaturas

Glosario de términos

Beats: “Son agentes de datos ligeros. es una plataforma gratuita y abierta para agentes de datos con solamente propósito. Envían datos de cientos o miles de máquinas y sistemas a Logstash o Elasticsearch.” [35]

Logstash: “Es uno de los productos principales del Elastic Stack, se usa para agregar y procesar datos y enviarlos a Elasticsearch. Logstash es una pipeline de procesamiento de datos open source y del lado del servidor que permite ingestar datos de múltiples fuentes simultáneamente y enriquecerlos y transformarlos antes de que se indexen en Elasticsearch.” [14]

Kibana: “Es una herramienta de visualización y gestión de datos para Elasticsearch que brinda histogramas en tiempo real, gráficos circulares y mapas. Kibana también incluye aplicaciones avanzadas, como Canvas, que permite a los usuarios crear infografías dinámicas personalizadas con base en sus datos, y Elastic Maps para visualizar los datos geoespaciales.” [14]

Debuggear: “El debugger o mejor dicho en español el depurador, es una Herramienta o Aplicación que permite la ejecución controlada de un programa o código para seguir cada instrucción ejecutada y localizar asi Bugs o errores, códigos de protección, etc.” [36]

Mappear: “El mapeo de datos es el proceso de extraer campos de datos de uno o varios archivos de origen y hacerlos coincidir con sus campos de destino relacionados en el destino” [37]

API (Interfaz de programación de aplicaciones): “Una API es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones.” [38]

API REST (Representational State Transfer): “Una API de transferencia de estado representacional (REST), o API de RESTful, es una interfaz de programación de aplicaciones (API o API web).” [39] Usa solicitudes HTTP que son las responsables de hacer las operaciones básicas necesarias (POST, GET, DELETE, PUT) para la manipulación de datos.

Dashboard: “Un dashboard o cuadro de mando es una herramienta de business intelligence que representa, de manera visual, los KPI’s o métricas que afectan en el logro de los objetivos de tu estrategia de Marketing digital. Con los dashboard podemos analizar los datos y detectar los posibles problemas, así como encontrar las acciones que podemos llevar a cabo para solucionarlos.” [40]

Framework: “Un framework es un marco o esquema de trabajo generalmente utilizado por programadores para realizar el desarrollo de software. Utilizar un framework permite agilizar los procesos de desarrollo ya que evita tener que escribir código de forma repetitiva, asegura unas buenas prácticas y la consistencia del código.” [41]

Frontend: Se le conoce como el “lado del usuario” porque incluye los elementos que se visualizan en la pantalla (colores, tipo de letra, animaciones…).[42]

Backend: Se le conoce como el “lado del servidor” y lo componen elementos que almacenan y organizan los datos en una web. [42]

1. Introducción

A lo largo de este documento se van a abordar diferentes aspectos que han hecho posible la implementación de módulos para el gestor de contenidos Magento. Ha sido desarrollado en la empresa Hiberus Tecnología con el fin de desarrollar un módulo genérico (en este trabajo cada funcionalidad representa un módulo) basado en el motor de búsqueda con la finalidad de ampliar su funcionalidad por defecto.

Para comprender los propósitos que se quieren conseguir con este trabajo (que se manifiestan en el apartado 1.2) se va a explicar a lo largo del trabajo qué es el comercio electrónico y cómo es su impacto en la actualidad, se va a destacar sus beneficios, las principales plataformas en este sector, cómo funciona la herramienta seleccionada para el desarrollo y cómo se ha llevado a cabo cada implementación.

1.1. Motivación

Desde principios del año 2020, se ha producido un aumento en las compras online debido a la pandemia del COVID-19. Por el consecuente confinamiento, al limitar el movimiento de las personas, ha derivado en un cambio de hábitos en el momento de realizar sus compras de forma más fácil y segura. [1]

La digitalización que ha ocurrido en el sector del eCommerce, también conocido como comercio electrónico, ha convertido que la opción de que un negocio pueda vender sus productos a través de Internet sea una necesidad. [2]

Dado el constante crecimiento de las tiendas online, la competencia también ha aumentado y todas quieren aumentar sus ventas. Para ello, además de que los servicios que ofrezca (productos, métodos de pago, envío y postventa) sean correctos, existen otros aspectos para su buen funcionamiento como lo es el buscador.

Haciendo una analogía con una tienda física, el buscador de una tienda online realiza la tarea de un dependiente, quien es el que ayuda a encontrar aquello que se necesita. Al igual que en una tienda tradicional, si esto falla, lo más probable es que no se realice la compra satisfactoriamente y que incluso, se abandone el sitio. [3]

Por este motivo, en un eCommerce es esencial tener un buscador que funcione eficazmente para mejorar la experiencia del cliente y, por ello, se ha decidido que el trabajo va a estar orientado al entorno eCommerce y, en especial a la funcionalidad de la búsqueda.

1.2. Objetivos

El **objetivo principal** de este trabajo es el desarrollo de módulos en la plataforma eCommerce de Magento en su versión 2.4 mediante las herramientas que pone a disposición el gestor de contenidos. Consiste en mejorar el funcionamiento del buscador por defecto que trae y también permitir un análisis de los usuarios y términos para mejorar la búsqueda o el catálogo de la tienda online.

Se han definido los siguientes **objetivos específicos** para el desarrollo del módulo:

Objetivo 1. Crear una redirección a las búsquedas que devuelvan un resultado a un producto en lugar de un listado de resultados

Objetivo 2. Crear una representación gráfica de los datos obtenidos de la sesión de los usuarios y de la búsqueda en la vista de administrador

Objetivo 3. Crear un desplegable de resultados en el buscador que además de mostrar la sugerencia de términos también muestre la imagen, el precio y el nombre del producto.

Objetivo 4. Añadir la opción de habilitar/deshabilitar estas funcionalidades en la vista de administrador y además, en los módulos con diversas configuraciones permitir habilitar/deshabilitar cada una de ellas o modificar los parámetros predefinidos.

El propósito de este trabajo es conseguir ampliar las funcionalidades del motor de búsqueda que trae por defecto Magento en su versión Open Source y poder aplicar las soluciones a trabajos reales de clientes en la empresa.

1.2.1. Diagrama de Gantt inicial

La planificación temporal prevista que se detalló en el anteproyecto fue la siguiente:

La 1ª fase se centra en el análisis:

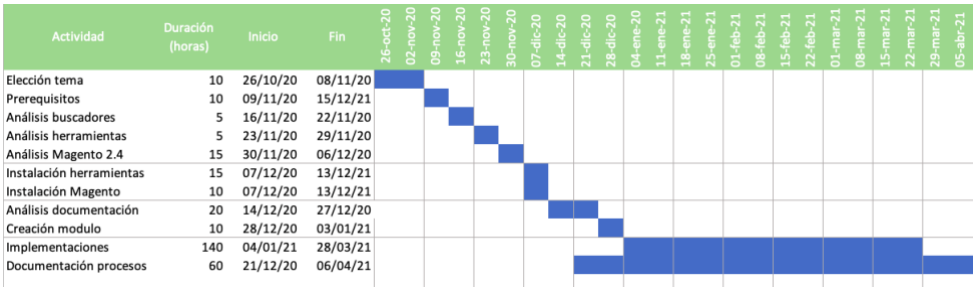
- Se estudian los posibles temas que se pueden realizar
- Una vez elegido el tema, se procede a analizar los requisitos de desarrollo que se van a utilizar para llevarlo a cabo. Se estudian las herramientas software, los lenguajes de programación y el entorno que se usará
- Según el tema elegido, en este caso, el desarrollo de un módulo avanzado para Magento 2.4, el cual se va a centrar en la funcionalidad del buscador, se pretende obtener información acerca de ello estudiando los buscadores actuales para evaluar que funciones se podrían implementar

La 2ª fase está enfocada a la instalación. Según los requisitos que han sido estudiados se procederá a la instalación de las herramientas y la plataforma Magento.

En la 3ª fase se estudia el entorno de Magento: la estructura básica de magento (árbol de directorios), los componentes más importantes del framework (layouts, bloques, templates, models, plugins, etc) y que funciones realizan.

Una vez que se ha comprendido la estructura y sus funcionalidades y ya se conocen los requisitos en Magento para crear el módulo, se comienza con la creación de este.

La 4ª fase se centra en el desarrollo de las funcionalidades descritas en los objetivos. El desarrollo de las implementaciones se irán haciendo paralelamente junto con la documentación de los procesos.



Actividad	Duración (horas)	Inicio	Fin
Elección tema	10	26/10/20	08/11/20
Prerequisitos	10	09/11/20	15/12/21
Análisis buscadores	5	16/11/20	22/11/20
Análisis herramientas	5	23/11/20	29/11/20
Análisis Magento 2.4	15	30/11/20	06/12/20
Instalación herramientas	15	07/12/20	13/12/21
Instalación Magento	10	07/12/20	13/12/21
Análisis documentación	20	14/12/20	27/12/20
Creación modulo	10	28/12/20	03/01/21
Implementaciones	140	04/01/21	28/03/21
Documentación procesos	60	21/12/20	06/04/21

Figura 1.3.1: Diagrama de Gantt inicial

1.2.2. Diagrama de Gantt final

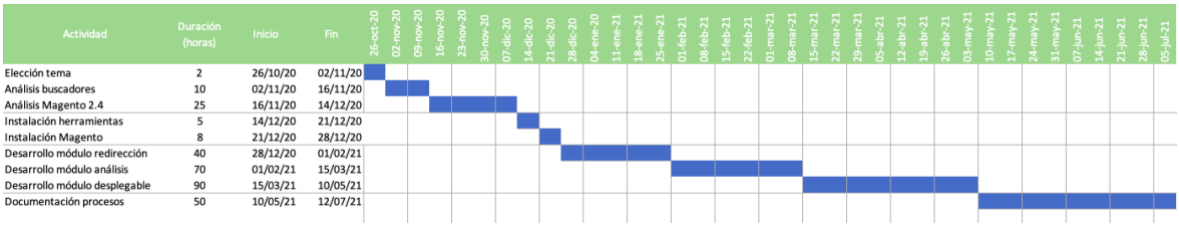
Se va a comparar la planificación inicial que se detalló en el anteproyecto contra la que se ha realizado realmente.

En la 1ª fase en el diagrama de Gantt inicial se detallaron más actividades que en el final. Se optó por minimizar el número de actividades en el diagrama de Gantt final ya que en el análisis de Magento 2.4. es dónde se ha investigado en conjunto qué herramientas se necesitaban tener antes de instalar el entorno y qué lenguajes van a utilizarse. También es la actividad que corresponde al estudio de cómo es la estructura de Magento y cómo funciona (anterior fase 3).

Respecto a la 2ª fase relacionada con las instalaciones, el tiempo que se utilizó realmente fue menor que el previsto y se realizaron consecutivamente. La instalación de herramientas se realizó primero para asegurar que la instalación de Magento fuera con los menores problemas posibles.

La nueva fase 3 se centra únicamente en el desarrollo de los módulos y se muestra cuál ha sido el tiempo real para cada uno. Se observa que el tiempo ha ido aumentando progresivamente, al igual que la complejidad que suponían.

La última fase se ha centrado únicamente en la redacción del documento del proyecto de fin de grado que se ha realizado al finalizar los desarrollos, a pesar de que en el diagrama inicial se pretendía hacer al mismo tiempo que los desarrollos.



Actividad	Duración (horas)	Inicio	Fin
Elección tema	2	26/10/20	02/11/20
Análisis buscadores	10	02/11/20	16/11/20
Análisis Magento 2.4	25	16/11/20	14/12/20
Instalación herramientas	5	14/12/20	21/12/20
Instalación Magento	8	21/12/20	28/12/20
Desarrollo módulo redirección	40	28/12/20	01/02/21
Desarrollo módulo análisis	70	01/02/21	15/03/21
Desarrollo módulo desplegable	90	15/03/21	10/05/21
Documentación procesos	50	10/05/21	12/07/21

Figura 1.3.2: Diagrama de Gantt final

1.3.3 Estimación de costes

El precio de un proyecto en Magento suele depender de la experiencia del programador y del lenguaje de programación. En el caso de un programador junior de Php/Javascript se ha obtenido un costo por hora de 20€ para el desarrollo (incluyen los desarrollos frontend y backend) y 10€ para la formación (adquisición de conocimientos y elaboración de un documento informativo).Además, se parte de un precio base que garantice la correcta instalación y puesta en marcha del entorno.

Debido a que el proyecto se basa en crear funcionalidades no se necesita la versión de pago (Enterprises Editions) ya que con la Open Source a pesar de ofrecer funcionalidades más básicas, se amplian mediante módulos. [4][5]

A continuación se muestran los costos desglosados:

Licencia Magento (versión gratuita)	0€
Magento base	2000€
Desarrollador	20€/h
Formación	10€/h

Según las horas definidas en el diagrama de Gantt final que se han invertido en el desarrollo (200h) y en la formación (100h) para poder llevar a cabo el trabajo, se ha estimado un coste total de 7000€.

1.3. Competencias

Durante la elaboración de este trabajo y en el transcurso del grado de ingeniería informática se han adquirido las siguientes competencias.

1.3.1. Transversales

UAL1 - Conocimientos básicos de la profesión

UAL2 - Habilidad en el uso de las TIC

UAL3 - Capacidad para resolver problemas

UAL4 - Comunicación oral y escrita en la propia lengua

UAL5 - Capacidad de critica y autocritica

UAL6 - Trabajo en equipo

UAL7 - Aprendizaje de una lengua extranjera*

UAL8 - Compromiso ético

UAL9 - Capacidad para aprender a trabajar de forma autónoma

UAL10 - Competencia social y ciudadanía global

1.3.2. Específicas

CC01 - Capacidad para diseñar, desarrollar, seleccionar y evaluar aplicaciones y sistemas informáticos, asegurando su fiabilidad, seguridad y calidad, conforme a principios éticos y a la legislación y normativa vigente.

CC03 - Capacidad para comprender la importancia de la negociación, los hábitos de trabajo efectivos, el liderazgo y las habilidades de comunicación en todos los entornos de desarrollo de software.

CC04 - Capacidad para elaborar el pliego de condiciones técnicas de una instalación informática que cumpla los estándares y normativas vigentes.

CC05 - Conocimiento, administración y mantenimiento sistemas, servicios y aplicaciones informáticas.

CC06 - Conocimiento y aplicación de los procedimientos algorítmicos básicos de las tecnologías informáticas para diseñar soluciones a problemas, analizando la idoneidad y complejidad de los algoritmos propuestos.

CC07 - Conocimiento, diseño y utilización de forma eficiente los tipos y estructuras de datos más adecuados a la resolución de un problema.

CC08 - Capacidad para analizar, diseñar, construir y mantener aplicaciones de forma robusta, segura y eficiente, eligiendo el paradigma y los lenguajes de programación más adecuados.

CC09 - Capacidad de conocer, comprender y evaluar la estructura y arquitectura de los computadores, así como los componentes básicos que los conforman.

CC12 - Conocimiento y aplicación de las características, funcionalidades y estructura de las bases de datos, que permitan su adecuado uso, y el diseño y el análisis e implementación de aplicaciones basadas en ellos.

CC13 - Conocimiento y aplicación de las herramientas necesarias para el almacenamiento, procesamiento y acceso a los Sistemas de información, incluidos los basados en web.

CT1 - Capacidad para concebir, redactar, organizar, planificar, desarrollar y firmar proyectos en el ámbito de la ingeniería en informática que tengan por objeto, de acuerdo con los conocimientos adquiridos según lo establecido en el apartado 5 de este anexo, la concepción, el desarrollo o la explotación de sistemas, servicios y aplicaciones informáticas.

CT3 - Capacidad para diseñar, desarrollar, evaluar y asegurar la accesibilidad, ergonomía, usabilidad y seguridad de los sistemas, servicios y aplicaciones informáticas, así como de la información que gestionan.

CT4 - Capacidad para definir, evaluar y seleccionar plataformas hardware y software para el desarrollo y la ejecución de sistemas, servicios y aplicaciones informáticas, de acuerdo con los conocimientos adquiridos según lo establecido en el apartado 5 de este anexo.

CT5 - Capacidad para concebir, desarrollar y mantener sistemas, servicios y aplicaciones informáticas empleando los métodos de la ingeniería del software como instrumento para el aseguramiento de su calidad, de acuerdo con los conocimientos adquiridos según lo establecido en el apartado 5 de este anexo.

CT8 - Conocimiento de las materias básicas y tecnologías, que capaciten para el aprendizaje y desarrollo de nuevos métodos y tecnologías, así como las que les doten de una gran versatilidad para adaptarse a nuevas situaciones.

CT9 - Capacidad para resolver problemas con iniciativa, toma de decisiones, autonomía y creatividad. Capacidad para saber comunicar y transmitir los conocimientos, habilidades y destrezas de la profesión de Ingeniero Técnico en Informática.

CB04 - Conocimientos básicos sobre el uso y programación de los ordenadores, sistemas operativos, bases de datos y programas informáticos con aplicación en ingeniería.

CB05 - Conocimiento de la estructura, organización, funcionamiento e interconexión de los sistemas informáticos, los fundamentos de su programación, y su aplicación para la resolución de problemas propios de la ingeniería.

2. Estado del arte

2.1. El eCommerce

"Podríamos definir un ecommerce como una tienda virtual. Un método de compraventa que utiliza Internet como medio para realizar transacciones y contactar con sus consumidores. No solo mediante una página web, sino también a través de las redes sociales. Estas suponen una fuente informativa con mucho impacto, y permiten acercarte y conocer más a tu público objetivo." [6]

2.1.1. Tipos de eCommerce

Existen varios modelos de negocio dependiendo del producto o servicio que se ofrezca y del negocio en sí. Los más comunes son los siguientes:

Según el perfil comercial:

"**B2B: Business to Business (de negocio a negocio).** La sigla B2B, del inglés Business to Business, se aplica a empresas que crean productos o servicios para vendérselos a otras empresas. Suele presentar transacciones de mayor valor, lo que exige un sistema más robusto para procesar los pagos." [7]

"**B2C: Business to Consumer (de negocio a consumidor).** El B2C es el modelo adoptado por empresas que efectúan ventas destinadas al consumidor final, lo cual representa la mayoría de empresas de comercio electrónico." [7]

"**C2B: Consumer to Business (de consumidor a negocio).** El C2B es una inversión del modelo de negocio tradicional en el que el consumidor pone su servicio a disposición de las empresas." [7]

"**C2C: Consumer to Consumer (de consumidor a consumidor).** Y finalmente el C2C, que comprende las relaciones comerciales realizadas entre consumidores y se da habitualmente en foros y mercados." [7]

"Además de estas formas de comercio electrónico, existen otras menos populares como la **G2C (Goverment-to-Consumer)**, **C2G (Consumer-to-Goverment)** o **B2E (Business-to-Employer).**" [8]

Según la plataforma o el canal que se utilice se puede distinguir entre

"**S-Commerce (Social Commerce):** Es el comercio electrónico basado en una red social que posee su propio espacio para alojar tiendas virtuales,como Twitter (Tcommerce), Facebook (Fcommerce) o Instagram (Instagram Shopping), entre otras. En Social eCommerce la venta de productos o servicios puede no ser lo prioritario." [9]

"**M-Commerce (Mobile Commerce):**Incluye las transacciones comerciales que se realizan mediante dispositivos móviles como smartphones y tabletas. Muchas empresas optan por facilitar una aplicación a los usuarios con el fin de facilitar la navegación y la compra." [9]

2.1.2. Ventajas del eCommerce

El eCommerce cuenta con una serie de ventajas respecto al comercio tradicional

1. "**Más clientes:** ni una tienda local ni una empresa con sedes en varias ciudades puede conseguir el alcance del e-commerce. La posibilidad de conseguir comprar y vender desde cualquier punto del mundo amplía el público objetivo y permite conseguir más clientes.
2. **Sin horarios:** el e-commerce no tiene horarios, mientras que rara vez hay tiendas o empresas que trabajen 24 horas al día. La web está abierta al público todo el día y el cliente puede comprar lo que quiera cuando quiera.
3. **Menos costes:** el simple hecho de no necesitar un establecimiento físico reduce los costes con respecto al negocio tradicional. Y cuando el e-commerce funciona poniendo en contacto a proveedores con compradores, ni siquiera hay gastos en producción.
4. **Más margen de beneficio:** la reducción de costes y el aumento del mercado de clientes provocan que, incluso bajando los precios, se pueda conseguir un margen mayor que con un establecimiento tradicional. Se vende más y se gana más dinero.
5. **Escalabilidad**: esto significa que se puede vender a una o a mil personas al mismo tiempo. En un negocio físico siempre hay un límite de cantidad de clientes que puedes atender a la vez, en e-commerce el límite lo pone la capacidad de atraer visitantes." [8]

2.1.3. Principales plataformas e-commerce y comparativa

Las plataformas eCommerce son un sistema software (CMS) que permite gestionar los contenidos, principalmente los del producto a vender, y mediante plantillas aportar diseño al aspecto visual de la tienda online.

En ocasiones, según la plataforma que se elija no será necesario tener conocimientos de programación ya que gracias al uso de plantillas solamente se deberá incorporar las características específicas que se necesiten. Hay plataformas gratuitas y de pago, en estas últimas se suele cobrar una mensualidad o comisiones por ventas.

Existen varias plataformas diferentes según las necesidades que requiera cada negocio. Se van a mostrar los 4 principales eCommerce según su cuota de mercado, comenzando por los más usados.

Shopify

"Shopify es una de las plataformas de ecommerce más utilizadas a nivel mundial, sobre todo por su facilidad a la hora de crear una tienda virtual. Sus grandes opciones de organización y personalización tanto de productos como de la tienda en general, así como los diferentes procesos de pago que ofrece o el seguimiento de los pedidos son algunos de sus puntos fuertes."[10]

Su funcionamiento es de tipo SAAS ya que permite a los usuarios conectarse a la nube a través de Internet sin la necesidad de descargarse ningún programa.

WooCommerce

"Básicamente, Woocomerce es el plugin para WordPress que puede convertir un sitio web en un eCommerce. La instalación se realiza con un par de clics, sin necesidad de saber de programación. Una vez instalado, ya se pueden añadir los productos, crear categorías o configurar los gastos de envío. Todo está preparado para poder usar la tienda de manera casi inmediata." [10]

Magento

"Magento, además de ser un reconocido CMS para generar páginas web, pone a disposición de sus usuarios miles de extensiones que incluyen funcionalidades para pagos, envíos, impuestos, analíticas y logística, fundamentales para competir en el comercio electrónico. Básicamente se trata de una solución integral open source (código liberado) para el desarrollo de páginas web orientadas a la venta online, pero Magento no es fácil de usar para personas que carecen de conocimientos de programación. [10]

Prestashop

"PrestaShop es una solución de comercio electrónico y código abierto (en constante evolución) que alimenta a más de 300.000 tiendas en línea, a día de hoy, en el mundo. Además, cuenta con partnership de marcas tan reconocidas en nuestros días como PayPal, Google, MailChimp o Facebook, todos ellos aportando su granito de arena a las páginas webs creadas a través de este CMS que, sin duda, ofrece un servicio integral para sus clientes." [11]

A continuación, se muestran las principales tecnologías eCommerce basadas en la cuota de mercado el año 2021:

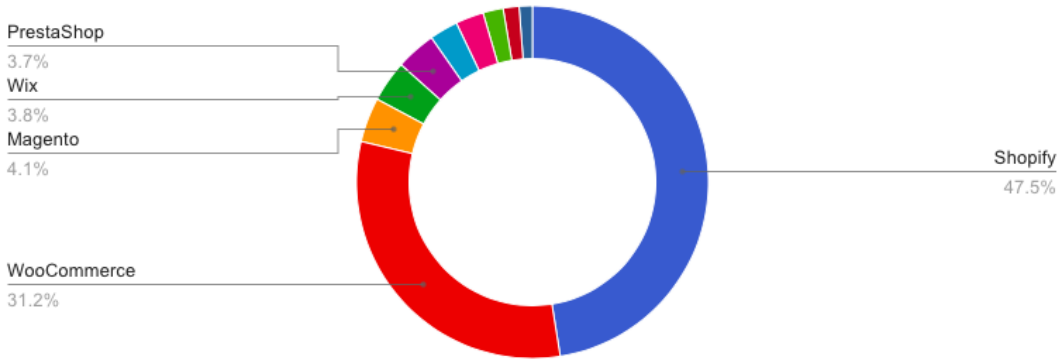


Figura 2.1.3: Principales tecnologías eCommerce [12]

A partir de la breve descripción de cada plataforma y conocer cuáles son las más usadas, se ha realizado una tabla en la que se ve de forma más resumida lo que ofrece cada plataforma.

Especificaciones	**Shopify**	**WooCommerce**	**Magento**	**Prestashop**
Tamaño proyecto	Pequeños	Pequeños y medianos	Grandes y medianos	Pequeños y medianos
Facilidad uso	Muy fácil	Fácil	Difícil	Media
Necesita hosting	No	Sí	Sí	Sí
Coste mensual	Sí	No	No	No
Multilenguaje	No	Sí	Sí	Sí
Seguridad	Alta	Alta	Muy alta	Alta
Open source	No	Sí	Sí	Sí
Personalización	Media	Alta	Muy alta	Alta

Tabla 2.1.3: Características de los principales eCommerce

2.2. Elección del CMS

Aunque en la imagen 2.1.3 la plataforma más usada ha sido Shopify y en tercer lugar Magento, la elección de la plataforma debe hacerse según las necesidades del proyecto y, en este caso se basa en la realización de módulos en una plataforma eCommerce.

Magento es un CMS que proporciona una avanzada personalización y abre la posibilidad de ampliar las funcionalidades que hagan única a la tienda online. Las características técnicas de esta plataforma la convierten en la mejor opción para la realización de este proyecto, gracias a ser un sistema escalable, lo que se ajusta a la necesidad de crear módulos personalizados. [13]

Las empresas que internacionales que apuestan por utilizar la plataforma Magento son: HP, Canon, Asus Technology, Tous Jewerly…

Figura 2.2: Magento

2.3. Elasticsearch

“Elasticsearch es un motor de analítica y análisis distribuido, gratuito y abierto para todos los tipos de datos, incluidos textuales, numéricos, geoespaciales, estructurados y no estructurados. Está desarrollado a partir de Apache Lucene y fue presentado por primera vez en 2010 por Elasticsearch N.V. [14] (ahora conocido como Elastic).

Las ventajas que ofrece son:

- “Elasticsearch es **rápido**. Como Elasticsearch está desarrollado sobre Lucene, es excelente en la búsqueda de texto completo. Elasticsearch también es una plataforma de búsqueda en casi tiempo real, lo que implica que la latencia entre el momento en que se indexa un documento hasta el momento en que se puede buscar en él es muy breve: típicamente, un segundo. Como resultado, Elasticsearch está bien preparado para casos de uso con restricciones de tiempo como analítica de seguridad y monitoreo de infraestructura.” [14]

- “Elasticsearch es **distribuido** por naturaleza. Los documentos almacenados en Elasticsearch se distribuyen en distintos contenedores conocidos como shards, que están duplicados para brindar copias redundantes de los datos en caso de que falle el hardware. La naturaleza distribuida de Elasticsearch le permite escalar horizontalmente a cientos (o incluso miles) de servidores y gestionar petabytes de datos.” [14]

- “Elasticsearch viene con un **amplio conjunto de características**. Además de su velocidad, la escalabilidad y la resistencia, Elasticsearch tiene una cantidad de características integradas poderosas que contribuyen a que el almacenamiento y la búsqueda de datos sean incluso más eficientes, como data rollup y gestión de ciclo de vida del índice. [14]

- “El **Elastic Stack simplifica la ingesta de datos, la visualización y el reporte**. La integración con Beats y Logstash facilita el proceso de datos antes de indexarlos en Elasticsearch. Y Kibana provee visualización en tiempo real de los datos de Elasticsearch así como UI para acceder rápidamente al monitoreo de rendimiento de aplicaciones (APM), los logs y los datos de métricas de infraestructura.” [14]

Anteriormente Magento utilizaba el motor de búsqueda de catálogo predeterminado de MySQL hasta que se lanzó la versión 2.4 que mejoraba la seguridad, las pasarelas de pago, el rendimiento y el stock (o gestión de inventario) entre otros, y fue reemplazado por el motor de búsqueda predeterminado de Elasticsearch.

2.4. Análisis buscadores

Para comenzar este capitulo, se recuerda que el objetivo de este trabajo se basa en la función de los buscadores y el objetivo de este en un eCommerce es conseguir llevar al cliente al producto que desea buscar en el menor número de clics.

Para la realización de los módulos primero se han analizado los principales buscadores para tiendas online: Doofinder y Empathy. Se ha elaborado un listado de características de ambos buscadores

que se han decidido implementar y otras que se han omitido debido a que Magento ya realizaba esas características o superaban las habilidades técnicas para su desarrollo.

En primer lugar se van a mencionar las características comunes de los dos buscadores y, a continuación se mencionará en los apartados correspondientes sus características propias.

Tanto Doofinder como Empathy coinciden en la característica de tener un dashboard que analice la geolocalización del usuario, los dispositivos y navegadores desde los que acceden a la página web, las consultas que se han realizado en el buscador y las consultas que no ofrecen resultados.

Otra coincidencia entre los dos buscadores es el autocompletado mediante un desplegable para que vaya ofreciendo sugerencias relacionadas con la búsqueda.

Por último, Empathy implementa la característica de que al buscar una consulta en particular, se redirige a una pagina diferente relacionada con esta consulta.

Además de haber mencionado las características que reúnen ambos buscadores anteriormente, se va a dar una breve descripción de cada uno para conocer que ofrecen.

2.4.1. Doofinder

Doofinder es un motor de búsqueda inteligente multiplataforma y multidioma para sitios de comercio electrónico. Promete aumentar las ventas, atraer usuarios y reducir los abandonos de la página, todo ello sin la necesidad de programar. Desde su sitio web se muestran las características que aporta Doofinder de forma resumida en la siguiente imagen.

“Desde 2014 Doofinder ha ayudado a los eCommerce a incrementar sus ventas, permitiendo que los consumidores utilicen su herramienta de búsquedas inteligente para ofrecer resultados más relevantes de forma personalizada. Doofinder nació en España como una vía para hacer asequibles a empresas de cualquier tamaño las tecnologías que requieren para crecer. Sin embargo, en la actualidad Doofinder se ha expandido creando una gran empresa global que cuenta con más de 5.000 clientes y tiene presencia en más de 50 países.” [15]

Grandes clientes que lo usan son Maserati, New Balance, Alain Afflelou, Store Volkswagen…

Figura 2.4.1: Características de Doofinder [16]

2.4.2. Empathy

"Empathy.co es una compañía asturiana que fue fundada en 2012 para implementar la tecnología de búsqueda en comercio electrónico. Empathy.co es el buscador online que se encuentra detrás de las webs y aplicaciones móviles de grandes marcas, como el grupo Inditex, Vodafone, Casa Del Libro o Interflora." [17]

Las especificaciones de Empathy se pueden visualizar en su página web desde el apartado de "Technology", al no estar tan resumido visualmente como en el caso de Doofinder, se va a mostrar a continuación una imagen con el nombre de sus principales especificaciones

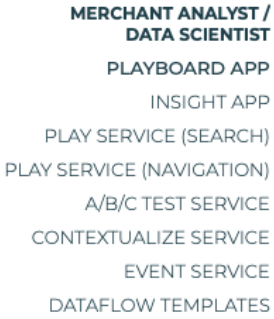


Figura 2.4.2: Características de Empathy [18]

3. Herramientas de desarrollo y metodología

3.1. Html, Less y Javascript

Los lenguajes que de programación web que se han usado son:

"**HTML** (lenguaje de marcas de hipertexto), es el lenguaje donde se define la información o el contenido del documento, el formato de los archivos es .html" [19]

Less (Leaner Style Sheets) es una extensión de lenguaje de CSS. Es sencillo ya que sólo se añaden algunas funciones, el formato de los archivos es .less. Permite definir variables, importaciones, anidamientos, operaciones… [20]

"**CSS** (cascading style sheets), es el lenguaje donde se especifica el diseño del documento, maneja todo lo relacionado con la parte visual, el formato de los archivos es .css" [19]

"**JavaScript**, el lenguaje que hace que todo sea interactivo, es realmente el lenguaje de programación que nos permite crear sitios web, el formato de los archivos es .js" [19]

El navegador utilizado para interpretar estos lenguajes ha sido Google Chrome.

Figura 3.1: Lenguajes de desarrollo web

3.2. Php y XML

Figura 3.2: Php

"PHP es un lenguaje de programación destinado a desarrollar aplicaciones para la web y crear páginas web, favoreciendo la conexión entre los servidores y la interfaz de usuario.

Una de las características principales de PHP es que es un lenguaje mucho más dinámico que la mayoría de las otras opciones que existen. Por lo tanto, es esencial para desarrollar sitios que tienen aplicaciones más complejas y, para eso, necesitamos dos cosas: agilidad en el tiempo de respuesta y conexión a una gran base de datos.

En la práctica, la idea de usar este lenguaje es disminuir el tiempo de carga de las páginas, permitiendo que el servidor trabaje con más suavidad para cargar plugins y aplicaciones en los sitios web.

De esta manera, es posible desarrollar con agilidad sitios con un gran rendimiento, incluso si están llenos de recursos, y con la garantía de la sostenibilidad del desempeño a largo plazo utilizando el lenguaje PHP.

Los ecommerces tienen una gran necesidad, que es la comunicación frecuente con bases de datos complejas y llenas de elementos importantes. Después de todo, hay muchas imágenes, videos y otros medios relacionados con los productos que se venden. Por lo tanto, cada vez que necesitas cargar páginas, debes conectarte con estas bases de datos, lo que podría dejarlas pesadas. Ante esto, el PHP se convierte en una gran alternativa para escapar de la posibilidad de tener una tienda que no le proporcione una experiencia de navegación satisfactoria al usuario.” [21]

Por otra parte, “XML es el acrónimo de Extensible Markup Language, es decir, es un lenguaje de marcado que define un conjunto de reglas para la codificación de documentos.

Un archivo XML se divide en dos partes: prolog y body. La parte prolog consiste en metadatos administrativos, como declaración XML, instrucción de procesamiento opcional, declaración de tipo de documento y comentarios. La parte del body se compone de dos partes: estructural y de contenido (presente en los textos simples).

El diseño XML se centra en la simplicidad, la generalidad y la facilidad de uso y, por lo tanto, se utiliza para varios servicios web.” [22]

3.3. Bases de datos

“MySQL Workbench es una herramienta visual unificada para arquitectos de bases de datos, desarrolladores y DBA (administradores de bases de datos). MySQL Workbench proporciona modelado de datos, desarrollo SQL y herramientas de administración integrales para la configuración del servidor, administración de usuarios, respaldo y mucho más. MySQL Workbench está disponible en Windows, Linux y Mac OS.” [23]

Figura 3.3: MySQL Workbench

3.4. IDE y SSOO

Para el desarrollo de los módulos se ha usado el sistema operativo Ubuntu 20.4 Desktop mientras que para realizar la documentación del proyecto se ha utilizado el sistema operativo Mac OS 11.4 junto con el programa de procesamiento de textos Microsoft Word.

El entorno de desarrollo integrado (IDE) que se ha empleado ha sido PhpStorm creado por la compañía JetBrains en el SO Ubuntu. No se trata de una versión gratuita y se ha adquirido la licencia de estudiante.

Es compatible para trabajar con grandes proyectos como Magento. PhpStorm proporciona un editor para PHP, HTML y JavaScript con análisis de código sobre la marcha, prevención de errores y refactorizaciones automatizadas para código PHP y JavaScript. Además de que incluye un editor SQL completo con resultados de consulta editables y herramientas de línea de comando. [24]

Figura 3.4: PhpStorm

3.5. Metodología

La metodología Agile es la que se ha aplicado en el desarrollo de este trabajo de fin de grado con la finalidad de analizar las necesidades del proyecto y dividirlo en pequeñas fases para realizar revisiones continuas y pruebas para controlar el avance del proyecto. Una vez que se terminan las fases, se muestra el proyecto completo. [25]

"La metodología Agile aplicada al desarrollo de proyectos permite:

- Tener una visión global del proyecto.
- Asignación de mini proyectos e hitos para el equipo.
- Incrementar el proyecto en fases.” [26]

“Las metodologías ágiles se empezaron a aplicar en la década de los 90 en el sector del software y las nuevas tecnologías debido a la insatisfacción de los trabajadores con los resultados de los modelos tradicionales de gestión. En el año 2001, un grupo de CEOs de las principales empresas del software en Utah, EE.UU., se reunieron para compartir las mejores prácticas con las que trabajaban y, de esta forma, descubrir la manera más eficiente de entregar un software. Como resultado, desarrollaron el “Manifiesto Ágil”, un documento que reúne los principios y valores de un conjunto de herramientas para mejorar la gestión de proyectos a través de la agilidad y el máximo rendimiento.” [27]

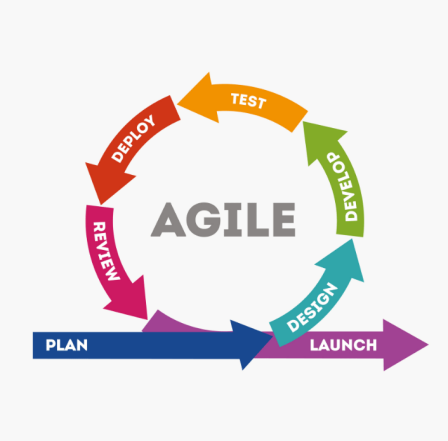


Figura 3.5: Metodología Agile

4. Desarrollo

Los pasos que se van a seguir para el desarrollo de este trabajo son, en primer lugar, comprender la estructura (modelo vista controlador y directorios) que ofrece el framework de Magento y después el desarrollo de cada una de las implementaciones (de los tres primeros objetivos específicos). En el Anexo II se continuará con las implementaciones de los desarrollos que aparecen en el panel de administración y las relacionadas con la vista al usuario. Como se puede observar en la siguiente imagen, los pasos son los siguientes:

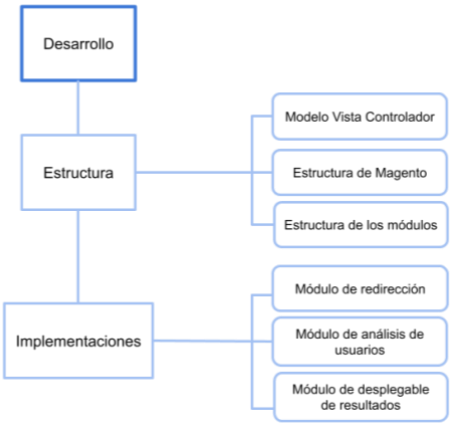


Figura 4-a: Diagrama de pasos para el desarrollo - Desarrollo

4.1. Estructura

En este apartado se va a exponer la arquitectura de software que utiliza Magento, el modelo vista controlador y la estructura de archivos de Magento 2.

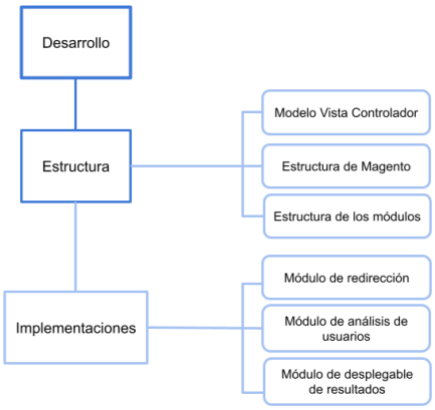


Figura 4-b: Diagrama de pasos para el desarrollo - Estructura

4.1.1. Modelo vista controlador

En la presente fase se expone la definición del modelo vista controlador.

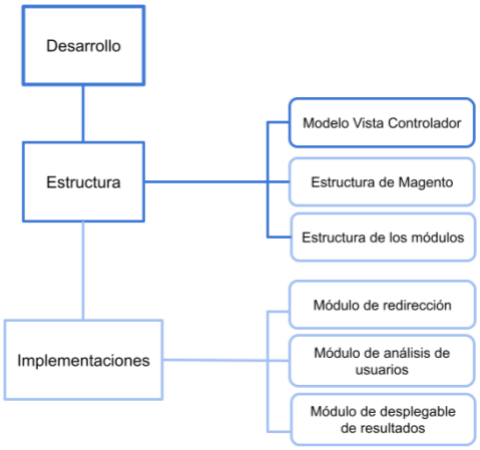


Figura 4-c: Diagrama de pasos para el desarrollo - MVC

Magento es una aplicación escrita en PHP que está basada en el modelo-vista-controlador (MVC) en la que la lógica, los datos y la interfaz están en carpetas separadas. El "Modelo Vista Controlador es un estilo de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. Se trata de un modelo muy maduro y que ha demostrado su validez a lo largo de los años en todo tipo de aplicaciones, y sobre multitud de lenguajes y plataformas de desarrollo." [28]

El código en esta plataforma se divide en módulos individuales. Para comprender en qué se basa este modelo, se aporta una breve descripción:

"El **Modelo** que contiene una representación de los datos que maneja el sistema, su lógica de negocio, y sus mecanismos de persistencia. Es responsable de: Acceder a la capa de almacenamiento de datos, definir las reglas de negocio, llevar un registro de las vistas y controladores del sistema.

La **Vista**, o interfaz de usuario, que compone la información que se envía al cliente y los mecanismos interacción con éste. Son responsables de: Recibir datos del modelo y los muestra al usuario y tienen un registro de su controlador asociado.

El **Controlador**, que actúa como intermediario entre el Modelo y la Vista, gestionando el flujo de información entre ellos y las transformaciones para adaptar los datos a las necesidades de cada uno. Es responsable de: Recibir los eventos de entrada y contiene las reglas de gestión de eventos." [28]

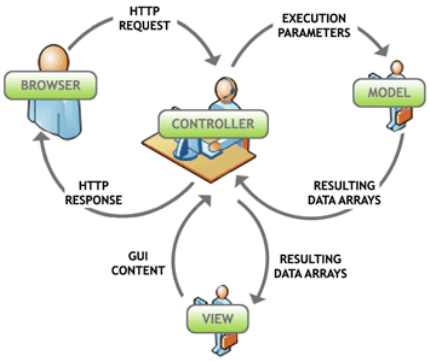


Figura 4.1.1: Funcionamiento MVC

4.1.2. Estructura de Magento

Ahora se va a mostrar cómo está organizada la estructura de Magento.

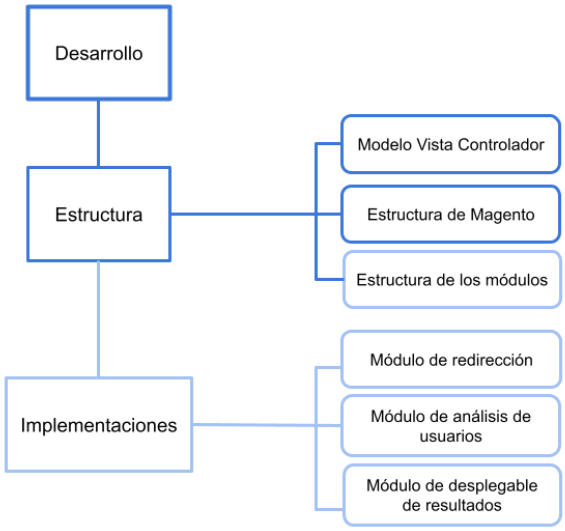


Figura 4-d: Diagrama de pasos para el desarrollo - Estructura de Magento

La estructura de Magento se compone de directorios (Figura 4.1.2.) y ficheros. A continuación, se van a exponer las funcionalidades de los principales directorios:

- **app**: Tiene 3 subdirectorios.
    - El subdirectorio **code** es dónde se crearán los módulos personalizados, las funcionalidades que se creen tienen que ir en la ruta app/code/Namespace/ModuleName.
    - El subdirectorio **design** tiene a su vez dos subdirectorios frontend y adminhtml, en los que se crearán los temas personalizados o alguna modificación de los temas que ya incluye Magento.
    - Por último, en el subdirectorio **etc** se encuentran los archivos de configuración.
- **bin**: En este directorio se almacenan los comandos de Magento.

- **dev**: Almacena los test que fueron ejecutados por Magento Test Framework.
- **generated**: Es donde se almacena el código generado por Magento. En la configuración predeterminada, si la clase se inyecta en un constructor, Magento generará el código para crear clases de fábrica inexistentes.
- **lib**: Contiene todos los archivos de librerías del vendor y el código de Magento no basado en módulos
- **phpserver**: Tiene el archivo **router.php** que puede ser usado para implementar un servidor PHP.
- **pub**: En este directorio se almacena un archivo **index.php** que sirve para ejecutar la aplicación en el modo de producción. También contiene los archivos estáticos generados.
- **setup**: Contiene todos los archivos relacionados con la configuración de instalación de Magento.
- **var**: Incluye clases generadas, sesiones de caché, copias de seguridad de base de datos e informes de errores almacenados en caché.
- **vendor**: Es el directorio que contiene el núcleo de la plataforma. También es dónde se instalan módulos adicionales.

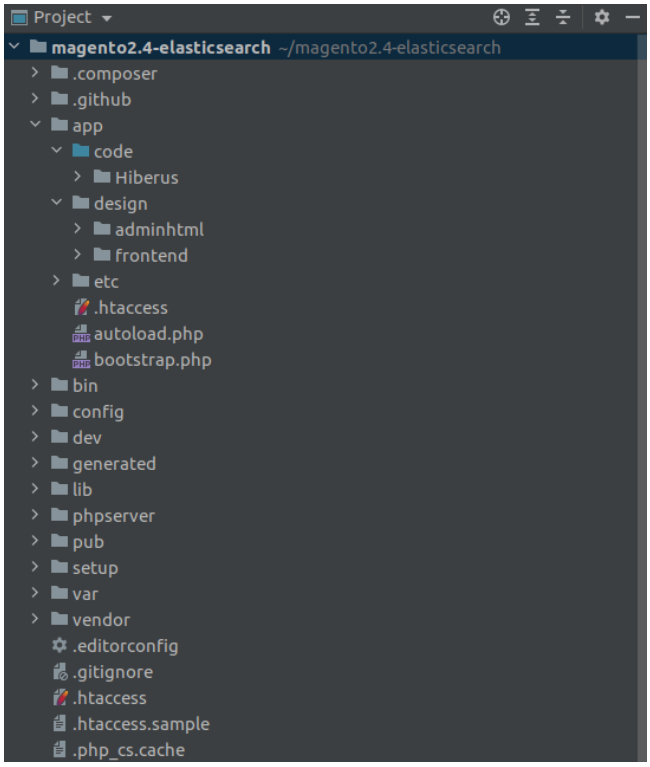


Figura 4.1.2: Estructura de Magento

La estructura de archivos de Magento 2 del módulo contiene bloques, controladores, helpers, models... En la carpeta etc se incluyen los archivos de configuración y de la administración.

Todos los archivos se combinan en función de sus funcionalidades (módulos) y se utilizan para mejorar el funcionamiento de la aplicación.

4.1.3. Estructura de los módulos

En esta parte se va a hacer referencia a los directorios que constituyen la estructura de los módulos del proyecto y a los dos ficheros que se necesitan en todos los módulos que se creen para que sean reconocidos por Magento.

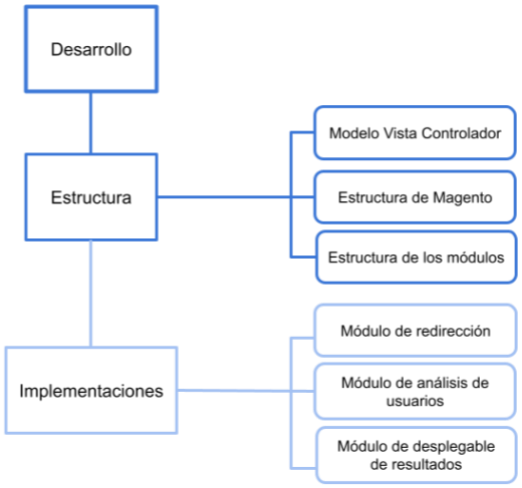


Figura 4-e: Diagrama de pasos para el desarrollo - Estructura de los módulos

Todos los módulos pertenecen al proyecto /Hiberus que a su vez se tiene que estar localizado dentro del directorio /app/code para que Magento reconozca la ruta de módulos personalizados. Básicamente, cada módulo tiene que seguir la ruta app/code/{NameSpace}/{Module_name}. Se explicará el contenido de las demás carpetas en sus capítulos correspondientes.

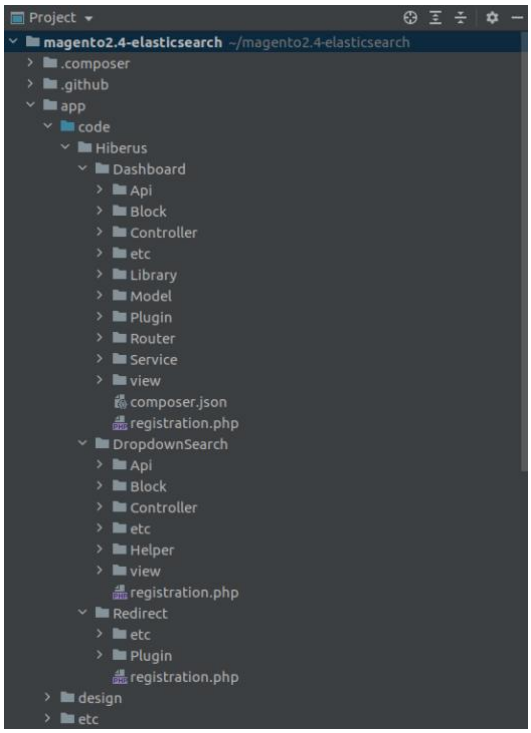


Figura 4.1.4: Estructura de los módulos

En cada desarrollo que se va a llevar a cabo en este proyecto será necesario repetir la creación de estos dos ficheros para que Magento reconozca que el módulo existe:

- **registration.php**: se crea este archivo en la raíz del módulo app/code/NameSpace/ModuleName. En él se registra que el componente a crear es de tipo MODULE y tendrá el nombre que se pasa en la definición. Se ve de la siguiente manera:

```
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
\Magento\Framework\Component\ComponentRegistrar::MODULE,
'Namespace_ModuleName',
__DIR__
);
```

- **module.xml**: se crea en la ruta app/code/NameSpace/ModuleName/etc/. Este archivo se encarga definir la información básica sobre el componente. Tiene el siguiente aspecto:

```
<?xml version="1.0"?>
<config    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/
module.xsd">
<module name="Namespace_ModuleName" />
</config>
```

  Donde:

  `<config>`: Nodo raíz de la configuración de módulos de Magento
  `<module>`: Información básica sobre el módulo

El Namespace es el nombre de la empresa, Hiberus. Dentro van los directorios de los módulos, separados por funcionalidades. Todos los módulos personalizados contienen los mismos archivos en común registration.xml y module.xml.

La versión elegida para desarrollar el proyecto es la gratuita Magento 2.4. Community Edition y se ha realizado mediante una instalación local para probar las funcionalidades. En este apartado se van a explicar los desarrollos de las funcionalidades principales de los módulos y más adelante, en los anexos se explicará cómo se realizó la instalación de Magento y los desarrollos relacionados con la vista de administración.

Los comandos que más se van a usar son los relacionados con borrar la caché y actualizar la base de datos de Magento:

```
dockergento magento cache:flush

dockergento magento cache:clean

dockergento magento setup:upgrade
```

4.2. Módulo de redirección

Se comienza con el desarrollo de los módulos con el módulo de redirección. Está organizado en dos secciones, en la primera se explica el funcionamiento actual y lo que se quiere conseguir y, en la segunda sección, se expone los pasos que se realizaron para el funcionamiento del módulo.

Se recuerda que el último objetivo específico consistía en que los módulos tuvieran la opción de poder ser habilitados o deshabilitados, esta funcionalidad se desarrolla en el Anexo II.

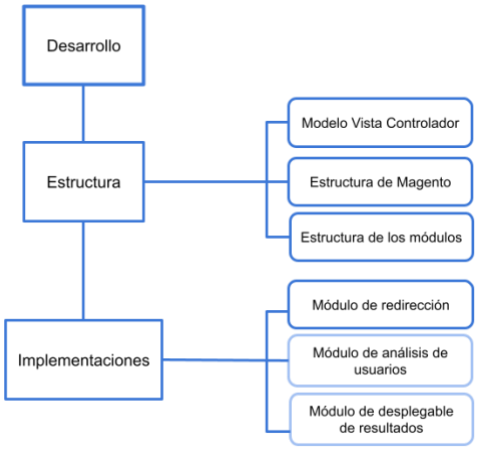


Figura 4-f: Diagrama de pasos para el desarrollo - Módulo de redirección

4.2.1. Contexto

En Magento cuando se realiza la búsqueda de productos, sea cual sea el número de resultados, dirige a los usuarios a un listado de resultados. Este módulo consiste en que, cuando se busque un producto, si sólo existe un resultado, lleve al usuario a la página del producto directamente, en lugar de al listado de productos.

El primer módulo está compuesto por sus archivos de reconocimiento (registration.php y module.xml en sus respectivas ubicaciones), dentro de la carpeta /etc los archivos di.xml, config.xml y el archivo system.xml que a su vez está dentro del directorio adminhtml.

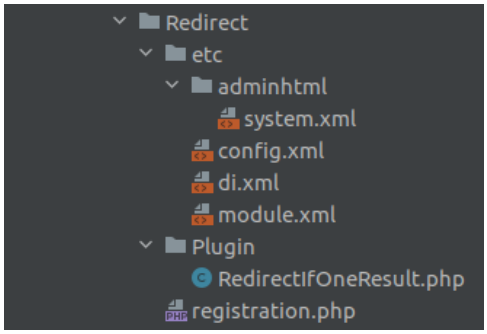


Figura 4.2.1-a: Estructura del módulo basado en la redirección

Para que Magento reconozca el módulo es necesario crear las carpetas registration.php y module.xml.

```php
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
    type: \Magento\Framework\Component\ComponentRegistrar::MODULE,
    componentName: 'Hiberus_Redirect',
    path: __DIR__
);
```

Figura 4.2.1-b: Contenido del archivo registration.php del módulo de redirección

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
    <module name="Hiberus_Redirect"/>
</config>
```

Figura 4.2.1-c: Contenido del archivo module.xml del módulo de redirección

Para habilitar el módulo, una vez que el módulo ha sido definido, se lanza el comando:

`dockergento magento setup:upgrade`

Este comando hace que Magento borre la caché, compila el código y actualiza la base de datos.

4.2.2. Implementación del Plugin

La funcionalidad de este módulo se centra en un Plugin que es una clase que permite modificar el comportamiento de funciones públicas de clases interceptando la llamada a una función y ejecutando código antes, durante o después. Esto permite extender o sustituir el comportamiento original del método.

Primero es necesario realizar la inyección de dependencias en el archivo di.xml ubicado en la ruta `app/code/Hiberus/Redirect/etc/di.xml`. Consiste en declarar que el primer parámetro que se coloca en type es la clase que va a ser interceptada, lo que da acceso a todas sus funciones públicas. A continuación, se identifica con un nombre el plugin creado y su type es la clase del plugin que observa.

```xml
<?xml version="1.0"?>
<config  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <type name="Magento\CatalogSearch\Controller\Result\Index">
        <plugin name="hiberus_one_result" type="Hiberus\Redirect\Plugin\RedirectIfOneResult" />
    </type>
</config>
```

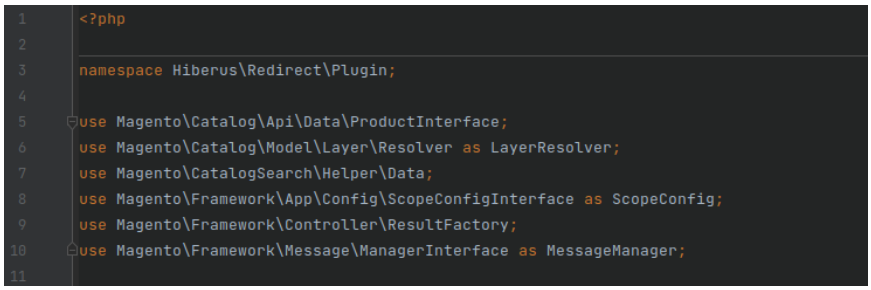
Figura 4.2.2-a: Contenido del archivo di.xml del módulo de redirección

En este caso se ha creado un plugin de tipo after que modifica el comportamiento final del método `Execute()` de la clase Index. Se requiere un valor de retorno `$result`, el método al que se

interviene debe tener el mismo nombre junto con el prefijo after y la variable `$subject` es el objeto original.

El archivo se ha creado dentro de la carpeta Plugin. Se ha usado el espacio de nombres (namespace) para definir la ubicación de este archivo y se han importado las clases de Magento que se van a usar:

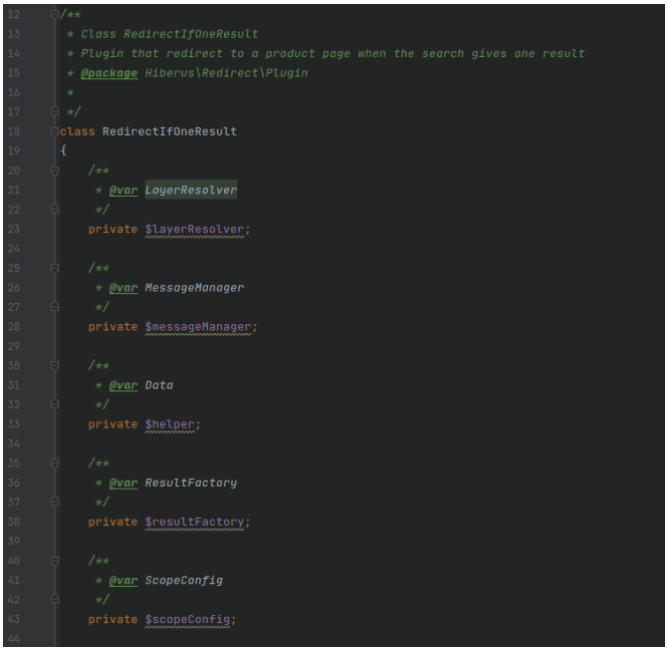
- ProductInterface: Contiene todos los atributos de un producto
- Resolver: Crea la capa actual del catálogo de productos
- Data: Ayuda a recuperar la url de búsqueda avanzada
- ScopeConfigInterface: Comprueba si el módulo está habilitado con la configuración por el árbol ruta del archivo system.xml
- ResultFactory: Crea una página nueva según su tipo (en este desarrollo de tipo redirect)
- ManagerInterface: Agrega diferentes tipos de mensajes (en este desarrollo de tipo success) a la sesión



```php
<?php

namespace Hiberus\Redirect\Plugin;

use Magento\Catalog\Api\Data\ProductInterface;
use Magento\Catalog\Model\Layer\Resolver as LayerResolver;
use Magento\CatalogSearch\Helper\Data;
use Magento\Framework\App\Config\ScopeConfigInterface as ScopeConfig;
use Magento\Framework\Controller\ResultFactory;
use Magento\Framework\Message\ManagerInterface as MessageManager;

```

Figura 4.2.2-b: Contenido del archivo RedirectIfOneResult.php (declaraciones)

En la siguiente imagen, se nombra a la clase que debe llamarse de la misma manera que se ha nombrado al fichero, en este caso, RedirectIfOneResult y se definen los atributos.



```php
/**
 * Class RedirectIfOneResult
 * Plugin that redirect to a product page when the search gives one result
 * @package Hiberus\Redirect\Plugin
 *
 */
class RedirectIfOneResult
{
    /**
     * @var LayerResolver
     */
    private $layerResolver;

    /**
     * @var MessageManager
     */
    private $messageManager;

    /**
     * @var Data
     */
    private $helper;

    /**
     * @var ResultFactory
     */
    private $resultFactory;

    /**
     * @var ScopeConfig
     */
    private $scopeConfig;

```

Figura 4.2.2-c: Contenido del archivo RedirectIfOneResult.php (instanciaciones)

Se establece el primer método que será el constructor, el cual inicializa los atributos que recibe por parámetro.

```php
/**
 * RedirectIfOneResult constructor.
 *
 * @param \Magento\Catalog\Model\Layer\Resolver               $layerResolver
 * @param \Magento\CatalogSearch\Helper\Data                  $catalogSearchHelper
 * @param \Magento\Framework\Message\ManagerInterface         $messageManager
 * @param \Magento\Framework\Controller\ResultFactory         $resultFactory
 * @param \Magento\Framework\App\Config\ScopeConfigInterface  $scopeConfig
 */
public function __construct(
    LayerResolver $layerResolver,
    Data $catalogSearchHelper,
    MessageManager $messageManager,
    ResultFactory $resultFactory,
    ScopeConfig $scopeConfig
) {
    $this->layerResolver  = $layerResolver;
    $this->messageManager = $messageManager;
    $this->helper         = $catalogSearchHelper;
    $this->resultFactory  = $resultFactory;
    $this->scopeConfig    = $scopeConfig;
}
```

Figura 4.2.2-d: Contenido del archivo RedirectIfOneResult.php (inicialización)

Le sigue la función que observa el método `Execute()` de la clase Index, añadiéndole al principio el tipo de plugin que se utiliza `afterExecute()`. Se le pasa por parámetro la clase con la variable `$subject` y la variable `$result` que devolverá el comportamiento que modifica el final original:

- En el primer if se comprueba que si no existe una redirección y si el módulo está habilitado, se obtiene la lista de productos en la variable `$productCollection`.
- En el segundo if, se verifica si existe un resultado del producto, y si es así se guarda en la variable `$product` el primer elemento.
- En el último if, según la id del producto se devuelve un mensaje y se realiza la redirección.

```php
/**
 * By a given search redirect to the product page if there is one result
 *
 * @param \Magento\CatalogSearch\Controller\Result\Index $subject
 * @param $result
 * @return \Magento\Framework\Controller\ResultInterface
 */
public function afterExecute(
    \Magento\CatalogSearch\Controller\Result\Index $subject,
    $result
) {
    if (!$subject->getResponse()->isRedirect() && $this->scopeConfig->isSetFlag( path: 'section_redirect/general_redirect/enable_redirect')) {
        $productCollection = $this->layerResolver->get()->getProductCollection();
        if ($productCollection->getCurPage() === 1 && $productCollection->getSize() === 1) {
            /** @var \Magento\Catalog\Api\Data\ProductInterface $product */
            $product = $productCollection->getFirstItem();
            if ($product->getId()) {
                $this->addRedirectMessage($product);
                $result = $this->resultFactory->create( type: ResultFactory::TYPE_REDIRECT);
                $result->setUrl($product->getProductUrl());
            }
        }
    }
    return $result;
}
```

Figura 4.2.2-e: Contenido del archivo RedirectIfOneResult.php (after)

En la última función del plugin, se crea el mensaje que va a lanzarse al producto cada vez que ocurra la redirección.

```
94      /**
95       * Session message to the product page that has been redirected
96       * @param ProductInterface $product
97       */
98      private function addRedirectMessage(ProductInterface $product)
99      {
100         $message = __("%1 es el unico resultado de tu busqueda '%2'", $product->getName(), $this->helper->getEscapedQueryText());
101         $this->messageManager->addSuccessMessage($message);
102     }
103 }
```

Figura 4.2.2-f: Contenido del archivo RedirectIfOneResult.php (mensaje de redirección)

Los demás archivos (config.xml y system.xml) corresponden al panel de administración y van a ser explicados en el Anexo II.

4.3. Módulo de análisis de usuarios

Esta parte del módulo de análisis de usuarios está formada en cuatro secciones. En la primera se contextualiza la finalidad del módulo y le siguen los tres directorios que conforman la lógica del módulo. El desarrollo de este módulo finaliza en el Anexo II con el resto de directorios y archivos que hacen posible la vista del dashboard en el panel de administrador.

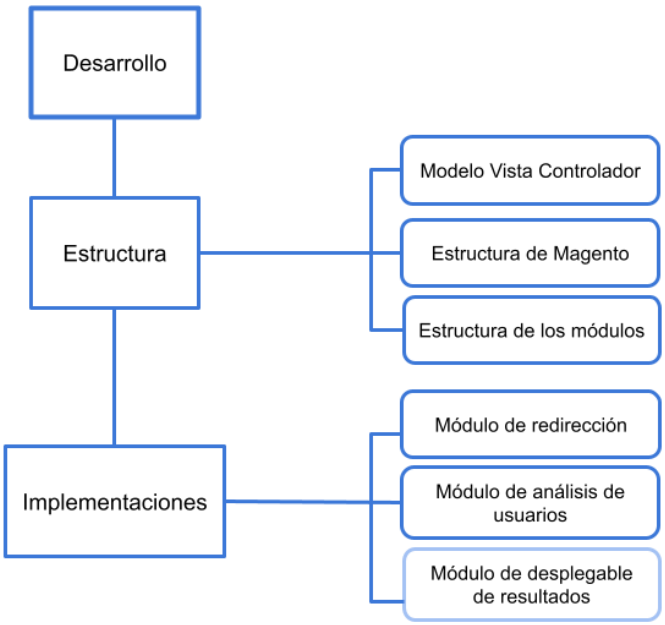


Figura 4-d: Diagrama de pasos para el desarrollo - Módulo de análisis de resultados

4.3.1. Contexto

La intención de este módulo es que la persona responsable de la gestión de la tienda online pueda observar el comportamiento de acceso de sus usuarios registrados. Los datos que se desean analizar son la ubicación, el dispositivo y el navegador desde el que el usuario accede al sitio web. El propósito es que, basándose en los resultados obtenidos, se permita a los responsables de la tienda online tomar decisiones para mejorar la experiencia del usuario, como por ejemplo, si gran parte de los usuarios acceden de Francia, lo ideal sería incorporar a la tienda online la opción de mostrar el contenido en ese idioma en el futuro; o si la mayoría accede desde un navegador determinado o dispositivos móviles, mejorar la experiencia de usuario.

La estructura que va a tener este módulo se muestra en la siguiente imagen:

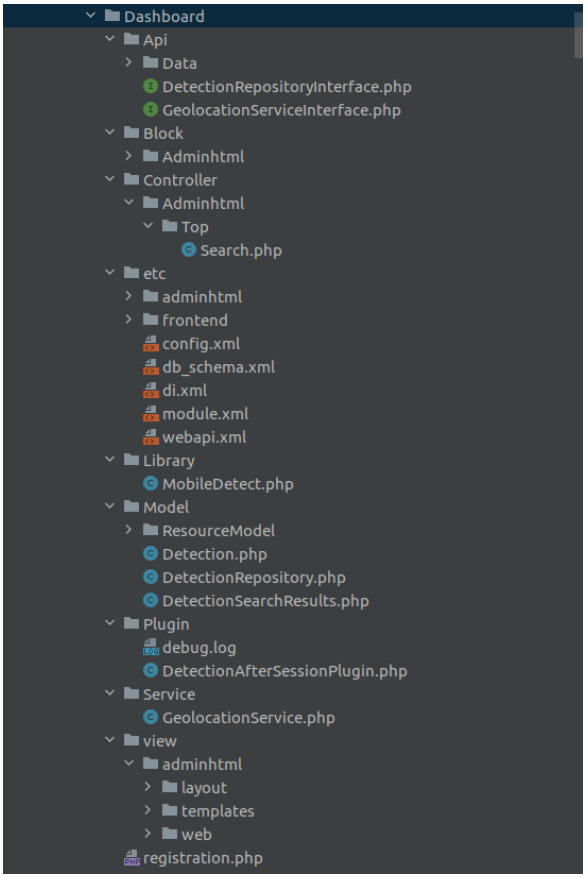


Figura 4.3.1-a: Estructura del módulo basado en el análisis de usuarios

El primer paso es crear los archivos de reconocimiento registration.php y etc/module.xml

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
    <module name="Hiberus_Dashboard"/>
</config>
```

Figura 4.3.1-b: Contenido del archivo module.xml del módulo de análisis de usuarios

```php
<?php

\Magento\Framework\Component\ComponentRegistrar::register(
    type: \Magento\Framework\Component\ComponentRegistrar::MODULE,
    componentName: 'Hiberus_Dashboard',
    path: __DIR__
);
```

Figura 4.3.1-c: Contenido del archivo registration.php del módulo de análisis de usuarios

Para la detección de los tipos de dispositivos y navegadores se ha utilizado la librería Mobile Detect [33]. Esta librería hace uso de las cabeceras HTTP que lee la información que recibe del servidor junto con el USER AGENT que identifica el sistema operativo, navegador, versión... Se ha ubicado en la carpeta Library.

Para la detección de la ubicación cuando un usuario visita la tienda online, se ha realizado un Plugin de tipo after que va a observar el comportamiento del método `start()` de la clase `SessionManager` de Magento. Este método almacena cuando se inicia sesión.

4.3.2. Implementación del Plugin

Puesto que se va a hacer uso nuevamente de un plugin, se requiere declarar en el archivo de configuración di.xml (que se ubica dentro del directorio `/etc/frontend` ya que la funcionalidad del plugin afecta únicamente a la vista de la tienda y no a la administración) para crear la inyección de dependencias entre el plugin que observa y la clase que da acceso a sus funciones públicas.

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <!-- Frontend plugin for setting store code based on IP -->
    <type name="Magento\Framework\Session\SessionManager">
        <plugin name="hiberus_dashboard_detectionaftersessionplugin" type="Hiberus\Dashboard\Plugin\DetectionAfterSessionPlugin"/>
    </type>
</config>
```

Figura 4.3.2-a: Archivo di.xml del área frontend del módulo de análisis de usuarios

La explicación del Plugin, debido a su extensión, se va a centrar en los métodos de la clase (excluyendo el constructor y la importación de las clases que aparecen al principio del archivo).

El método es el `afterStart()`, en donde se desea modificar el comportamiento final del método `Start()` de la clase SessionManager.

Primero, mediante la variable scopeConfig se comprueba si el módulo está habilitado, si es el caso, se realiza el plugin.

En la variable `$storedStoreCode` se almacena el código de país y se verifica si ya está definida en la sesión, si es el caso, el plugin finalizaría aquí.

Sino, se guarda en la variable `$storeCode` el código del país que se ha obtenido en la función `countryCode()`. Esta variable va a ser almacenada como la nueva `'store_code'` para que cuando se vuelva a iniciar sesión, hacer la comprobación anterior de la variable `$storedStoreCode`.

Finalmente, se almacenan en la base de datos, mediante la llamada a la interfaz DetectionInterface, la variable del código del país, la ip del usuario, el tipo de dispositivo y de navegador. El tipo de dispositivo y de navegador se encuentran en funciones en la misma clase mientras que la ip del usuario se ha obtenido en la clase GeolocationService, la cual se va a explicar al finalizar el plugin.

```php
/**
 * After SessionManager Start method get and save the values
 *
 * @param \Magento\Framework\Session\SessionManager $subject
 * @param \Magento\Framework\Session\SessionManager $result
 * @return SessionManager
 */
public function afterStart(
    SessionManager $subject,
    SessionManager $result
) {
    if ($this->scopeConfig->isSetFlag( path: self::PATH_DASH)) {
        $storedStoreCode = $this->storage->getData( key: 'store_code');
        if (isset($storedStoreCode)) {
            return $result;
        }
        try {
            $storeCode = $this->countryCode();
            $this->storage->setData('store_code', $storeCode);
            $this->detection->setCountryCode($storeCode);
            $ip = $this->geolocationService->getClientIp();
            $this->detection->setUserIp($ip);
            $device = $this->deviceType();
            $this->detection->setDevice($device);
            $browser = $this->browserType();
            $this->detection->setBrowser($browser);
            $this->detectionRepository->save($this->detection);
            return $result;
        }catch (\Magento\Framework\Exception\CouldNotSaveException $couldNotSaveException) {}
    }
    return $result;
}
```

Figura 4.3.2-b: Método afterStart() del Plugin

El siguiente método, recoge los código del país cercanos a España que se han obtenido en la clase `GeolocationService()` para mapearlos.

```php
/**
 * Mapping country codes
 *
 * @return string|null
 */
private function countryCode(): ?string
{
    $countryCode = $this->geolocationService->getCountryCodeByIp();

    // Main countries
    switch ($countryCode) {
        case 'US':
            return 'us';
        case 'DE':
            return 'de';
        case 'ES':
            return 'es';
        case 'FR':
            return 'fr';
        case 'GB':
            return 'gb';
        case 'PT':
            return 'pt';
        case 'IT':
            return 'it';
        case 'AD':
            return 'ad';
        default:
            return '';
    }
}
```

Figura 4.3.2-c: Método countryCode() del Plugin

Los dos últimos métodos hacen uso de la librería Mobile Detect y para poder usarla se ha importado al principio de la clase. El método `deviceType()` se asegura mediante una llamada a los métodos `isMobile()` y `isTablet()` de la librería que tipo de dispositivos son; si móvil, tablet u ordenador.

```php
/**
 * Determines the type of device using the mobile library
 *
 * @return string
 */
private function deviceType()
{
    if ($this->mobileDetect->isMobile() && !$this->mobileDetect->isTablet()) {
        return 'mobile';
    } elseif ($this->mobileDetect->isTablet()) {
        return 'tablet';
    } else {
        return 'computer';
    }
}
```

Figura 4.3.2.-d: Método deviceType() del Plugin

En el último método, se determina si el contenido del *User Agent* coincide con las expresiones regulares para determinar qué tipo de dispositivo se usa: Chrome, Firefox, safari, edge u Opera.

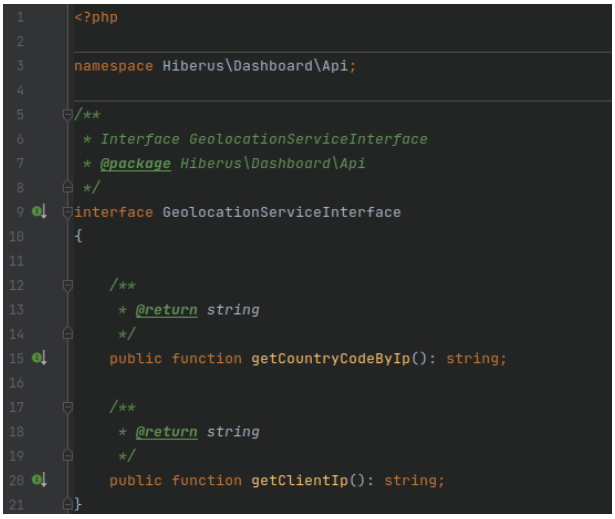
```php
/**
 * Determines the type of browser trhough regular expressions
 *
 * @return string
 */
public function browserType()
{
    $agent = htmlentities($_SERVER['HTTP_USER_AGENT']);

    if (preg_match( pattern: '/Chrome\/[.0-9]* Mobile/', $agent) || preg_match( pattern: '/Chrome\/[.0-9]* (?!Mobile)*Safari/', $agent) ||
        preg_match( pattern: '/CriOS\/[.0-9]*Mobile/', $agent)) {
        return 'chrome';
    }
    if (preg_match( pattern: '/(Firefox|(?:Focus).*(Safari)|(?:FxiOS).*(Safari))/', $agent)) {
        return 'firefox';
    }
    if (preg_match( pattern: '/(?:Mobile).*(Safari)|(Safari)/', $agent)) {
        echo 'safari';
    }
    if (preg_match( pattern: '/(Trident)|(IEMobile)/', $agent)) {
        echo 'edge';
    }
    if (preg_match( pattern: '/(?:Chrome).*(OPR)/', $agent)) {
        echo 'opera';
    }
    return 'other';
}
}
```

Figura 4.3.2-e: Método browserType() del Plugin

4.3.3. Implementación del Service

Se ha utilizado un Service para proveer al Plugin de más lógica y la clase no quede sobrecargada.

En la clase GeolocationService se importa la interfaz GeolocationServiceInterface que aporta la ventaja de que los métodos puedan ser utilizados en cualquier parte del código (se usan en esta clase y el plugin).



```php
<?php

namespace Hiberus\Dashboard\Api;

/**
 * Interface GeolocationServiceInterface
 * @package Hiberus\Dashboard\Api
 */
interface GeolocationServiceInterface
{

    /**
     * @return string
     */
    public function getCountryCodeByIp(): string;

    /**
     * @return string
     */
    public function getClientIp(): string;
}
```

Figura 4.3.3-a: Interfaz de GeolocationService

También se importa a clase Curl de Magento que trabaja con los protocolos HTTP mediante la librería curl y la clase logger que permite *debuggear* el código para revisar los resultados que se van consiguiendo.

En el primer método se realiza la llamada al método `getClientIp()` que es el que obtiene la ip del usuario y se la pasa al siguiente método.

```php
public function getCountryCodeByIp(): string
{
    $ipAddress = $this->getClientIp();
    $this->logger->debug('IP address is: ' . $ipAddress);
    if ($ipAddress !== 'UNKNOWN') {
        return $this->getCountryCodeFromIpStack($ipAddress);
    }
    return '';
}
```

Figura 4.3.3-b: Método getCountryCodeByIp() de la clase GeolocationService

El siguiente método hace uso de la API ipstack para detectar la ubicación de los usuarios que visitan la página web mediante su IP. Para usar ipstack, se ha tenido que crear una cuenta para conseguir la clave de acceso. [34]

En la variable `$requestUrl` se enlaza a la url de ipstack, la IP que se ha obtenido, la clave de acceso y la respuesta que se quiere filtrar `country_code`. El resultado es un JSON con el código del país según la IP.

Se crea la condición de que en 5 segundos tiene que conseguirse el código del país mediante la llamada al método de petición HTTP GET de la url, se decodifica el resultado json en un array asociativo y se guarda en la variable $response. Si en la variable $response existe el valor `country_code`, se devuelve el código del país en mayúsculas . Si se supera el tiempo, se devuelve una respuesta vacía.

```php
/**
 * Get country code from api ip stach
 * @param string $ipAddress
 * @return string
 */
private function getCountryCodeFromIpStack(string $ipAddress): string
{
    $requestUrl = 'http://api.ipstack.com/' . $ipAddress . '?access_key=5906bc343b3dc58785fc861aa1e79023&fields=country_code';
    $this->logger->debug('URL: ' . var_export($requestUrl, return: true));
    /** @var Curl $curl */
    $curl = $this->curlFactory->create();
    $curl->setTimeout( value: 5);
    try {
        $curl->get($requestUrl);
        $body = $curl->getBody();
        $response = json_decode($body, associative: true);
        $this->logger->debug('Store_code by IpStack in method: ' . var_export($response, return: true));
        if (isset($response['country_code'])) {
            return strtoupper($response['country_code']);
        }
    } catch (Exception $ex) {
        return '';
    }
}
```

Figura 4.3.3-c: Método getCountryCodeFromIpStack() de la clase GeolocationService

```
{"country_code":"ES"}
```

Figura 4.3.3-d: Resultado de la url de ipstack

```
array(1) { ["country_code"]=> string(2) "ES" }
```

Figura 4.3.3-e: Array asociativo que contiene el código del país

Debido a que los módulos de este trabajo se han desarrollado con intención de que en un futuro sirvan de ayuda para proyectos reales, al haber trabajado en un entorno local y utilizado una API y el servidor Docker, todo en conjunto, hace que no sea posible obtener la IP de acceso de usuario. Para solventar el problema se ha obtenido la IP pública del dispositivo local para que sea viable realizar el resto de funcionalidades.

```php
    /**
     * Client IP Address
     * @return string
     */
    public function getClientIp():string
    {
        /**
         * IN DEVELOPMENT
         * TODO: With Docker IP is not possible to get the public address in order to get a feed from the IP API
         * $ipAddress= $this->remote->getRemoteAddress();
         * return $ipAddress;
         * TODO: If the ip address doesn't work search the new ip address
         */
        return '188.78.110.29';
    }
}
```

Figura 4.3.3-f: Método getClientIp() de la clase GeolocationService

4.3.4. Implementación de la base de datos

Con los pasos anteriores ya se han obtenido los datos que se deseaban, ahora se necesitan almacenar en una base de datos para poder mostrar los resultados. La configuración de la conexión con la base de datos se almacena en el archivo /etc/env.php en el directorio raíz de Magento.

En Magento las entidades se crean en repositorios y usan el protocolo REST API para realizar las solicitudes. Esas entidades tienen que estar declaradas en una base de datos para que se intercambie la información mediante un ORM (Object Relational Mapping) que mapea los datos.

Los pasos que se deben llevar a cabo se dividen en crear la base de datos, crear las interfaces, repositorios y modelos. Esto va a permitir ocultar la parte lógica.

Paso 1. Creación de la base de datos

El archivo declarative schema (db_schema.xml) simplifica los procesos de instalación y actualización de Magento que antes se realizaban mediante scripts. La gestión de los CRUD en el framework permite que no se escriba líneas SQL directamente haciendo que se declare el estado final de la base de datos y el propio sistema se reajusta en base a los cambios al definir el `db_schema`.

Para ello, la tabla de base de datos se ha creado dentro del directorio `/etc`. Se aporta un nombre a la tabla y los valores de cada columna que se quieren almacenar, en este caso, la id del usuario, la ip del usuario, la ubicación del usuario, el navegador y el dispositivo (la columna de fecha sólo fue creada para comprobar los accesos). Se establece como clave primaria la id y para verificar que no se repitan varias veces la misma IP se declara como única la columna de ip del usuario. Una vez que se ha definido se ejecuta el siguiente comando para crear la nueva tabla.

```
dockergento magento setup:upgrade
```

```xml
<?xml version="1.0"?>
<schema xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:Setup/Declaration/Schema/etc/schema.xsd">
    <table name="hiberus_dashboard_table" resource="default" engine="innodb" comment="User Information Dashboard Backend Table">
        <column xsi:type="int" name="key_id" padding="10" unsigned="true" nullable="false" identity="true" comment="Primary key"/>
        <column xsi:type="varchar" name="user_ip" length="20" nullable="true" comment="User IP"/>
        <column xsi:type="varchar" name="country_code" length="5" nullable="true" comment="User Location Code"/>
        <column xsi:type="varchar" name="browser" length="10" nullable="true" comment="User Browser"/>
        <column xsi:type="varchar" name="device" length="10" nullable="true" comment="User Device"/>
        <column xsi:type="datetime" name="date_time" on_update="false" default="CURRENT_TIMESTAMP" nullable="true" comment="Actual Date Time"/>
        <constraint xsi:type="primary" referenceId="PRIMARY">
            <column name="key_id"/>
        </constraint>
        <constraint xsi:type="unique" referenceId="HIBERUS_DASHBOARD_TABLE_USER_IP">
            <column name="user_ip"/>
        </constraint>
    </table>
</schema>
```

Figura 4.3.4-a: Creación de tabla de base de datos

Paso 2. Crear las interfaces del repositorio

Una vez que se tiene la tabla creada, se tienen que crear las interfaces del modelo de datos y de los repositorios.

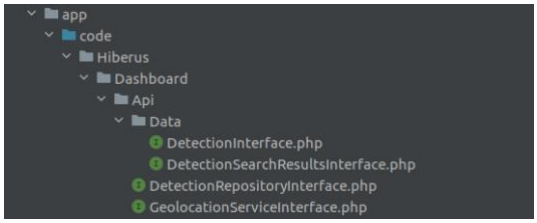


Figura 4.3.4-b: Estructura del directorio Api

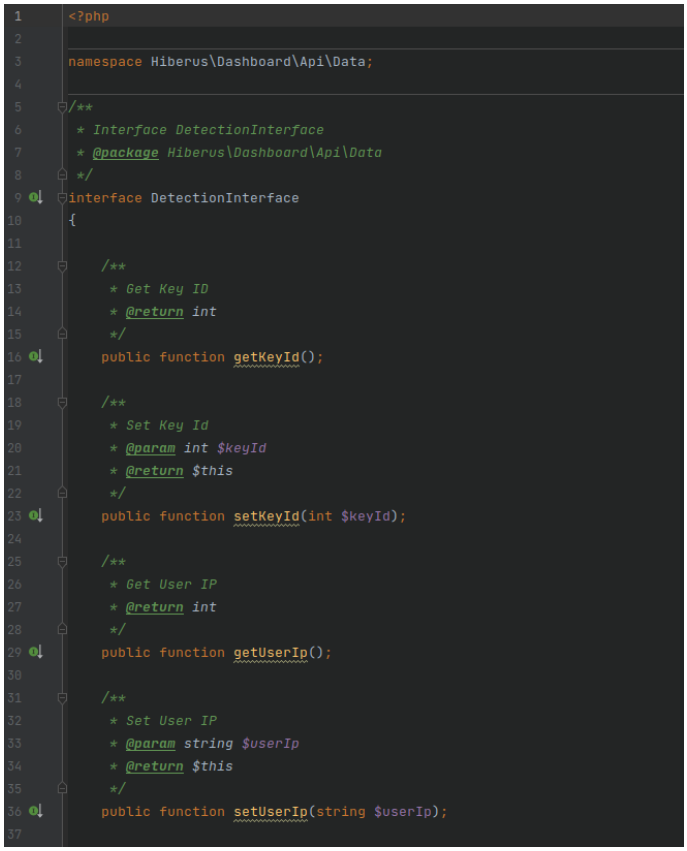
En /Api se crea la interfaz de los repositorios de la entidad que se ha denominado Detection y debe ir seguido por RepositoryInterface: DetectionRepositoryInterface.php. En esta clase se definen los métodos de las funciones CRUD mediante los objetos de acceso a datos (DAO). Se le pasa como parámetro a las funciones save y delete la clase DTO y a la función getList la instancia de SearchCriteria de Magento que devuelve un array de objetos (filtros, ordenaciones, tamaño de página o la página actual). En cada método se utilizan anotaciones para los argumentos y los tipos de valores que se devuelven para que el framework de Magento determine como convertir los datos a un JSON o XML.

```php
<?php

namespace Hiberus\Dashboard\Api;

use Hiberus\Dashboard\Api\Data\DetectionInterface;
use Magento\Framework\Api\SearchCriteria;

/**
 * Interface DetectionRepositoryInterface
 * @package Hiberus\Dashboard\Api
 */
interface DetectionRepositoryInterface
{
    /**
     * Save the result detections
     *
     * @param \Hiberus\Dashboard\Api\Data\DetectionInterface $detection
     * @return \Hiberus\Dashboard\Api\Data\DetectionInterface
     */
    public function save(DetectionInterface $detection) : DetectionInterface;

    /**
     * Delete the result detections
     *
     * @param \Hiberus\Dashboard\Api\Data\DetectionInterface $detection
     * @return bool
     */
    public function delete(DetectionInterface $detection) : bool;

    /**
     * Gets objects result detections list
     *
     * @param \Magento\Framework\Api\SearchCriteria $criteria
     * @return \Hiberus\Dashboard\Api\Data\DetectionSearchResultsInterface
     */
    public function getList(SearchCriteria $searchCriteria);
}
```

Figura 4.3.4-c: Clase DetectionRepositoryInterface

En el subdirectorio `/Api/Data` se crean las clases DTO que contienen los métodos getters y setters de todas las propiedades definidas en la tabla de la base de datos. En este caso, el nombre del archivo va a ser la entidad seguido por Interface: DetectionInterface.php.

Se recuerdan las propiedades que se quieren manejar: `key_id`, `user_ip`, `country_code`, `browser` y `device`. Cada una de estas propiedades tendrán sus correspondientes métodos getters y setters que permitirán devolver el valor de las variables (getter) y establecer los valores (setter).



```php
<?php

namespace Hiberus\Dashboard\Api\Data;

/**
 * Interface DetectionInterface
 * @package Hiberus\Dashboard\Api\Data
 */
interface DetectionInterface
{

    /**
     * Get Key ID
     * @return int
     */
    public function getKeyId();

    /**
     * Set Key Id
     * @param int $keyId
     * @return $this
     */
    public function setKeyId(int $keyId);

    /**
     * Get User IP
     * @return int
     */
    public function getUserIp();

    /**
     * Set User IP
     * @param string $userIp
     * @return $this
     */
    public function setUserIp(string $userIp);

```

Figura 4.3.4-d: Clase DetectionInterface

En este directorio también se va a crear la interfaz de la entidad que extiende el SearchResultsInterface de Magento ya que al usar el método `getList()` se necesita esta configuración para su correcto funcionamiento. Esta interfaz consiste en reescribir los tipos de los dos métodos `getItems()` y `setItems()` de la interfaz principal de la que hereda para el propósito de la base de datos creada.

```php
<?php

namespace Hiberus\Dashboard\Api\Data;

use Magento\Framework\Api\SearchResultsInterface;

/**
 * Interface DetectionSearchResultsInterface
 * @package Hiberus\Dashboard\Api\Data
 */
interface DetectionSearchResultsInterface extends SearchResultsInterface
{
    /**
     * Get detection list.
     *
     * @return \Hiberus\Dashboard\Api\Data\DetectionInterface[]
     */
    public function getItems();

    /**
     * Set detection list.
     *
     * @param \Hiberus\Dashboard\Api\Data\DetectionInterface[] $items
     * @return $this
     */
    public function setItems(array $items);
}
```

Figura 4.3.4-e: Clase DetectionSearchResultsInterface

Paso 3. Crear los repositorios

El último paso para gestionar los datos de la tabla `'hiberus_dashboard_table'` se basa en crear la implementación de las interfaces que se han establecido. Para ello, en Magento se hace uso del ORM (Object Relational Mapping) que es una técnica de conversión entre los tipos de datos y objetos en una programación orientada a objetos (OOP). En Magento el ORM consiste en Models, ResourceModels y Collections.

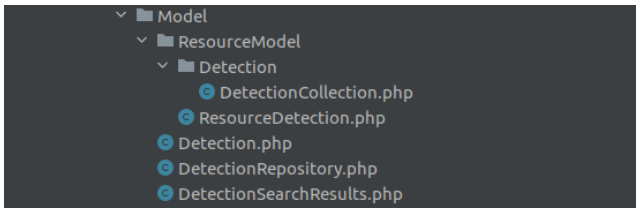


Figura 4.3.4-f: Estructura de un ORM

Los modelos representan la entidad y proporcionan acceso a los datos de la entidad. El primer archivo Detection.php extiende de la clase de Magento AbstractModel para heredar sus métodos y lógica e implementa la interfaz de Detection (DetectionInterface) para realizar la funcionaliad de los métodos getters y setters creados.

El modelo se suele usar en conjunto con la clase del ResourceModel (se declara más adelante) que es la que sabe cómo cargar y almacenar los datos en la base de datos.

Para su buen funcionamiento, en el método `_construct()` (no es el método público `__construct()`) se llama al método `_init()` (método que se ha heredado del AbstractModel) con el nombre de la clase del ResourceModel.

```php
<?php

namespace Hiberus\Dashboard\Model;

use Hiberus\Dashboard\Api\Data\DetectionInterface;
use Hiberus\Dashboard\Model\ResourceModel\ResourceDetection;
use Magento\Framework\Model\AbstractModel;

/**
 * Class Detection
 * @package Hiberus\Dashboard\Model
 */
class Detection extends AbstractModel implements DetectionInterface
{

    /**
     * Detection Construct
     */
    protected function _construct()
    {
        $this->_init( resourceModel: ResourceDetection::class);
    }
```

Figura 4.3.4-g: Modelo Detection (constructor)

Los demás métodos son los relacionados con los que se implementa de la interfaz DTO, DetectionInterface. Se determina el funcionamiento de cada getter y setter.

```php
    /**
     * @return int|mixed
     */
    public function getKeyId()
    {
        return $this->_getData( key: 'key_id');
    }

    /**
     * @param int $keyId
     * @return $this|DetectionInterface
     */
    public function setKeyId(int $keyId)
    {
        $this->setData('key_id', $keyId);
        return $this;
    }

    /**
     * @return mixed|string
     */
    public function getUserIp()
    {
        return $this->_getData( key: 'user_ip');
    }

    /**
     * @param string $userIp
     * @return $this|DetectionInterface
     */
    public function setUserIp(string $userIp)
    {
        $this->setData('user_ip', $userIp);
        return $this;
    }
```

Figura 4.3.4-h: Modelo Detection (getters y setters)

El siguiente modelo está relacionado con la interfaz del repositorio DAO, DetectionRepositoryInterface. Los métodos son los mismos que se definieron en la interfaz

(`save()`, `delete()` y `getList()`) pero además se implementan nuevos métodos que permitir filtrar los resultados, ordenarlos y convertir los datos en objetos.

En la función save y delete se les pasa por parámetro la entidad DetectionInterface y mediante el resourceDetection crea las funciones de guardar y eliminar.

Figura 4.3.4-i: Modelo DetectionRepository (save)

Figura 4.3.4-j: Modelo DetectionRepository (delete)

El framework de Magento 2 generará la interfaz, la implementación y el `factory` si en las implementaciones de interfaces, repositorios y ORM se han nombrado correctamente.

En el método `getList()` se le pasa por parámetro el `SearchCriteria` y tiene que traducir las condiciones de SearchCriteria en llamadas a métodos en la colección.

En la `DetectionCollection()` se va a crear el objeto con el Factory que permite auto-generar la clase para crear nuevas instancias. Se le pasa el método `getSearchResultsCollection()` que contiene los métodos con los criterios de filtros y orden que se establecen más adelante, mientras que el número de resultados de la página actual y la paginación se establecen por defecto.

Y devuelve la interfaz `detectionSearchResults` que establece los valores de los modelos a objetos con el método `turnModelObjectIntoDataObect()`.

```php
/**
 * @param SearchCriteria $searchCriteria
 * @return \Hiberus\Dashboard\Api\Data\DetectionSearchResultsInterface|DetectionSearchResults
 */
public function getList(SearchCriteria $searchCriteria)
{
    $this->collection = $this->detectionCollectionFactory->create();
    $this->getSearchResultsCollection($searchCriteria);
    return $this->detectionSearchResults->setItems(
        $this->turnModelObjectIntoDataObject()
    );
}
```

Figura 4.3.4-k: Modelo DetectionRepository (getList)

El método `getSearchResultsCollection()` se le pasa por parámetro el SearchCriteria y devuelve para el método `getList()` las instancias que se añade el searchCriteria.

```php
/**
 * @param SearchCriteria $searchCriteria
 * @return DetectionRepositoryInterface
 */
protected function getSearchResultsCollection(SearchCriteria $searchCriteria): DetectionRepositoryInterface
{
    $this->detectionSearchResults->setSearchCriteria($searchCriteria);
    $this->setFilters($searchCriteria);
    $this->detectionSearchResults->setTotalCount($this->collection->getSize());
    $this->setSortOrders($searchCriteria, $this->collection);
    $this->collection->setCurPage($searchCriteria->getCurrentPage());
    $this->collection->setPageSize($searchCriteria->getPageSize());

    return $this;
}
```

Figura 4.3.4-l: Modelo DetectionRepository (getSearchResultsCollection)

En este método se establece el orden para mostrar los campos de la colección es el. El campo que va a ir en orden ascendente es el de la clave primaria.

```php
/**
 * @param SearchCriteria $searchCriteria
 * @param DetectionCollection $collection
 */
protected function setSortOrders(SearchCriteria $searchCriteria, DetectionCollection $collection): void
{
    $sortOrders = $searchCriteria->getSortOrders();

    if ($sortOrders) {
        foreach ($sortOrders as $sortOrder) {
            $collection->addOrder(
                $sortOrder->getField(),
                direction: ($sortOrder->getDirection() == SortOrder::SORT_ASC) ? 'ASC' : 'DESC'
            );
        }
    }
}
```

Figura 4.3.4-m. Modelo DetectionRepository (setSortOrders)

En el método setFilters() se establece que el filtro que se va a aplicar a los campos de la colección es en el que el valor es igual a la condición.

```php
/**
 * @param SearchCriteria $searchCriteria
 */
protected function setFilters(SearchCriteria $searchCriteria): void
{
    foreach ($searchCriteria->getFilterGroups() as $filterGroup) {
        $fields = [];
        $conditions = [];
        foreach ($filterGroup->getFilters() as $filter) {
            $condition = $filter->getConditionType() ? $filter->getConditionType() : 'eq';
            $fields[] = $filter->getField();
            $conditions[] = [$condition => $filter->getValue()];
        }
        if ($fields) {
            $this->collection->addFieldToFilter($fields, $conditions);
        }
    }
}
```

Figura 4.3.4-n: Modelo DetectionRepository (setFilters)

Este último método es el que se encarga de convertir los datos de la entidad en objetos y este resultado es pasado al método `getList()` para que aplique los filtros en forma de objetos.

```php
public function turnModelObjectIntoDataObject(): array
{
    $items = [];
    /**@var DetectionInterface $modelObject */
    foreach ($this->collection as $modelObject) {
        /** Data Object
         * @var DetectionInterface $dataObject
         */
        $dataObject = $this->detectionInterfaceFactory->create();
        $dataObject->setKeyId($modelObject->getKeyId())
            ->setUserIp($modelObject->getUserIp())
            ->setCountryCode($modelObject->getCountryCode())
            ->setDevice($modelObject->getDevice())
            ->setBrowser($modelObject->getBrowser());

        $items[] = $dataObject;
    }
    return $items;
}
```

Figura 4.3.4-ñ: Modelo DetectionRepository (turnModelObjectIntoDataObject)

El último modelo es el más sencillo debido a que hereda toda la funcionalidad de los métodos del framework con la clase SearchResults e implementa las funciones declaradas de la interfaz. Está vacío porque no ha sido necesario crear una funcionalidad, ya que la declaración de los métodos fueron creados en la interfaz y la funcionalidad de estos métodos se heredan del SearchResults.

```php
<?php
namespace Hiberus\Dashboard\Model;

use Hiberus\Dashboard\Api\Data\DetectionSearchResultsInterface;
use Magento\Framework\Api\SearchResults;

/**
 * Class DetectionSearchResults
 * @package Hiberus\Dashboard\Model
 */
class DetectionSearchResults extends SearchResults implements DetectionSearchResultsInterface
{

}
```

Figura 4.3.4-o: Modelo DetectionSearchResults

Una vez que se han definido los Modelos, quedan por detallar el ResourceModel y Collection. Estos archivos se crean al mismo tiempo ya que los modelos dependen del ResourceModel y Collection, y para poder utilizarlos se deben crear.

El ResourceModel hereda los métodos de la clase AbstractDb de Magento para utilizarlo en el constructor con el método `_init()`. En este método se inicializa la tabla y su clave primaria para saber con que tabla y clave se tiene que usar para trabajar con ella.

```php
<?php

namespace Hiberus\Dashboard\Model\ResourceModel;

use Magento\Framework\Model\ResourceModel\Db\AbstractDb;

/**
 * Class ResourceDetection
 * @package Hiberus\Dashboard\Model\ResourceModel
 */
class ResourceDetection extends AbstractDb
{
    /**
     * Constant db_schema table name
     */
    const TABLE = 'hiberus_dashboard_table';

    /**
     * @var string
     */
    protected $_IdFieldName = 'key_id';

    /**
     * ResourceDetection Construct
     */
    protected function _construct()
    {
        $this->_init( mainTable: self::TABLE, $this->_IdFieldName);
    }
}
```

Figura 4.3.4-p: ResourceDetection

La clase `Collection` se necesita sólo en caso de que se quieran mostrar todos los datos (`getList()`). Su ubicación se encuentra dentro del directorio `/Model/ResourceModel` y el nombre de la entidad debe ir seguido de Collection, DetectionCollection. Debe especificar a qué modelo pertenece, para cargar las clases adecuadas

Inicializa en el constructor con el método heredado `init()` la clase de la entidad y el resource model para encapsular los datos y poder acceder a la base de datos. Con la creación del Collection también se genera el código de la entidad del DetectionCollectionFactory automáticamente (que es usado en el modelo de la interfaz DAO).

```php
<?php

namespace Hiberus\Dashboard\Model\ResourceModel\Detection;

use Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection;
use Hiberus\Dashboard\Model\Detection;
use Hiberus\Dashboard\Model\ResourceModel\ResourceDetection;

/**
 * Class DetectionCollection
 * @package Hiberus\Dashboard\Model\ResourceModel\Detection
 */
class DetectionCollection extends AbstractCollection
{
    /**
     * Construct that define model and resource model
     */
    protected function _construct()
    {
        $this->_init( model: Detection::class, resourceModel: ResourceDetection::class);
    }
}
```

Figura 4.3.4-q: DetectionCollection

Paso 4. Configurar las dependencias

Una vez que las implementaciones de las interfaces y ORM se han completado, se deben instanciar las interfaces de las dependencias de otras clases en el archivo di.xml ya que todavía el framework no es capaz de procesar que implementaciones debe usar.

En el contenido del archivo di.xml es diferente al del plugin, en este se usa la etiqueta `<preference>` para indicar que cuando haya una instancia de la interfaz es cree la instancia del objeto.

```xml
<?xml version="1.0"?>
<config  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.xsd">
    <!-- Interface that are going to use Service class-->
    <preference for="Hiberus\Dashboard\Api\GeolocationServiceInterface" type="Hiberus\Dashboard\Service\GeolocationService"/>
    <!-- DB implementations -->
    <preference for="Hiberus\Dashboard\Api\Data\DetectionInterface" type="Hiberus\Dashboard\Model\Detection"/>
    <preference for="Hiberus\Dashboard\Api\DetectionRepositoryInterface" type="Hiberus\Dashboard\Model\DetectionRepository"/>
</config>
```

Figura 4.3.4-r: Archivo di.xml de las interfaces de Detection y GeolocationService

Tras crear las dependencias de las implementaciones, se debe definir las interfaces creadas como un recurso API mediante el archivo webapi.xml ubicado en el directorio `/etc` para que Magento haga que las clases ubicadas en la etiqueta `<service>` estén disponibles de forma dinámica.

`<routes>`: Es el elemento raíz que define el namespace y ubicación del archivo XML.

`<route>`: Define la ruta HTTP para el método API. Para cada route se ha fijado el método correspondiente a la función. Los métodos válidos son GET, POST, PUT y DELETE. La url tiene que empezar obligatoriamente con “/V1” con una cadena identificativa.

`<service>`: Define ubicación de la interfaz implementada y el nombre del método de la API.

`<resource>`: Define un recurso al que debe tener acceso la persona que llama. Los valores válidos son self, anonymous o un recurso de Magento (como `Magento_Customer::group`)

```xml
<?xml version="1.0"?>
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Webapi:etc/webapi.xsd">

    <!-- SAVE -->
        <route method="POST" url="/V1/hiberus/dashboard/detections">
            <service class="Hiberus\Dashboard\Api\DetectionRepositoryInterface" method="save"/>
            <resources>
                <resource ref="anonymous"/>
            </resources>
        </route>

    <!-- DELETE -->
    <route method="DELETE" url="/V1/hiberus/dashboard/detections">
        <service class="Hiberus\Dashboard\Api\DetectionRepositoryInterface" method="delete"/>
        <resources>
            <resource ref="anonymous"/>
        </resources>
    </route>

    <!-- GETLIST-->
    <route method="GET" url="/V1/hiberus/dashboard/detections">
        <service class="Hiberus\Dashboard\Api\DetectionRepositoryInterface" method="getList"/>
        <resources>
            <resource ref="anonymous"/>
        </resources>
    </route>

</routes>
```

Figura 4.3.4-s: Archivo webapi.xml de las interfaces

4.4. Módulo de desplegable de resultados

El desarrollo de los módulos finaliza con este apartado, el módulo de desplegable de resultados. Está divido en cinco subapartados, siendo los últimos cuatro los que definen los directorios de la principal finalidad de mostrar un desplegable de resultados personalizado en el buscador. En el Anexo II se continúa con la funcionalidad de mostrar en el dashboard los términos consultados (con y sin resultados) y los estilos para la maquetación de la página del dashboard.

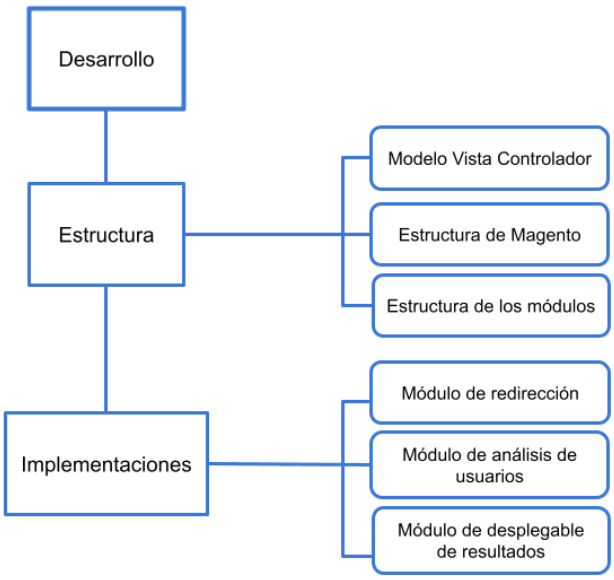


Figura 4-d: Diagrama de pasos para el desarrollo - Módulo de desplegable de resultados

4.4.1. Contexto

El propósito principal de este módulo consiste en ampliar la funcionalidad del desplegable de resultados del buscador de Magento. Al realizar una consulta en el buscador de la tienda online, el desplegable de resultados de Magento muestra los términos sugeridos y el número de resultados que tiene. Se pretende crear un módulo que además de mostrar los términos y cantidad, muestre la imagen de una selección de productos y el precio.

Conforme se realizaba el desarrollo del módulo, se observó la necesidad de añadir en el panel de administración un dashboard que le sirva al gestor de la tienda para analizar las palabras que más se buscan y ver cuales arrojan resultados de productos y cuales no. Los resultados se van a mostrar en tablas y así el gestor podrá tomar decisiones como promocionar los productos que más se buscan y ofrecen resultados, y para los productos que más se buscan y no ofrecen resultados deliberar si añadirlo a su catálogo de productos o si corresponde a otro producto pero tiene una denominación diferente (es un sinónimo). También se añadió la posibilidad de descargar los resultados en formato .csv.

En este apartado de desarrollo se va a exponer la principal idea, el desplegable de resultados y en el Anexo II el análisis de estas palabras.

La estructura completa del módulo se muestra en la siguiente imagen:

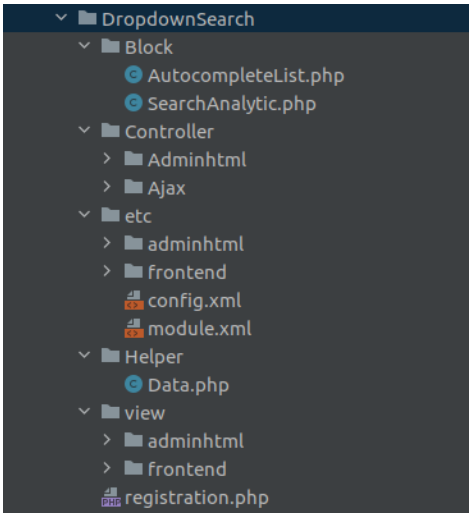


Figura 4.4.1-a: Estructura del módulo desplegable de resultados

Como se ha mencionado anteriormente, para que Magento reconozca que existe un nuevo módulo, se necesitan los archivos registration.php y module.xml:

```php
<?php

\Magento\Framework\Component\ComponentRegistrar::register(
    type: \Magento\Framework\Component\ComponentRegistrar::MODULE,
    componentName: 'Hiberus_DropdownSearch',
    path: __DIR__
);
```

Figura 4.4.1-b: Contenido del archivo registration.php del módulo del desplegable

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
    <module name="Hiberus_DropdownSearch" setup_version="1.0.1">
    </module>
</config>
```

Figura 4.4.1-c: Contenido del archivo module.xml del módulo del desplegable

4.4.2. Implementación del Helper

En el desarrollo de este módulo la lógica reside en los directorios `/Block`, `/Helper` y `/Controller` (junto con `/etc/frontend/routes.xml` y `view/frontend/layout` para crear la ruta de la url) y la vista dentro del directorio `/view`.

En la carpeta Helper se han creado métodos estáticos cuyos elementos van a estar disponibles globalmente. Va a servir de ayuda complementaria para la funcionalidad que se establezca en el bloque y reducir las líneas de código en este. El archivo Data.php va a comprobar tanto si el módulo como algunos parámetros (resultados de términos y productos) están habilitados y el valor que tienen para realizar la consulta en el desplegable.

En primer lugar, establece como constantes de las rutas del archivo system.xml para utilizarlas en los métodos correspondientes que devolverán, gracias al ScopeConfig, si el módulo está habilitado o no. Estos resultados se les pasarán al bloque, que a su vez en los métodos que este cree se les pasarán a la plantilla para trabajar con ellas mediante Javascript.

```php
<?php

namespace Hiberus\DropdownSearch\Helper;

use Magento\Store\Model\ScopeInterface;

class Data extends \Magento\Framework\App\Helper\AbstractHelper
{
    const PATH_ENABLE_SEARCH = 'section_dropdown/general_dropdown/enable_dropdown';

    const PATH_SEARCH_DELAY = 'section_dropdown/general_dropdown/delay_time';

    const PATH_MIN_CHAR = 'section_dropdown/general_dropdown/min_characters';

    const PATH_NO_RESULT = 'section_dropdown/general_dropdown/no_result';

    const PATH_SEARCH_TERM_ENABLE = 'section_dropdown/terms_settings/enable_terms';

    const PATH_PRODUCT_ENABLE = 'section_dropdown/products_settings/enable_products';

    const PATH_PRODUCT_RESULTS = 'section_dropdown/products_settings/products_max';
```

Figura 4.4.2-a: Constantes del archivo Data.php del módulo del desplegable

Como los funcionamientos de los métodos son los mismos, se van a mostrar los 2 primeros pero se van a mencionar qué hacen los demás que se han creado:

`getIsSearchEnable()`: Verifica si el módulo está habilitado

`getDelayValue()`: Fija el tiempo de retraso para comenzar la búsqueda

`getMinCharactersValue()`: Establece el número de caracteres para comenzar la búsqueda

`getNoResultValue()`: Determina el mensaje que se va a mostrar cuando no existan resultados

`getIsSearchTermsEnable()`: Verifica si los resultados de los términos está activado

`getIsProductSearchEnable()`: Verifica si los resultados de los productos (imagen, titulo y precio) están activados.

`getMaxResuxtsValue()`: Establece el valor de los productos a mostrar.

```php
/**
 * Retrieve search enable
 *
 * @param int|null $storeId storeId
 *
 * @return int
 */
public function getIsSearchEnable($storeId = null)
{
    return $this->scopeConfig->getValue(
        path: self::PATH_ENABLE_SEARCH,
        scopeType: ScopeInterface::SCOPE_STORE,
        $storeId
    );
}

/**
 * Retrieve search delay default 1000ms
 *
 * @param int|null $storeId storeId
 *
 * @return int
 */
public function getDelayValue($storeId = null)
{
    return (int)$this->scopeConfig->getValue(
        path: self::PATH_SEARCH_DELAY,
        scopeType: ScopeInterface::SCOPE_STORE,
        $storeId
    );
}
```

Figura 4.4.2-b Contenido del archivo Data.php del módulo del desplegable

4.4.3. Implementación del Route y Controller

Al realizarse una consulta y cargarse resultados, en este caso, en forma de desplegable, también se necesita cargar una url para poder hacer la consulta. Para obtener la url se requiere crear la ruta `'route_name/controller/action'`.

Para el `'route_name'` se ha creado en el directorio `/etc/frontend` el archivo routes.xml ya que sólo va a ser usado en la vista de la tienda y así se minimizan los procesos de carga (en lugar de crearlo en el directorio `/etc`).

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:App/etc/routes.xsd">
    <router id="standard">
        <route id="dropdown_search" frontName="dropdown_search">
            <module name="Hiberus_DropdownSearch"/>
        </route>
    </router>
</config>
```

Figura 4.4.3-a: Contenido de /etc/frontend/routes.php del módulo del desplegable

Para el `'controller/action'` se ha creado el directorio `/Controller/Ajax` y el fichero Index.php, esto corresponde a `'ajax/index'`. El contenido del archivo devuelve el resultado en formato JSON de la consulta que cargará los términos y productos (si están activados)

```php
public function execute()
{
    $query = $this->queryFactory->get();

    $responseData = [];

    if ($query->getQueryText() != '') {
        $responseData['result'] = $this->autocompleteBlock->getResponseData();
    }

    $resultJson = $this->resultFactory->create( type: ResultFactory::TYPE_JSON);
    $resultJson->setData($responseData);

    return $resultJson;
}
}
```

Figura 4.4.3-b: Contenido del archivo Index.php del módulo del desplegable

4.4.4. Implementación del Block

Una vez que se tiene la ruta de la url, se procede a crear el desplegable de resultados mediante el uso de bloques que devolverá la lista de productos y términos. Los métodos en esta clase se intercambiarán en la plantilla para darle funcionalidad y con el uso de Javascript se va a conseguir mantener una comunicación asíncrona.

En la carpeta `/Block` se han implementado las clases de Magento:

- Del directorio /Search el fichero Data.php: Se va a utilizar para obtener la url de la página de resultados
- AutocompleteInterface.php: Se encarga de obtener el array de ítems
- QueryFactory.php: Se encarga de obtener el objeto mediante una clave y guardar la consulta con el número de resultados y en orden ascendente de popularidad.
- Product.php: Se encarga de obtener mediante la url propiedades de productos, para el desarrollo se obtendrá la imagen.
- Resolver.php: Crea la capa actual del catálogo de productos

Se recuerda que los valores que se definen por defecto en la administración, están en el archivo /etc/config.xml. El método `getDelay()` devuelve por defecto el tiempo en milisegundos para que la consulta comience; el método `getSearchPageUrl()` devuelve la url creada con las rutas

y controlador; el método getMincharacters() devuelve el número mínimo de caracteres para comenzar la consulta y el getNoResults() devuelve un mensaje en caso de que no hayan resultados.

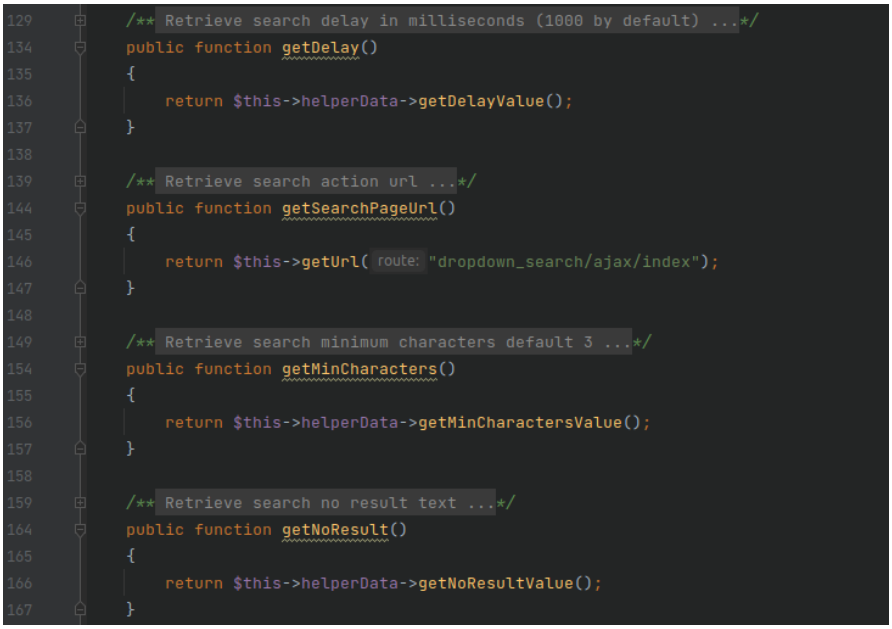


Figura 4.4.4-a: Contenido del archivo AutocompleteList.php del módulo del desplegable (1)

El método `getResponseData()` determina si la búsqueda de términos y de productos está activada para devolver un array con los resultados (mediante la llamada a sus respectivos métodos).

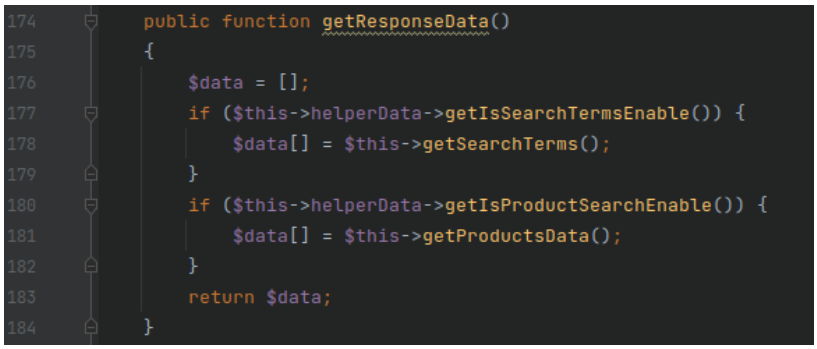


Figura 4.4.4-b: Contenido del archivo AutocompleteList.php del módulo del desplegable (2)

En el método `getSearchTerms()` se establece que cuando '`code`' tenga el valor de '`term`', que se obtengan los resultados (mediante la interfaz que provee Magento) y que se guarde cada uno de los valores con su url.

```php
public function getSearchTerms()
{
    $termsData['code'] = 'term';
    $termsData['data'] = [];
    $autocompleteData = $this->autocomplete->getItems();
    foreach ($autocompleteData as $itemData) {
        $itemData                 = $itemData->toArray();
        $itemData['url']          = $this->searchHelper->getResultUrl($itemData['title']);
        $termsData['data'][] = $itemData;
    }

    return $termsData;
}
```

Figura 4.4.4-c: Contenido del archivo AutocompleteList.php del módulo del desplegable (3)

El método `getProductData()` devuelve un array para cada resultado de producto con: el nombre del producto, la imagen del producto, el precio y la url que se creó (con los archivos routes y controller). En el caso de la imagen primero obtiene el tipo de moneda y si son varios productos iguales no se puede obtener el precio. Sin embargo, si el producto contiene un precio se carga la moneda (según la tienda) y se concatena las etiquetas HTML que se van a pasar al Javascript.

```php
public function getProductData($product)
{
    $price = $product->getFormattedPrice();
    if ($product->getTypeId() == 'bundle' || $product->getTypeId() == 'grouped') {
        $price = '';
        if (!empty($product->getMinimalPrice())) {
            $currencyCode = $this->_storeManager->getStore()->getCurrentCurrencyCode();
            $currency = $this->currencyFactory->load($currencyCode);
            $price .= '<span class="price">'.$currency->getCurrencySymbol().number_format($product->getMinimalPrice(), decimals: 2).'</span>';
        }
    }
    $data = [
        'name'              => $product->getName(),
        'image'             => $this->productHelper->getSmallImageUrl($product),
        'price'             => $price,
        'url'               => $product->getProductUrl()
    ];
    return $data;
}
```

Figura 4.4.4-d: Contenido del archivo AutocompleteList.php del módulo del desplegable (4)

Al tener todos los atributos que se desean de cada producto, se tiene que guardar, al igual que con los términos, para cuando el valor ‘`code`’ coincida con ‘`product`’.

El proceso continúa creando la capa actual de la colección de productos con todos sus atributos guardándolo en formato JSON y pasándole el límite del número de productos que se van a mostrar. Se recorre la colección de resultados de productos (producto con los atributos de nombre, precio, imagen y url que se establecieron en el anterior método) para obtener un array que en el valor ‘`data`’ va a guardar de cada producto los valores del producto.

También guarda en el valor ‘`size`’ el número de veces que aparece y si hay resultados, en el valor ‘`url`’ guarda la url de la página de resultados, sino devuelve un string vacío para que se muestre el mensaje personalizado. Finalmente se ha establecido que los resultados de productos se muestren por popularidad.

```php
public function getProductsData()
{
    $productData['code'] = 'product';
    $productData['data'] = [];

    $productResultNumber = $this->helperData->getMaxResultsValue();
    $productResultFields[]   = 'url';
    $query                   = $this->queryFactory->get();
    $queryText               = $query->getQueryText();

    $this->layerResolver->create( layerType: LayerResolver::CATALOG_LAYER_SEARCH);

    $productCollection = $this->layerResolver->get()
        ->getProductCollection()
        ->addAttributeToSelect( attribute: '*')
        ->addSearchFilter($queryText);
    $productCollection->getSelect()->limit($productResultNumber);

    foreach ($productCollection as $product) {
        $productData['data'][] = $this->getProductData($product);
    }

    $productData['size'] = $productCollection->getSize();
    $productData['url']  = ($productCollection->getSize() > 0) ?
        $this->searchHelper->getResultUrl($queryText) :
        '';

    $query->saveNumResults($productData['size']);
    $query->saveIncrementalPopularity();
    return $productData;
}
```

Figura 4.4.4-e: Contenido del archivo AutocompleteList.php del módulo del desplegable (5)

4.4.5. Implementación del View

Ahora que se tiene la lógica lista, se continúa con la vista para establecer que va a hacer visualmente el desplegable. Se ha creado en el directorio `/frontend` los siguientes archivos:

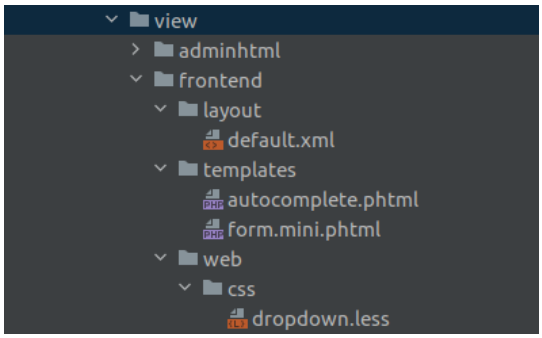
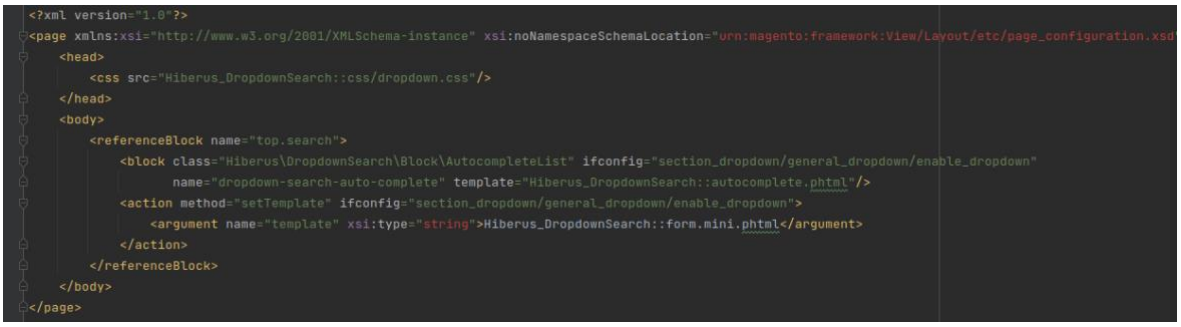


Figura 4.4.5-a: Contenido de la estructura de /view/frontend del módulo del desplegable

Se ha replicado el funcionamiento del desplegable de resultados de Magento (form.mini.phtml) copiando el mismo código y se ha añadido nombres a las clases para que reconozca el archivo del módulo y no el de Magento. Para que sea así se ha tenido que definir en el layout que este sea el formulario que se utilice para enviar las consultas (dentro de las etiquetas `<action>`). También en la etiqueta `<block>` se establece que el bloque sea el que le pase la lógica a la plantilla

autocomplete.phtml y tanto en el `<block>` como en el `<action>` se ha establecido la condición de que se haga si el módulo está habilitado.

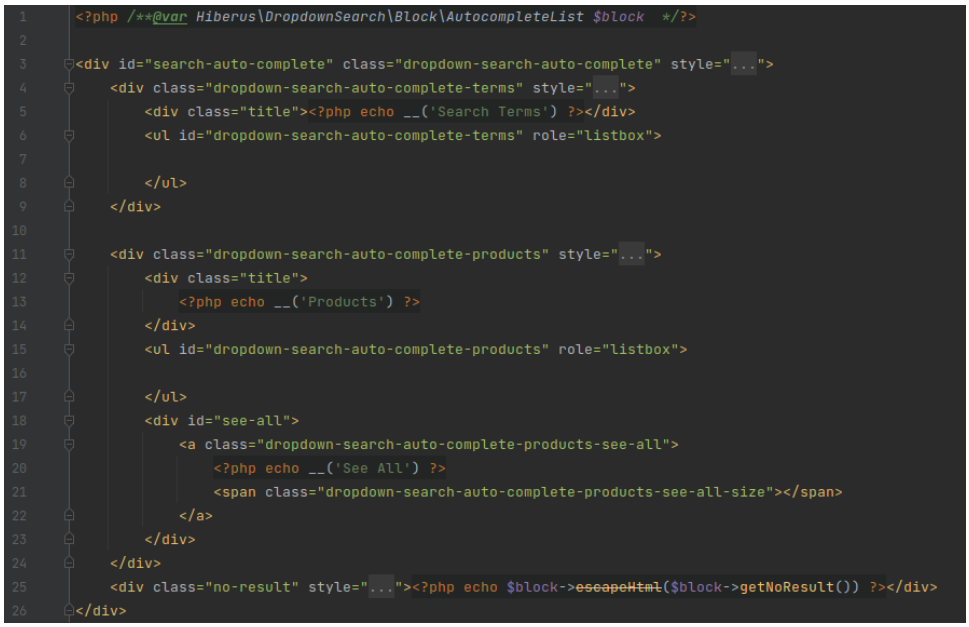


```xml
<?xml version="1.0"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configuration.xsd
    <head>
        <css src="Hiberus_DropdownSearch::css/dropdown.css"/>
    </head>
    <body>
        <referenceBlock name="top.search">
            <block class="Hiberus\DropdownSearch\Block\AutocompleteList" ifconfig="section_dropdown/general_dropdown/enable_dropdown"
                   name="dropdown-search-auto-complete" template="Hiberus_DropdownSearch::autocomplete.phtml"/>
            <action method="setTemplate" ifconfig="section_dropdown/general_dropdown/enable_dropdown">
                <argument name="template" xsi:type="string">Hiberus_DropdownSearch::form.mini.phtml</argument>
            </action>
        </referenceBlock>
    </body>
</page>
```

Figura 4.4.5-b: Contenido del archivo layout.xml del módulo del desplegable

El archivo form.mini.phtml no se va a explicar ya que es el mismo que el de Magento lo que va a permitir la extensión del archivo autocomplete.phtml. Tampoco se va a explicar el contenido del directorio /web que sólo contiene los estilos que se han definido para el desplegable por su tamaño.

Este archivo comprende en la primera parte el contenido PHP y HTML y en la segunda parte el contenido en Javascript. En el contenido phtml se ha definido la estructura del desplegable mostrando en primer lugar el listado de los términos de productos, seguido del listado de los productos y con un botón que permite redirigir a la página de resultados. También se ha creado el contenedor para que se pinte el texto cuando no hayan resultados.



```php
<?php /**@var Hiberus\DropdownSearch\Block\AutocompleteList $block  */?>

<div id="search-auto-complete" class="dropdown-search-auto-complete" style="...">
    <div class="dropdown-search-auto-complete-terms" style="...">
        <div class="title"><?php echo __('Search Terms') ?></div>
        <ul id="dropdown-search-auto-complete-terms" role="listbox">

        </ul>
    </div>

    <div class="dropdown-search-auto-complete-products" style="...">
        <div class="title">
            <?php echo __('Products') ?>
        </div>
        <ul id="dropdown-search-auto-complete-products" role="listbox">

        </ul>
        <div id="see-all">
            <a class="dropdown-search-auto-complete-products-see-all">
                <?php echo __('See All') ?>
                <span class="dropdown-search-auto-complete-products-see-all-size"></span>
            </a>
        </div>
    </div>
    <div class="no-result" style="..."><?php echo $block->~~escapeHtml~~($block->getNoResult()) ?></div>
</div>
```

Figura 4.4.5-c: Contenido del archivo autocomplete.phtml del módulo del desplegable (1)

En primer lugar, se han establecido mediante los bloques los caracteres mínimos y el tiempo de retraso para comenzar la consulta. Se ha definido que cuando ya no se escriba, si la longitud de los caracteres es menor al que se ha establecido para que no se muestre aún el desplegable y si lo supera que realice la función `doneTyping()` con el tiempo de retraso definido.

```
<script>
    require(
        [
            'jquery'
        ],
        function(
            $,
        ) {
            let minCharacters = "<?php echo $block->escapeHtml($block->getMinCharacters()) ?>";
            let timeLapse = "<?php echo $block->getDelay() ? $block->escapeHtml($block->getDelay()) : 0 ?>";
            let timer = null;
            $("#search").keyup(function(){
                    clearTimeout(timer);
                    if($("#search").val().length < minCharacters){
                        $("#search-auto-complete").hide();
                    }
                    timer = setTimeout(doneTyping, timeLapse)
            });
```

Figura 4.4.5-d: Contenido del archivo autocomplete.phtml del módulo del desplegable (2)

La función `doneTyping()` comprueba nuevamente el tamaño de caracteres y en el caso de sea mayor realiza la llamada Ajax al controlador. Si el resultado de la llamada ha sido satisfactorio y si contiene elementos, se llama a la función `DropdownSearchParseData()` con su contenido.

```
function doneTyping () {
    if($("#search").val().length < minCharacters){
        $("#search-auto-complete").hide();
        return false;
    }

    if($("#search").val().length >= minCharacters){
        $.ajax({
            url: "<?php echo $block->escapeUrl($block->getSearchPageUrl()) ?>",
            data: {q: $("#search").val()},
            dataType: "json",
            type: 'GET',
            success: function (response) {
                if(typeof response.result != 'undefined' && response.result.length > 0){
                    DropdownSearchParseData(response.result);
                }
            }
        });
        return false;
    }else{
        $("#search-auto-complete").hide();
    }
}
```

Figura 4.4.5-e: Contenido del archivo autocomplete.phtml del módulo del desplegable (3)

También se ha creado la opción de que si al estar el desplegable expandido y se hace click fuera de él, este se oculta.

```
$(document).on("focusout","#search",function(){
    if(!jQuery("#search-auto-complete").is(":hover")){
        $("#search-auto-complete").hide();
    }
});
```

Figura 4.4.5-f: Contenido del archivo autocomplete.phtml del módulo del desplegable (4)

Por último, en la función `DropdownSearchParseData()` se obtienen los datos de los resultados de los términos y productos. Para llevar la cuenta del número de veces que salen los términos y productos se establecen contadores y para poder realizar las concatenaciones del contenido de los resultados se inicializan los dos strings vacíos. En un inicio los contenedores de términos y productos van a estar vacíos y ocultos.

```
function DropdownSearchParseData(result) {
    console.log(result);
    let termsCount = 0;
    let productsCount = 0;
    let htmlTerms = '';
    let htmlProducts = '';

    $("#dropdown-search-auto-complete-terms").html('');
    $("#dropdown-search-auto-complete-products").html('');

    $(".dropdown-search-auto-complete-terms").hide();
    $(".dropdown-search-auto-complete-products").hide();
```

Figura 4.4.5-g: Contenido del archivo autocomplete.phtml del módulo del desplegable (5)

Se accede al array de resultados. Este array tiene dos resultados (con el valor '`code`') en el primer nivel que corresponde a la identificación de si se trata de términos o productos. El segundo nivel del array de términos devuelve los arrays con los datos de términos y número de resultados. En el segundo nivel del array de productos el array de datos contiene los atributos nombre, imagen, precio y url.

```
▼Array(2)
  ▼0:
    code: "term"
    ▼data: Array(2)
      ▶0: {title: "pant", num_results: "16", url: "http://www.magento2-elastic-local/catalogsearch/result/?q=pant"}
      ▶1: {title: "pants", num_results: "14", url: "http://www.magento2-elastic-local/catalogsearch/result/?q=pants"}
       length: 2
      ▶__proto__: Array(0)
    ▶__proto__: Object
  ▼1:
    code: "product"
    ▼data: Array(5)
      ▼0:
         image: "http://www.magento2-elastic-local/media/catalog/product/m/p/mp12-black_main_1.jpg"
         name: "Cronus Yoga Pant "
         price: "<span class=\"price\">48,00 €</span>"
         url: "http://www.magento2-elastic-local/cronus-yoga-pant.html"
        ▶__proto__: Object
      ▼1:
         image: "http://www.magento2-elastic-local/media/catalog/product/m/p/mp11-brown_main_1.jpg"
         name: "Aether Gym Pant "
         price: "<span class=\"price\">74,00 €</span>"
         url: "http://www.magento2-elastic-local/aether-gym-pant.html"
        ▶__proto__: Object
      ▶2: {name: "Orestes Yoga Pant ", image: "http://www.magento2-elastic-local/media/catalog/product/m/p/mp10-black
```

Figura 4.4.5-h: Contenido del archivo autocomplete.phtml del módulo del desplegable (6)

Se comprueba si hay resultados de términos y se recorre el array para sacar cada elemento (título, número de resultados y url) y estructurar el código HTML. En el atributo href del enlace se establece el contenido de la url del término y, en el listado se concatena el nombre y número de resultados y se muestra en el desplegable. Si no hay resultados, el contenedor de términos sigue oculto.

```
result.forEach(function(res){
    if(res.code === 'term'){
        termsCount +=  res.data.length;
        if(res.data.length > 0){
            res.data.forEach(function(termsRes){
                console.log('eyy: '+termsRes.title);
                htmlTerms += '<a href="'+termsRes.url+'" title="'+termsRes.title+'">';
                htmlTerms += '<li>';
                htmlTerms += '<span class="qs-option-name">'+termsRes.title+'</span> <span class="amount">'+termsRes.num_results+'</span>';
                htmlTerms += '</li>';
                htmlTerms += '</a>';
            });

            $("#dropdown-search-auto-complete-terms").append(htmlTerms);
            $(".dropdown-search-auto-complete-terms").show();
        }else{
            $(".dropdown-search-auto-complete-terms").hide();
        }
    }
```

Figura 4.4.5-i: Contenido del archivo autocomplete.phtml del módulo del desplegable (7)

Para los resultados de productos, se realizan las mismas comprobaciones que en los términos. Se concatena al código HTML cada enlace de producto con su imagen, nombre y precio y se añade un botón de mostrar todos los resultados siempre que existan resultados, y sino los hay, se oculta el contenedor de productos.

```
if(res.code === 'product'){
    productsCount += res.data.length;
    if(res.data.length > 0){
        $(".dropdown-search-auto-complete-products-see-all").attr("href",res.url);
        $(".dropdown-search-auto-complete-products-see-all-size").text(res.size);
        $(".dropdown-search-auto-complete-products-see-all").show();
        res.data.forEach(function(productRes){
            htmlProducts += '<a href="'+productRes.url+'" title="'+productRes.name+'">';
            htmlProducts += '<li>';
            if(productRes.image){
                htmlProducts += '<div class="qs-option-image">';
                htmlProducts += '<img src="'+productRes.image+'" title="'+productRes.name+'" />';
                htmlProducts += '</div>';
            }
            htmlProducts += '<div class="qs-option-info">';
            if(productRes.name){
                htmlProducts += '<div class="qs-option-title">';
                htmlProducts += productRes.name;
                htmlProducts += '</div>';
            }
            if(productRes.price){
                htmlProducts += '<div class="qs-option-price">'+productRes.price+'</div>';
            }
            }
            htmlProducts += '</div>';
            htmlProducts += '</li>';
            htmlProducts += '</a>';
        });
        $("#dropdown-search-auto-complete-products").append(htmlProducts);
        $(".dropdown-search-auto-complete-products").show();
    }else{
        $(".dropdown-search-auto-complete-products").hide();
    }
}
```

Figura 4.4.5-j: Contenido del archivo autocomplete.phtml del módulo del desplegable (8)

Para mostrar los resultados de términos y productos juntos, se comprueba si hay contenido entre los dos y se despliega el desplegable, si no existen resultados, se muestra el mensaje personalizado en el desplegable.

```
    let sum = termsCount + productsCount;
    if (sum > 0)
    {
        $(".dropdown-search-auto-complete").show();
        $(".no-result").hide();

    }else{
        $(".no-result").show();
        $(".dropdown-search-auto-complete").hide();
    }
    //Muestra el desplegable
    $(".dropdown-search-auto-complete").show();
}
```

Figura 4.4.5-k: Contenido del archivo autocomplete.phtml del módulo del desplegable (9)

5. Pruebas realizadas

5.1. Valores obtenidos

Para realizar las comprobaciones de código y los resultados que se iban obteniendo se ha hecho uso de las variables:

En PHP

`var_dump($valor1,$valor2)`: Muestra el contenido, tipo de una variable o array y su valor.

`echo`: Esta variable seguida de un string o valor se ha utilizado para verificar que el código estaba llegando hasta la línea fijada en la que se lanzó.

`var_export($valor, bool)`: Muestra el contenido de la variable y su valor.

Para debuggear el código (a pesar de que el IDE PhpStorm incorpora la herramienta Debugger) en los módulos y que este genere un log propio (separado de los logs de Magento) se ha usado la clase Psr/Log/LoggerInterface de Magento que junto con la clase Logger de Monolog permite crear la funcionalidad 'DEBUG' y con la clase StreamHandler (también de Monolog) permite indicar en qué archivo de tipo log se guardará la información.

```
$log = new \Monolog\Logger('Debugging');
$log->pushHandler(new \Monolog\Handler\StreamHandler(__DIR__ .
'/debug.log', /Monolog/Logger::DEBUG));
$log->debug(var_export($valor,bool));
```

En Javascript:

console.log(): Desde el DOM del navegador en la ventana console se pueden leer los mensajes que saquen la función console.log() desde el códido de un script.

5.1.1. Módulo de análisis de datos

Para registrar los datos, la funcionalidad se ha basado en el desarrollo del plugin efectuándose al final de la ejecución del método `start()` de la clase SessionManager de Magento. Este método se asegura de saber cuando un cliente registrado accede por primera vez y mantiene en la variable de sesión esa información para que no se almacene la misma información cada vez que el usuario navegue por la tienda online.

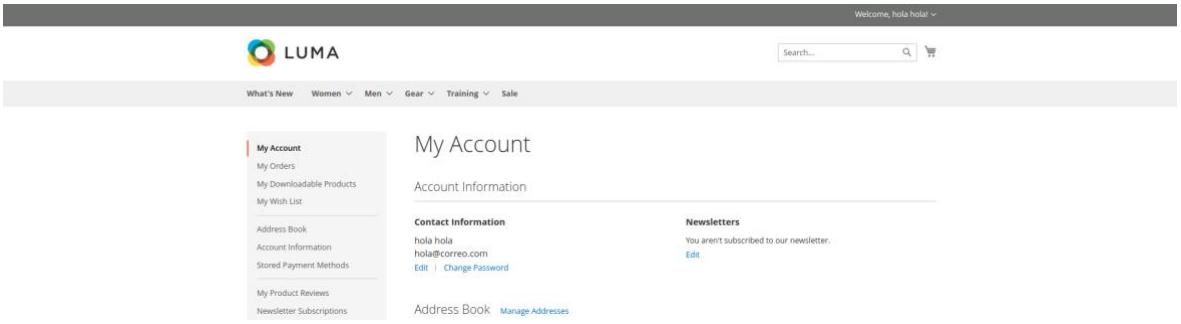


Figura 5.1.1: Inicio sesión del usuario

Desde el programa MySQL Workbench se ha realizado la consulta de ver todas las filas de la tabla que se creó `hiberus_dashboard_table`.

Los registros de los accesos de todos los campos (ip del usuario, código del país, tipo de navegador y tipo de dispositivo) se guardaban satisfactoriamente, pero como el campo de la ip del usuario debía ser único, sólo se hubiera registrado un resultado, para seguir obteniendo datos, se ha modificado manualmente la ip hasta conseguir una serie de registros que permitan realizar el análisis. También se han modificado los campos del `country_code`, `browser` y `device` ya que al hacer el acceso en el entorno local siempre daría los mismos resultados: en `country_code` devolvería '`es`'; en `browser` los dos tipos de navegadores que se tienen instalados `Firefox` o `Chrome` y en device `computer`.

```
1 •  select * from hiberus_dashboard_table;
2
3
```

Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

#	key_id	user_ip	country_code	browser	device	date_time
1	26	188.78.110.01	fr	firefox	mobile	2021-02-16 13:45:22
2	27	188.78.110.02	fr	firefox	tablet	2021-02-16 13:45:22
3	28	188.78.110.03	gb	firefox	computer	2021-02-16 13:45:22
4	29	188.78.110.04	es	firefox	computer	2021-02-16 13:45:22
5	34	188.78.110.05	es	firefox	mobile	2021-02-16 13:47:18
6	35	188.78.110.06	es	firefox	mobile	2021-02-16 13:47:26
7	37	188.78.110.07	fr	safari	mobile	2021-02-16 13:52:37
8	38	188.78.110.08	fr	safari	computer	2021-02-16 14:04:26
9	39	188.78.110.09	gb	safari	mobile	2021-02-16 15:10:02
10	40	188.78.110.10	gb	safari	mobile	2021-02-17 08:13:01
11	41	188.78.110.11	es	chrome	computer	2021-02-17 13:05:42
12	42	188.78.110.12	es	chrome	tablet	2021-02-17 13:08:57
13	43	188.78.110.13	es	chrome	computer	2021-02-17 15:16:51
14	44	188.78.110.14	it	chrome	computer	2021-02-18 07:16:21
15	45	188.78.110.15	es	chrome	computer	2021-02-18 11:55:56
16	46	188.78.110.16	de	edge	computer	2021-02-19 07:13:58
17	47	188.78.110.17	de	edge	computer	2021-02-23 07:14:06
18	99	188.78.110.18	de	chrome	computer	2021-03-01 15:53:41
19	103	188.78.110.19	es	opera	computer	2021-03-02 12:45:34
20	116	188.78.110.29	es	firefox	computer	2021-03-02 16:33:51
*	NULL	NULL	NULL	NULL	NULL	NULL

Figura 5.1.2: Registro de la tabla dashboard

5.2. Administrador

5.2.1. Módulo de redirección

Cuando se realiza una busca una palabra que contiene un resultado en Magento, se redirige a una página de resultados

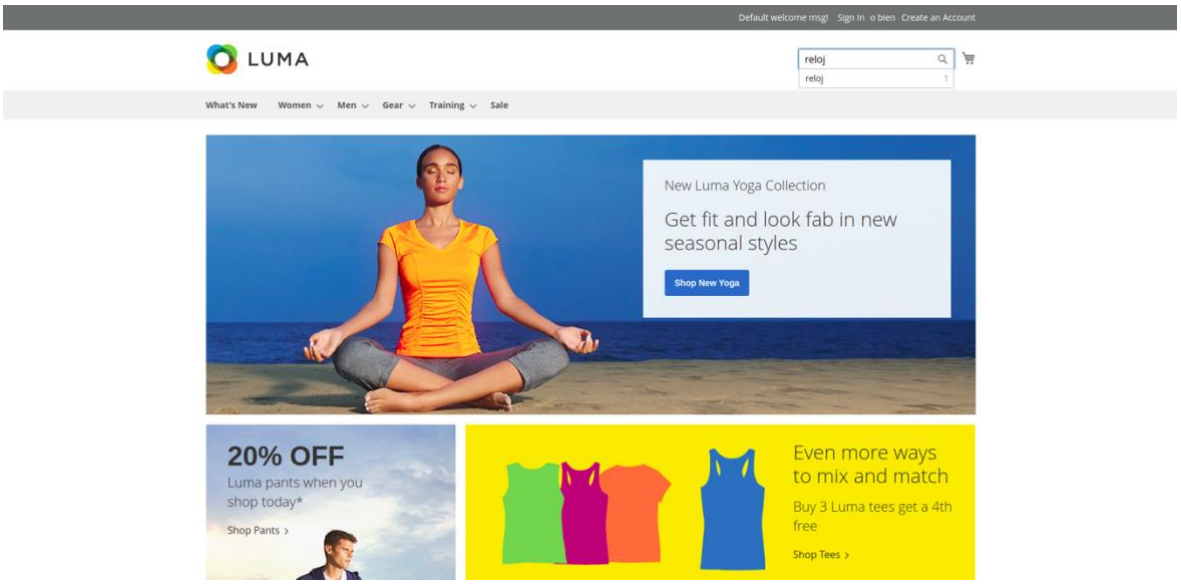


Figura 5.2.1-a: Búsqueda de un producto (Magento)

La palabra ‘reloj’ que sólo tiene un resultado se devuelve en un listado de productos. Para ver el contenido completo de la página del producto habría que acceder al él.

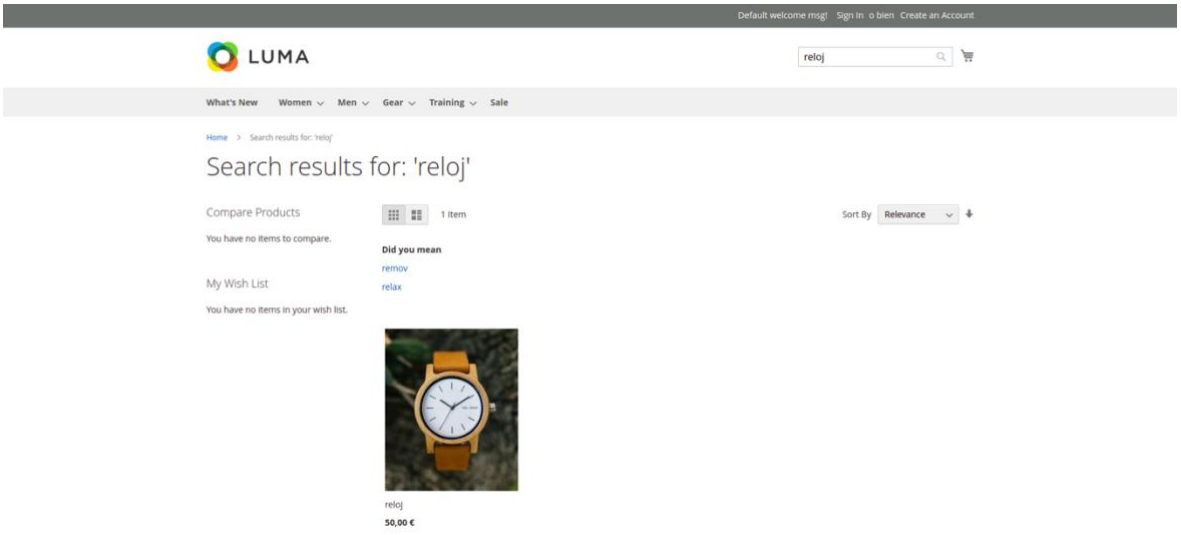


Figura 5.2.1-b: Página con un resultado (Magento)

Desde el menú que se ha desarrollado en el siguiente módulo, se ha creado el acceso a la configuración de los 3 módulos (y los dos dashboard, el de análisis de usuarios y el de las palabras que buscan los usuarios). En la configuración “One Result Configuration” se puede habilitar el módulo de la redirección.

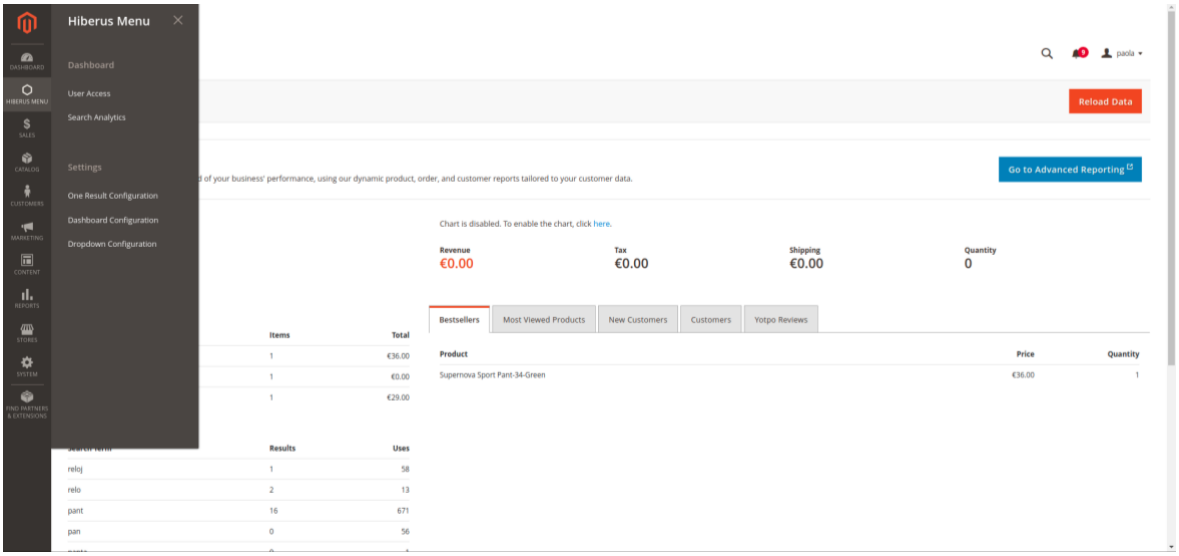


Figura 5.2.1-c: Menú en la vista de administración

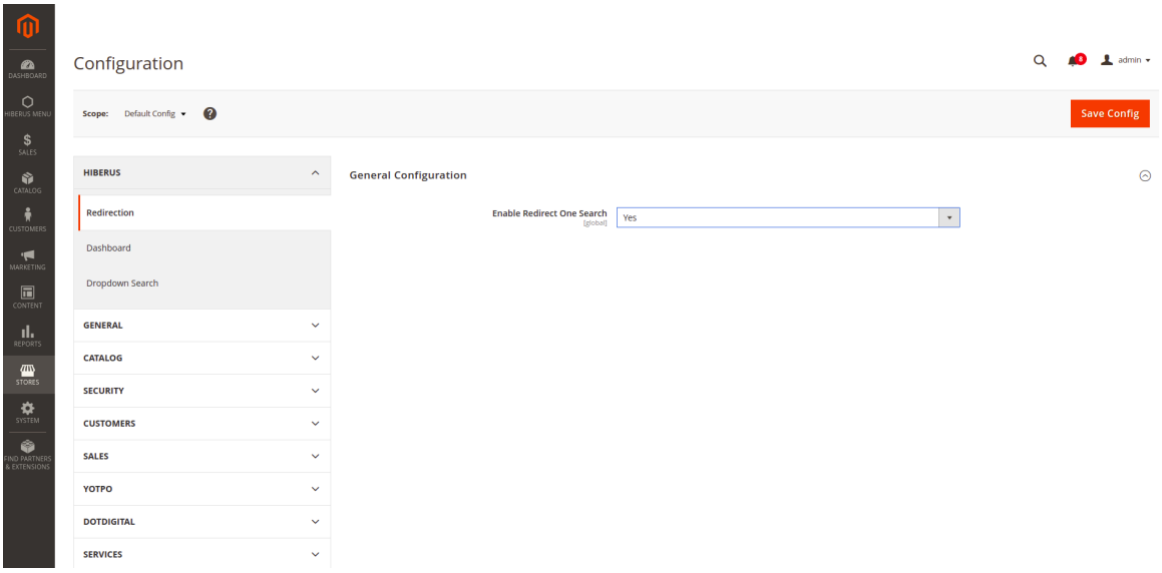


Figura 5.2.1-d: Módulo de redirección habilitado

Ahora que se ha comprobado que el módulo está activado, se vuelve a la vista principal de la tienda online para realizar la misma búsqueda y ver si ha cambiado el funcionamiento.

Como se puede ver en la siguiente imagen, al realizar la búsqueda de la palabra ‘reloj’ que sólo tiene un resultado, el plugin que se ha desarrollado ha conseguido redirigir al usuario a la página del producto y mostrar el mensaje personalizado.

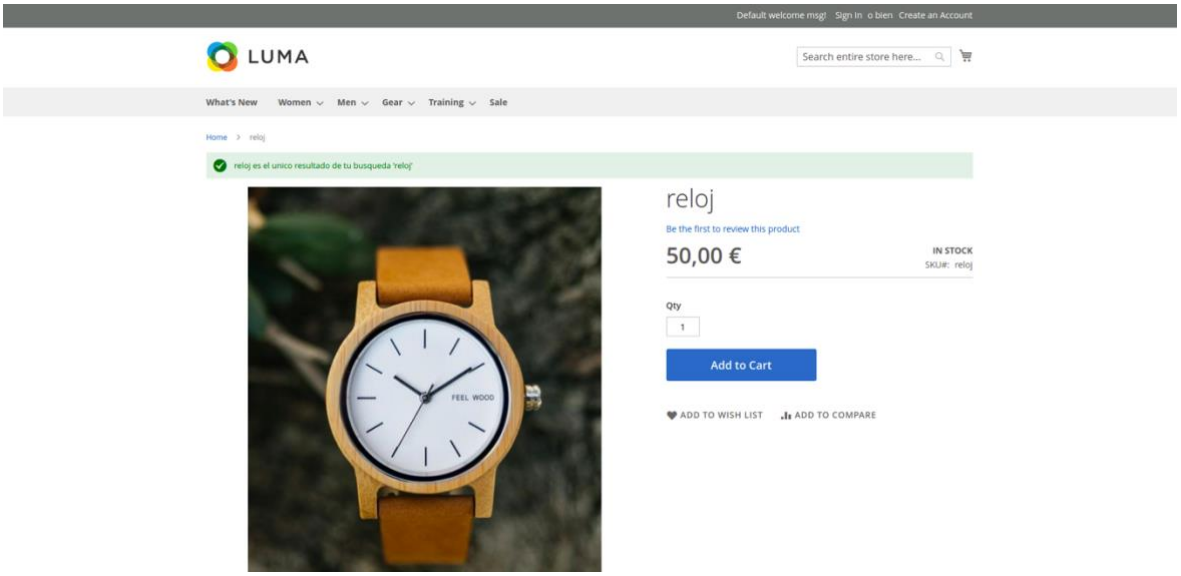


Figura 5.2.1-e: Redirección a la página de producto

5.2.2. Módulo de análisis de usuarios

Si el módulo se encuentra deshabilitado en la configuración, se ha programado de forma que no se muestren las tablas y gráficos.

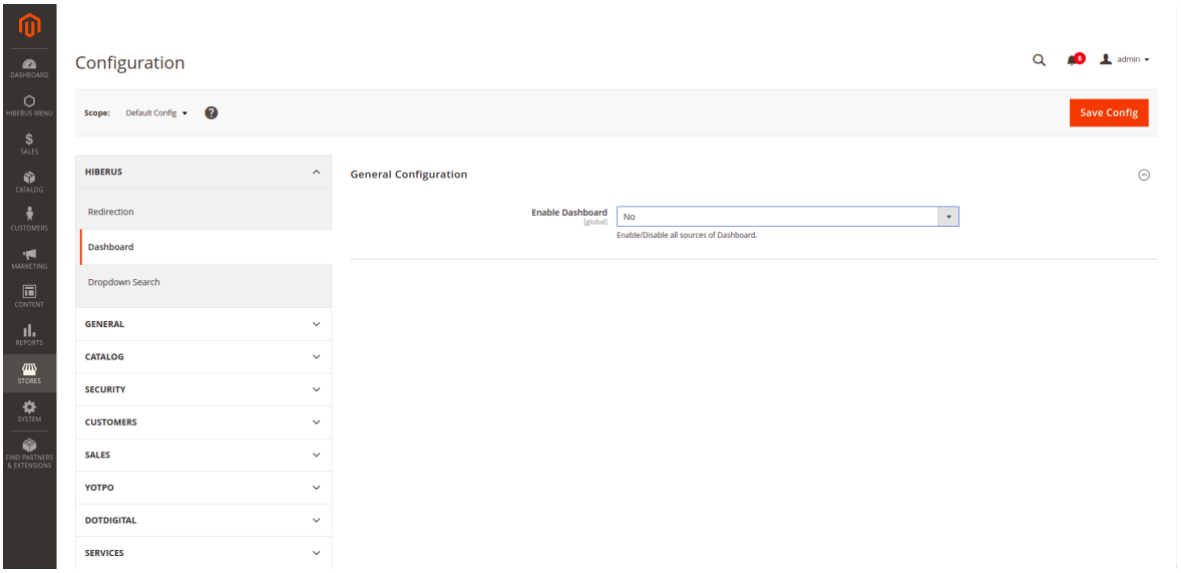


Figura 5.2.2-a: Módulo de análisis de usuario deshabilitado

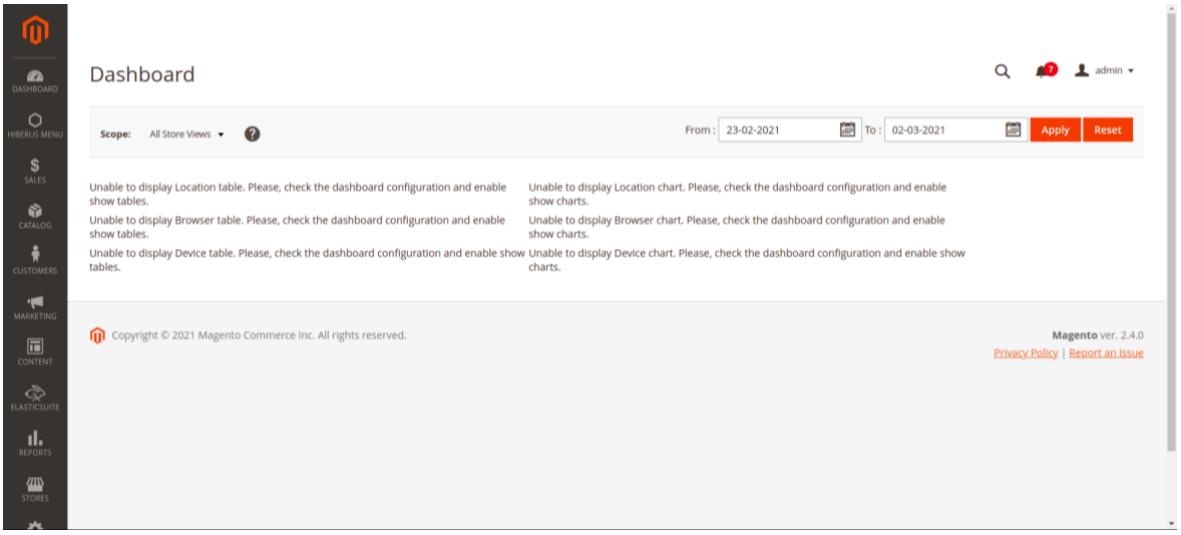


Figura 5.2.2-b: Tablas y gráficos no disponibles

La configuración de este módulo se ha desarrollado con la finalidad de que una vez que se active, se desplieguen las opciones de poder habilitar/deshabilitar la visualización de las tablas o de los gráficos, siendo estas dos visualizaciones independientes.

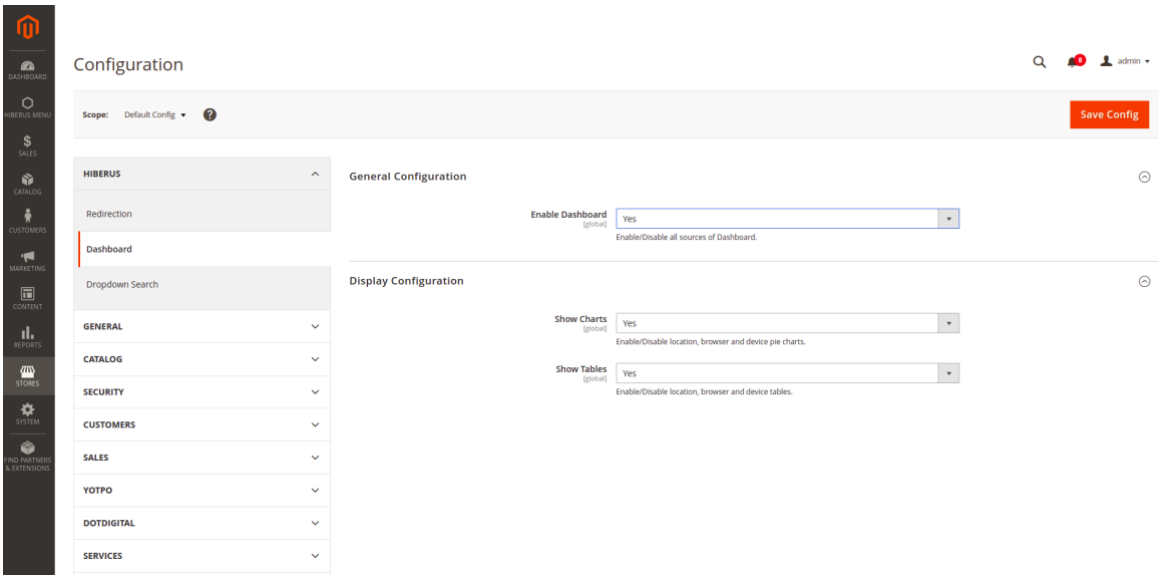


Figura 5.2.2-c: Módulo de análisis de usuarios habilitado

Al acceder al Dashboard de análisis de usuarios (User Access), por defecto, se van a mostrar los registros de todas las filas en las tablas y gráficos. El filtro de fechas se mantendrá en la semana que se accede pero hasta que no se le de al botón ‘Apply’ no se aplicará el filtrado.

Los resultados de las tablas se han programado con Javascript para que se muestren los 3 primeros resultados y con el botón ‘Show more’ se vayan mostrando más resultados de 3 en 3.

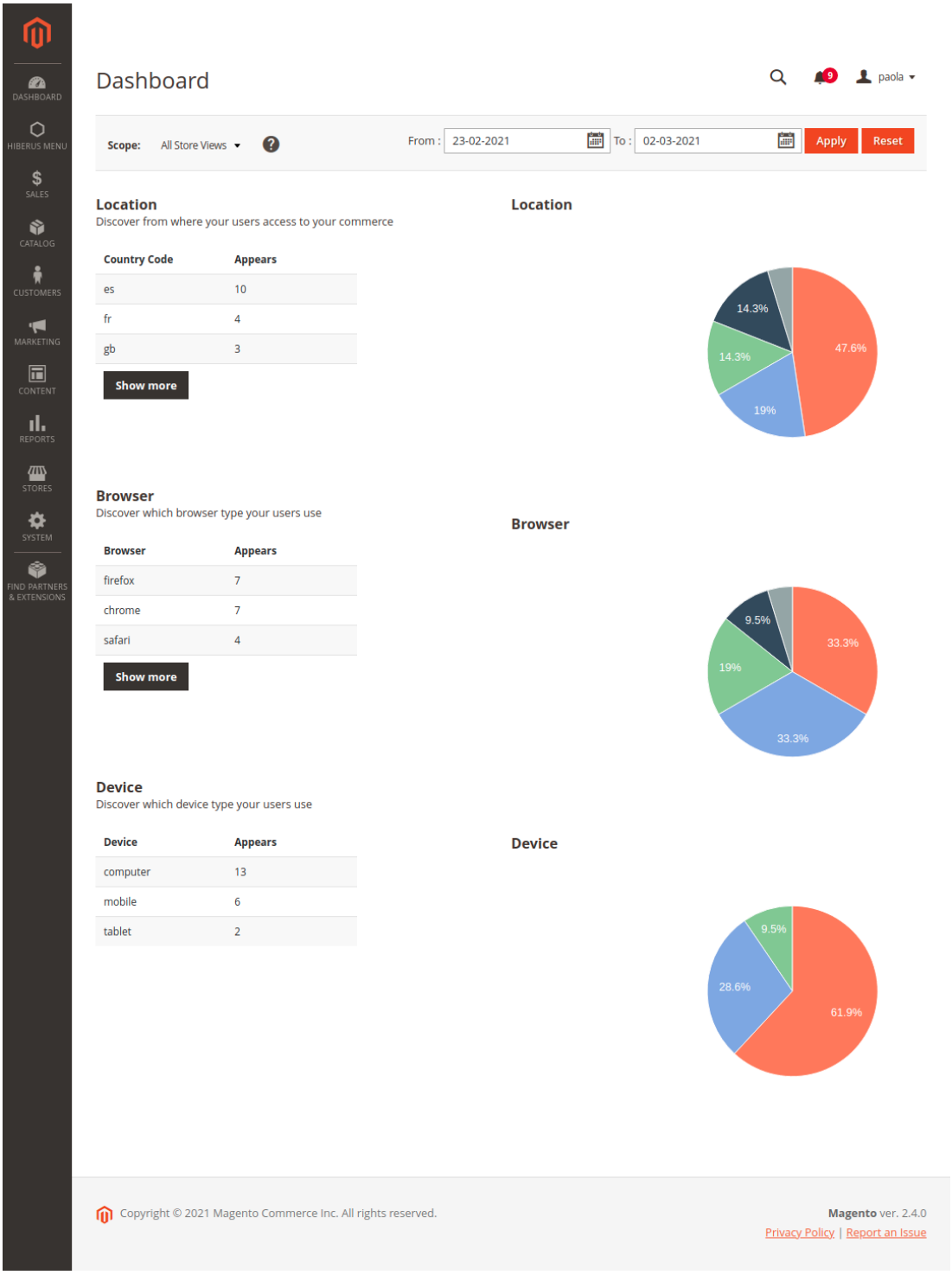


Figura 5.2.2-d: Tablas y gráficos por defecto

Si se selecciona un rango de fechas, es posible filtrar los resultados que se hayan realizado en entre esas fechas. Por ejemplo, se ha seleccionado en la primera semana de Marzo 2021 las fechas desde el 01-03-2021 hasta el 07-03-2021 (no se tienen que seleccionar de forma obligatoría por semanas)

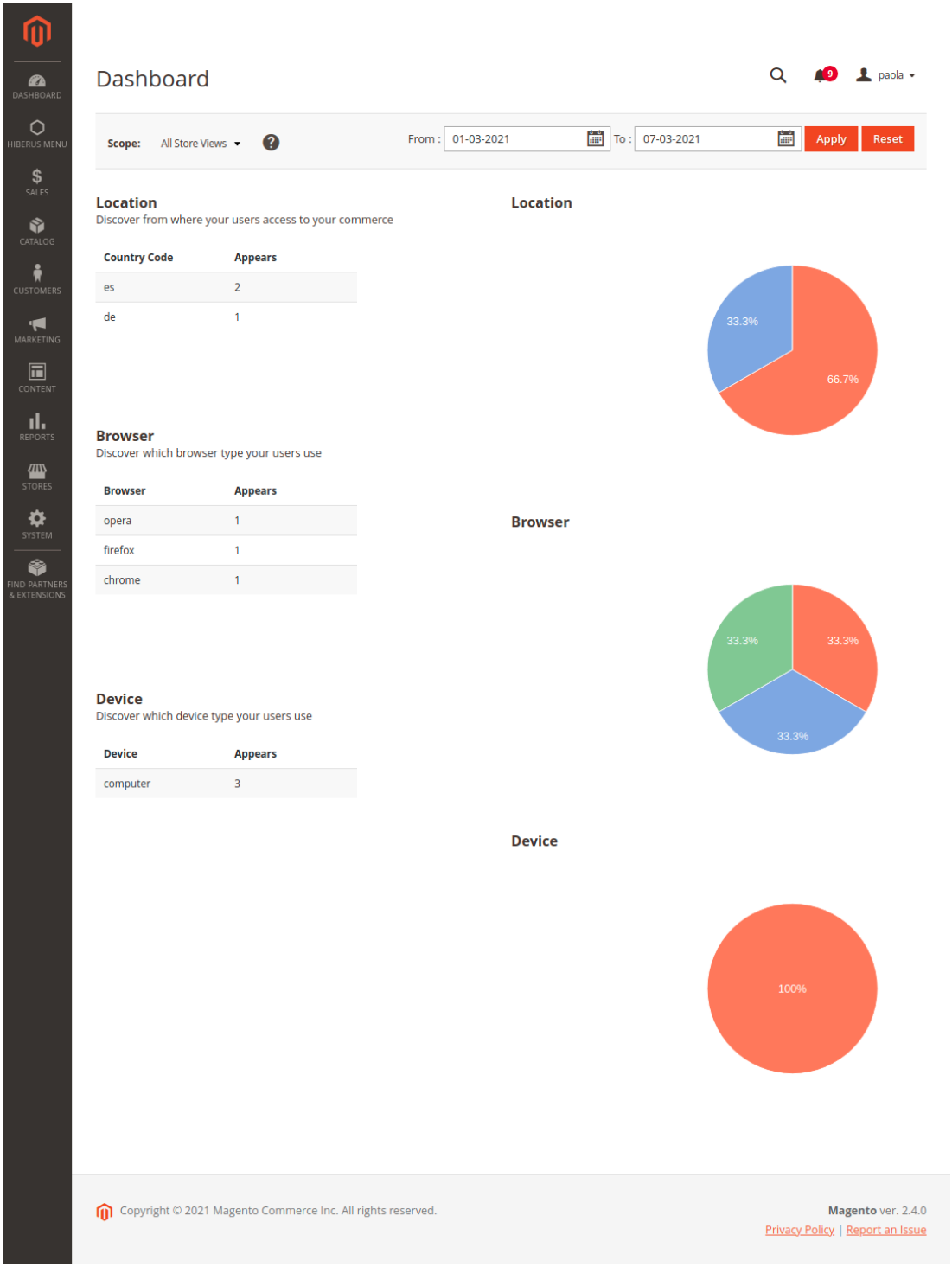


Figura 5.2.2-e: Tablas y gráficos con el filtro de fechas aplicado

5.2.3. Módulo de desplegable de resultados

Cuando se hace una consulta en el buscador de Magento, devuelve un listado de resultados con los términos y el número de resultados.

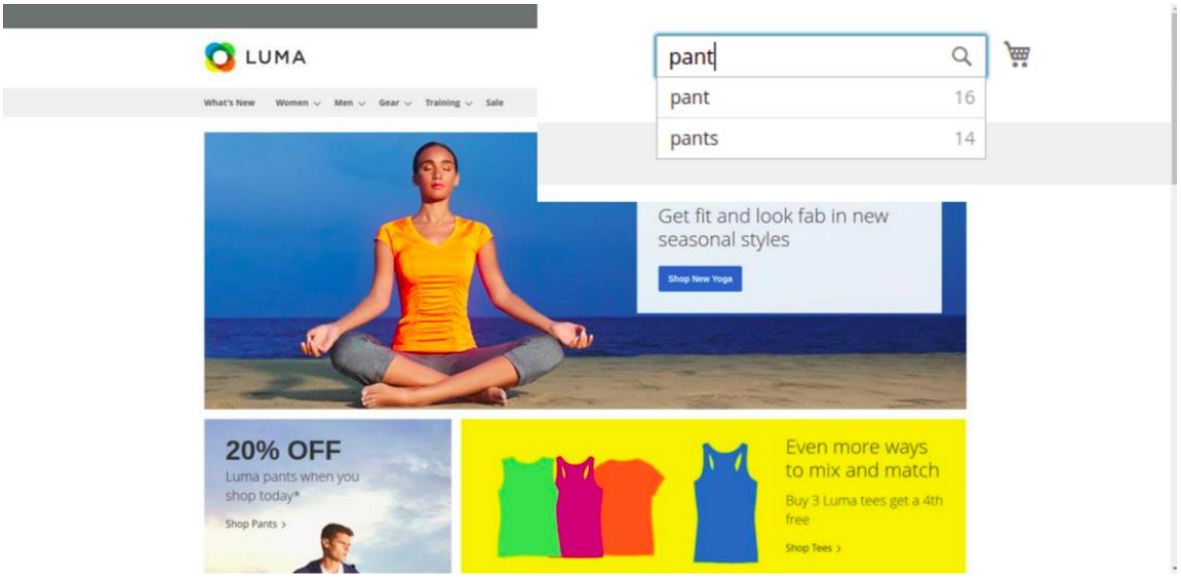


Figura 5.2.3-a: Resultados de una consulta en Magento

Desde la vista de administrador, al habilitar el módulo, se despliegan las opciones de configuración. En la configuración general se puede modificar el tiempo de retraso y los caracteres mínimos para que empiece la búsqueda y el mensaje personalizado en caso de que no haya resultados. También se despliegan las demás configuraciones.

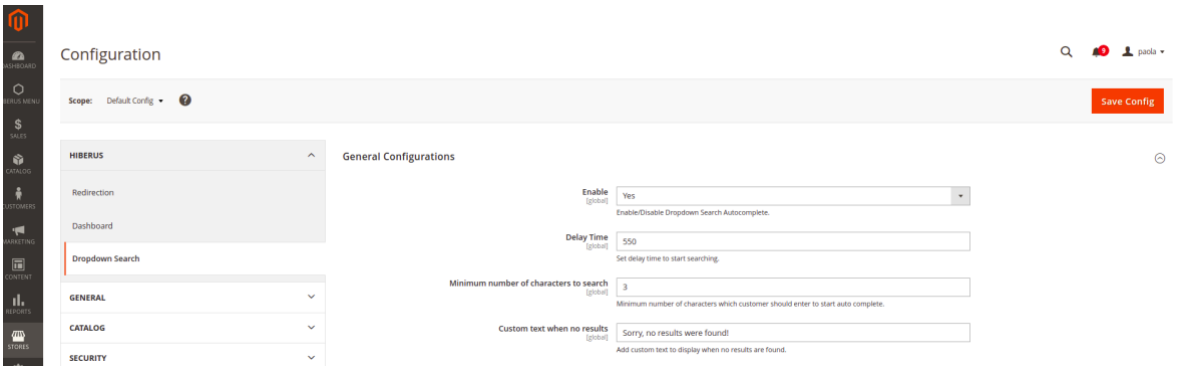


Figura 5.2.3-b: Configuración general del módulo del desplegable de resultados

En la configuración de términos y productos se habilita el campo para activar/desactivar que se muestren en el desplegable de resultados. Asimismo, en la configuración de productos se habilita el campo que permite modificar el número de resultados de producto que se muestren.

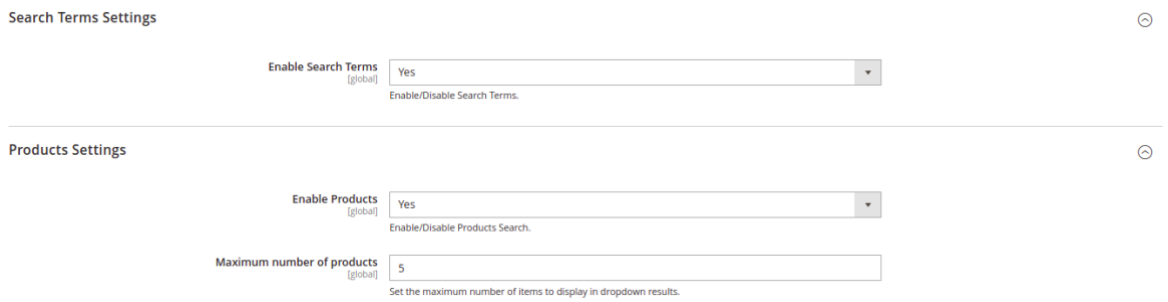


Figura 5.2.3-c: Configuración de términos y productos del módulo del desplegable de resultados

Finalmente se muestra el campo que habilita/deshabilita la visualización del dashboard de las palabras más usadas en las consultas.

Figura 5.2.3-d: Configuración del dashboard del módulo del desplegable de resultados

Al tener todas las opciones de configuración habilitadas y estableciendo que el número de resultados de productos sean 5. Cuando se realiza una consulta en el buscador, este ofrece los resultados con los términos y 5 productos con su imagen, nombre del producto y precio.

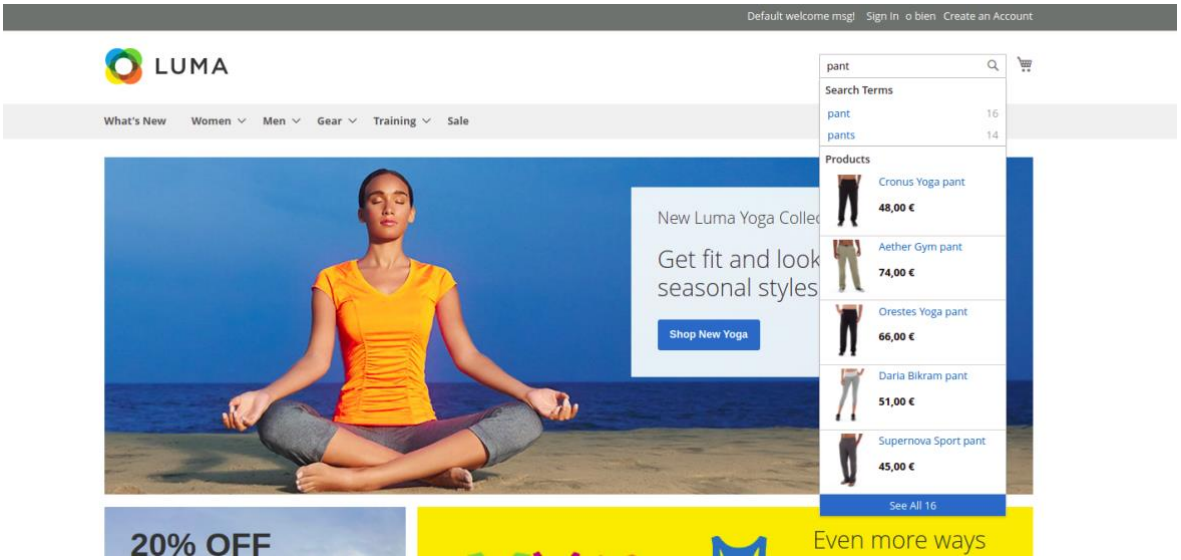


Figura 5.2.3-e: Resultado de una consulta con el módulo de desplegable de resultados

Por otra parte, se recogen los resultados de las consultas que se hacen en el buscador y se muestran en el dashboard en el panel de administración (en la imagen 5.2.1-c se puede ver su acceso directo en “Search Analytics” desde el menú que se creó).

En este dashboard existe en la parte superior un botón que actualiza los datos y se visualizan los valores de las consultas que dan resultados y las que no. También, se aportan tablas con un listado de los 15 términos más buscados (con resultados y sin resultados) y debajo un botón que permite la descarga en formato .csv de estos resultados.

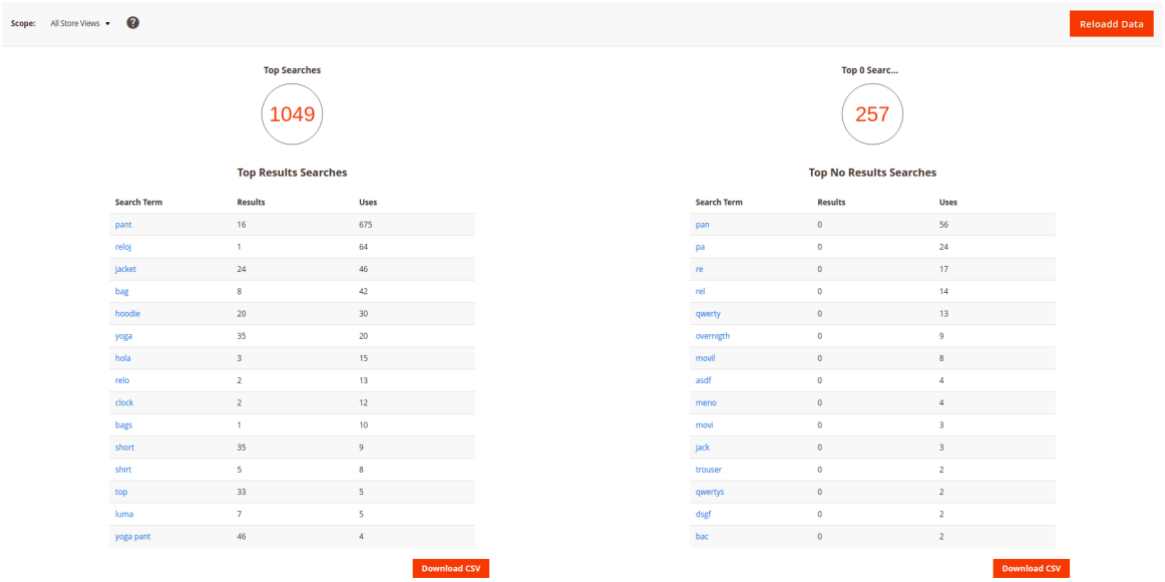


Figura 5.2.3-f: Tablas y valores de los resultados de las consultas

6. Conclusiones

6.1. Conclusión

El desarrollo de módulos personalizados permite ampliar las características de una tienda online agregando nuevas funcionalidades. Brinda una experiencia única a los usuarios que la visiten y ofrece un panel de administración que da acceso a todos los módulos y sus funciones.

La curva de aprendizaje de Magento ha sido bastante elevada. Los factores que considero que me han influenciado han sido la poca experiencia como programadora (puesto que con conocer el funcionamiento de los lenguajes de programación parecía no ser suficiente para avanzar) y el completo desconocimiento de esta herramienta ya que he tenido que aprender cómo funcionaba y qué hacía cada fichero.

Se trata de una plataforma tan completa (y compleja a la vez) que el primer encuentro me produjo una total incertidumbre de no saber por dónde empezar, cosa que fue solucionándose leyendo (mucho) la documentación oficial de Magento, foros y dudas puntuales a algún compañero de la compañía para la que se realizó los módulos.

Tras la realización de este trabajo, considero que he prosperado potencialmente mis aptitudes de programación web, dado que he mejorado mi nivel de programación en los lenguajes que ya conocía.

He aplicado conceptos de bases de datos para realizar las consultas de tablas y desarrollado el lenguaje PHP ya que durante la carrera aprendí a gestionar la comunicación de la aplicación con la base de datos y con este trabajo he sido capaz de crear una comunicación de datos entre bloques y plantillas y cambiar comportamientos de funciones. He aprendido el lenguaje LESS que es una evolución del CSS y con Javascript me ha permitido darle funcionalidad dinámica a las páginas webs de la tienda online (como el uso de botones). Gracias al uso de XML he podido declarar dependencias de clases, configuraciones y fijar componentes en una página.

En relación a lo expuesto, me ha aportado una gran satisfacción personal y profesional poder haber conseguido terminar los objetivos iniciales que me había propuesto (y no haber desarrollado antes nada parecido) e ir comprendiendo cada día más a esta plataforma. Me gustaría seguir aprendiendo su funcionamiento ya que me resulta una plataforma bastante atractiva por todo lo que tiene por ofrecer y mejorar profesionalmente para poder dedicarme a ello muchos años más.

6.2. Trabajos futuros

En cuanto a los posibles desarrollos que se pueden realizar en el futuro sobre este trabajo, a pesar de que se ha intentado seguir las buenas prácticas de programación y de la documentación de Magento y, gracias a la experiencia que sigo desarrollando en este sector, he podido comprobar que puedo mejorar la declaración de nombres de archivos o identificaciones y, en los archivos de diseño .less el uso de variables ya que simplifica la repetición del código. También poder darle un diseño responsive a las páginas creadas de dashboard y al desplegable de resultados, debido a que su desarrollo ha sido sólo con la vista en web.

Bibliografía de páginas webs

[1] Bello, E. (2020, 29 octubre). ¿Cómo ha afectado el Covid-19 al aumento de eCommerce? Thinking for Innovation. https://www.iebschool.com/blog/aumento-ecommerce-e-commerce/

[2] ¿Cómo ha cambiado la necesidad de tener un ecommerce? (2020, 5 octubre). Financial Food. https://financialfood.es/como-ha-cambiado-la-necesidad-de-tener-un-ecommerce/

[3] R. (2017, 4 diciembre). La importancia de tener un buen buscador en un ecommerce. Comercio 360 Galicia. https://comercio360.gal/es/a-importancia-de-tener-un-bo-buscador-nun-ecommerce/

[4] Villalonga, A. (2020, 5 mayo). ¿Cuál es el precio hora de un programador? Kaira. https://kaira.es/cual-es-el-precio-hora-de-un-programador/

[5] Risueño, P. (2020, 15 septiembre). Cuánto cuesta una tienda online | 4webs.es. 4webs. https://www.4webs.es/blog/cuanto-cuesta-una-tienda-online

[6] Podríamos definir un ecommerce… (Bello, E. (2021, 26 mayo). ¿Qué es eCommerce y cómo crear tu propio comercio electrónico? Thinking for Innovation. https://www.iebschool.com/blog/comercio-online-ecommerce/)

[7] B2B: Business to Business… (A. (2021, 8 julio). Ecommerce para principiantes: Tipos y tendencias. Agencia ékiba. https://www.agenciaekiba.com/marketing-online/ecommerce-para-principiantes-tipos-y-tendencias/)

[8] Además de estas formas… (Gamella, N. (2021, 1 marzo). Qué es un e-commerce: tipos de negocios y pasos para crearlo. Doofinder. https://www.doofinder.com/es/blog/que-es-e-commerce)

[9] S-Commerce… **(**O. (2015, 15 septiembre). Tipos de eCommerce. Observatorio eCommerce. https://observatorioecommerce.com/tipos-de-ecommerce/)

[10] Shopify es una de las… (Las 5 mejores plataformas para crear tu e-commerce. (s. f.). Intelequia. Recuperado 9 de julio de 2021, de https://intelequia.com/blog/post/2061/las-5-mejores-plataformas-para-crear-tu-e-commerce9

[11] PrestaShop es una solución… (de la Hera, C. (2021, 27 abril). Top 20: los mejores CMS para eCommerce (2021). Marketing 4 Ecommerce - Tu revista de marketing online para e-commerce. https://marketing4ecommerce.net/mejores-cms-para-ecommerce/#woo)

[12] Ecommerce market share, websites and contacts - Wappalyzer. (s. f.). wappalyzer. Recuperado 9 de julio de 2021, de https://www.wappalyzer.com/technologies/ecommerce/

[13] Borges, E. (2019, 12 junio). Magento: Características, Ventajas y Desventajas. Infranetworking. https://blog.infranetworking.com/magento-caracteristicas-ventajas-desventajas/

[14] ¿Qué es Elasticsearch? (s. f.). Elastic. Recuperado 9 de julio de 2021, de https://www.elastic.co/es/what-is/elasticsearch

[15] Desde 2014 Doofinder… (Galeano, S. (2020, 1 octubre). Doofinder abre una nueva etapa estrenando identidad visual, web y herramientas para sus clientes. Marketing 4 Ecommerce - Tu revista de marketing online para e-commerce. https://marketing4ecommerce.net/doofinder-abre-una-nueva-etapa-estrenando-identidad-visual-web-y-herramientas-para-sus-clientes/)

[16] Doofinder. (2021, 31 marzo). Doofinder ▷ El Mejor Buscador Interno para Web y eCommerce. https://www.doofinder.com/es/

[17] Empathy.co es una compañía… (Reina, C. (2020, 5 mayo). Así es la exitosa empresa asturiana que está detrás de los buscadores online de Inditex o Kroger. elEconomista.es. https://www.eleconomista.es/status/noticias/10523929/05/20/Asi-es-la-exitosa-empresa-asturiana-que-esta-detras-de-los-buscadores-online-de-Inditex-o-Kroger.html)

[18] Technology. (s. f.). Empathy.Co. Recuperado 9 de julio de 2021, de https://empathy.co/technology/
[19] HTML (lenguaje… (Platzi: Cursos online profesionales de tecnología. (s. f.). Platzi. Recuperado 9 de julio de 2021, de https://platzi.com/clases/1050-programacion-basica/5104-que-es-htmlcssjs/)
[20] Team, T. C. L. (s. f.). Getting started | Less.js. Lesscss. Recuperado 9 de julio de 2021, de https://lesscss.org
[21] PHP es un lenguaje de… (de Souza, I. (2021, 12 febrero). Descubre qué es el lenguaje de programación PHP y en qué situaciones se hace útil. Rock Content - ES. https://rockcontent.com/es/blog/php/)
[22] XML es el acrónimo de… (de Souza, I. (2021b, febrero 12). XML: ¿qué es y para qué sirve este lenguaje de marcado? Rock Content - ES. https://rockcontent.com/es/blog/que-es-xml/)
[23] MySQL Workbench es… (ProgSoft.net. (2019, 16 agosto). MySQL Workbench Alternativas y software similar. https://progsoft.net/es/software/mysql-workbench)
[24] PhpStorm: el IDE rápido e inteligente para programación en PHP de. (2021, 2 junio). JetBrains. https://www.jetbrains.com/es-es/phpstorm/
[25] Monster Digital Agency. (2021, 25 mayo). Metodología Agile: Desarrollo y gestión de proyectos. Epitech España. https://www.epitech-it.es/metodologia-agile/
[26] La metodología Agile… (C. (2019b, julio 18). 12 principios de la metodología agile en el desarrollo de proyectos. cognodata. https://www.cognodata.com/blog/principios-metodologia-agile-desarrollo-proyectos/)
[27] Las metodologías ágiles… (E. (2021b, abril 29). Origen de las metodologías ágiles | EnFormación Alicante. Enformacion. https://www.enformacion.es/origen-de-las-metodologias-agiles/)
[28] Modelo Vista Controlador… (Modelo vista controlador (MVC). Servicio de Informática ASP.NET MVC 3 Framework. (s. f.). ua. Recuperado 9 de julio de 2021, de https://si.ua.es/es/documentacion/asp-net-mvc-3/1-dia/modelo-vista-controlador-mvc.html)
[29] Composer. (s. f.). Composer. Recuperado 9 de julio de 2021, de https://getcomposer.org/download/
[30] La tecnología Docker… (¿Qué es Docker? (s. f.). redhat. Recuperado 9 de julio de 2021, de https://www.redhat.com/es/topics/containers/what-is-docker)
[31] Docker se instala en… (Contenedores de Docker | ¿Qué es Docker? | AWS. (s. f.). Amazon Web Services, Inc. Recuperado 9 de julio de 2021, de https://aws.amazon.com/es/docker/)
[32] Magento_SampleData | Adobe Commerce Developer Guide. (2020, 16 junio). Devdocs. https://devdocs.magento.com/guides/v2.4/mrg/ce/SampleData.html
[33] S. (s. f.). serbanghita/Mobile-Detect. GitHub. Recuperado 9 de julio de 2021, de https://github.com/serbanghita/Mobile-Detect
[34] API Documentation - ipstack. (s. f.). ipstack. Recuperado 9 de julio de 2021, de https://ipstack.com/documentation
[35] Son agentes de datos ligeros… (Beats: Agentes de datos para search. (s. f.). Elastic. Recuperado 9 de julio de 2021, de https://www.elastic.co/es/beats/)
[36] El debugger o mejor dicho… (¿Que es debuggear? (s. f.). blogspot. Recuperado 9 de julio de 2021, de http://java-white-box.blogspot.com/2012/05/eclipse-que-es-debuggear-que-es-un.html)
[37] El mapeo de datos… (Fatima, N. (2021, 29 junio). ¿Qué es el mapeo de datos? Herramientas, tutoriales y plantillas de asignación de datos. Astera. https://www.astera.com/es/type/blog/understanding-data-mapping-and-its-techniques/)

[38] Una API es un conjunto… (¿Qué es una API? (s. f.). redhat. Recuperado 9 de julio de 2021, de https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces)
[39] Una API de transferencia… (¿Qué es una API de REST? (s. f.). redhat. Recuperado 9 de julio de 2021, de https://www.redhat.com/es/topics/api/what-is-a-rest-api)
[40] Un dashboard o cuadro… (Abellán, E. (2020, 16 enero). Qué es un dashboard de negocios y cuáles sus beneficios. wearemarketing. https://www.wearemarketing.com/es/blog/que-es-un-dashboard-de-negocios-y-cuales-sus-beneficios.html)
[41] Un framework es… (Qué es Framework. (s. f.). Arimetrics. Recuperado 9 de julio de 2021, de https://www.arimetrics.com/glosario-digital/framework)
[42] Se le conoce como… (I. (2020, 1 octubre). Front End vs Back End. Viewnext. https://www.viewnext.com/front-end-vs-back-end/)
[43] UNO. (2020, 19 junio). INFORGES. https://www.inforges.es/post/ecommerce-en-tiempos-del-covid-19
[44] M. (2018, 18 julio). ¿Por qué deberías integrar un CMS en tu e-commerce? •. El VALOR DE LA ENTREGA. https://www.elvalordelaentrega.com/por-que-deberias-integrar-un-cms-en-tu-e-commerce/

Anexo I. Instalación de Magento

En esta sección del proyecto se va a mostrar cómo se ha instalado la plataforma Magento versión 2.4 en Ubuntu 20.4 utilizando Docker, Composer, Git y la línea de comandos en Terminal. Todos los pasos de la instalación van a ser llevados a cabo usando el Terminal:

1. Instalar git

Primero, para asegurarse que los paquetes que se descarguen sean la última versión, se necesita actualizar los paquetes ejecutando el comando:

```
sudo apt update
```

Y lanzar el siguiente comando para instalar git:

```
sudo apt install git
```

2. Instalar PHP

El framework de Magento está escrita en PHP, por lo que es necesario tener el lenguaje en el entorno local. Se procede a instalar la versión de PHP 7.4:

```
sudo apt -y install php7.4
```

Para comprobar que se ha instalado correctamente y ver su versión se lanza el siguiente comando:

```
php --version
```

Como se puede comprobar el paquete de PHP ha sido instalado satisfactoriamente:

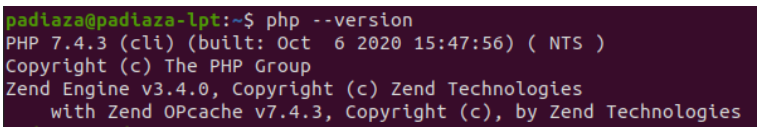


Figura Anexo I-a: Versión Php

3. Descargar Composer

Composer es una herramienta que gestiona las dependencias de PHP y componentes en Magento. Permite declarar las librerías de las que depende el proyecto y se encarga de la instalación y actualización de estos paquetes. Para obtener Composer se deben seguir los pasos de su página web [29]:

```
php -r "copy('https://getcomposer.org/installer', 'composer-setup.php');"
```

```
php -r "if (hash_file('sha384', 'composer-setup.php') === '756890a4488ce9024fc62c56153228907f1545c228516cbf63f885e036d37e9a59d27d63f46af1d4d07ee0f76181c7d3') { echo 'Installer verified'; } else { echo 'Installer corrupt'; unlink('composer-setup.php'); } echo PHP_EOL;"

php composer-setup.php

php -r "unlink('composer-setup.php');"
```

Como se puede observar, se ha instalado correctamente y se tiene la versión 2.0.8.

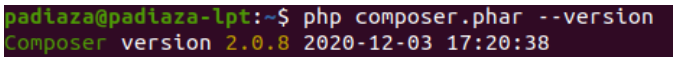


Figura Anexo I-b: Versión Composer

4. Instalar docker

Docker es un proyecto software open source que permite usar contenedores de manera ligera y modular.
"La tecnología Docker usa el kernel de Linux y las funciones de este, como cgroups y namespaces, para segregar los procesos, de modo que puedan ejecutarse de manera independiente. El propósito de los contenedores es esta independencia: la capacidad de ejecutar varios procesos y aplicaciones por separado para hacer un mejor uso de su infraestructura y, al mismo tiempo, conservar la seguridad que tendría con sistemas separados.

Las herramientas del contenedor, como Docker, ofrecen un modelo de implementación basado en imágenes. Esto permite compartir una aplicación, o un conjunto de servicios, con todas sus dependencias en varios entornos. Docker también automatiza la implementación de la aplicación (o conjuntos combinados de procesos que constituyen una aplicación) en este entorno de contenedores." [30]

"Docker se instala en cada servidor en el que se desee ejecutar contenedores y proporciona un conjunto sencillo de comandos que puede utilizar para crear, iniciar o detener contenedores." [31]

```
apt install docker.io
```

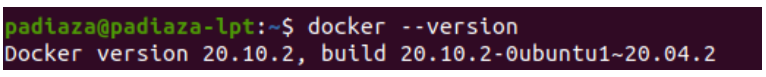


Figura Anexo I-c: Versión Docker

5. Instalar docker-compose

Docker Compose es una herramienta que permite simplificar el uso de Docker. A partir de archivos YAML es mas sencillo crear contendores, conectarlos, habilitar puertos, volumenes, etc

```
Docker-compose up -d
```

```
padiaza@padiaza-lpt:~$ docker-compose --version
docker-compose version 1.25.0, build unknown
```

Figura Anexo I-d: Versión docker-compose

6. Instalar dockergento

Esta herramienta ha sido desarrollada en la empresa Hiberus por un compañero del departamento de Magento. Consiste en simplificar las ordenes de docker-compose y optimizar su rendimiento tanto en el momento de ejecutar la orden en el terminal como al recargar la página. También contiene el servidor web (ngnix), el servidor de base de datos (mysql) y la imagen de la última versión de PHP.

Para instalar dockergento primero hay que clonar el repositorio en la máquina local, añadirlo al $path .

```
git clone https://github.com/ModestCoders/magento2-dockergento.git
```

```
sudo ln -s $(pwd)/magento2-dockergento/bin/dockergento /usr/local/bin
```

Para comprobar que funciona, se tiene que abrir una nueva ventana en el terminal y ejecutar el siguiente comando:

```
which dockergento
```

Esta herramienta permite sustituir el principio de los comandos de magento como por ejemplo, el de borrar cache `php bin/magento clean:cache` por el uso únicamente de `dockergento clean:cache`.

Cada vez que se quiera iniciar un proyecto se escribirá el comando `dockergento start`, por el contrario cuando se quiera finalizar los servicios se escribirá `dockergento stop` para apagar los contenedores.

Los comandos que integra esta herramienta se aprecian en la imagen Anexo I-e.

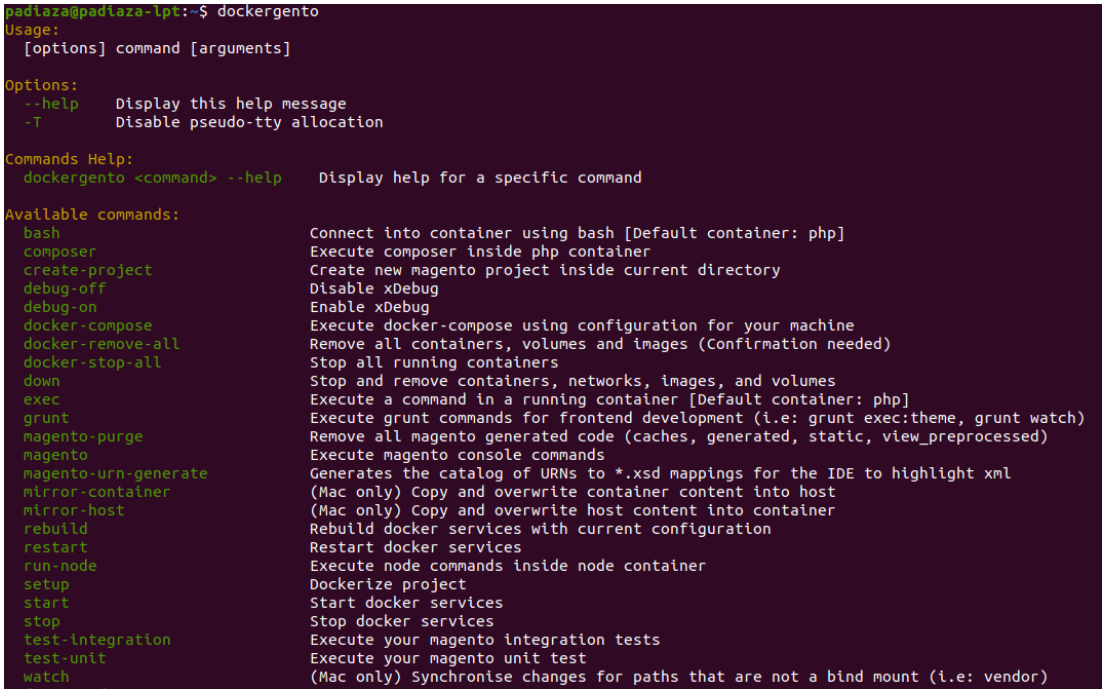


Figura Anexo I-e: Comandos dockergento

7. Instalar el código de Magento

La versión que se ha instalado es la de Community Edition, es decir, la Open Source y se ha hecho mediante la herramienta composer:

```
composer create-project --repository-url=https://repo.magento.com/
magento/project-community-edition=2.4.0 .
```

Durante el proceso de instalación se van a solicitar las claves de autenticación pública y privada que se han configurado en el Magento Marketplace.

Una vez se ha descargado Magento, se instala también datos de muestra (sample data) para que el contenido de la página creada incluya un catálogo de muestra con más de 250 productos, categorías, reglas de precios promocionales, páginas CMS, banners, etc. Técnicamente, los datos de muestra son un conjunto de módulos de Magento. [32]

Como la instalación de Magento se ha realizado a través de composer, se descargará con el comando:

```
bin/magento sampledata:deploy
```

Antes de finalizar la instalación, se accede al archivo /etc/hosts con nano para añadir la url del proyecto:

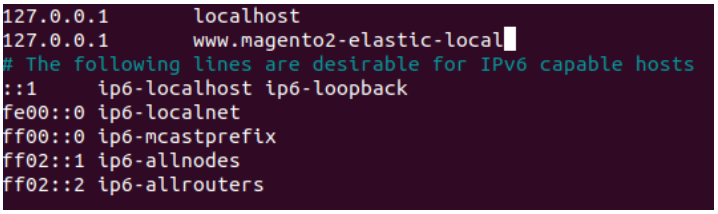


Figura Anexo I-f: Contenido /etc/hosts

Con el último comando se va a terminar el proceso de instalación declarando el nombre de la url, la base de datos, el nombre del usuario y contraseña, el usuario del administrador y contraseña, el lenguaje…

```
dockergento magento setup:install
--base-url=http://www.magento2-elastic.local/
--db-host=db
--db-name=magento
--db-user=magento
--db-password=magento
--admin-firstname=paola
--admin-lastname=diaz
--admin-email=padiaza@hiberus.com
--admin-user=paola
--admin-password=misco1234
--language=es_ES
--currency=EUR
--timezone=Europe/Madrid
--use-rewrites=1
--backend-frontname="admin"
--elasticsearch-host=elasticsearch
```

Cuando la instalación termine, se debe ejecutar el siguiente comando para actualizar el software:

```
dockergento magento setup:upgrade
```

Anexo II. Código de la vista de administración y frontend

En este apartado se va a explicar cómo se ha programado las funcionalidades de la vista del panel de administración y el desarrollo del diseño del software en el que los usuarios interactúan (frontend).

El primer módulo (redirección) va a contener el desarrollo únicamente de la vista de administración y en los otros dos módulos, los desarrollos de la vista de administración y frontend. En el módulo de análisis de usuarios el desarrollo frontend se va a implementar en los diseños de las tablas y, en el módulo del desplegable de resultados el desarrollo frontend va a ir tanto a los diseños del dashboard (en la vista de administración) como para el estilo del desplegable, cuya intención es hacerlo lo más similar al tema de Magento.

Para comprender que es la vista de administración, es la parte en la que el gestor de la tienda online tiene acceso a la configuración para realizar cambios según sus necesidades y comprobar en tiempo real los usuarios, ventas, stock...

Lo que se va a implementar en la administración es la incorporación de las opciones de configuración de habilitar/deshabilitar los módulos y opciones que tengan, crear un menú para tener un acceso más directo a dicha configuración y la creación de páginas en las que se ha creado los dashboard de análisis de usuarios y el de las palabras que usan los usuarios.

Para acceder al panel de administración se hace mediante la url que se establecido en la carpeta /etc/hosts, www.magento2-elastic-local/ seguido de admin. Las credenciales para acceder a esta vista, se han creado en el comando de la instalación.

En el panel de administrador y en Stores -> Settings -> Configuration se accede a la página de configuración de Magento.

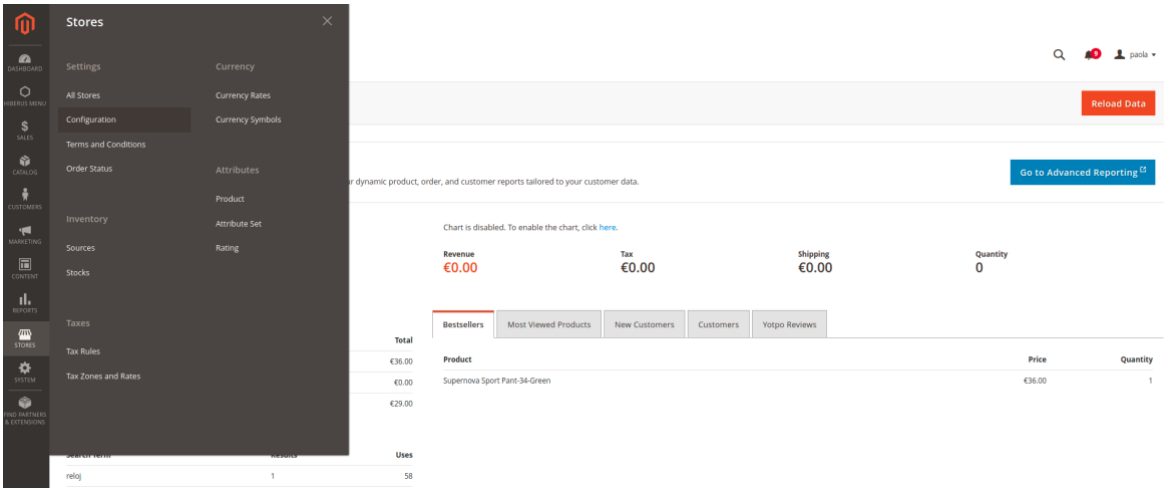


Figura Anexo II-a: Menú de configuración de Magento

La página de configuración se personaliza en el archivo system.xml que contiene los siguientes elementos:

<tab>: Hace referencia a una pestaña nueva o existente en la configuración. Los tres módulos se han creado bajo el mismo <tab>. Tiene el atributo obligatorio id y los atributos opcionales translate, type, sortOrder y class

<section>: Se encuentran dentro de cada <tab>. Tiene el atributo obligatorio id y los opcionales translate, sortOrder, showInDefault, showInStore y showInWebsite entre otros. En cuanto a los hijos label, tab, resource, group…

<group>: Se encuentra dentro del nodo <section>. Tiene el atributo obligatorio id y algunos de los opcionales son translate, type, sortOrder, showInDefault, showInStore y showInWebsite. Puede tener los hijos label, field y depends principalmente.

<fields>: Se usa dentro de la etiqueta <group> para definir la configuración. Puede ser del tipo text, textarea, select, multiselect… Los atributos que se han usado son los mismos que los mencionados en las anteriores etiquetas.

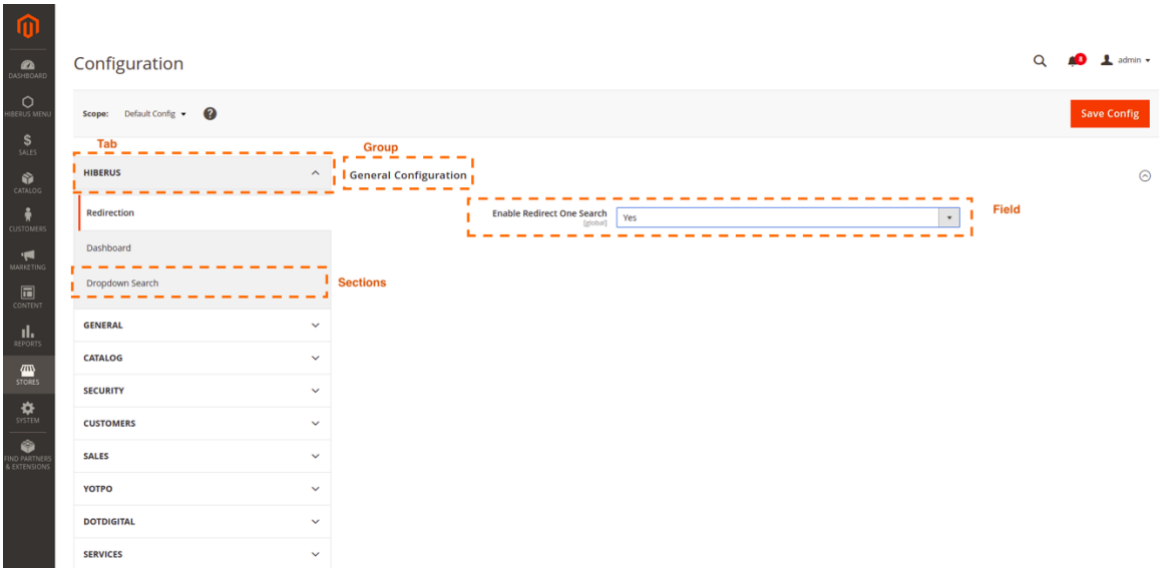


Figura Anexo II-b: Partes de la página de configuración

Módulo de redirección

Para gestionar la configuración en Magento y poder tener la opción de habilitar/deshabilitar el módulo en la vista de administración se ha hecho mediante el archivo system.xml que está ubicado en `/etc/adminhtml`. Los elementos de este archivo son:

`<tab>`: Se ha creado un tab personalizado para los módulos cuyo orden es el primero que va a aparecer y con la etiqueta `<label>` contendrá el nombre que se mostrará (Hiberus) en la `<tab>`.

`<section>`: Es el primero de las 3 sections que se han creado, el orden que se ha establecido hace que salga en primer lugar.

`<group>`: Se define el título de la configuración inicial que tendrá el módulo

`<field>`: Se establece la posibilidad de habilitar o no el módulo

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Config:etc/system_file.xsd">
    <system>
        <tab id="hiberus_general" translate="label" sortOrder="100">
            <label>Hiberus</label>
        </tab>

        <section id="section_redirect" translate="label" sortOrder="10" showInDefault="1" showInWebsite="1" showInStore="1">

            <label>Redirection</label>
            <tab>hiberus_general</tab>
            <resource>Magento_Catalog::config_catalog</resource>

            <group id="general_redirect" translate="label" type="text" sortOrder="10" showInDefault="1" showInWebsite="0" showInStore="0">
                <label>General Configuration</label>

                <field id="enable_redirect" translate="label" type="select" sortOrder="1" showInDefault="1" showInWebsite="0" showInStore="0">
                    <label>Enable Redirect One Search</label>
                    <source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
                </field>
            </group>

        </section>
    </system>
</config>
```

Figura Anexo II-c: Archivo system.xml del módulo de redirección

En el archivo config.xml ubicado en el directorio `/etc` del módulo. Establece que por defecto el módulo deba activarse antes de poder ser usado.

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Store:etc/config.xsd">
    <default>
        <section_redirect>
            <general_redirect>
                <enable_redirect>0</enable_redirect>
            </general_redirect>
        </section_redirect>
    </default>
</config>
```

Figura Anexo II-d: Archivo config.xml del módulo de redirección

Para activar o desactivar los módulos se puede acceder al panel de administración y seleccionar que se desea o mediante comandos. Primero para comprobar que módulos están habilitados se lanza el comando:

```
dockergento magento module:status
```

Este comando lanza un listado con los módulos activados y otro con los módulos desactivados. Para activar el módulo basta con usar el comando seguido del nombre del módulo (y lanzar el comando de actualización):

```
dockergento magento module:enable Hiberus_Redirect
```

Módulo de análisis de usuarios

Para pintar los datos, en Magento la lógica está separada de la vista por lo que, según el módelo vista controlador, se creará un controlador que crea la página, la vista donde se muestran los datos (mediante plantillas o templates) y la lógica dónde se crean los datos para pasárselos a la vista (por bloques).

Bloques

En el directorio `/block`, se ha creado el directorio `/Adminhtml` ya que los datos van a ir exclusivamente a la vista de administrador. En el archivo Detections.php se van a pasar los datos a la tabla y dentro del directorio `/Chart` irá la lógica de cada uno de las gráficas.

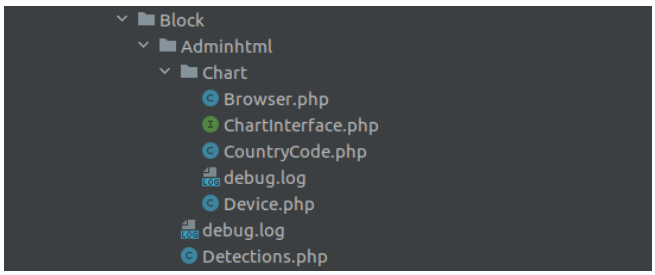


Figura Anexo II-e: Estructura del bloque

En el archivo Detections, primero comprueba si está activado tanto el módulo como la sección de las tablas con el scopeConfig que recupera la configuración por el árbol ruta del archivo system.xml.

```
64      /**
65       * Enable/Disable tables
66       * @return mixed
67       */
68      public function getTables()
69      {
70          return $this->scopeConfig->getValue( path: self::TABLE_PATH,  scopeType: \Magento\Store\Model\ScopeInterface::SCOPE_STORE);
71      }
72
73      /**
74       * Enable/Disable tables depending on plugin
75       * @return mixed
76       */
77      public function isDashboardEnable()
78      {
79          return $this->scopeConfig->getValue( path: self::DASH_PATH,  scopeType: \Magento\Store\Model\ScopeInterface::SCOPE_STORE);
80      }
```

Figura Anexo II-f: Detections.php (1)

En primer lugar, si no se ha realizado el filtrado por fechas, se llama al método `getList()` del repositorio que devuelve todos los elementos filtrados y se recogen los que contengan los campos `'country_code'` de forma ordenada descendentemente de más a menos apariciones.

Sino, si en el rango de fechas decodificadas previamente contienen alguna selección, se le pasa al filtro el rango de fechas. Para que no de errores, se comprueba que hay resultados en el rango de fechas filtrado y los resultados se devuelve en orden descendente.

```php
    /**
     * If date range is null gives all country codes saved, otherwise return country codes on date range
     * @return string
     * @throws \Exception
     */
    public function getCountryCode()
    {
        list($decodeFrom, $decodeTo)= $this->getSelectedDates();

        if ($decodeFrom == null && $decodeTo == null) {
            $detections =  $this->detectionRepository->getList($this->searchCriteriaBuilder->create())->getItems();
            foreach ($detections as $item) {
                $codes[$item['country_code']][] = $item['country_code'];
            }
            rsort( &array: $codes);
            return $codes;
        } else {
            $fromDate = date( format: 'Y-m-d H:i:s', strtotime($decodeFrom));
            $toDate = date( format: 'Y-m-d H:i:s', strtotime( datetime: $decodeTo . '+1 days'));
            $filters = $this->searchCriteriaBuilder
                ->addFilter( field: 'date_time', $fromDate,  conditionType: 'gteq')
                ->addFilter( field: 'date_time', $toDate,  conditionType: 'lteq')
                ->create();
            $items =  $this->detectionRepository->getList($filters)->getItems();
            //$log = new \Monolog\Logger('Debugging');
            //$log->pushHandler(new \Monolog\Handler\StreamHandler(__DIR__ . '/debug.log', \Monolog\Logger::DEBUG));

            if (!empty($items)) {
                foreach ($items as $item) {
                    $codes[$item['country_code']][] = $item['country_code'];
                    //$log->debug(var_export($codes[$item['country_code']][] = $item['country_code'], true));
                }
                rsort( &array: $codes);
                return $codes;
            } else {
                return '';
            }
        }
    }
```

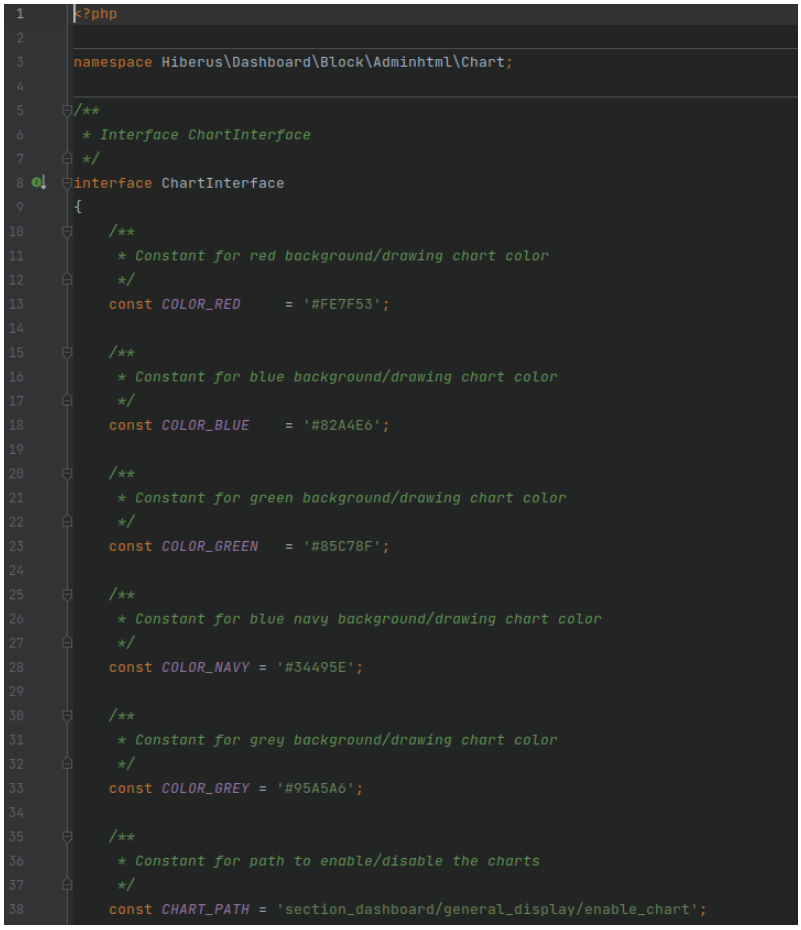
Figura Anexo II-g: Detections.php (2)

El último método es el filtrado de los resultados por fecha, por lo que en el método `getSelectedDates()` se van a decodificar las fechas que se pasan por Javascript de la plantilla.

```php

    /**
     * Decode the dates
     * @return array
     */
    public function getSelectedDates()
    {
        $decodeFrom = base64_decode($this->getRequest()->getParam( key: 'from'));
        $decodeTo = base64_decode($this->getRequest()->getParam( key: 'to'));

        return [$decodeFrom,$decodeTo];
    }
}
```

Figura Anexo II-h: Detections.php (3)

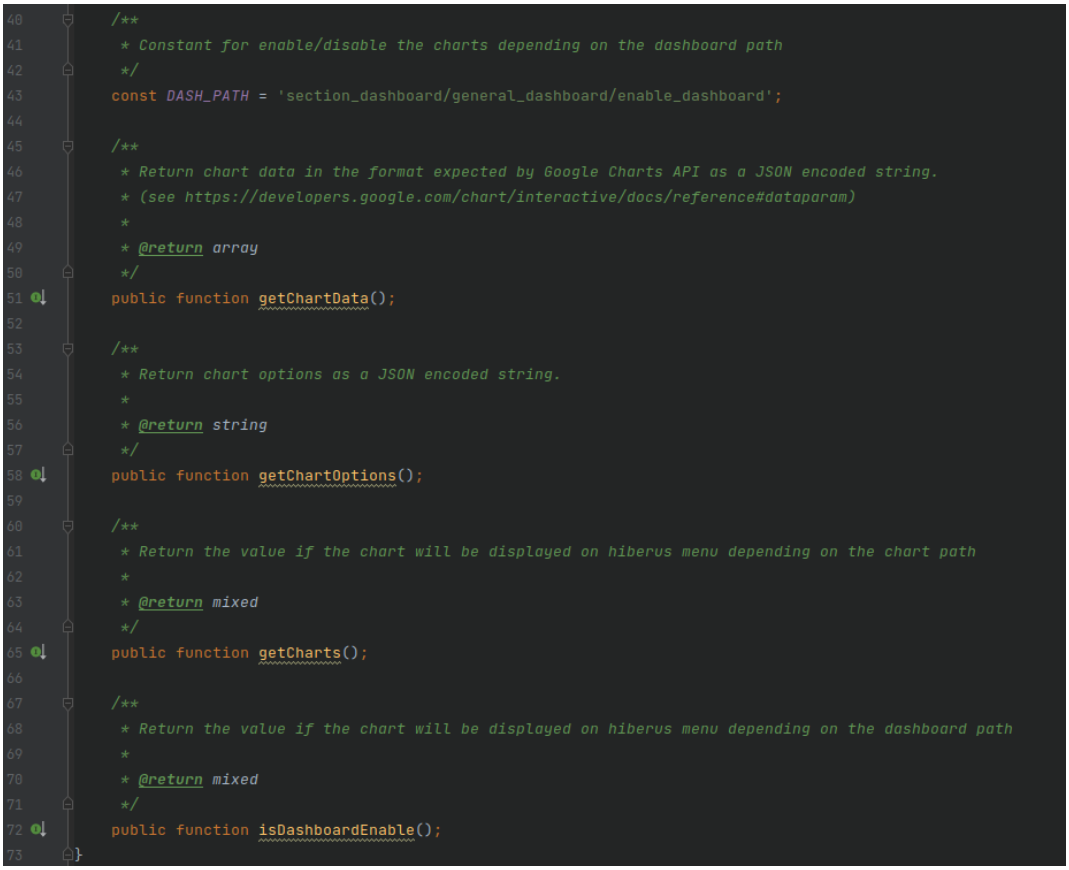
En el directorio `/Chart`, la clase ChartInterface.php define las constantes de los colores que van a ir en los gráficos, las rutas que comprueban si el módulo y los gráficos están habilitados o no y, la definición de los métodos que se van a usar en sus clases.



```php
<?php

namespace Hiberus\Dashboard\Block\Adminhtml\Chart;

/**
 * Interface ChartInterface
 */
interface ChartInterface
{
    /**
     * Constant for red background/drawing chart color
     */
    const COLOR_RED     = '#FE7F53';

    /**
     * Constant for blue background/drawing chart color
     */
    const COLOR_BLUE    = '#82A4E6';

    /**
     * Constant for green background/drawing chart color
     */
    const COLOR_GREEN   = '#85C78F';

    /**
     * Constant for blue navy background/drawing chart color
     */
    const COLOR_NAVY = '#34495E';

    /**
     * Constant for grey background/drawing chart color
     */
    const COLOR_GREY = '#95A5A6';

    /**
     * Constant for path to enable/disable the charts
     */
    const CHART_PATH = 'section_dashboard/general_display/enable_chart';
```

Figura Anexo II-i: Colores de los gráficos y ruta de los gráficos

Como los gráficos se han realizado con la API de Google, su documentación provee la sintaxis de obtener los datos y pintarlos (se ha realizado en el método `getChartData()`). Los datos se cogen primero de una tabla (la de la clase Detections) para pintar las filas y columnas en el gráfico.



```
    /**
     * Constant for enable/disable the charts depending on the dashboard path
     */
    const DASH_PATH = 'section_dashboard/general_dashboard/enable_dashboard';

    /**
     * Return chart data in the format expected by Google Charts API as a JSON encoded string.
     * (see https://developers.google.com/chart/interactive/docs/reference#dataparam)
     *
     * @return array
     */
    public function getChartData();

    /**
     * Return chart options as a JSON encoded string.
     *
     * @return string
     */
    public function getChartOptions();

    /**
     * Return the value if the chart will be displayed on hiberus menu depending on the chart path
     *
     * @return mixed
     */
    public function getCharts();

    /**
     * Return the value if the chart will be displayed on hiberus menu depending on the dashboard path
     *
     * @return mixed
     */
    public function isDashboardEnable();
}
```

Figura Anexo II-j: Métodos de los gráficos y ruta del módulo

A continuación, el contenido de las 3 tablas es el mismo pero cambiando los `'country_code'` por `'browser'` y `'device'`. Se va a explicar solamente el contenido del método getCountryCode() ya que en las demás clases se sustituirá en el método `getChartData()` las llamadas a sus respectivos métodos.

Para el desarrollo de los gráficos, primero se ha comprobado si está habilitado el módulo y los gráficos en la administración.

Después, se han obtenido los valores de la tabla de `'country_code'` y en la variable `$data` se han definido las columnas y en las filas un array que, si no está vacío, va guardando los valores de los `'country_code'` y el número de veces que aparece. Finalmente, este resultado es convertido al formato JSON para que la plantilla trabaje con ellos y pueda crear el gráfico.

```php
    /**
     * Chard Data with country code values
     * @return array|bool|false|string
     */
    public function getChartData()
    {
        $countryCodes= $this->detections->getCountryCode();
        $data = [
            'cols' => [
                ['type' => 'string', 'label' => __('Country codes')],
                ['type' => 'number', 'label' => __('Frequency')],
            ],
            'rows' => [],
        ];
        if($countryCodes != null){
            foreach ($countryCodes as $codes) {
                $data['rows'][]['c'] = [
                    ['v' => __('Country code: ' . $codes[0])],
                    ['v' => count($codes)]
                ];
            }
        }else{
            $data['rows'][]['c'] = [['v' => 1],['v' =>'No Data']];
        }

        return $this->serializer->serialize($data);
    }
}
```

Figura Anexo II-k: Método getChartData() en la clase CountryCode.php

Routes y Controller

Una vez se tiene la lógica, para ver el código ejecutado (mediante plantillas) de los resultados en una página, se necesitan los directorios: Controller, Router y Layout. La url de la página que se va a crear es la siguiente:

```
www.magento2-elastic-local/admin/hiberus_dashboard_searches/top/search/key/f87570bd36103816c1d3738ebb788d50fa6514afb495d45b854f3ed2c864d6ff/
```

Figura Anexo II-l: url

Dónde cada instancia significa: `http://url/route_name/controller/action`

Ya que la ubicación de la página va a estar en la vista de administrador, el archivo routes.xml debe ir en `/etc/adminhtml` y la id de `<router>` debe ser admin. En este archivo se define el `route_name` de la url.

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:App/etc/routes.xsd">
    <router id="admin">
        <route id="hiberus_dashboard_searches" frontName="hiberus_dashboard_searches">
            <module name="Hiberus_Dashboard"/>
        </route>
    </router>

</config>
```

Figura Anexo II-m: route.xml

En el Controller se crea el resto de la ruta de la url. Seguido del `route_name` la url continuará con `top/search`, es decir, el nombre del directorio y del fichero.

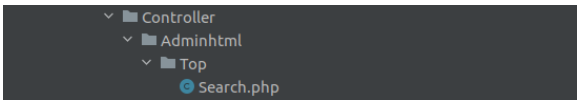


Figura Anexo II-n: Estructura Controller

Su funcionalidad reside en que recibe la solicitud, la procesa y carga la página. Todos los controladores heredan de la clase `Action` de Magento para poder procesar la URL mediante el método `execute()`. Este método se ejecuta cuando el router coincide con la clase de acción del controller.

```php
namespace Hiberus\Dashboard\Controller\Adminhtml\Top;

use Magento\Backend\App\Action;
use Magento\Backend\App\Action\Context;
use Magento\Framework\View\Result\PageFactory;

/**
 * Class Top Search dashboard controller
 * @package Hiberus\Dashboard\Controller\Adminhtml\Search
 */
class Search extends Action
{
    /**
     * @var bool|PageFactory
     */
    protected $pageFactory = false;

    /**
     * Search constructor.
     * @param Context $context
     * @param PageFactory $pageFactory
     */
    public function __construct(
        Context $context,
        PageFactory $pageFactory
    ) {
        $this->pageFactory = $pageFactory;
        parent::__construct($context);
    }

    /**
     * @return \Magento\Framework\App\ResponseInterface|\Magento\Framework\Controller\ResultInterface|\Magento\Framework\View\Result\Page
     */
    public function execute()
    {
        return $this->pageFactory->create();
    }
}
```

Figura Anexo II-o: Controller Módulo de análisis de usuarios

Layout, templates y web

Finalmente para mostrar los datos, en el directorio `/view/adminhtml` (ya que el destino es el panelde administrador) se encuentra el directorio `/layout`, el directorio `/template` y el directorio `/web`.

El directorio layout representa la estructura de la página que va a cargarse con los datos de los bloques y mostrarse según se defina en el template. En el directorio `/web` se definen los estilos.

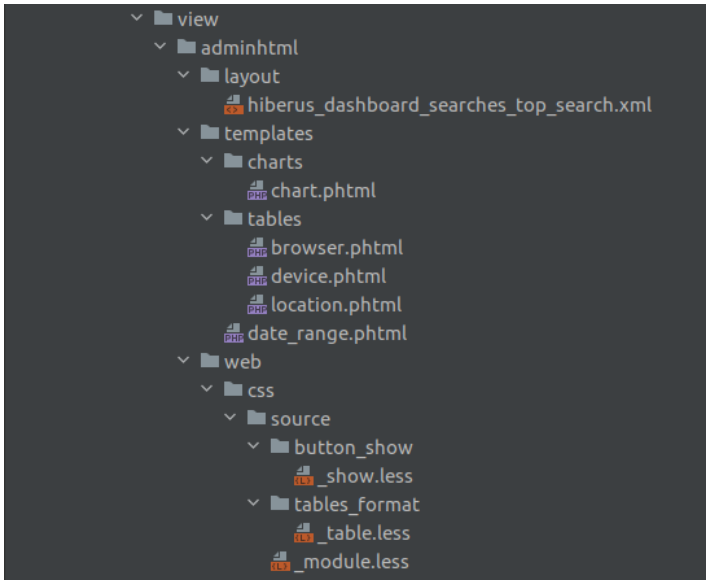


Figura Anexo II-p: Estructura del directorio View

El archivo contenido en el directorio `/layout` va a llamarse con el `route_name` y el controller. Dentro, divide la cabecera de la página y el cuerpo. En la cabecera se va a llamar a la API de Google que va a ayudar a pintar los gráficos.

```
<?xml version="1.0"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configuration.xsd">
    <head>
        <title>Dashboard</title>
        <script src="https://www.gstatic.com/charts/loader.js" src_type="url" />
    </head>
```

Figura Anexo II-q: Layout (<head>)

En el cuerpo se van a definir las secciones que va a contener la página. En el `<referenceContainer>` se aplican `<block>` o `<container>`. En las etiquetas `<block>` se introducen contenidos distintivos, mientras que en el `<container>` puede estar vacío o contener otros elementos <block> para mostrar la lógica de los bloques.

Como se ve en la siguiente imagen, el primer contenido es el filtro del rango de fechas que coge la lógica de la clase del bloque Detections.php (la lógica en sí está en el método `getSelectedDates()`) y la va a pintar en la template (o plantilla) de la clase date_range.phtml

```
<body>
    <!-- DATE RANGE-->
    <referenceContainer name="page.main.actions">
        <block class="Magento\Backend\Block\Store\Switcher" name="store.switcher">
            <arguments>
                <argument name="use_confirm" xsi:type="boolean">false</argument>
            </arguments>
        </block>
        <block class="Hiberus\Dashboard\Block\Adminhtml\Detections" template="Hiberus_Dashboard::date_range.phtml" name="dateRange"/>
    </referenceContainer>
```

Figura Anexo II-r: Layout (rango de fechas)

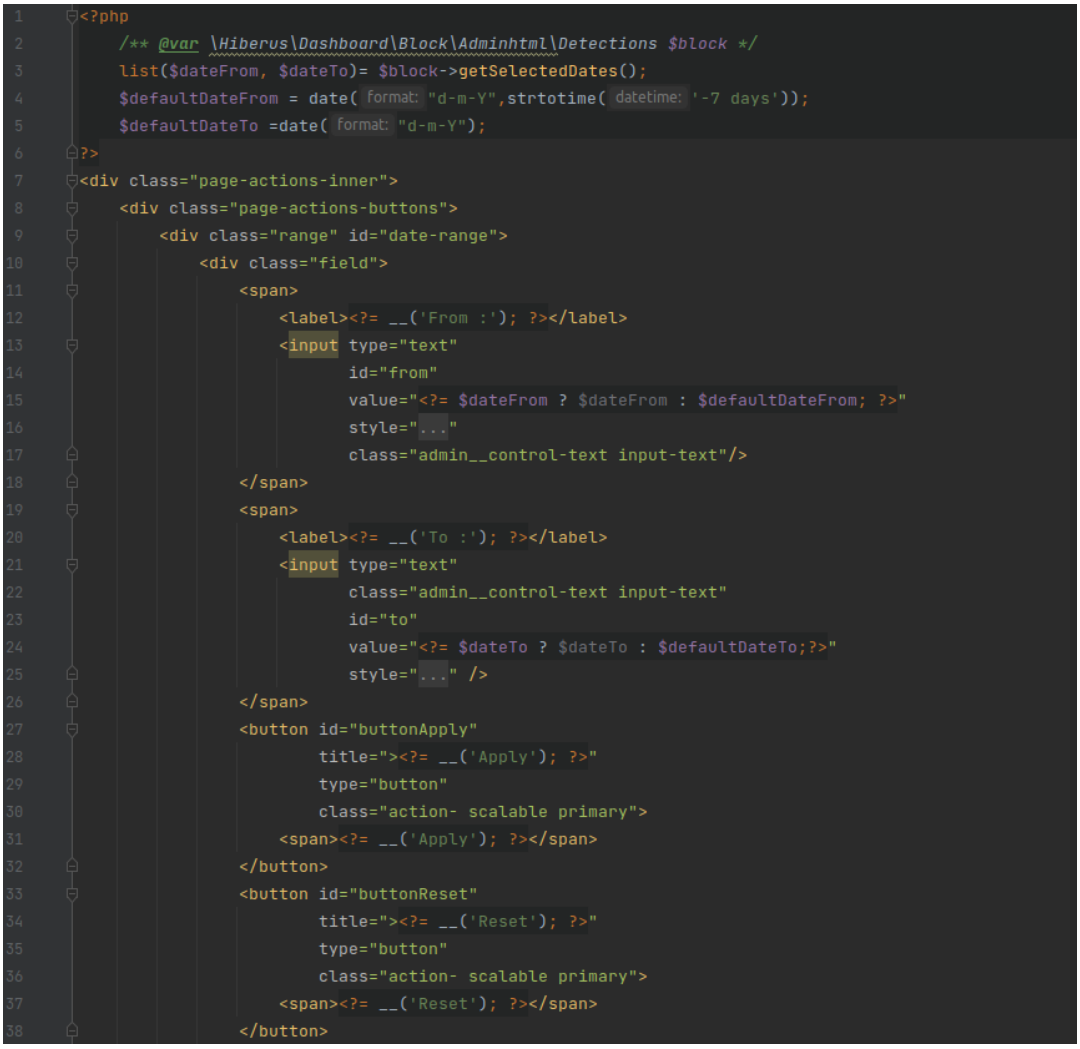
El siguiente `<referenceContainer>` se ha dividido en dos `<container>` (que divide el aspecto de la página en dos columnas) y va a pintar las tablas y gráficos llamando a la lógica de sus respectivos bloques y pintándolos en las respectivas templates. En la etiqueta `<argument>` se ha definido el título de cada tabla y gráfico y, además en los gráficos se ha establecido el tipo de gráfico, de tipo circular.

```
        <referenceContainer name="content">
            <!-- TABLES -->
            <container name="dashboard-right-col" htmlTag="div" htmlClass="dashboard-main col-m-6">
                <!-- TOP LOCATIONS -->
                <block name="dashboard_table_location" class="Hiberus\Dashboard\Block\Adminhtml\Detections" template="Hiberus_Dashboard::tables/location.phtml
                    <arguments>
                        <argument name="title" xsi:type="string" translate="true">Location</argument>
                    </arguments>
                </block>
                <!-- TOP BROWSERS -->
                <block name="dashboard_table_browser" class="Hiberus\Dashboard\Block\Adminhtml\Detections"  template="Hiberus_Dashboard::tables/browser.phtml
                    <arguments>
                        <argument name="title" xsi:type="string" translate="true">Browser</argument>
                    </arguments>
                </block>
                <!-- TOP DEVICES -->
                <block name="dashboard_table_devices" class="Hiberus\Dashboard\Block\Adminhtml\Detections"  template="Hiberus_Dashboard::tables/device.phtml"
                    <arguments>
                        <argument name="title" xsi:type="string" translate="true">Device</argument>
                    </arguments>
                </block>
            </container>
            <!-- CHARTS -->
            <container name="dashboard-left-col" htmlTag="div" htmlClass="dashboard-secondary col-m-6">
                <!-- LOCATION CHART -->
                <block name="dashboard_chart_location" class="Hiberus\Dashboard\Block\Adminhtml\Chart\CountryCode" template="Hiberus_Dashboard::charts/chart
                    <arguments>
                        <argument name="title" xsi:type="string" translate="true">Location</argument>
                        <argument name="chart_type" xsi:type="string" translate="true">PieChart</argument>
                    </arguments>
                </block>
                <!-- BROWSER CHART -->
                <block name="dashboard_chart_browser" class="Hiberus\Dashboard\Block\Adminhtml\Chart\Browser" template="Hiberus_Dashboard::charts/chart.phtml
                    <arguments>
                        <argument name="title" xsi:type="string" translate="true">Browser</argument>
                        <argument name="chart_type" xsi:type="string" translate="true">PieChart</argument>
                    </arguments>
                </block>
                <!-- DEVICE CHART -->
                <block name="dashboard_chart_device" class="Hiberus\Dashboard\Block\Adminhtml\Chart\Device" template="Hiberus_Dashboard::charts/chart.phtml"
                    <arguments>
                        <argument name="title" xsi:type="string" translate="true">Device</argument>
                        <argument name="chart_type" xsi:type="string" translate="true">PieChart</argument>
                    </arguments>
                </block>
            </container >
        </referenceContainer>
    </body>
</page>
```

Figura Anexo II-s: Layout (tablas y gráficos)

Una vez que se han definido desde que plantillas se va a cargar su lógica en el archivo de layout, se va a explicar el contenido de estas plantillas.

La plantilla del rango de fechas implementa el bloque Detections.php para hacer uso del método `getSelectedDates()` y establece que el formato del rango de fechas por defecto sea desde hace 7 días hasta el día actual. En HTML se han definido los rangos de fecha y los botones para aplicar o cancelar. En PHP se han definido los valores de los rangos de fecha.



```php
<?php
    /** @var \Hiberus\Dashboard\Block\Adminhtml\Detections $block */
    list($dateFrom, $dateTo)= $block->getSelectedDates();
    $defaultDateFrom = date( format: "d-m-Y",strtotime( datetime: '-7 days'));
    $defaultDateTo =date( format: "d-m-Y");
?>
<div class="page-actions-inner">
    <div class="page-actions-buttons">
        <div class="range" id="date-range">
            <div class="field">
                <span>
                    <label><?= __('From :'); ?></label>
                    <input type="text"
                           id="from"
                           value="<?= $dateFrom ? $dateFrom : $defaultDateFrom; ?>"
                           style="..."
                           class="admin__control-text input-text"/>
                </span>
                <span>
                    <label><?= __('To :'); ?></label>
                    <input type="text"
                           class="admin__control-text input-text"
                           id="to"
                           value="<?= $dateTo ? $dateTo : $defaultDateTo;?>"
                           style="..." />
                </span>
                <button id="buttonApply"
                        title="><?= __('Apply'); ?>"
                        type="button"
                        class="action- scalable primary">
                    <span><?= __('Apply'); ?></span>
                </button>
                <button id="buttonReset"
                        title="><?= __('Reset'); ?>"
                        type="button"
                        class="action- scalable primary">
                    <span><?= __('Reset'); ?></span>
                </button>
```

Figura Anexo II-t: Plantilla del rango de fechas (phtml)

Al final del archivo, en la siguiente imagen, se escribe el código Javascript. Se han definido la librería jquery para interactuar con el HTML y el widget de Magento del calendario que contiene los parámetros para inicializar el calendario con el rango de fechas.

Las id definidas en el código HTML se guardan en variables en el código Javascript. En las variables `'dateFrom'` y `'dateTo'` se indica que se pueda cambiar de mes, de año, que se muestre los botones (predefinidos en Magento) de `'go today'` y `'close'` y el formato de la fecha que se va a mostrar en el selector de fechas. Además, en el `'dateTo'` se añade el parámetro que como máximo se va a mostrar el mes siguiente disponible.

Por último, se asigna funcionalidad a los botones. Al botón de aplicar la selección se guardan los valores codificados en la url y son los que se mandan al bloque para decodificarlos y filtrar los resultados. En el botón de resetear las fechas, se llama a la función de jquery `datepicker()` que mediante los valores del rango de fechas establezca los valores a `null`. Como recordatorio en el bloque se establecía que si la selección de las dos fechas era null, se devolvía el listado completo de resultados.

```
<script>
    require([
        'jquery',
        'mage/translate',
        'mage/calendar',
        'mage/adminhtml/tools'
    ], function ($, $t) {

        let dateFrom = $("#from");
        let dateTo = $("#to");
        let applyBtn = $('#buttonApply');
        let resetBtn = $('#buttonReset');

        dateFrom.calendar({
            changeMonth: true,
            changeYear: true,
            showButtonPanel: true,
            currentText: $t('Go Today'),
            closeText: $t('Close'),
            dateFormat: 'dd-mm-Y'
        });

        dateTo.calendar({
            changeMonth: true,
            changeYear: true,
            showButtonPanel: true,
            currentText: $t('Go Today'),
            closeText: $t('Close'),
            dateFormat: 'dd-mm-Y',
            maxDate: '+1m'
        });
        applyBtn.on('click', function() {
            let url = "<?= $block->getUrl( route: '*/*/*', ['from' => '__from__', 'to' => '__to__']); ?>"
                .replace('__from__', btoa(($("#from").val())))
                .replace('__to__', btoa(($("#to").val())));
            window.location = url;
        });

        let dates = $('#from, #to').datepicker();
        resetBtn.on('click', function () {
            dates.datepicker('setDate', null);
        })
    })
</script>
```

Figura Anexo II-u: Plantilla del rango de fechas (Javascript)

La funcionalidad de las plantillas de las tablas son similares, por lo que se va a continuar la explicación con la tabla de `'country_code'`. Para las otras tablas habría que llamar a sus métodos respectivos y en el contenido de la tabla `<tbody>` establecer una clase propia para cada una (para no solapar la información de unas con otras).

En primer lugar, se implementa la clase del bloque Detections.php para llamar al correspondiente método para obtener los valores de la tabla. Al principio también se llaman a los métodos de ese bloque que comprueban si el módulo y las tablas están habilitadas, si es el caso, se realiza todo el código que se muestra.

La llamada al método `getJsId()` obtiene a través del layout el name dinámicamente y el método `getTitle()` localiza el `name='title'` que se estableció dentro la etiqueta argumento para colocarlo dinámicamente también. Se crea la tabla y se verifica que si no hay registros lance un mensaje en el cuerpo de la tabla, si existen registros se crearán en Javascript mediante la clase del `<tbody>`.

```php
<?php

/** @var \Hiberus\Dashboard\Block\Adminhtml\Detections $block */
if($block->getTables() && $block->isDashboardEnable()){
    $countrycodes = $block->getCountryCode();
?>
<div class="dashboard-wrapper">
    <div class="dashboard-item" id="<?= $block->getJsId() ?>">
        <div class="dashboard-item-title">
            <span><?= $block->getTitle();?></span>
        </div>
        <p>Discover from where your users access to your commerce</p>
        <table class="dashboard-table-wrapper" >
            <thead>
            <tr>
                <th class="table-col"><span><?= __('Country Code')?></span></th>
                <th class="table-col"><span><?= __('Appears')?></span></th>
            </tr>
            </thead>
            <?php if (empty($countrycodes)) : ?>
                <tbody>
                <tr>
                    <td colspan="10" style="text-align:center; padding: 20px;">
                        <em><?= __('No data over the selected period.'); ?></em>
                    </td>
                </tr>
                </tbody>
            <?php else : ?>
                <tbody class="item-list-cc">
                </tbody>
                <tfoot>
                <tr>
                    <td class="show-buttons">
                        <input type="button" id="button_more_cc" value="Show more">
                        <input type="button" id="button_less_cc" value="Show less">
                    </td>
                    <td></td>
                </tr>
                </tfoot>
            <?php endif; ?>
        </table>
    </div>
</div>
```

Figura Anexo II-v: Plantilla del código del país (phtml)

Para trabajar en el Javascript con los datos del bloque, se codifican los resultados, que devuelve un array en formato JSON, se recorre el array (que estaba ordenado en descendente) y se adjunte a la tabla el código de país y las veces que aparece.

```javascript
<script>
    require(['jquery'], function($){
        $(document).ready(function(){
            let codes2 = <?php echo json_encode($countrycodes); ?>;
            //console.log(codes2);
            for(var index in codes2) {
                var attr = codes2[index];
                for (var i in attr){
                    var res = attr[i];
                    ++i
                }
                $( ".item-list-cc" ).append(
                    '<tr class="list-cc">' +
                    '   <td class="item-cc">'+res+'</td>' +
                    '   <td class="appears-cc">'+ i +'</td>' +
                    '</tr>'
                );
            }
```

Figura Anexo II-w: Plantilla del código del país (Javascript-listado)

La otra funcionalidad en Javascipt es la del botón `'Show more'`. Para ello se ha establecido que se muestren inicialmente tres filas (x=6 porque cada fila tiene dos celdas), se guarden en variables las id y clases y, se obtenga el tamaño de los elementos. En este caso el tamaño de las filas es de 10.

Existen otro botón `'Show less'` que inicialmente está oculto. Se comprueba si el tamaño de las filas es mayor al tamaño que no se quiere superar para que sólo muestre el tamaño deseado (6 filas) y que el botón `'Show more'` realice la función de ir mostrando los demás elementos. Si están todos los elementos desplegados, el botón `'Show less'` ocultará las filas que sobren. Si los elementos no sobrepasan el tamaño deseado, no se va a mostrar el botón `'Show more'` y se mostrará todo el listado.

Al finalizar el código Javascript, se muestra un mensaje en caso de que la primera condición del código phtml sea que ni el módulo ni las tablas estén activadas.

```
            let x = 6;
            let list = $(".list-cc td");
            let buttonMore = $("#button_more_cc");
            let buttonLess = $("#button_less_cc");
            let listEle = list.length;
            list.hide();
            buttonLess.hide();

            if (listEle > x) {
                list.slice(0, x).show();
                buttonMore.click(function () {
                    list.show();
                    buttonMore.hide();
                    buttonLess.show();
                });
                buttonLess.click(function () {
                    list.not(':lt(6)').hide();
                    buttonLess.hide();
                    buttonMore.show();
                });
            }else{
                buttonMore.hide();
                list.show();
            }
        });
    });
</script>

<?php }else{
    echo '<p>Unable to display '. $block->getTitle() . ' table. Please, check the dashboard configuration and enable show tables.</p>';
}?>
```

Figura Anexo II-x: Plantilla del código del país (Javascript-botón)

Los gráficos implementan la interfaz para tener acceso a todos los métodos que se han usado y así recuperar la información de manera más eficaz y evitar llamar a cada clase del bloque. Al igual que se ha hecho en las tablas, en los gráficos primero se va a comprobar que estén activados tanto el módulo como los gráficos y sino se va a lanzar un mensaje.

Para mostrar los gráficos se establecen dos variables que necesita la API de Google, la id (es el `name` del `<block>` en el layout) y el tipo de gráfico. Por otra parte, se comprueba si en el método `getChartData()` se han registrado valores para crear el gráfico, sino no se va a crear ningún gráfico.

```php
<?php
/**
 * @var Hiberus\Dashboard\Block\Adminhtml\Chart\ChartInterface $block
 */
if($block->getCharts() && $block->isDashboardEnable()){

$chartContainerId = $block->getJsId();
$chartType = $block->getChartType() ?: 'PieChart';
?>
<div class="dashboard-item">
    <div class="dashboard-item-title"><?= $block->getTitle() ;?></div>
    <?php if (empty($block->getChartData())) : ?>
    <div class="dashboard-no-data">
    </div>
    <?php else : ?>
    <div class="dashboard-item-content">
        <div id="<?= $chartContainerId ?>" style="..."></div>
    </div>
    <?php endif; ?>
</div>
```

Figura Anexo II-y: Plantilla del gráfico circular (phtml)

Para el desarrollo en Javascript con el que se muestran los gráficos se ha visitado la documentación de Google Charts en: https://developers.google.com/chart/interactive/docs/gallery/piechart

```javascript
<script type="text/javascript">
    google.charts.load('current', {'packages':['corechart']});
    google.charts.setOnLoadCallback(function () {

        var containerId = '<?= $chartContainerId ?>';

        var wrapper = new google.visualization.ChartWrapper({
            chartType: '<?= $block->escapeJs($chartType) ?>',
            dataTable: <?= $block->getChartData() ?>,
            options: <?= $block->getChartOptions() ?>,
            containerId: containerId
        });
        //console.log(wrapper);
        if (wrapper.getDataTable().getNumberOfRows() === 0) {
            document.getElementById(containerId).append('No data over the selected period.');
        } else {
            wrapper.draw();
        }
    });
</script>

<?php } else{
    echo '<p>Unable to display '. $block->getTitle() . ' chart. Please, check the dashboard configuration and enable show charts.</p>';
}?>
```

Figura Anexo II-z: Plantilla del gráfico circular (javascript)

Por último, en el subdirectorio `/web` se han creado los estilos personalizados para las tablas y botones de `'Show more'` y `'Show less'`.

```less
& when (@media-common = true) {
  body {
  //#button_more_cc #button_less_cc #button_more_br #button_less_br
    .show-buttons{
      input {
        background-color: #373330; /* Dark Grey */
        border: none;
        color: white;
        padding: 8px 15px;
        text-align: center;
        text-decoration: none;
        display: inline-block;
        font-size: 14px;
        transition-duration: 0.4s;
        position: relative;
        float:left;
        font-weight: bold;
      }

      input:hover {
        background-color: #696562; /* Clear Green */
      }
    }
  }
}
```

Figura Anexo II-aa: Estilos de los botones

```less
& when (@media-common = true) {
  body {

    .dashboard-wrapper {
      margin-bottom: 100px;

      p {
        margin-bottom: 20px;
      }

      .dashboard-table-wrapper {
        border-collapse: collapse;
        border-spacing: 0;
        width: 63%;
        font-size: 13px;
        margin-bottom: 20px;
        table-layout: fixed;
        position: relative;
        text-align-last: left;

        tbody {
          cursor: pointer;
        }

        th, td {
          padding: 10px;
          border-bottom: 1px solid #e3e3e3;
        }

        tbody tr:nth-child(odd) {
          background-color: #f8f8f8;
          width: 100px;
        }

        tr:last-child td {
          border-bottom: 0;
        }
```

Figura Anexo II-bb: Estilos de las tablas

Para que Magento reconozca los estilos no basta con crearlos, también se tienen que importar en el archivo _module.less

```less
@import "button_show/_show.less";
@import "tables_format/_table.less";
```

Figura Anexo II-cc: Archivo _module.less

Finalizada la explicación de cómo se han pintado los datos, se comienza con la de cómo se ha creado el menú lateral que permite acceder a las configuraciones de los módulos y a los dashboards de análisis de usuarios y resultados de búsqueda. También se explicará cómo se han desarrollado las opciones de habilitar/deshabilitar el módulo, las tablas y gráficos.

En el directorio `/etc/adminhtml` se ha creado el archivo menu.xml, este arhivo se organiza por niveles. El nivel 0 es el que se muestra a la izquierda del menú y en el código corresponde a Hiberus Menu.

Los atributos que posee son el identificador que debe ser único y tener el formato del módulo seguido del nombre del menú (`Hiberus_Menu::menu`), el titulo que es el que se mostrará en el menú lateral, el modulo que es dónde se encuentra el archivo, el orden con el que se va a mostrar y en el atributo se define la regla ACL que el administrador debe tener para ver y acceder al menú.

Los siguientes niveles se definen de la misma manera pero indicándole que pertenecen a un padre (de nivel 0). Para el submenú, los niveles 1 contendrán la id del padre al que pertenecen "`Hiberus_Dashboard::menu`" y los niveles 2 contendrán la id del padre de los niveles 1 que corresponden.

En la siguiente imagen, los últimos niveles son las páginas del dashboard que realizan el análisis de usuarios y palabras buscadas. Para que se pueda acceder desde el menú (de nivel 2) a la página se debe colocar además el atributo '`action`' que especifica el formato '`route_name/controller/action`' que se definió al crear la página.

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Backend:etc/menu.xsd">
    <menu>
        <!-- MENU -->
        <add id="Hiberus_Dashboard::menu"
             title="Hiberus Menu"
             translate="title"
             module="Hiberus_Dashboard"
             sortOrder="10"
             resource="Hiberus_Dashboard::content"
        />

        <!-- DASHBOARD TOP SEARCH ANALYTICS -->

        <add id="Hiberus_Dashboard::detections"
             title="Dashboard"
             translate="title"
             module="Hiberus_Dashboard"
             sortOrder="10"
             parent="Hiberus_Dashboard::menu"
             resource="Hiberus_Dashboard::detections"
        />
        <add id="Hiberus_Dashboard::user_access"
             title="User Access"
             module="Hiberus_Dashboard"
             sortOrder="10"
             parent="Hiberus_Dashboard::detections"
             action="hiberus_dashboard_searches/top/search"
             resource="Hiberus_Dashboard::user_access"
        />
        <add id="Hiberus_DropdownSearch::search_analytics"
             title="Search Analytics"
             module="Hiberus_Dashboard"
             sortOrder="20"
             parent="Hiberus_Dashboard::detections"
             action="hiberus_dashboard_dropdown/search/analytics"
             resource="Hiberus_Dashboard::search_analytics"
        />
```

Figura Anexo II-dd: Configuración del menú (menú y páginas creadas)

A continuación del mismo archivo, se ha creado de la misma manera el nivel 1 que indica que los niveles 2 van a ser enlaces de configuración.

Para definir los atributos `action` de las configuraciones de los módulos para poder acceder a la página de configuración de cada módulo, se ha observado en la url del navegador (se recuerda que para el panel de administración la url es www.magento2-elastic-local/admin) cómo continúa.

Por ejemplo, en este módulo la url de su configuración es `'admin/system_config/edit/section/section_dashboard'` y este es el action que se le ha establecido. Se realiza la misma dinámica con los demás módulos.

```
        <!-- REDIRECTIONS TO CONFIGURATIONS -->

        <add id="Hiberus_Dashboard::settings"
             title="Settings"
             translate="title"
             module="Hiberus_Dashboard"
             sortOrder="20"
             parent="Hiberus_Dashboard::menu"
             resource="Hiberus_Dashboard::settings"
        />
        <add id="Hiberus_Dashboard::redirect_config"
             title="One Result Configuration"
             module="Hiberus_Dashboard"
             sortOrder="10"
             parent="Hiberus_Dashboard::settings"
             action="adminhtml/system_config/edit/section/section_redirect"
             resource="Hiberus_Dashboard::redirect_config"
        />
        <add id="Hiberus_Dashboard::dashboard_config"
             title="Dashboard Configuration"
             module="Hiberus_Dashboard"
             sortOrder="20"
             parent="Hiberus_Dashboard::settings"
             action="adminhtml/system_config/edit/section/section_dashboard"
             resource="Hiberus_Dashboard::dashboard_config"
        />
        <add id="Hiberus_Dashboard::autocomplete_config"
             title="Dropdown Configuration"
             module="Hiberus_Dashboard"
             sortOrder="30"
             parent="Hiberus_Dashboard::settings"
             action="adminhtml/system_config/edit/section/section_dropdown"
             resource="Hiberus_Dashboard::autocomplete_config"
        />
    </menu>
</config>
```

Figura Anexo II-ee: Configuración del Menú (menú y acceso a ajustes)

El archivo de configuración system.xml está ubicado en el directorio `/etc/adminhtml` y las nuevas funcionalidades que se han añadido respecto al system.xml del módulo de redirección son:

El atributo `<comment>` añade un comentario debajo de la etiqueta `<label>`

El atributo `<depends>` declara las dependencias con otros campos. Si el campo del que se depende está con el valor 1 (en el archivo config.xml), se mostrará el `<field>`.

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Config:etc/system_file.xsd">
    <system>
        <tab id="hiberus_general" translate="label" sortOrder="100">
            <label>Hiberus</label>
        </tab>
        <section id="section_dashboard" translate="label" type="text" sortOrder="20" showInDefault="1" showInWebsite="1" showInStore="1">
            <tab>hiberus_general</tab>
            <label>Dashboard</label>
            <resource>Magento_Catalog::config_catalog</resource>
            <group id="general_dashboard" translate="label" type="text" sortOrder="1" showInDefault="1" showInWebsite="0" showInStore="0">
                <label>General Configuration</label>
                <field id="enable_dashboard" translate="label" type="select" sortOrder="1" showInDefault="1" showInWebsite="0" showInStore="0">
                    <label>Enable Dashboard</label>
                    <source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
                    <comment>Enable/Disable all sources of Dashboard.</comment>
                </field>
            </group>
            <group id="general_display" translate="label" type="text" sortOrder="2" showInDefault="1" showInWebsite="0" showInStore="0">
                <label>Display Configuration</label>
                <field id="enable_table" translate="label" type="select" sortOrder="3" showInDefault="1" showInWebsite="0" showInStore="0">
                    <label>Show Tables</label>
                    <source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
                    <comment>Enable/Disable location, browser and device tables.</comment>
                    <depends>
                        <field id="section_dashboard/general_dashboard/enable_dashboard">1</field>
                    </depends>
                </field>
                <field id="enable_chart" translate="label" type="select" sortOrder="2" showInDefault="1" showInWebsite="0" showInStore="0">
                    <label>Show Charts</label>
                    <source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
                    <comment>Enable/Disable location, browser and device pie charts.</comment>
                    <depends>
                        <field id="section_dashboard/general_dashboard/enable_dashboard">1</field>
                    </depends>
                </field>
                <depends>
                    <field id="section_dashboard/general_dashboard/enable_dashboard">1</field>
```

Figura Anexo II-ff: Contenido del archivo system.xml del módulo de análisis de usuarios

En este caso, en el archivo /etc/config.xml de este módulo se ha determinado que al habilitar el módulo, las tablas y gráficos estén activos.

```xml
<?xml version="1.0"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Store:etc/config.xsd">
    <default>
        <section_dashboard>
            <general_dashboard>
                <enable_dashboard>1</enable_dashboard>
            </general_dashboard>
            <general_display>
                <enable_chart>1</enable_chart>
                <enable_table>1</enable_table>
            </general_display>
        </section_dashboard>
    </default>
</config>
```

Figura Anexo II-gg: Contenido del archivo config.xml del módulo de análisis de usuarios

Módulo de desplegable de resultados

Puesto que en los anteriores módulos se ha explicado qué archivos se necesitan para crear una página en la vista de administrador (este módulo además va a ir en la misma ubicación del menú personalizado) en este módulo no se va a proceder a su explicación por no repetir las mismas funcionalidades.

Los archivos para crear la ruta de una página en el panel de administración "Search Analytics" ubicada en el menú (imagen 5.2.1-c) son etc/adminhtml/routes.xml, Controller/Adminhtml/Search/Analytics.php y view/adminhtml/layout que se corresponden a la url 'route_name/controller/action'.

Por el mismo motivo, los archivos etc/adminhtml/system.xml y etc/config.xml en los que se definen los campos de configuración y los valores por defecto tampoco van a explicarse. Sin embargo, del archivo system.xml se va a mostrar los campos <field> para que el gestor pueda modificar los valores principalmente de la configuración general del módulo (ya que el otro campo que recibe parámetros configurables es el del número de productos que se quieren mostrar y se realiza de la misma manera).

Todos los campos <field> dependen de que el módulo esté habilitado para poder desplegarse sus opciones de configuración y los que tienen la id 'delay_time', 'min_characters' y 'no_result' son los campos que restringen su entrada de datos en campos específicos. Se ha usado la clase de Magento validate-zero-or-grater para validar números cuyos valores deben ser mayores que 0 en los campos 'delay_time' y 'min_characters' mientras que para el campo 'no_result' se colocará el texto personalizado a continuación de la etiqueta <label>.

La comunicación entre el panel de configuración con estos campos editables y el desplegable en la tienda online se realiza gracias a la clase ScopeConfig en el Helper que es dónde se comprueban los valores para luego pasarlos al Block y que finalmente desde la template puedan ser llamados.

```
<group id="general_dropdown" translate="label" type="text" sortOrder="1" showInDefault="1" showInWebsite="0" showInStore="0">
    <label>General Configurations</label>
    <field id="enable_dropdown" translate="label" type="select" sortOrder="1" showInDefault="1" showInWebsite="0" showInStore="0">
        <label>Enable</label>
        <source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
        <comment>Enable/Disable Dropdown Search Autocomplete.</comment>
    </field>
    <field id="delay_time" translate="label" type="text" sortOrder="2" showInDefault="1" showInWebsite="0" showInStore="0">
        <label>Delay Time</label>
        <validate>validate-zero-or-greater integer</validate>
        <comment>Set delay time to start searching.</comment>
        <depends>
            <field id="enable_dropdown">1</field>
        </depends>
    </field>
    <field id="min_characters" translate="label" type="text" sortOrder="3" showInDefault="1" showInWebsite="0" showInStore="0">
        <label>Minimum number of characters to search</label>
        <validate>validate-zero-or-greater integer</validate>
        <comment>Minimum number of characters which customer should enter to start auto complete.</comment>
        <depends>
            <field id="enable_dropdown">1</field>
        </depends>
    </field>
    <field id="no_result" translate="label" type="text" sortOrder="4" showInDefault="1" showInWebsite="0" showInStore="0">
        <label>Custom text when no results</label>
        <comment>Add custom text to display when no results are found.</comment>
        <depends>
            <field id="enable_dropdown">1</field>
        </depends>
    </field>
</group>
```

Figura Anexo II-hh: Contenido del archivo system.xml del módulo de desplegable de resultados

Para el análisis de las consultas realizadas en el dashboard se ha usado la clase del bloque SearchAnalytic.php que implementa del módulo de búsqueda de Magento la colección de consultas de búsqueda.

En la plantilla se muestran los valores de las consultas con resultados y sin resultados y las tablas con las palabras de consultas con resultados y sin resultados. Esta clase contiene 4 métodos que se corresponden con lo que se quiere mostrar en las plantillas. Se van a explicar el funcionamiento de dos de ellos (valores y tabla de consultas sin resultados) ya que los otros dos contienen el mismo código pero con la condición del filtro del número de resultados diferente.

Mientras que en los métodos que devuelven las consultas sin resultados la condición es 'eq' => 0, lo que significa que se va a mostrar las consultas cuyos números de resultados sean 0 (al contrario en los que no devuelven resultados cuya condición cambia a `'neq' => 0`.

En el método `getNoResultsTable()` de la clase CollectionFactory de Magento que se implementa, se obtienen los 15 términos más usados de las consultas sin resultados. En el método `getNoResultsValue()` se obtiene el número de veces que se han realizado consultas sin resultados.

```
49    public function getNoResultsTable()
50    {
51        $this->_collection = $this->_queriesFactory->create()->addOrder( field: "popularity")->setPageSize( size: 15);
52
53        if ($this->getRequest()->getParam( key: 'store')) {
54            $this->_collection->addFieldToFilter( field: 'store_id', $this->getRequest()->getParam( key: 'store'));
55        } elseif ($this->getRequest()->getParam( key: 'website')) {
56            $storeIds = $this->_storeManager->getWebsite($this->getRequest()->getParam( key: 'website'))->getStoreIds();
57            $this->_collection->addFieldToFilter( field: 'store_id', ['in' => $storeIds]);
58        } elseif ($this->getRequest()->getParam( key: 'group')) {
59            $storeIds = $this->_storeManager->getGroup($this->getRequest()->getParam( key: 'group'))->getStoreIds();
60            $this->_collection->addFieldToFilter( field: 'store_id', ['in' => $storeIds]);
61        }
62
63        return $this->_collection->addFieldToFilter( field: 'num_results', ['eq'=> 0])->getItems();
64    }
65
66    public function getNoResultsValue()
67    {
68        $this->_collection = $this->_queriesFactory->create();
69
70        if ($this->getRequest()->getParam( key: 'store')) {
71            $this->_collection->addFieldToFilter( field: 'store_id', $this->getRequest()->getParam( key: 'store'));
72        } elseif ($this->getRequest()->getParam( key: 'website')) {
73            $storeIds = $this->_storeManager->getWebsite($this->getRequest()->getParam( key: 'website'))->getStoreIds();
74            $this->_collection->addFieldToFilter( field: 'store_id', ['in' => $storeIds]);
75        } elseif ($this->getRequest()->getParam( key: 'group')) {
76            $storeIds = $this->_storeManager->getGroup($this->getRequest()->getParam( key: 'group'))->getStoreIds();
77            $this->_collection->addFieldToFilter( field: 'store_id', ['in' => $storeIds]);
78        }
79
80        return $this->_collection->addFieldToFilter( field: 'num_results', ['eq'=> 0])->addFieldToSelect( field: 'popularity')->getItems();
81    }
```

Figura Anexo II-ii: Métodos getNoResultsValue() y getNoResultsTable() del bloque SearchAnalytic

También se comprueba si en la configuración del panel de administración está habilitado gracias al `ScopeConfig`.

Figura Anexo II-jj: Método getIsDashboardEnable() del bloque SearchAnalytic

El botón que aparece debajo de las tablas en el dashboard y que descarga su contenido se ha realizado en la ruta `Controller/TopSearches/CsvNoResults.php`. La explicación va a continuar sobre el mismo tipo de resultados (sin resultados). Esta clase implementa las clases de Magento:

FileFactory: Sirve para declarar el contenido del archivo en respuesta a la descarga

FileSystem: Crea la instancia del directorio con permisos de escritura

CollectionFactory: Sirve para crear una instancia con los parámetros específicos

El método getColumnHeader() define los cabeceros de las columnas para el archivo CSV.

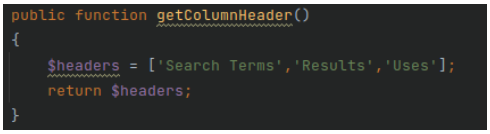


Figura Anexo II-kk: Método getColumnHeader() del controlador

El otro método que se utiliza `getNoResults()` hace exactamente la misma funcionalidad que en el `getNoResultsTable()` del bloque excepto que se ha eliminado el filtro del tamaño de resultados, ya que se desea descargar todos los resultados (y no los 15 primeros). La línea cambia de la siguiente manera (el resto de las líneas del método son las mismas que las de la tabla):

Figura Anexo II-ll: Método getResults() del controlador CsvNoResults

Finalmente, el método `execute()` es donde se realiza el proceso de exportación al fichero CSV. Se crea una ruta para crear la exportación del archivo y se establece que tenga permisos de lectura y escritura. Se define el cabecero que se va a mostrar en la primera fila de la tabla y todos los datos que se han obtenido de los términos sin resultados, sacando sólo el texto, número de resultados y número de veces que aparece. Tras escribir la cabecera y el contenido del CSV se crea el archivo de exportación.

```php
public function execute()
{
    $name = date( format: 'm-d-Y_H:i:s');
    $filepath = 'export/custom' . $name . '.csv';
    $this->directory->create( path: 'export');
    /* Open file */
    $stream = $this->directory->openFile($filepath, mode: 'w+');
    $stream->lock();
    $columns = $this->getColumnHeader();
            foreach ($columns as $column) {
        $header[] = $column;
    }

    $stream->writeCsv($header);
    $terms[] = $this->getNoResults();

    foreach ($terms as $item) {
        foreach ($item as $val) {
            //var_export($itemData);
            $itemData = [];
            $itemData[] = $val->getData()["query_text"];
            $itemData[] = $val->getData()["num_results"];
            $itemData[] = $val->getData()["popularity"];
            $stream->writeCsv($itemData);
        }
    }

    $content = [];
    $content['type'] = 'filename';
    $content['value'] = $filepath;
    $content['rm'] = '1';
    $csvfilename = 'Searches0Results.csv';
    return $this->_fileFactory->create($csvfilename, $content, baseDir: DirectoryList::VAR_DIR);
}
```

Figura Anexo II-mm: Método execute() del controlador CsvNoResults

El contenido del archivo view del panel de administración tiene el siguiente aspecto:

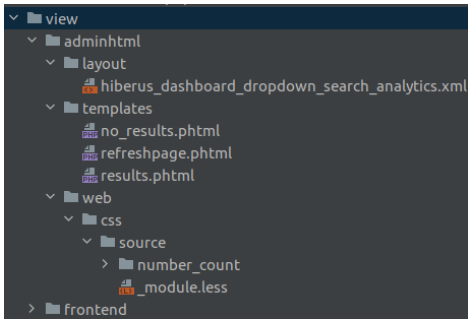


Figura Anexo II-nn: Estructura del directorio /view/adminhtml

En el layout, la estructuración de la página va a contener al principio `<referenceContainer>` con la plantilla delbotón que recarga y actualiza los resultados de la página (datos que se consiguen en el bloque SearchAnalytic). En el siguiente `<referenceContainer>` se van a incluir la lógica de los bloques de valores y tablas y la ruta de la plantilla como van a pintarse.

```xml
<?xml version="1.0"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_configuration.xsd">
    <head>
        <title>Dashboard</title>
    </head>
    <body>
        <referenceContainer name="page.main.actions">
            <block class="Magento\Backend\Block\Store\Switcher" name="store.switcher">
                <arguments>
                    <argument name="use_confirm" xsi:type="boolean">false</argument>
                </arguments>
            </block>
            <block class="Hiberus\DropdownSearch\Block\SearchAnalytic" name="refresh_page" after="store_switcher" template="Hiberus_DropdownSearch::refreshpage.phtml"/>
        </referenceContainer>

        <referenceContainer name="content">
                <container name="dashboard-left-col" htmlTag="div" htmlClass="dashboard-main col-m-6">
                    <block class="Hiberus\DropdownSearch\Block\SearchAnalytic" name="dashboard-dropdown-table-results" template="Hiberus_DropdownSearch::results.phtml">
                        <arguments>
                            <argument name="title" xsi:type="string" translate="true">Top Results Searches  </argument>
                        </arguments>
                    </block>
                </container>

                <container name="dashboard-right-col" htmlTag="div" htmlClass="dashboard-secondary col-m-6">
                    <block class="Hiberus\DropdownSearch\Block\SearchAnalytic" name="dashboard-dropdown-table-no-results" template="Hiberus_DropdownSearch::no_results.phtml">
                        <arguments>
                            <argument name="title" xsi:type="string" translate="true">Top No Results Searches </argument>
                        </arguments>
                    </block>
                </container>
        </referenceContainer>

    </body>
</page>
```

Figura Anexo II-oo: Contenido del layout del módulo de desplegable de resultados

El botón de refrescar la página hace que al darle al botón `'Reload Data'` se envía la petición y recarga la url que coincide con la ruta del controlador. La maquetación se ha extendido de la plantilla de Magento cambiando el atributo `action`.

```php
<?php /**@var Hiberus\DropdownSearch\Block\SearchAnalytic $block  */
/**
 *  @var \Magento\Framework\Escaper $escaper
 */
$escaper?>
<div class="page-actions">
    <div class="page-actions-inner">
        <div class="page-actions-buttons">

            <form class="action-element-my"
                  action="<?= $escaper->escapeUrl($block->getUrl( route: '*/*/analytics')) ?>"
                  method="post">
                <input
                    name="form_key"
                    type="hidden"
                    value="<?= $escaper->escapeHtmlAttr($block->getFormKey()) ?>"/>
                <button
                    class="action-primary"
                    type="submit"
                    title="<?= $escaper->escapeHtmlAttr(__('Reload Data')) ?>">
                    <?= $escaper->escapeHtml(__('Reloadd Data')) ?>
                </button>
            </form>
        </div>
    </div>
</div>
```

Figura Anexo II-pp: Maquetación del botón del módulo de desplegable de resultados

Para mostrar los valores sin resultados, en el primer contenedor `<div>` se realiza la llamada al bloque SearchAnalytic para hacer la llamada a su respectivo método, `getNoResultsValue()` para obtener el número total de veces que hay consultas sin resultados.

```php
<?php /**@var Hibervs\DropdownSearch\Block\SearchAnalytic $block  */
    if ($block->getIsDashboardEnable()) :
?>
<div class="col-s-2 col-m-6 col-l-6 col-xl-12" id="dashboard-counters">
    <ul>
        <li class="dashboard-counter-item">
            <span class="dashboard-totals-label"><?= __('Top 0 Searches') ?></span>
            <?php
            $noRes=0;
            /**
             * @var Magento\Search\Model\Query $item
             */
            foreach ($block->getNoResultsValue() as $item):
                foreach ($item->getData() as $value):
                    $noRes +=$value;
                endforeach;
            endforeach;
            ?>
            <div class="numberCircle"><?=$noRes?></div>
        </li>
    </ul>
</div>
```

Figura Anexo II-qq: Maquetación de los valores de términos sin resultados

Para la tabla con las consultas de los términos sin resultados más usadas, en otro contenedor se ha pintado la tabla con las columnas `'Search Term'` dónde van a ir los términos de búsqueda, `'Results'` dónde se muestran si hay o no resultados (en este caso siempre va a devolver 0 a diferencia de la tabla de resultados) y `'Uses'` que mostrará las repeticiones con las que se ha hecho la consulta.

```php
<div class="dashboard-wrapper">
    <div class="dashboard-item" id="<?= $block->getJsId() ?>">
        <div class="dashboard-item-title">
            <?= $block->getTitle(); ?>
        </div>
        <table class="dashboard-table-wrapper" id="dashboard-table-wrapper-top">
            <thead>
            <tr>
                <th class="table-col"><span><?= __('Search Term'); ?></span></th>
                <th class="table-col"><span><?= __('Results'); ?></span></th>
                <th class="table-col"><span><?= __('Uses'); ?></span></th>
            </thead>
            <?php if (empty($block->getNoResultsTable())) : ?>
                <tbody>
                <tr>
                    <td colspan="10" style="...">
                        <em><?= __('No data yet'); ?></em>
                    </td>
                </tr>
                </tbody>
            <?php else : ?>
                <tbody class="item-list-no-results">
                <?php
                /**
                 * @var Magento\Search\Model\Query $item
                 */
                foreach ($block->getNoResultsTable() as $item):?>
                    <tr>
                        <td><a href="<?= $block->escapeUrl($block->getUrl( route: 'search/term/edit', ['id' => $item->getId()])) ?>">
                                <?=$item->getData()["query_text"]?></a></td>
                        <td><?=$item->getData()["num_results"]?></td>
                        <td><?=$item->getData()["popularity"]?></td>
                    </tr>
                <?php endforeach; ?>
                </tbody>
            <?php endif; ?>
        </table>
```

Figura Anexo II-rr: Maquetación de la tabla de términos sin resultados

Al continuación de la tabla se ha creado un link que llamará a la ruta del archivo csv para proceder a su descarga. Su ruta será la del mismo archivo routes.xml (se ha reutilizado) junto con la ruta de TopSearches/CsvNoResults.php:
`hiberus_dashboard_dropdown/TopSearches/CsvNoResults`.

También se ha comprobado si está el módulo habilitado para mostrar o no los resultados.

```
        <div id="btn-csv-download">
            <a href="<?=$this->getUrl("hiberus_dashboard_dropdown/TopSearches/CsvNoResults")?>" class="csv-button-res" >Download CSV</a>
        </div>
    </div>
</div>
<?php else:
        echo '<p>Unable to display ' . $block->getTitle() . ' table. Please, check the dashboard configuration.</p>';
    endif; ?>
```

Figura Anexo II-ss: Maquetación del botón de descarga del csv

En cuanto a los estilos se ha aprovechado los mismo estilos de las tablas del módulo anterior y se han creado estilos para el botón csv y para los círculos que van a contener los valores de consultas con y sin resultados.

```
#dashboard-dropdown-table-results, #dashboard-dropdown-table-no-results{
    .csv-button-res {
        padding: 7px 15px;
        background: #ea5323;
        color: #fff;
        text-decoration:none;
        font-weight: bold;
        position: relative;
        float: right;
        right: 158px;
    }
    .csv-button-res:hover{
        background-color: #ba4000;
        box-shadow: 0 0 0 1px #007bdb;
        text-decoration: none;
    }
}
```

Figura Anexo II-tt: Estilos del botón de descarga del csv

```
        .numberCircle {
          font: 16px Arial, sans-serif;
          color: #ea5323;
          border: 2px solid #666;
          border-radius: 50%;
          font-size: 3.5rem;
          display: flex;
          justify-content: center;
          align-items: center;
          width: 3em;
          height: 3em;
        }
      }

    }

  }
```

Figura Anexo II-uu: Estilos del circulo para los valores