UNIVERSIDAD AUTONOMA DE MADRID

Escuela Politécnica Superior

19
20

# Trabajo fin de grado

## Desarrollo de un plugin en WordPress para análisis y minería de datos


Alfonso de Paz García

Escuela Politécnica Superior
Universidad Autónoma de Madrid
C\Francisco Tomás y Valiente nº 11


Campus Internacional excelencia UAM CSIC

www.uam.es

**UNIVERSIDAD AUTÓNOMA DE MADRID**
**ESCUELA POLITÉCNICA SUPERIOR**


**Grado en Ingeniería Informática**

# TRABAJO FIN DE GRADO

## Desarrollo de un plugin en WordPress para análisis y minería de datos

**Autor: Alfonso de Paz García**
**Tutor: Alejandro Bellogín Kouki**

**mayo 2020**

**Alfonso de Paz García**
***Desarrollo de un plugin en WordPress para análisis y minería de datos***

**Alfonso de Paz García**
C\ Francisco Tomás y Valiente Nº 11

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

# Agradecimientos

En primer lugar me gustaría agradecer a Alejandro Bellogín por todo el buen trato que ha tenido conmigo, tanto por ser un excelente profesor, siempre disponible en primero de carrera; como tutor, por guiarme en este proyecto desde el principio y siempre estar dispuesto a ayudarme y responder mis dudas hasta el final.

Quería agradecer a toda mi familia por apoyarme desde el principio y haber estado encima ayudando cuando lo necesitaba, desde que comencé mis estudios en la educación primaria hasta la actualidad, gracias por enseñarme la importancia de la constancia y el trabajo si uno quieres lograr sus objetivos.

# Resumen

La minería de datos es un proceso que consiste en la recuperación de información no estructurada para obtener de ella ciertos patrones tanto en la estructura como en el contenido. Es un proceso que se ha extendido en el último siglo con la migración masiva de todos los servicios a Internet, principalmente en aquellas páginas web donde interesa conocer al usuario.

Sin embargo, es necesario darle uso a toda esta información obtenida para que sea útil y, es en este momento, donde entran en juego los sistemas de recomendación. Estas herramientas se encargan, mediante el uso de la información extraída previamente, de sacar conclusiones y obtener un patrón con los gustos de los usuarios para posteriormente recomendarles ítems (productos, canciones, relaciones sociales, etc.) acorde a ellos.

A lo largo de este Trabajo de Fin de Grado definiremos una arquitectura para un sistema de recomendación con minería de datos, la cual desarrollaremos al completo. Para ello, se ha creado un plugin en WordPress, el mayor gestor de páginas web de Internet, tanto para obtener información de los usuarios que visiten el sitio web como para mostrarles las recomendaciones, las cuales podrán ser tanto personalizadas como no personalizadas.

En cuanto al proceso de minería de datos, se ha desarrollado un crawler que rastree un sitio web para obtener de él los elementos a recomendar y, además, toda esta información recogida será procesada por un índice invertido el cual calculará las similitudes entre elementos, que serán usados en el motor de recomendación. En este contexto, la construcción modular del sistema anterior y el uso del plugin aportan una gran flexibilidad y sencillez a la hora de instalarse en una o varias máquinas, dando al administrador una gran capacidad de personalización y haciendo que sea la herramienta quien se adapte a la página web y no al revés. Además de usar dos tipos de recomendación, personalizada y no personalizada, la primera mezcla los resultados de tres estrategias distintas a la recomendación personalizada que le aporta una combinación ideal y sólida que ayuda a mantener la congruencia del resultado final en todo momento.

# Palabras clave

Minería de datos, recuperación de información, sistemas de recomendación, plugin, WordPress, página web, flexibilidad

# Abstract

Data mining is a process that consists in retrieving unstructured information to obtain certain patterns in both structure and content. This process has been extended in the last century with the massive migration of all services to the Internet, mainly in those web pages where it is more interesting to know about the user.

However, it is necessary to make use of all this information in order to make it useful, and it is at this point that the recommender systems come into play. These tools are in charge, through the use of the information previously extracted, of drawing conclusions and obtaining patterns with the tastes of the users, to later recommend items (products, songs, social links, etc.) according to them.

Throughout this Bachelor Thesis we will define an architecture for a recommendation system with data mining, which we will fully develop. For that purpose, we have created a plugin in WordPress, the largest web page manager on the Internet, both to obtain information from users who visit the website and to show them the recommendations, which can be personalized or non-personalized.

As far as the data mining process is concerned, a crawler has been developed to crawl the entire website and obtain from it the elements to be recommended. In addition, all this collected information will be processed by an inverted index which will calculate the similarities between elements, which will be further used by the recommendation engine. In this context, the modular construction of the previous system and the use of the plugin provide great flexibility and simplicity when we have to install it in one or more machines, providing the administrator a great capacity for customization, since the tool is the one that adapts to the website and not the other way around. In addition to using two types of recommendation, personalized and non-personalized, the first one mixes the results of three different approaches of personalized recommendation that provides an ideal and solid combination that helps to maintain the consistency of the final result at any moment.

# Keywords

Data mining, information retrieval, recommender system, plugin, WordPress, website, flexibility

# Índice

# Listas

## Lista de ecuaciones



## Lista de figuras

# 1 Introducción

Primer capítulo donde se detallará la motivación que ha dado lugar a este proyecto, los objetivos que se buscan alcanzar y la estructura general de este documento.

## 1.1. Motivación del proyecto

Los sistemas de recomendación han dado lugar a nuevos métodos para encontrar contenido que nos sea interesante. Comenzaron como un complemento ideal a la búsqueda por consulta para espacios muy masivos y dinámicos donde el usuario desconoce qué buscar y poco a poco han ganado en importancia a la propia búsqueda tradicional. Esto se debe a que es el sistema quien toma la iniciativa y se encarga de la recuperación de información sin consulta. Para ello, se requiere de observar al usuario, detectar patrones de conocimiento y finalmente predecir y sugerir posible contenido interesante.

Para ejemplificar podemos recurrir a cualquier e-commerce, como Amazon, que nos mostrará productos similares al que estamos viendo y también productos similares relacionados con nuestro historial, e incluso, relacionado con el historial de otros usuarios que han adquirido ese producto. También, cualquier web de ocio que vive de maximizar nuestra estancia, como YouTube, Spotify, El País, etc., intentan mantenernos entretenidos el máximo tiempo posible tratando de acertar en las recomendaciones y así evitar que abandonemos su web [1]. Estos sistemas de recomendación tan profesionales contrastan con la poca capacidad que tienen la mayoría de páginas web para ponerse a la altura en cuanto a ofrecer la misma calidad de servicio que pueden brindar a sus usuarios.

En este proyecto se buscará desarrollar un completo motor de recomendación que, a partir de un plugin de WordPress fácil de instalar, sea completamente libre y gratuito, con una gran capacidad de personalización. De forma que cualquier página web, principalmente pensada para aquellas webs formadas por entradas/posts, tenga la capacidad de adquirir la funcionalidad de recomendar a sus usuarios elementos del sitio web en función de sus gustos.

## 1.2. Objetivos

Por tanto, el objetivo principal de este trabajo es el desarrollo de un sistema de recomendación funcional que sea capaz de enviar recomendaciones a una página web y que estas sean mostradas al usuario.

Un objetivo fundamental será desarrollar un plugin de WordPress que, de una forma sencilla, sea capaz de vincularse a casi cualquier página web y que aporte dos funcionalidades principales:

1.– Recoger la información necesaria del usuario, tales como sus movimientos en el sitio web.

2.– Ser capaz de mostrar las recomendaciones calculadas de una forma clara, y que el administrador sea capaz de personalizarlas y insertarlas donde le sea más interesante.

Toda la información recogida por la herramienta anterior deberá ser enviada y procesada a un sistema externo al cliente y realizar los cálculos necesarios para poder devolver recomendaciones cuando este las solicite.

Sin embargo, para poder recomendar algo es necesario conocer los elementos que se van a recomendar, en este caso posts de la página web, para ello, este sistema deberá ser capaz de recoger todo el contenido del sitio web y estructurarlo para que, junto con la información de los usuarios, poder realizar las recomendaciones correctamente.

## 1.3. Estructura del trabajo

El documento actual está dividido en varios capítulos detallados a continuación:

**Capítulo 1 - Introducción** Primer acercamiento al proyecto, donde se introducen las ideas iniciales, motivación y objetivos de este, y la estructura del documento.

**Capítulo 2 - Estado del arte y de las tecnologías** Se realiza un estudio sobre la situación actual de las tecnologías usadas en este proyecto, como los sistemas de recomendación, o los procesos de minería de datos e indexación. Además de repasar la creación de páginas web en la actualidad para conocer las posibles herramientas sobre las que construir las bases de este trabajo.

**Capítulo 3 - Diseño y desarrollo** En este capítulo se desarrolla todo el proyecto, desde explicar cómo se ha decidido modular el sistema de recomendación, establecer los distintos requisitos que debe cumplir y mostrar su arquitectura, además de explicar en profundidad tanto la tecnología usada como el desarrollo de cada módulo.

**Capítulo 4 - Instalación, pruebas y resultados** Se establecen los requisitos necesarios de cada componente para un correcto funcionamiento, además, se detalla el proceso de configuración para iniciar el mismo y se realizan las pertinentes pruebas para verificar el correcto funcionamiento del proyecto.

**Capítulo 5 - Conclusiones y trabajo futuro** Se establecen las conclusiones finales de este trabajo y se realiza un breve estudio sobre las posibles mejoras que se podrían desarrollar en el futuro.

# 2

# Estado del arte y de las tecnologías

En este capítulo se va a hablar sobre las distintas metodologías y tecnologías usadas a lo largo del proyecto, tales como los sistemas de recomendación, el proceso de minería web y la gestión y creación actual de sitios web. Además, se expondrán ejemplos de las herramientas existentes así como de las usadas.

## 2.1. Minería Web

La minería web es una de las técnicas más importantes para la recuperación de información y obtención de patrones vía Internet, se centra en la obtención de información no estructurada de forma automática tanto del contenido, como de la estructura y los usuarios [2].

Es una de las actividades principales de los motores de búsqueda, tanto para el posicionamiento de contenidos como para su indexación. Además, forma parte de cualquier actividad donde intervengan grandes cantidades de información, como las tiendas online, los servicios de publicidad y cualquier web con contenido que requiera estructurarlo, por ejemplo, Amazon, Youtube, etc.

Podemos diferenciar tres tipos de minería web en función del tipo de información que se desea obtener:

**Minería de contenido** La web no solo está formada por HTML, sino que también hay una gran cantidad de documentos, como textos, PDF, imágenes, vídeos, audios. Además, todos estos documentos contienen información adicional como son los metadatos, donde podemos encontrar autores, fecha de creación, de modificación, etc. Toda esta información se puede recopilar y estructurar para un fácil acceso o para sacar ciertas conclusiones de su procesamiento.

**Minería de estructura** A partir del uso de hipervínculos se trata de ir saltando entre páginas e ir conociendo la estructura de cualquier web. Esto puede servir para conocer la distribución y tener una idea de cómo fluye el tráfico en ella, además de encontrar posibles errores que dificulten el acceso a subpáginas importantes.

**Minería de uso** En este caso se trata de obtener la mayor cantidad de información posible sobre las acciones que realizan los usuarios, para ello, se suele hacer uso de los logs, que registran las acciones sobre la web, la fecha de acceso, nombre de usuario o IP que realizó dicha acción, entre otras cosas.

En el proceso de minería web se pueden distinguir cuatro acciones principales [3]:

1.– Obtención de la información

2.– Procesamiento

3.– Descubrimiento de patrones

4.– Análisis de patrones

Una de las mayores ventajas de esta tecnología es la capacidad para estructurar y detectar patrones en grandes cantidades de información de forma automática. Esto es muy útil para una gran cantidad de negocios online, como el marketing digital personalizado que nos aporta recomendaciones más útiles y permite un mayor conocimiento del cliente, lo que provoca una mayor fidelidad que se traduce en mayores beneficios. Sin embargo, no todo está relacionado con el comercio, también se trabaja en agencias de seguridad que permiten descubrir y pelear contra movimientos terroristas, detectar actividades criminales, como fraude fiscal y blanqueo de dinero o incluso aplicaciones en la salud, para realizar diagnósticos más exactos y mejorar la calidad de vida [4–6].

No obstante, toda herramienta tiene su parte negativa. Cuando se hace uso de estas tecnologías para procesar información de ámbito personal sin el debido consentimiento de los usuarios o sin garantizar una correcta seguridad de los datos puede dar lugar a una violación de la privacidad. No solo eso, el simple tratamiento de información privada, aunque sea tratada de forma anónima para crear un perfil digital de los usuarios, está sujeto a ciertas licencias éticas con las que no todo el mundo está de acuerdo [7].

### 2.1.1. Obtención de información: crawling

Uno de los componentes clave de la minería web es el Crawler, también llamado araña o spiderbot, en inglés. Este módulo se encarga de solicitar páginas web, descargarlas y tras esto acceder a todos los hipervínculos de esta para descargarlos y así sucesivamente.

El crawler se inicia con un grupo de URLs, llamadas semillas, puestas manualmente. Tras esto, se procede a descargar su contenido y recorrer sus hipervínculos recursivamente siguiendo una serie de políticas establecidas previamente. Estas políticas pueden ir desde normas muy generales, como aquellas marcadas en el propio HTML, por ejemplo, en las etiquetas de hipervínculos se puede establecer el atributo follow/nofollow para decirle a la araña si deberá seguir o no ese enlace. También tenemos la posibilidad de indicar en la web el archivo `robots.txt`, orientado a los bots de los buscadores, donde se establecen una serie normas como páginas anidadas que deberá rastrear y cuáles no. Además, podemos establecer en el crawler políticas más específicas, como mantenerse dentro de un dominio, detenerse al cumplirse cierta condición o evitar repetir enlaces [8].

A la par, toda esta información que se va descargando se suele filtrar. Mediante lenguajes como XPath [9], los crawlers son capaces de acceder a información concreta de archivos XML, como es el caso del HTML con la que están construidas las páginas. Después de extraer el contenido deseado se puede pasar a su análisis o almacenarlo en una base de datos hasta que se requiera su uso. En resumen, el crawling va a permitir a este proyecto conocer los contenidos con los que trabajar y poder realizar recomendaciones a partir de ellos. Por tanto, es un proceso importante, pues sin esta herramienta el proyecto estaría ciego y no podría realizar su trabajo. Hay muchos crawlers para usar en distintos lenguajes, sin embargo, se puede desarrollar uno propio usando librerías como JSoup en Java o Pyspider y Scrapy en Python.

**JSoup** Es una biblioteca de Java cuya función es la de extraer y tratar datos de la web, principalmente documentos HTML [10].

**Pyspider** Una biblioteca de Python para realizar web crawling, cuenta con una interfaz de usuario web para hacer más accesible el trabajo. Sin embargo, la comunidad que hay detrás no es la más grande y todavía está en desarrollo [11].

**Scrapy** Otra librería de Python para desarrollar un crawler, pero, a diferencia de la anterior, cuenta con una gran comunidad detrás y una muy buena documentación, además de ya ser un proyecto sólido [12].

Tanto JSoup como Scrapy aportan la calidad y facilidades necesarias para desarrollar un crawler simple y eficiente. En este sentido, la decisión de usar Scrapy se sustenta exclusivamente en la preferencia del lenguaje, Python.

### 2.1.2. Búsqueda de información: indexación

Una vez se ha recopilado una cantidad importante de información, normalmente después del crawling, es necesario tratar todos esos datos para que puedan ser útiles, aquí entra en juego el procesamiento y descubrimiento de patrones. Esto se debe a que tener los datos en crudo, tanto en archivos como base de datos, no es útil, debido a que para extraer cierto conocimiento de ellos no sería eficiente, se tardaría mucho y la respuesta podría no ser de calidad.

Con lo cual, aquí entran en juego los índices. Son estructuras de datos que permiten la recuperación de información de forma rápida. A diferencia de las bases de datos, la información es divida en términos, los cuales son filtrados, limpiados y agrupados en familias de palabras. Además, se almacena información adicional, como la frecuencia, posición y documento donde se encuentra cada término, etc. Cuando se realiza una búsqueda, se obtienen los términos y a partir de ellos, mediante ciertos cálculos de su información asociada, se devuelven los documentos más relacionados con la consulta [13].

Cualquier buscador asociado a una gran cantidad de información maneja sus datos con un índice, por ejemplo, Google, Bing, Twitter, etc. También existen ciertas herramientas que permiten la creación de tu propio índice, como Apache Lucene o ElasticSearch.

**Apache Lucene** Es una de las librerías mas importantes y potentes para la creación de índices. A parte de Java, Lucene es compatible con una gran variedad de lenguajes, como Pascal, C++, C#, PHP o Python [14].

**ElasticSearch** Es un motor de analítica y análisis de datos construido con Apache Lucene. Para usarlo e insertar, solicitar datos, crear índices y realizar búsquedas se hace a partir de su API Rest [15].

En conclusión, toda la información recogida con el crawler es inútil de primeras, hay que analizarla y procesarla, y una de las mejores herramientas para procesar datos son los índices. Se ha decidido usar Apache Lucene por su importancia y, a diferencia de ElasticSearch, poder partir de cero y crear un índice a medida de las necesidades de nuestro proyecto. En concreto, se ha usado la versión en Java [1], pues, aunque existe una preferencia de Python sobre Java, la versión en Python [2] no es más que una envoltura y por debajo trabaja la de Java, además, está peor documentada.

## 2.2. Desarrollo y gestión de sitios web

A partir de 1991 la web es públicamente accesible vía Internet formada por páginas en HTML plano, desde ese momento comenzó a usarse como un ventana para compartir información a nivel mundial. Surgieron sobre todo páginas de universidades y laboratorios científicos donde publicaban

[1] Apache Lucene for Java version: https://lucene.apache.org/core/

[2] Apache Lucene for Python version: https://lucene.apache.org/pylucene/index.html

sus investigaciones. Sin embargo, a partir de 1996 y con la expansión, cada vez más rápida, de Internet a los hogares se comenzó a ver un gran potencial comercial en ella.

Desde entonces, la web ha evolucionado y crecido exponencialmente y con ella todas las páginas que la componen. Estas han pasado de estar formadas por un conjunto de archivos HTML y manejadas por un par de personas, llamadas webmasters, a ser servicios gigantes que requieren la colaboración de varios individuos.

Debido a esta nueva necesidad surgen aplicaciones de gestión de entornos web que facilitan el trabajo haciendo que cualquier persona sin conocimientos previos de informática pueda participar en labores de creación de contenido, mantenimiento y supervisión.

### 2.2.1. Visión general

Como se ha mencionado anteriormente, con la evolución de la web, desarrollar un sitio web podía resultar una tarea compleja y que requería bastante trabajo. Desde la construcción de la página, que requería conocer diversas tecnologías como HTML, CSS, javascript, como la creación de la lógica del sitio en Python, PHP o NodeJS, hasta la tarea de actualizar sus contenidos, que a medida que crecía se complicaba la estructura. Por todo esto, surgen herramientas que ayudan con su gestión, podemos encontrar dos tipos: CMS y frameworks.

Los CMS, en inglés “Content Management System”, son aplicaciones para crear, administrar y gestionar una página web a un alto nivel, lo que permite hacer sitios sin ningún conocimiento previo, mas allá de adaptarse al CMS. En consecuencia, limitan mucho la personalización y, a menos que se añadan extensiones o se modifique el código fuente, dan lugar a entornos muy similares.

Sin embargo, los framework son un conjunto de herramientas a más bajo nivel que aportan la estructura general y los funcionamientos básicos, como la conexión de los clientes, estructura modelo-vista-controlador, conexión con base de datos, e incluso incluyen algunas capas de seguridad adicionales. Requieren, dependiendo del framework, conocimientos en Python, PHP, Javascript [16].

De hecho, se podría decir que los CMS están creados partiendo de un framework. Algunos ejemplos de los CMS más populares son: WordPress [17], Joomla [18] y Blogger [19]. En cuanto a los frameworks, podemos destacar: Symfony [20] (PHP), AngularJS [21] (Javascript) y Django [22] (Python).

Para el proyecto desarrollado se decidió trabajar sobre el CMS Wordpress, como veremos a continuación se debe a que se buscaba una gran compatibilidad con la mayoría de las páginas web, además de facilitar al máximo la instalación del proyecto en un sitio web.

### 2.2.2. Wordpress

Wordpress, o WP, nació en 2003 como una aplicación para facilitar la creación de blogs, muy populares en ese momento. Sin embargo, gracias a su filosofía y arquitectura, donde todos los usuarios podían contribuir a su evolución, impulsó tanto a este CMS que rápidamente abarcó todos los ámbitos y es usado en todo tipo de sitios web. Incluso, se ha ganado la confianza de grandes empresas, como Sony, Disney, BBC, etc.

Sus estadísticas son demoledoras [23]: El *14.7 %* del top 100 de sitios web usan WordPress; se crean *500 sitios web al día* y se publican *17 entradas cada segundo*; y existe una comunidad de *75 millones de portales* web usando WordPress. Como se puede ver en la Figura 2.1, es usado por muchos sitios web, acumulando bastante tráfico de dichos sitios. Para poner en contexto, representa

el *63.2 %* de los sitios web creados con CMS, seguido por Joomla con un discreto *4.2 %* [24].

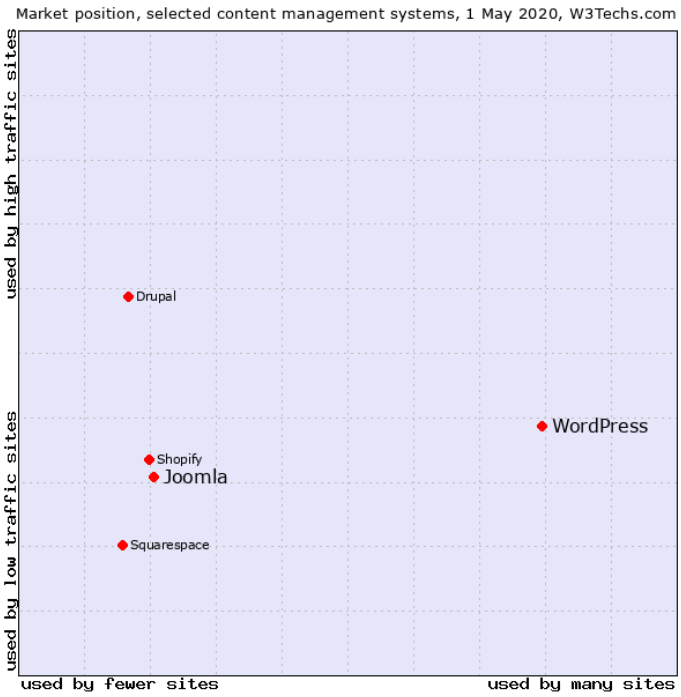


**Figura 2.1:** Posición de los gestores de páginas web en el mercado. Gráfica que relaciona el uso en relación a la cantidad de tráfico que suelen soportar, creada por w3tech.com [24].

Este éxito se debe a una gran cantidad de factores. Entre ellos, la versatilidad que aporta, mientras que otros prefieren centrarse en una finalidad determinada, WordPress busca quitarle cualquier barrera a sus usuarios. También, su sistema de código abierto que permite tener un conocimiento mayor del funcionamiento del sistema.

Sin embargo, uno de los mejores elementos de WordPress, y clave en su éxito, son los complementos, que partiendo de una instalación limpia permite transformar la página en casi cualquier servicio, como webs de entretenimiento, corporativas, blogs e incluso tiendas online.

### 2.2.3. Complementos en Wordpress

Los complementos, o plugins, de WordPress son una de las principales razones de su éxito, permite añadir casi cualquier funcionalidad, modificar el aspecto y cambiar la finalidad de la página en poco tiempo y sin conocimientos de programación, simplemente buscando en la tienda y añadiendo la extensión con un clic.

Principalmente, esto es gracias a la enorme comunidad que hay detrás de desarrolladores que WordPress permite que se ganen la vida creando para su plataforma.

Uno de los ejemplos más impresionantes del poder de estos complementos llega con la aparición de WooCommerce [25], lanzado en Wordpress para la creación de comercios en línea, que ha logrado que una herramienta para crear blogs se encuentre en el 28.24 % de las tiendas online [26].

En consecuencia, esta es la principal razón para que este proyecto sea implementado en las páginas web a través de un plugin de WordPress pues las ventajas son muy grandes, instalación rápida y sencillas desde cualquier página WP, muchas facilidades a la hora de crearlo y estar disponible para una gran cantidad de usuarios.

## 2.3. Sistemas de recomendación

A partir del siglo XXI, con la masificación de Internet a los hogares, comenzaron a surgir una gran cantidad de servicios que buscaban aprovecharse de todo este tráfico. Las tiendas, la prensa e incluso el ocio poco a poco se fueron digitalizando para tratar de ganar todos los posibles clientes. Sin embargo, al mismo tiempo que crecía rápidamente el número de usuarios también crecía una competencia feroz por el clic.

Llegan los teléfonos móviles inteligentes, todo el mundo se transforma en un posible cliente desde cualquier sitio y en cualquier momento y surge el sentimiento de la inmediatez. Los consumidores, que buscan un producto, cierta información o entretenerse no solo no quieren esperar, sino que tampoco harán mucho esfuerzo en encontrarlo hasta saltar a otra web. Las consultas explícitas, y las recomendaciones basadas en lo más visto o comprado comienzan a quedarse obsoletas pues requieren de la participación de un usuario cada vez más exigente.

Se produce una guerra contra el tiempo, por lo general este nuevo tráfico será muy efímero y para nada fiel, por lo tanto hay que capitalizar al máximo sus visitas. Para ello, es necesario que su estancia en la web tenga la mayor duración posible, tanto para monetizar al máximo su estancia como para aumentar la probabilidad de que adquiera un posible producto. Con lo cual, dentro del marco de mantener a los usuarios sin siquiera ellos buscarlo entran los sistemas de recomendación.

En los siguientes subapartados vamos a hablar brevemente de los distintos tipos de recomendación que existen, recomendación no personalizada y personalizada, en función de si se usa información del usuario, así como de los distintos algoritmos usados.

### 2.3.1. Recomendación no personalizada

En la recomendación no personalizada no se usa información acerca del usuario por lo que hay que trabajar con el entorno. Por ello, se basa en calcular la similitud del ítem actual, que en el caso de este proyecto serán los posts de un sitio web, con la del resto de items y obteniendo los N elementos más similares. Como es una similitud, se va a calcular mediante la fórmula de similitud coseno que calcula el coseno de los vectores de dos ítems para obtener su similitud.

$$sim(i,j) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\| \cdot \|\vec{j}\|} \tag{2.1}$$

En este caso, al tratarse de entradas de una página web, es decir, de documentos, sus vectores están formados por las frecuencias de las palabras que los componen. Tiene como desventaja recomendar elementos del mismo tipo sin innovar, esto se debe a usar exclusivamente el contenido, lo que provoca que los elementos resultantes sean muy parecidos en temática [27].

### 2.3.2. Recomendación personalizada

En cuanto a la recomendación personalizada, es necesario usar cierta información del usuario, para ello, necesitamos registrar algunas acciones de estos, desde información más relevante como puntaciones explícitas, pero que requieren más participación, hasta un simple log con las páginas que visita, este será el caso de este proyecto. A partir de los datos, podemos obtener unas puntuaciones finales de cada usuario para cada ítem, por ejemplo, en el caso de que un usuario no haya interactuado nunca con un item esta puntuación será de cero. Y finalmente, con esto, obtener una matriz que

relacione usuarios (filas) con ítems (columnas), como se puede observar en la figura 2.2.

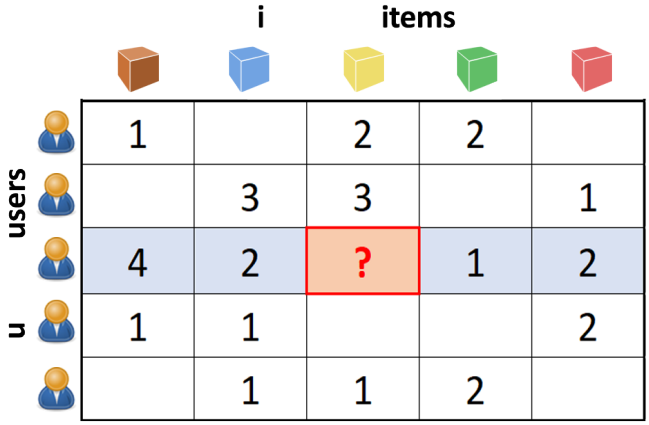


**Figura 2.2:** Matriz de puntuaciones de usuarios e items.

Esta matriz servirá para el cálculo de los ratings con los distintos tipos de algoritmos usados, que veremos a continuación.

### 2.3.3. Algoritmos de recomendación personalizada

Para la recomendación personalizada existen una serie de algoritmos para obtener una puntuación. Dependiendo de cómo calculemos la similitud podemos encontrar dos tipos, filtrado colaborativo y basado en contenido [28, 29].

#### Filtrado colaborativo

El filtrado colaborativo usa exclusivamente la matriz de usuarios-items, figura 2.2, e ignora el contenido de los elementos, podemos encontrar dos tipos basados en la misma técnica, KNN basado en usuarios y basado en ítems.

KNN basado en usuarios: Se obtienen los N usuarios más similares (usando los vectores de usuarios de la tabla) y a partir de ellos se calcula la puntuación con el ítem. Es decir, la puntuación que se espera de un usuario para un ítem depende de la puntuación que tienen los usuarios más similares a él.

$$r(u,i) = c \sum_{v}^{users} sim(u,i) \cdot r(v,i) \tag{2.2}$$

KNN basado en ítems: Igual que basado en usuarios, pero aplicado a ítems. Sin embargo, la similitud coseno entre ítems no se calcula usando sus vectores de contenido sino los de la matriz.

$$r(u,i) = c \sum_{v}^{items} sim(i,j) \cdot r(u,j) \tag{2.3}$$

Una de las ventajas de este tipo de recomendación es la capacidad de obtener resultados novedosos, que permiten al usuario no cerrarse en un tema, debido a que no se usa el contenido para nada,

sino las interacciones de toda la comunidad de usuarios e items [29].

### Basado en contenido

Para este tipo se recomienda a los usuarios sin mirar a los demás, sigue siendo personalizada pues se sigue usando el historial de estos, sin embargo, vuelve a ocurrir el problema de estancamiento en un tema porque se calcula usando exclusivamente el contenido de los items del historial del usuario.

Principalmente existen dos tipos, centroides y KNN, aunque en este trabajo solo se ha hecho uso del KNN. Este método usa la misma fórmula que la de KNN basado en ítem de filtrado colaborativo, ecuación 2.3, con una diferencia: para la similitud coseno entre ítems sim(i,j) se usa el contenido del ítem, es decir, la misma que en la recomendación no personalizada, ecuación 2.1.

A este respecto, podemos concluir que la recomendación personalizada que se ha usado en este proyecto se sustenta en el uso de tres métodos de recomendación, los dos de filtrado colaborativo y KNN basado en contenido. De esta forma, se busca atacar al problema desde distintos flancos para que la mezcla logre ser lo más efectiva posible.

# 3 Diseño y desarrollo

La metodología usada para el desarrollo de este proyecto ha sido la incremental iterativa. Es decir, se ha dividido el proyecto en funcionalidades y en cada iteración se desarrollaba cada una o se mejoraba otra.

La idea de dividir el proyecto en funcionalidades, o módulos, parte del objetivo principal de buscar la mayor generalidad posible para el proyecto y que este pueda ser adaptado a multitud de entornos. Además, es una buena forma de dividir un problema complejo en varios más pequeños y sencillos.

Recapitulando, este proyecto propone el desarrollo de un plugin de WordPress que implemente, en la página donde está instalado, un sistema con la funcionalidad de recopilar información, realizar minería de datos para explotar esa información y recomendar a los usuarios. Está pensado principalmente para cualquier tipo de página donde se publiquen noticias, entradas o algún tipo de publicación la cual mantenga una misma unidad estructural. Por ejemplo, las noticias de un periódico digital, la página de cualquier video de Youtube, etc.

Por ello, se ha dividido el proyecto en cuatro módulos que abarcan todas las áreas para este sistema:

**Plugin de WordPress** Es uno de los módulos más importantes, se encarga de la comunicación entre el cliente y el sistema. Este módulo se dedica a recopilar las acciones del usuario, es decir, enviar logs de información con la página actual, el destino y el identificador de este. Además, solicita recomendaciones que mostrará, mediante herramientas de WordPress, en el cliente.

**Módulo núcleo/principal** Este subsistema se encarga de conectar el resto de los elementos. Recibe logs del plugin y posts del crawler y los almacena en la base de datos, al mismo tiempo, envía documentos al índice. Pero no solo se encarga de almacenar y enviar información, sino que también se encarga de solicitar similitudes y hacer aprendizaje automático para recomendar y enviar estas recomendaciones al plugin según se vayan requiriendo.

**Módulo crawler** Para poder recomendar entradas de un sitio web, el sistema debe contener toda la información posible de ese sitio. Para ello, se usa un crawler, en este caso se trata de una aplicación que se encarga de recorrer todas las páginas del sitio web y, tras aplicar un filtro para solo quedarse con las entradas/posts, se envían al módulo principal.

**Módulo del índice** Este último módulo se encarga de darle un uso a todas las entradas/posts que se encuentran en la base de datos. Por lo tanto, se crea un índice que permite manejar toda esa información de forma eficiente y se calculan todas las similitudes posibles entre documentos. A su vez, se mantiene a la espera de recibir peticiones de similitudes para enviarlas.

## 3.1. Análisis de requisitos

A continuación, se detallará los requisitos que debe desempeñar el proyecto, en concreto, veremos los distintos requisitos, tanto funcionales como no funcionales, que deberá cumplir el proyecto para su correcto funcionamiento. Además, estos requisitos están divididos en los distintos procesos de los que se encarga el sistema y que conforman los subsistemas en los que está estructurado este proyecto.

### 3.1.1. Requisitos funcionales

#### Requisitos generales del sistema

**RF-1.–** El sistema deberá ser capaz de aportar recomendaciones a través de un plugin de WordPress en cualquier página web (con entradas/posts) desarrollada en este CMS.

**RF-2.–** El plugin será la única vía de comunicación entre usuarios y sistema, por tanto, también deberá ser capaz de obtener información de estos.

**RF-3.–** Será necesario un crawler que obtenga las páginas web y así conocer el contenido del entorno donde se mueven los usuarios.

**RF-4.–** El sistema deberá dividirse en módulos que se comunicarán entre ellos, dividiendo la funcionalidad, para facilitar la implementación y optimizando el diseño y la gestión de memoria.

#### Requisitos para la obtención de datos

**RF-5.–** El crawler solo deberá recoger las páginas que contengan los elementos especificados en la configuración. Esto permite recolectar solo un tipo de páginas concreto, como por ejemplo, solo las entradas de una web de publicaciones evitando páginas intermedias, menús, página principal.

**RF-6.–** El crawler evitará devolver duplicados, es decir, no pasará por la misma página dos veces.

**RF-7.–** Antes de realizar el crawling, se deberán eliminar todas las entradas almacenadas previamente en la base de datos. De esta forma se evita la creación de duplicados involuntariamente o de mantener entradas antiguas que ya han sido eliminadas de la web.

**RF-8.–** El plugin será capaz de crear un log cada vez que un usuario visite una página del sitio con el id del usuario, la página de donde viene y la página destino.

**RF-9.–** Los movimientos de los usuarios no logueados se representarán con un id especial general para todos.

**RF-10.–** El perfil de un usuario estará formado por las páginas que visita y la frecuencia con la que lo hace.

#### Requisitos para el manejo de datos

**RF-11.–** Los datos obtenidos del crawler serán procesados y analizados para crear un índice.

**RF-12.–** Los datos obtenidos de los usuarios serán usados para la recomendación.

**RF-13.–** Durante la creación del índice se calcularán las estadísticas necesarias para el resto de módulos, cómo las similitudes entre las entradas/posts de la web.

**RF-14.–** Para el cálculo de similitudes se usará la similitud coseno y para el vector de los documentos se usará tf-idf.

**RF-15.–** El log de los usuarios se filtrará para eliminar aquellos en los que el usuario haya visitado páginas que no correspondían con documentos.

**RF-16.–** Se creará una matriz formada por usuarios (filas) y documentos (columnas) siendo la puntuación de un usuario a un documento la frecuencia de veces que lo ha visitado. Esta matriz se usará para la recomendación personalizada.

#### Requisitos de recomendación

**RF-17.–** Para iniciar el proceso de recomendación es necesario que el índice esté creado con antelación.

**RF-18.–** Además, para poder recomendar un administrador deberá iniciar el entrenamiento del sistema de recomendaciones. Este proceso debe hacerse una vez el índice está creado.

**RF-19.–** Para activar el entrenamiento deberá poder realizarse mediante una llamada remota, por ejemplo, con un método de una API REST.

**RF-20.–** Habrá dos tipos de recomendaciones: personalizada y no personalizada.

**RF-21.–** La recomendación personalizada estará formada por una mezcla de tres recomendaciones. evitando repetición de elementos en el resultado final: Filtrado colaborativo, KNN basada en ítems; filtrado colaborativo, KNN basada en usuarios; y Basado en contenido, KNN basado en ítems.

**RF-22.–** La recomendación no personalizada se basará en un ranking con las entradas/posts más similares a la actual.

**RF-23.–** El plugin mostrará las recomendaciones mediante un widget.

**RF-24.–** El widget se podrá configurar para elegir el tipo de recomendación (personalizada o no) y el número de elementos a devolver (cutoff).

**RF-25.–** Si el usuario no está logueado un widget configurado con recomendación personalizada deberá devolver recomendación no personalizada, pero con las posiciones randomizadas (para evitar devolver lo mismo que un widget configurado para ello).

**RF-26.–** Cuando se va a cargar una página, si el usuario está logueado el plugin enviará el nombre del usuario y el cutoff al sistema para que este le devuelva las recomendaciones personalizadas.

#### Requisitos para la comunicación entre sistemas

**RF-27.–** Habrá un módulo central, que hará de puente entre las comunicaciones de todos los demás módulos. De esta forma se obtiene una mejor organización y una arquitectura más limpia.

**RF-28.–** Habrá que crear una interfaz REST API con métodos para obtener, insertar y eliminar datos, solicitar recomendaciones y similitudes y crear o leer índices. Por lo tanto, esta REST API será necesaria en todos aquellos módulos que hagan alguna de la funcionalidad anterior.

**RF-29.–** Una vez finalice el crawler habrá que llamar al sistema para iniciar la indexación de los datos.

### 3.1.2. Requisitos no funcionales

**RNF-1.–** **Persistencia de datos**: Para almacenar los logs o posts antes de ser usados es necesario usar una base de datos relacional compatible con Django REST.

**RNF-2.–** **Compatibilidad**: El sistema deberá funcionar sobre cualquier sistema operativo compatible con Python 3 y Java 8.

**RNF-3.–** **Especificaciones**: El ordenador, u ordenadores, sobre los que se monte el sistema deberán tener las especificaciones técnicas suficientes para que este funcione correctamente, sin que se produzcan fallos, cortes, más CPU o más RAM.

**RNF-4.–** **Velocidad**: Es requisito fundamental que el sistema de recomendación no empeore el tiempo de carga de la página web.

**RNF-5.–** **Comunicación**: Debe existir comunicación constante entre todos los módulos, ya estén en el mismo entorno o montados sobre distintos ordenadores.

**RNF-6.–** **Lenguaje**: El lenguaje de la página web no debería ser un problema para el correcto funcionamiento del sistema, por lo que debería funcionar bien sobre cualquier alfabeto latino, en concreto español o inglés.

## 3.2. Arquitectura del proyecto

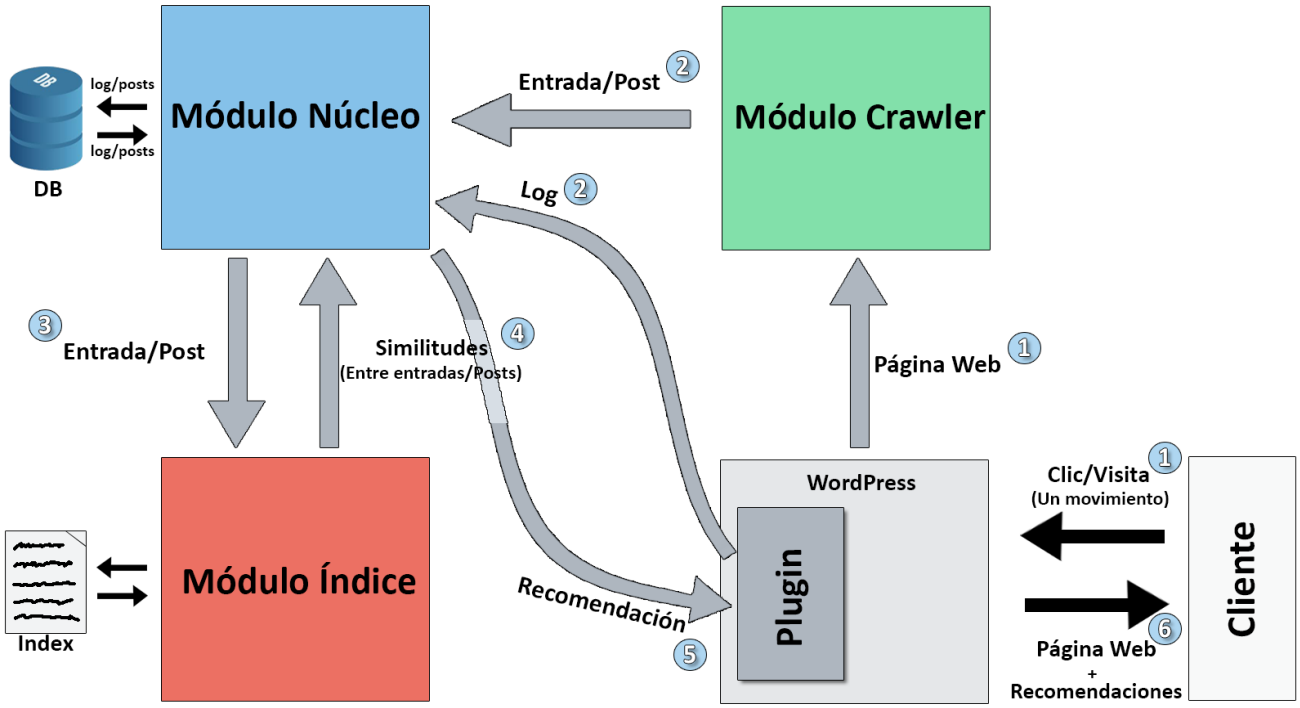


**Figura 3.1:** Arquitectura del proyecto. Los marcadores numéricos sirven para dar una idea de la secuencia de los procesos del sistema y de las llamadas que pueden realizarse a la vez (indicadas usando el mismo número).

En la figura 3.1 superior, a partir de los marcadores numéricos, se puede ver que primero se obtienen los datos, los logs de los clientes por parte del plugin y cada página del sitio web por parte del crawler. Después, los logs y posts son enviados al núcleo y serán almacenados en la base de datos. Tras esto, el módulo índice puede comenzar a indexar, solicitando al núcleo los documentos y, una vez haya indexado, podrá aportar similitudes. Finalmente, una vez se haya entrenado, se podrán enviar recomendaciones tanto personalizadas como no personalizadas al plugin, que se encargará de mostrarlo al cliente.

Como podemos ver, el módulo núcleo es el elemento principal del proyecto, actúa de puente de comunicación para el resto del sistema, hace de almacén de los logs y posts y, además, entrena las recomendaciones y se encarga de enviarlas.

Para las comunicaciones entre módulos se usa HTTP utilizando la arquitectura REST, lo que permite una comunicación remota entre distintas aplicaciones que pueden estar escritas en cualquier lenguaje corriendo bajo cualquier sistema operativo. En concreto, los subsistemas que contienen una API Rest son el módulo núcleo y el módulo índice, el resto se comunican con ellos haciendo uso de dichas APIs.

Una de las ventajas de esta arquitectura es la libertad a la hora de ser instalado, desde tener el proyecto en una solo máquina, como es el caso de las pruebas, hasta ser desplegado cada módulo en una máquina distinta, mientras exista comunicación entre ellas. Por ejemplo, si el ordenador tiene suficiente hardware que lo soporte, el módulo núcleo e índice pueden ir en la misma máquina, mientras que el plugin se encontrará junto al WordPress en cualquier host de páginas web, y por último, el crawler podría lanzarse en cualquier ordenador del que disponga el administrador (siempre y cuando exista conexión con la web y el módulo núcleo), incluso en el mismo que el índice y núcleo.

Se hablará más en detalle de cada módulo, su funcionamiento y comunicación a continuación.

### 3.2.1. Plugin

#### Explicación de la tecnología usada

Como ya mencionamos en el estado del arte, WordPress es uno de los gigantes actuales para la gestión y creación de páginas web y ha conseguido ser utilizado en una gran cantidad de sitios. Por ello, la decisión de construir este proyecto bajo este entorno de trabajo estaba claro, más aún cuando se buscaba ser lo más general posible.

Sin embargo, a pesar de que WordPress sea de código abierto y sea sencillo editar sus archivos, pues está construido sobre un framework en PHP, modificar la funcionalidad desde su propio esqueleto puede suponer un problema. Cuando se tenga que actualizar o al cambiar de tema se pueden perder los cambios o simplemente dejar de funcionar. Además de tener en cuenta que la simple complicación de tener que tocar código va en contra de la filosofía de hacer un sistema lo más simple posible de instalar.

Aquí entran en juego los ya mencionados plugins de WordPress, que permiten añadir a cualquier instalación funcionalidad añadida simplemente descargando y activando la extensión, sin necesidad de tocar el código del CMS. Para ello, WordPress cuenta con ciertas funciones, a priori decorativas, llamadas Hooks, a las que podemos engancharnos para añadir ahí nuestro código. Existen otros métodos para añadir funcionalidad, como los shortcodes, aunque no han sido necesarios.

Para engancharse a los hooks simplemente hay que hacer uso de una de las dos funciones según el hook que se utilice:

**add_action (nombre_del_action, función_para_añadir)** Para añadir funcionalidad propia a partir de un action hook. Este será el tipo de enganche usado para el proyecto y sirve simplemente para añadir funcionalidad.

**add_filter (nombre_del_filter, función_para_añadir)** Los filter hooks sirven, como su nombre indica, para filtrar información, es decir, recibir información y, a través de nuestra funcionalidad, modificarla. Para localizar todos los hooks disponibles podemos bucear en el código en su búsqueda o visitar el siguiente sitio web [1] que dispone de un listado con todos ellos.

También se van a usar uno de los elementos más comunes de WordPress, los widgets, son trozos de web, con formularios, imágenes, listas, que se pueden anclar a determinadas zonas del sitio web y que aparezcan siempre ahí en todas las páginas del sitio. De hecho, WordPress cuenta con una sección en el menú de configuración donde se puede elegir en qué localización colocar qué widget e incluso, ciertos widgets admiten ser configurados.

Además, al estar en un entorno que utiliza PHP se pueden usar, incluso dentro de los plugins, variables globales como las siguientes:

**$_SESSION** Para acceder a variables de sesión, es decir, a la información de las cookies.

**$_REQUEST** Para acceder a todo lo relacionado con una petición, cabeceras, URL, contenido del GET y del POST.

**$_SERVER** Información del servidor.

[1] Hookr website: `http://hookr.io/`

Aunque hay muchas más, las anteriores servirán para que la extensión creada pueda obtener cierta información necesaria como veremos a continuación.

### Explicación del diseño y desarrollo del plugin

Para empezar, con la primera iteración de este módulo se comenzó con la obtención del log del cliente y su envío al módulo núcleo. Para ello, se creaba una función que se llamaba en cualquier momento durante la creación de la respuesta del usuario por parte de WordPress y recogía la IP, la URL origen y URL destino gracias a la variable $_SERVER.

Sin embargo, la IP no nos aporta prácticamente casi ningún tipo de identificación por parte del usuario, es una información muy ambigua debido a que varios individuos de una misma red la comparten, y además, cambia cada cierto tiempo, con lo cual, es necesario un inicio de sesión en la web para poder distinguir y hacer en un futuro recomendaciones personalizadas.

Para obtener el id del usuario se extrae de la cookie de WordPress, wordpress_logged_in_ + [hash], es decir que cualquier plugin que añada un inicio de sesión y use esa cookie será compatible con este proyecto. Por lo tanto, para acceder al identificador del usuario contenido en la cookie mencionada anteriormente se puede hacer a través de la variable $_COOKIE.

Para realizar la petición HTTP al módulo núcleo se empezó por usar el método file_get_contents. No obstante, esta función nos deja poca versatilidad, además, si la petición falla, esto se verá reflejado en la web con una advertencia. Así pues, se ha decidido usar curl en PHP [30], es una biblioteca que permite abrir comunicación con numerosos tipos de servidores y protocolos y ofrece mucha más versatilidad: tiene una gran capacidad de configuración, tanto de la petición como de la respuesta, se pueden estudiar las cabeceras, el contenido, tanto de POST como GET, y da igual el código de error que se devuelva ya que no salta una advertencia.

Después de tener toda la funcionalidad de recogida de información y envío surgía un problema: cuando el usuario solicitaba una página web, antes de cargarse totalmente se ejecutaba la funcionalidad explicada arriba, en consecuencia, si había algún tipo de problema de comunicación con el módulo núcleo la web tardaría demasiado en cargar, provocando una mala experiencia a los visitantes.

Por ello, hay que ejecutar esa parte de forma asíncrona, para lo cual hay varias opciones, una de ellas es la de tener toda la funcionalidad en JavaScript, lo que da lugar a un problema de poca abstracción hacia los usuarios, pues podrían ver cierta información sensible. Finalmente, el método elegido es ejecutar de forma asíncrona código PHP llamándolo a partir de AJAX.

Para hacer esto último, primero hay que insertar el script con AJAX en la web, se puede usar el hook wp_head para que se añada en la cabecera de la página. Después, como estamos usando WordPress no basta con llamar al script PHP a partir de su ruta, sino que hay que hacer uso del archivo **admin-ajax.php** de WP el cual se encarga de recibir peticiones AJAX y llamar a unos hooks a los cuales previamente hemos enganchado la funcionalidad del log: **wp_ajax_ + [nombre único]** y **wp_ajax_nopriv_ + [nombre único]**, este último registra los logs cuando el usuario no está registrado, si lo obviamos no se enviarán logs de sus movimientos.

En este caso, ese nombre único es `create_log_entry`, por tanto, cada vez que llamemos a `www.example.com/admin-ajax.php?action=create_log_entry&extra=...` se ejecutarán los hooks `wp_ajax_create_log_entry` y `wp_ajax_nopriv_create_log_entry`. Además, se pueden añadir todos los argumentos que queramos a esa petición GET pues serán accesibles a partir de la variable $_REQUEST. En este caso se pasarán como argumentos adicionales la web origen y destino.

También se pasará un argumento adicional, nonce, para agregar una capa de seguridad. Su signi-

ficado es “number used once” y es una especie de checksum que sirve para evitar peticiones indeseadas.

Una vez se ha acabado con la parte de recopilación de información de los clientes, toca desarrollar los mecanismos necesarios para mostrar recomendaciones. Para esta tarea se ha decidido usar widgets, pues aportan muchas facilidades como la de mantenerse siempre en un mismo sitio de la web.

Primero, hay crear una clase que extienda de **WP_Widget**. Aquí hay dos métodos principales para el funcionamiento del plugin:

**Método form** En este método se crea el formulario de configuración del widget, que se podrá ver al ser agregado en el menú de WordPress. En este caso, se van a añadir tres elementos: un título, el tipo de recomendación y el número de elementos a devolver.

**Método widget** Es el método principal, se encarga de crear la vista del usuario, por tanto, toda la funcionalidad de recomendación se llama aquí.

Para la recomendación se ha creado una función principal, **getRecomendationHTML**, que recibe dos parámetros de la configuración, el tipo de recomendación y el número de elementos a devolver. Este método llamará a una de las dos funciones creadas, **getRecomendationPersonalized** y **getRecomendationNotPersonalized**, las cuales se encargan, mediante la API Rest del módulo núcleo, de solicitar las recomendaciones correctamente y devolver el listado al método principal que lo insertará con marcado HTML.

Además, en el caso de tener un widget configurado con recomendaciones personalizadas y no estar dado de alta en la web, se devolverán diez recomendaciones no personalizadas en orden aleatorio, esto se hace para no obtener un listado vacío y que sea diferente al de un posible widget con recomendación no personalizada, tal como se indicaba en el requisito **RF25**.

### 3.2.2. Módulo núcleo

#### Explicación de la tecnología usada

Como ya se ha mencionado, este es uno de los módulos que cuenta con una API REST con la que el resto del sistema interactúa. Para ello, se va a hacer uso de uno de los frameworks mas completos y potentes de Python, Django, en concreto se va a usar la versión de **Django Rest** enfocada exclusivamente para el desarrollo de API’s REST y que cuenta con todas las facilidades para ello.

Para iniciar este framework, primero hay que instalar el paquete de python con **pip install djangorestframework**. Una vez hecho, podremos crear un proyecto Django con **python manage.py startapp [nombre del modulo]**.

Una de las filosofías de Django es la de tener una estructura muy modular, separando la lógica en pequeñas aplicaciones, llamadas apps. Con lo cual, estos submódulos se pueden crear mediante el comando: **python manage.py startapp [nombre del módulo]**.

Al final, tendremos una serie de archivos en la carpeta principal, como settings.py o urls.py, y en la carpeta de cada app, como models.py, views.py, serializers.py, etc., que veremos más en detalle a continuación.

**Settings** Es el archivo de configuración del proyecto Django donde se encuentran las variables globales de este. Podemos configurar desde los hosts donde permitir el despliegue de este módulo, hasta establecer cual serán las bases de datos, tanto local como remota, que

se van a usar. En concreto, nos centraremos en INSTALLED_APPS, una lista donde habrá que añadir rest_framework y el nombre de las apps que creemos.

**Urls** Se encarga de redirigir las peticiones recibidas, según su URL, hacia su controlador.

**Routers** Es una clase que simplifica el trabajo que habrá que hacer en Urls, permite declarar rápidamente todas las rutas necesarias para un controlador mediante expresiones regulares, asociando automáticamente los métodos HTTP con métodos de este, aunque también permite hacerlo manualmente.

**Models** Sirve para definir la estructura de los elementos que haya que crear, sus campos, y cómo se almacenarán en la base de datos.

**Serializers** Es el traductor de la aplicación, sirve para traducir los elementos de una base de datos, a partir de sus modelos, a archivos JSON o XML. Y viceversa.

**Views** Es el controlador de las peticiones que se reciben sobre una cierta URL. Aquí se añadirá la lógica deseada y dependiendo del tipo de método HTTP usado, ejecutar una funcionalidad u otra, solicitar o enviar elementos al modelo por medio de serializers, y devolver la respuesta pertinente.

**Django Rest**, además de la interfaz web para acceder a la base de datos del framework Django, cuenta con otra donde consultar los métodos y modelos creados. Basta con acceder a la IP y puerto donde está desplegado.

Por lo general, se ha usado el esqueleto básico que aporta el framework, las clases que proporciona y su documentación, pues es suficiente para tener la funcionalidad esencial requerida. No obstante, a continuación, se explicarán aquellos elementos que han requerido de más investigación y desarrollo.

Para la vista se ha decidido extender la clase **ViewSet**, esta clase cuenta con todas las funciones necesarias para manejar cualquier consulta sobre un modelo, es decir, consultar o eliminar un elemento, listar o eliminar el modelo, crear y actualizar. Estos métodos de python ya están enlazados con su respectivos métodos HTTP, por tanto, al usar un router con esta clase se creará toda la tabla de rutas de forma automática y cada método enlazado correctamente.

Además, se puede añadir más funcionalidad a la ViewSet, bastaría con usar encima de la función deseada la etiqueta **action()** con el parámetro methods=[], una lista con todos los métodos HTTP a los que va a responder esa función. De esta forma se creará una URL que contendrá el nombre del método al final y enrutará las peticiones, con el método HTTP correcto, a esa funcionalidad.

Sin embargo, estas funciones están por defecto vacías y habrá que añadir la lógica, si se deja alguna vacía Django detecta que esa funcionalidad no ha querido ser implementada y por tanto no estará disponible para enrutar.

También definiremos un router de forma manual para poder crear URL más personalizadas y pudiendo elegir que método HTTP usar en cada caso. Para ello, hay que crear una clase que extienda de **SimpleRouter** y dentro crear una lista llamada **routes** con todos los enrutamientos deseados. Estos enrutamientos son instancias de la clase **Route** donde habrá que insertar los siguientes argumentos:

**url** Contiene un string, que puede ser una expresión regular, con la URL que será enrutada.

**mapping** Mediante un diccionario, donde las claves son los métodos HTTP y el valor la función a ejecutar, mapeamos qué hará el framework en cada caso cuando llamemos a la url anterior con distintos métodos HTTP. Los que se dejen vacíos simplemente no estarán habilitados para esta llamada.

**name** El identificador de este enrutamiento.

**detail** Pensado para solicitudes para listar o solicitar un elemento. En general estará a false.

**initkwargs** Un diccionario de cualquier argumento adicional que se deba pasar al instanciar la vista.

Esta clase que se acaba de crear, con el nombre que hayamos deseado, habrá que instanciarla y registrarla con una vista que cuente con los métodos establecidos en el mapeo. Y una vez que se incluya este router a la lista de urls, ya estará disponible para ser usados.

Para la parte de recomendación se va a hacer uso de la teoría desarrollada en el apartado 2.3.

#### Explicación del diseño y desarrollo del módulo

Este módulo, al ser el principal, cuenta con una gran funcionalidad detrás. Sin embargo, se va a dividir esta funcionalidad en dos bloques claramente diferenciados, un bloque centrado en la recolección y proporción de datos, y otro bloque centrado en la recomendación.

Por ello, haciendo uso de la división en aplicaciones que incita Django, mencionada anteriormente, crearemos dos, **collectors_app** para el bloque de datos y **recommend_app** para el bloque de recomendaciones. A continuación, se explicará más en detalle cada bloque.

Bloque **collectors_app**

Este bloque, pensado para la transferencia de información, usa principalmente las funcionalidades básicas que ofrecen por defecto el framework. Contiene dos modelos, vinculados con sus respectivas tablas a la base de datos, para manejar los dos tipos de información que se pueden recibir:

**Modelo de Log** Almacena las acciones de los usuarios y se usará para crear las recomendaciones personalizadas. Cada tupla, enviada por el plugin, está formada por los siguientes campos: id usuario, IP, URL origen, URL destino, host e instante de tiempo. Como veremos más adelante, la versión de este proyecto solo usa la URL destino de la información recogida.

**Modelo de Post** Se encarga de almacenar todas las entradas recibidas por parte del crawler, en principio, deberían estar todos lo posts de la web donde se ha montado el sistema. Cada tupla está formada por: la ruta del post, host, titulo, autor, contenido, categoría y fecha de publicación. Como veremos en el módulo del índice, este se encargará de solicitar estos posts y, en concreto, en esta versión del proyecto, usará la ruta, título, contenido y categoría.

Los métodos de las API REST implementados en este bloque son:

**/api/logs/** Método principal para manejar el modelo de Log, si se usa con GET devolverá un listado JSON con todos los elementos existentes para este modelo. Si se usa con POST y se envía con una tupla JSON, la insertará en la base de datos.

**/api/logs/<id>** Este método solo está disponible con GET y sirve para obtener, en formato JSON, una tupla concreta de los logs. El identificador es un valor numérico que añade Django a cada modelo como clave primaria.

**/api/posts/** Método principal para manejar el modelo de Post, si se usa con GET devolverá un listado JSON con todos los elementos existentes para este modelo. Si se usa con POST y se envía con una tupla JSON, la insertará en la base de datos.

**/api/posts/<id>** Solo está disponible con GET y sirve para obtener, en formato JSON, una tupla concreta de los posts. El identificador es un valor numérico que añade Django a cada modelo como clave primaria.

**/api/posts/remove/** En este caso, se trata de un método adicional que sirve para vaciar la tabla de posts al completo. No ha sido creado por defecto, sino que se ha usado la etiqueta

**action**, mencionada anteriormente, sobre una función llamada **remove**. Funciona exclusivamente con **DELETE** y es usado por el crawler, antes de iniciar, para vaciar de la base de datos todos los posts e insertarlos de cero en el sistema.

Bloque **recommend_app**

Lo primero ha sido la creación de una clase que se encarga tanto de entrenar, como de calcular y aportar las recomendaciones. Para ello, puesto que este bloque no dispone de modelos, se ha aprovechado el archivo de **models.py** para crear ahí dicha clase, llamada **Recommendation**.

**Recommendation** es la clase principal de este bloque y aporta toda la funcionalidad necesaria para que casi todos los métodos API REST relacionados con la recomendación la usen. Crear, usar, almacenar y modificar una matriz de valoraciones durante el entrenamiento, y devolver recomendaciones tras ciertos cálculos en tiempo de consulta, es un resumen de su funcionalidad. A continuación, se van a definir las funciones principales:

**__init__** Para inicializar la clase es necesario pasarle la lista de usuarios y posts que van a participar. Con ellos, creará la matriz vacía de valoraciones asociando un identificador a cada usuario y post, siendo los primeros las filas y los otros las columnas. Además, se inicializarán todos los atributos.

**create** Es uno de los métodos más importantes, sirve para entrenar el sistema cargando la información de los logs en la matriz. Recibe una la lista con todos los logs, es decir, todas las acciones de los usuarios, y crea las valoraciones, las cuales son el número de veces que se visitó un post. Esta lista recibida tiene como formato una lista de diccionarios los cuales tienen las claves user (id del usuario) y dst (post que visita). Además, en este momento también se calcula el vecindario, es decir, dos listas de diccionarios donde se almacenan las similitudes que hay entre usuarios y posts, respectivamente.

**add_and_refresh** Igual que create, pero solo para una tupla de log.

**create_neighborhood** Se encarga de crear el vecindario de similitudes tanto para los usuarios como para los posts. Para ello, se usa la fórmula de similitud coseno explicada en el estado del arte, sección 2.3, donde el vector de los usuarios es su fila de la matriz y el de los posts es su columna de la matriz.

**getRatingUserBased y getRatingUserBasedCutOff** Dado un usuario y un post, obtiene su puntuación basado en usuarios, como se vio en el apartado 2.3 del estado del arte.

**getRatingItemBased y getRatingItemBasedCutOff** Igual que las funciones anteriores, la única diferencia es que en este caso se basan en items.

Existe una bandera, **content_based**, que al estar en **False** calculará todo basado en ítems de filtrado colaborativo, sin embargo, al poner a **True**, hará los cálculos basado en ítems de basado en colaborativo.

En este último caso, para obtener las similitudes entre ítems basadas en el contenido es necesario solicitarlas al módulo del índice, no obstante, esto daba lugar a que las recomendaciones con este tipo tardasen ente 10-20 segundos, por lo que ahora, al crear el vecindario también se creará un vecindario para las similitudes basadas en contenidos y así, durante la ejecución, las recomendaciones ya están cargadas en un diccionario y conseguimos tiempos de respuesta mucho más cortos.

Una vez tenemos la clase creada es tiempo de desarrollar los métodos de la API REST necesarios para usar las recomendaciones. Como se quieren métodos muy personalizados se ha tomado la decisión de desarrollar una clase de router personalizada que extienda de **SimpleRouter**, como se explicó anteriormente.

Los métodos de las API REST implementados en este bloque serán definidos a continuación.

Los dos siguientes métodos no tienen relación directa con las recomendaciones, sin embargo, son necesarios para que estas funcionen correctamente. De hecho, su funcionalidad se basa simplemente en llamar a métodos iguales de la API REST del índice, pero, como queremos esta estructura donde haya un núcleo que haga de puente para todas las comunicaciones, se requiere de este tipo de implementaciones.

**/api/index/create** Usa GET. Hace una llamada al método /index/create del módulo índice que se encarga de crear el índice y dejarlo listo para poder ser usado y recoger similitudes.

**/api/index/read** Usa GET. Al igual que antes, hace una llamada al método /index/read del módulo índice que se encarga de leer un índice ya creado y listo.

A continuación, veremos los tres únicos métodos enfocados a la recomendación:

**/api/recommendation/train** Usa GET. Al igual que se necesitan los métodos anteriores para tener listo el índice, se necesita este para tener listas las recomendaciones. Por ello, se filtran los modelos para obtener todos los usuarios y posts que hubiese en la base de datos en el momento de la llamada y se crea una instancia de la clase **Recommendation**. Como cada vez que se hace una llamada esta es independiente al resto, es necesario que la instancia de recomendación sea una variable global para que pueda ser accesible por todas las llamadas que se hagan.
Después, se llama a la función **create** pasándole todos los logs con el formato requerido, mencionado anteriormente. Una vez finalice, devolverá el código de estado **201 CREATED**.

**/api/recommendation/nopersonalizada** Usa el método POST con formato JSON con los parámetros **url** y **cutoff**, el primero para obtener de él otros posts similares y el segundo es el número de elementos a devolver.
Esta información se extrae y se envía, en formato **x-www-form-urlencoded**, con el método **/index/sim_top** al índice y se devuelve lo mismo que retorna esta llamada, una lista JSON con el título del post, su URL y la posición que ha quedado.

**/api/recommendation/personalizada** Usa GET con tres parámetros: **userid**, **cutoff** y **type**. Como ya se ha mencionado, se hace uso de tres tipos de recomendación, para ello es necesario tener ya instanciada la clase **Recommendation** y haber creado las recomendaciones.
Además, se utiliza el algoritmo **Round-Robin** sin repetición para coger recomendaciones de cada tipo, aunque, mediante el parámetro **type**, podemos escoger un tipo en concreto asignándole el valor de: **ufc** (basado en usuarios de filtrado colaborativo), **ifc** (basasado en ítems de filtrado colaborativo) o **ibc** (basado en contenido).
Al igual que en la no personalizada, se devuelve una lista JSON con el título del post, su URL y la posición que ha quedado.

Para tener una idea más visual y clara de como son utilizados estos métodos se ha realizado el diagrama A.3 que explica el flujo del entrenamiento y el diagrama 3.5 enfocado a la recomendación de los usuarios.

Cabe mencionar que para obtener la ip, puerto y métodos necesarios para la comunicación con la API del índice, se ha usado el archivo de configuración **settings.py**, donde se han insertado todas las variables requeridas y así ser accesibles desde cualquier punto del módulo.

### 3.2.3. Módulo crawler

#### Explicación de la tecnología usada

Para poder recoger todos los posts de una página web es necesario hacer uso de rastreadores web. En este caso se ha optado por usar **Scrapy**, un potente framework de Python para la recolección y filtrado de páginas web que será configurado para recoger únicamente los posts de un sitio web y extraer de ellos cierta información.

Scrapy es una herramienta cuya arquitectura está formada por cuatro elementos que trabajan en conjunto [12]:

**Motor Scrapy** Encargado de controlar correctamente el flujo de información entre componentes y de realizar algunas acciones al producirse ciertas situaciones.

**Planificador** Recibe solicitudes del motor y las pone en cola, también se las envía cuando este las solicita.

**Descargador** Como su nombre indica, se encarga de obtener la página web y enviarla al motor, el cual se las envía a las arañas.

**Arañas** Son clases personalizadas por los usuarios de este framework para analizar y extraer elementos de las páginas webs (ya descargadas), crear los items y seleccionar qué solicitudes adicionales hacer a partir de estas. Scrapy dispone de un amplio abanico de clases de arañas que podremos extender y personalizar para suplir nuestras necesidades.

**Tubería de ítems** Este último elemento se encarga del post-procesado. Recibe los ítems de la araña y suele encargarse de validar, parsear y almacenar.

Esta estructura permite una gran personalización, se pueden añadir varias arañas que hagan diferentes funcionalidades o que creen distintos ítems, o también, establecer varios pipelines para tener varias soluciones en caso de algún fallo en los anteriores.

Los ítems son estructuras de información extraídas por las arañas, podemos crear varios ítems con los campos deseados escribiendo una clase en **item.py** que extienda de **scrapy.Item**.

Para desarrollar una araña adaptada para este proyecto, el framework ofrece una gran variedad clases sobre las que se puede partir para personalizarla con las necesidades que se requieran. Todas estas clases comparten funcionamiento: nombre de la araña, una lista de dominios permitidos, semillas donde comenzar el rastreo, varias opciones de configuración y una función, llamada **parse** o **parse_item**, para parsear cada elemento web y crear el item.

**Spider** Es la araña más básica, sobre la que están basadas el resto y no provee de mucha más funcionalidad a la mencionada en el párrafo anterior.

**CrawlSpider** Es la más común, añade una funcionalidad para el seguimiento de links, llamada **rules**. Es una lista con objetos Rule donde se definen ciertos comportamientos que seguir, o evitar, para el seguimiento de links dentro de la página web recibida. Esta araña es la elegida para el crawler de este proyecto.

**XMLFeedSpider** Especializado en parsear XML.

**CSVFeedSpider** Especializado en parsear csv.

En cuanto a la clase **Rule**, para **CrawlSpider**, tiene varios argumentos para establecer ciertas reglas, las usadas son:

**LinkExtractor** Objeto para definir cómo se van a extraer los elementos, con parámetros des-

tacados como **allow_domains**, para establecer los dominios permitidos, o **unique**, un booleano para deshabilitar o habilitar la repetición.

**Callback** Establecer la función donde redirigir la web extraída.

**Follow** Para especificar si los links de la respuesta deben ser seguidos.

Para desarrollar el pipeline basta con hacer una clase en **pipelines.py** con el método **process_item**, que recibirá el ítem, la araña y se encargará de post procesar los elementos recogidos por esta. Y habrá que registrar cada nuevo pipeline creado en **ITEM_PIPELINES** de **settings.py** junto con un valor que determina el orden a ejecutar estos pipelines.

Para usar scrapy, primero hay que instalarlo, **pip install scrapy**. También se hará uso de otra biblioteca para hacer peticiones HTTP, **pip install requests**, para enviar los ítems en el pipeline. Después, creamos el proyecto con **scrapy startproject [nombre del proyecto]**

Finalmente, una vez se ha terminado de desarrollar, se pueden lanzar las arañas con **scrapy crawl [Nombre de la araña]**. Cabe mencionar que durante la manipulación de las páginas web recibidas por parte de la araña, se ha hecho uso de la herramienta **XPATH** que permite extraer y filtrar elementos deseados de la web.

#### Explicación del diseño y desarrollo del módulo

En primer lugar, se decidió la creación de un archivo de configuración JSON, llamado **conf.json**, donde se encontrarían todas las variables configurables del proyecto. Este archivo está formado por los siguientes elementos:

**Etiquetas para localizar la web** Usadas en la araña para establecer los dominios permitidos, **allowed_domain_to_crawl**, y la URL semilla donde comenzar el crawling, **start_url_to_crawl**.

**Etiquetas para la conexión con el núcleo** En este caso, se establecen las vías de conexión con el módulo núcleo para el envío de los posts: **rest_api_host** y **rest_api_create_post_method**, es decir, el host con el puerto y el método de su API para insertar posts. También está **rest_api_remove_post_method**, una etiqueta usada en la araña para limpiar todas las entradas antes de realizar el crawling, y **rest_api_create_index**, para enviar una petición que crea el índice cuando el crawler finaliza.

**Etiquetas para la extracción de datos** Establecer localizaciones XPath para extraer diferente información, permitiendo adaptarse mejor a cualquier web y así buscar la mayor compatibilidad posible. Las etiquetas son: **author_full_xpath**, **category_full_path**, **time_full_xpath**, para el autor, categoría y fecha respectivamente.

Después, para la creación de la araña, la cual se ha llamado **WebCrawler**, se ha optado por extender la clase **CrawlSpider**, principalmente por el añadido de las reglas. De hecho, se insertará una regla usando **LinkExtractor** permitiendo extraer cualquier enlace, sin repetición, que forme parte del dominio establecido en el archivo de configuración y cuya respuesta se enviará al método **parse_item**.

En el constructor de la araña, a parte de cargar las variables del JSON de configuración, como las etiquetas para localizar la web y las etiquetas para extracción de datos, también se va a cargar **rest_api_host** y **rest_api_remove_post_method** para enviar una petición al núcleo y eliminar todos los posts antes de comenzar con el crawling y se vuelvan a insertar. Limpiar la base de datos antes de llenarla de nuevo sirve para evitar duplicados y que posts que ya no están en la web se sigan recomendando.

Además, se ha añadido un trigger que responde a la señal **spider_closed**, es decir, cuando finaliza el crawler. Este trigger se crea sobrescribiendo la función de clase **from_crawler** y al activarse llama

a una función propia **on_quit** que hará la petición al núcleo para crear el índice.

Este crawler dispondrá de un ítem, llamado Post, formado por las mismas variables que el modelo Post del módulo principal: path, host, título, autor, contenido, categorías y tiempo de publicación. El método **parse_item** es el encargado de recibir los archivos web y crear instancias de ese ítem que se enviarán al pipeline. Para ello, mediante xpath se extraerá toda la información de la página. Hay cierta información que es fácil de extraer, pues sigue la misma mecánica en todas las webs, como la url, el host, o incluso el título, con las etiquetas **h1** de HTML.

Sin embargo, hay elementos más concretos que debido a la gran maleabilidad que tienen las páginas web es imposible obtenerlos de la misma forma siempre, es por eso por lo que el archivo de configuración solicita esas etiquetas con las rutas concretas XPath del autor, categoría y tiempo. También es así como se detecta si la página recibida forma parte de un post o se trata de otra cosa y así poder filtrar y solo quedarse con las entradas. Para poder obtener la ruta XPath de uno de estos elementos simplemente hay que:

1.– Subrayar el texto deseado, por ejemplo, el autor

2.– Botón derecho, inspeccionar elemento y se abrirá el código HTML con el elemento subrayado

3.– Botón derecho al HTML subrayado, copiar, copiar XPath completo.

Para el atributo contenido, es simple, recoge todo el texto de la web sin ningún tipo de marcado HTML. Todos los ítems creados se irán enviando al pipeline, esta clase llamada **RestApiPipeline**, ya registrada en la lista **ITEM_PIPELINES**, cargará en el constructor la ruta y método donde enviar los ítems del archivo de configuración y con la función **process_item**, recibirá uno a uno los elementos que, antes de enviarlos al núcleo, comprobará si todos los atributos contienen información, a modo de filtro, para asegurarse que se trata de un post y no una página cualquiera.

Se puede ejecutar con **scrapy crawl WebCrawler**.

Una vez probado, se pudo solucionar un error que afectaba al modelo de Post del módulo principal. Tras el crawling, había en concreto dos posts que no acaban insertados en la base de datos pero que sí habían sido recogidos por el crawler. El error se encontraba en la longitud del campo contenido que tuvo que ser aumentado pues esos dos posts superaban en caracteres lo establecido inicialmente.

Una vez recibido todo el contenido en el módulo principal hubo que pensar cómo almacenarlo, si se piensa desde una perspectiva de crear un correcto diagrama entidad-relación surgió la idea de tener una tabla de contenidos y otra de categorías con una relación, sin embargo, no interesaba organizar correctamente los datos pues se va a hacer minería con estos en masa usando Lucene. Es por eso que finalmente todas las entradas acaban almacenadas en una sola tabla siendo cada fila un documento que posteriormente será indexado como veremos en el siguiente módulo.

### 3.2.4. Módulo índice

#### Explicación de la tecnología usada

Toda la información recogida por el crawler y usada para recomendar en el módulo principal, debe ser estructurada, procesada y filtrada para un eficiente uso. Por ello, este módulo se encargará de recoger todos los posts, crear un índice, obtener datos a partir de ese índice y ofrecer una API REST para hacer solicitudes.

A diferencia del resto de módulos, este está desarrollado usando el lenguaje JAVA, principalmente debido a la biblioteca elegida para la creación del índice, **Apache Lucene**. Esta biblioteca hace uso

del llamado **índice invertido**, que, al contrario de los usados en bases de datos, está formado por una tabla, o catálogo, de términos al que llamaremos diccionario, junto a información adicional, como los documentos donde aparece ese término, frecuencia, posición, etc.

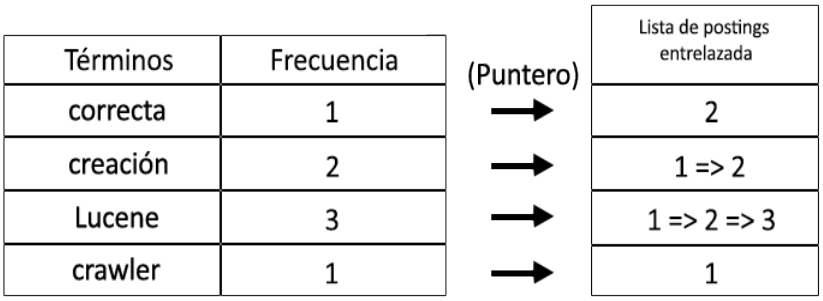


**Figura 3.2:** Catálogo de un índice invertido (con una parte en disco).

Sin embargo, para la correcta creación de este diccionario, Lucene tiene que analizar previamente el texto y así extraer de forma apropiada los términos de cada post. Con lo cual, primero se realiza un proceso de **tokenización**, es decir, extracción de palabras del documento, un proceso de **stopwords**, esto es eliminación de términos sin importancia: conjunciones, determinantes, preposiciones, artículos, adverbios, etc., y por último, una **normalización** de los términos resultantes, la cual está formada por varios procesos: unificar la forma de los términos con el mismo significado, ignorar mayúsculas o minúsculas, agrupación en familias de palabras y **stemming**, reducción de las palabras a su raíz [13, 31].

Una de las principales ventajas de los índices invertidos es su velocidad, así pues, son usados en los motores de búsqueda para optimizar al máximo el tiempo de respuesta. En la Figura 3.2 se puede ver una representación de cómo se almacena un índice invertido. En consecuencia, requieren de un mayor espacio en RAM, y por ello, se eligen distintas implementaciones en función del espacio disponible, desde extremos como tener todo el índice en RAM o en disco, hasta mezclas como la de tener únicamente el diccionario en RAM y que toda la información adicional a la que apunte se encuentre en disco, subiendo a RAM únicamente lo necesario.

**Apache Lucene** [2] es una biblioteca muy extensa que cuenta con una gran cantidad de clases y métodos para aportar todo tipo de funcionalidades, desde crear simplemente un índice para obtener métricas hasta crear un completo motor de búsqueda. A continuación, se resumirá brevemente la funcionalidad básica de Lucene utilizada para crear y cargar el índice.

La arquitectura de un índice basado en Lucene está formado por **documentos**, donde se encuentra el contenido encapsulado y, a su vez, estos pueden estar divididos por **campos**. El uso de campos produce una cierta independencia de la información entre ellos, por ejemplo, las métricas de frecuencia asociadas a un término están diferenciadas por campos, además, al realizar búsquedas hay que especificar en cual se va a buscar, pues los resultados variarán. Una buena práctica es dividir la información en campos y mezclar las respuestas obtenidas por cada uno de ellos en función de la importancia que tengan.

Primero, hay que montar el índice y para ello hay que crear una instancia de **IndexWriter**, pasándole el directorio donde se va a escribir y un **IndexWriterConfig** con algunos parámetros de configuración y el tipo de analizador a usar, por ejemplo, **StandardAnalyzer**.

[2] Apache Lucene offical website `https://lucene.apache.org/`

Después, para insertar documentos hay que instanciar la clase **Document**, a su vez, en esta clase hay que introducir el contenido por campos con el método **add** y un objeto de la clase **Field**, donde hay que pasarle como parámetros el nombre del campo, el contenido y un **FieldType** con parámetros de configuración, como la de especificar qué tipo de información se va a manejar en el diccionario de ese campo: frecuencias, posición, postings.

Una vez tenemos la instancia de Document con todos los campos insertados, podemos insertar el post en la instancia de IndexWriter con el método **add**. Finalmente, una vez se han insertado todos los posts, se puede concluir cerrando con el método **close** y se habrán creado una serie de directorios y archivos que usa Lucene para almacenar el índice.

Después, para cargar el índice basta con instanciar la clase **IndexReader** mediante el método **open** de **DirectoryReade**r pasándole la ruta del directorio. A partir de aquí, esta clase cuenta con una gran cantidad de métodos y clases de apoyo para obtener información del índice: se pueden obtener el vector de términos de cada campo, la frecuencia total y el número de documentos de los términos por campo. Por ejemplo, tenemos la clase **Term**, donde se especifica el campo y el término, y se usa en algunos métodos de IndexReader, como **docFreq**, para obtener información.

Sin embargo, a la hora de crear el índice se va a aprovechar para obtener ciertas métricas que en tiempo de ejecución darían lugar a un elevado tiempo de respuesta. Esto provoca que el índice tarde más en crearse, pero optimizamos las consultas que se harán posteriormente. Una de estas métricas es la similitud entre documentos, que como se mencionó en el estado del arte se usará la similitud coseno, ecuación 2.1, y donde el vector de cada documento esta formado por el **tf-idf** de cada término, es decir, frecuencia del término multiplicada por frecuencia inversa del documento.

El **tf-idf** es una medida que representa la importancia de una palabra para un documento en toda la colección de un campo concreto. Abreviando, **tf** es la frecuencia del término en el documento, e **idf** es la frecuencia inversa del documento, o sea cómo de común es el término en toda la colección.

$$TF - IDF = tf * idf \tag{3.1}$$

$$tf(\text{term}, \text{doc}) = \begin{cases} 1 + \log_2 \text{frecuencia}(\text{term}, \text{doc}) & si \quad \text{frecuencia}(\text{term}, \text{doc}) > 0 \\ 0 & si \quad \text{frecuencia}(\text{term}, \text{doc}) \leq 0 \end{cases} \tag{3.2}$$

$$idf(\text{term}) = \frac{\|D\| + 1}{\|D_{\text{term}}\| + 0{,}5} \tag{3.3}$$

En la fórmula 3.3, |D| es el número total de documentos y $|D_{\text{term}}|$ es el número de documentos con un cierto término. Por ejemplo, si no se eliminan los términos muy comunes y sin significado, como es el caso del determinante “el”, sus puntuaciones tf-idf, es decir su importancia, serán muy bajas pues es una palabra muy común en general. Sin embargo, si la palabra “física” aparece mucho en el documento **D1**, pero casi no se encuentra en el resto, será un término relevante para **D1**, y, por tanto, tendrá en él un alto tf-idf. [13].

Como ya se ha comentado, solo hay dos módulos en todo el proyecto que requieren de una API REST, estos son el núcleo y el módulo índice. El índice requiere de esta arquitectura debido a que es necesario solicitarle información, como son las similitudes, y por tanto debe tener a disposición una API de métodos para comunicarse con él. La herramienta elegida para esta tarea es el framework de

Dropwizard [3].

**DropWizard** cuenta con una clase, llamada **Configuration**, que se encarga de deserializar un archivo de configuración YAML y así almacenar la configuración establecida en la aplicación. Por defecto, este archivo de configuración está formado por elementos de configuración que ofrece el propio framework, como el puerto y protocolos a usar, sin embargo, se puede extender la clase Configuration e implementar una propia que acepte parámetros personalizados.

Uno de los elementos principales de los microservicios REST de DropWizard son los **recursos**, son clases con métodos que responden a ciertas URLs, procesan la solicitud y devuelven una respuesta. Para ello, este framework utiliza las anotaciones **@Path**, **@POST**, **@GET**, etc., de **JAX-RS**, una API de Java para la creación de servicios web.

El elemento fundamental es la clase de la aplicación, que deberá extender de **Application** y pasándole como genérico la clase Configuration a usar. Esta clase se encarga de establecer todos los elementos, como registrar los recursos, e iniciar el servicio. De hecho, para iniciar este módulo el main debe llamar al método **run** de la clase [32].

Para este componente se va a partir del proyecto desarrollado en el artículo [33], que se basa en un TFG desarrollado en esta Escuela y dirigido por el mismo tutor que este trabajo, aunque como se puede observar en su proyecto, este framework cuenta con mucha más funcionalidad disponible, como vistas, modelos, sistemas de almacenamiento, obtención de métricas, etc., que no van a ser necesarios en este caso.

Además, para la solicitud de post al módulo principal se ha hecho uso de la librería Apache HttpComponents [4] que ofrece un variado conjunto de herramientas para el manejo de peticiones HTTP. También se ha utilizado la librería oficial de JSON [5] para el manejo y extracción de información de los post devueltos.

### Explicación del diseño y desarrollo del módulo

Este módulo está diferenciado por dos secciones que funcionan en conjunto, un bloque de información donde se construye y utiliza el índice y un bloque que aporta un servicio web RESTful para la comunicación con el exterior.

Para facilitar la tarea de configurar el módulo sin tener que tocar código ni volver a compilar se usa un archivo de configuración, **server.yml**. Este archivo contiene todo lo necesario para configurar el índice y elementos necesarios para el servidor RESTful, como los puertos. Esto se ha realizado con una extensión de Configuration de DropWizard, llamada **ServerConfiguration**, para que lea también nuevos campos personalizados del archivo de configuración. Una vez se inicia la aplicación y se lee la configuración, estos nuevos atributos serán insertados en una clase estática, llamada **IndexConfiguration**, para que puedan ser accesibles desde cualquier lado del proyecto. La información que almacenará será: IP, puerto y método del módulo principal para obtener los posts, una lista con los campos del modelo de post que va a usar el índice y los pesos que tendrán cada uno de estos para indicar su importancia, por ejemplo, a la hora de calcular las similitudes totales. Esta lista de campos se va a iterar cada vez que se requiera realizar la misma tarea en cada uno de ellos y, de esta forma, cuando se quite o añada un campo, por ejemplo, el título, basta con cambiar el YAML y de forma automática el índice se creará y se calcularán las métricas únicamente con los campos establecidos, sin tener que tocar código.

[3] DropWizard offical website https://www.dropwizard.io/en/latest/

[4] HttpComponents official website https://hc.apache.org/

[5] JSON offical website https://www.json.org/json-en.html

El primer bloque está formado por dos clases principales, **LuceneIndexBuilder** y **LuceneIndex**, con sus respectivas interfaces, **IndexBuilder** y **Index**, para un correcto diseño de clases y que cualquier implementación que parta de estas interfaces sea compatible. A continuación, se detallará la funcionalidad de la clase y los métodos principales.

**LuceneIndexBuilder** se encarga de construir el índice. Primero hay que llamar al método **build**, pasando la IP, puerto, método y directorio donde almacenar el índice. Esta función inicializa el IndexWriter, después llama a **restAPIConnection** que, con ayuda de subfunciones adicionales, realiza la petición para obtener los posts del módulo principal, itera el JSON devuelto, extrae los campos que hayamos solicitado en el archivo de configuración y crea documentos que los inserta en el índice. Finalmente, el método build cierra la instancia de IndexWriter y el índice ha finalizado de crearse.

Sin embargo, hay cierta información que antes o después era necesario calcular y requiere de cierto tiempo computacional que no sería aceptable a la hora de responder consultas, por tanto, se realiza aquí. Estas métricas son la similitud y por tanto el módulo de los documentos. Para ello, después de finalizar la creación del índice, se llama a dos métodos, **calculateModule** y **calculateSimilarity**, que lo cargarán de nuevo para usarlo.

**calculateModule** Se llama primero, pues este cálculo luego se va a usar en la similitud y así evitar calcularlo dos veces. Estando formada cada componente del vector documento de tf-idf, calcula el módulo de cada documento para cada campo y lo inserta en un archivo del directorio del índice, **modulo.mod**, cuyo formato podemos ver en la figura 3.3, y en una variable con doble diccionario anidado (la cual se usará en las similitudes).
Este doble diccionario, llamado **doc_mod**, contiene como claves todos los ids de los documentos asociados a otro diccionario cuya clave serán los campos y cuyo valor será el módulo de ese campo para ese documento. Este tipo de variables, con un diccionario anidado donde las claves son los campos, van a ser usadas en varias ocasiones para poder ampliarlos o reducirlos desde el archivo de configuración y automáticamente la variable regula su tamaño sin necesidad de modificar código y crear una variable para cada uno.

modulo.mod

| Doc ID | Módulo <campo1> | ... | Módulo <campo i> |
|---|---|---|---|
| 1 | 6.867 | ... | 43.867 |
| 2 | 8.357 | ... | 26.357 |
| 3 | 10.123 | ... | 32.123 |
| 4 | 9.479 | ... | 56.479 |

**Figura 3.3:** Ejemplo de un archivo de módulos.

**calculateSimilarity** Realiza el mismo proceso, calcula la similitud coseno, ecuación 2.1, de todos los pares de documento para cada uno de sus campos, pero esta vez, se almacena en un archivo para cada campo, llamado **sim_<campo>.sim** , donde se guardará cada par y su similitud en ese campo, podemos ver su formato en la figura 3.4.
Para la similitud, se obtiene el vector de cada documento y se hace su producto escalar, y finalmente, se divide entre el producto de sus módulos, almacenados previamente en **doc_mod**. De nuevo, cada componente del vector está formada por tf-idf.

**LuceneIndex** se encarga de cargar el índice y de devolver información sobre este. Gracias a un booleano en el constructor se puede cargar de dos formas, estando a True lo cargará de forma rápida,

sim_<campo x>.sim

| DocID i | DocID j | Similitud <campo x> |
|---|---|---|
| 0 | 1 | 0.256 |
| 0 | 2 | 0.123 |
| 0 | 3 | 0.567 |
| 0 | 4 | 0.064 |

**Figura 3.4:** Ejemplo de un archivo de similitudes para un campo.

únicamente la instancia del IndexReader sin leer nada adicional como las similitudes, pensado para cargar el índice rápido en las funciones de calcular el módulo y similitud; y estando a False, lo carga completamente con todos los archivos adicionales.

Esta clase contiene todos los métodos para obtener la información necesaria del índice, por ejemplo, el número total de documentos; especificando el campo, la colección de términos; precisando además el término, la frecuencia en documentos de esa palabra y con el id del documento también la frecuencia en ese post, su vector o su módulo. Sin embargo, se va a detallar dos métodos pensados para la obtención de similitudes.

**getDocSimilarity** Devuelve similitudes usando la variable **docsSim**. Esta variable, cargada al leer los archivos de similitudes, es un diccionario cuyas claves son los campos y los valores otro diccionario con un id del par con otro diccionario con la otra id del par y el valor de la similitud para ese par de documentos en ese campo.
Este método tiene tres variantes; si recibe como argumento un par de dos ids, devuelve la similitud total para ese par; con un par de dos URLs, también devuelve la similitud total; y si recibe un par de dos ids y se especifica un campo, devuelve la similitud de ese par para ese campo. Cabe decir que para comprobar el valor de un par [i,j], primero se comprueba en ese orden y si devuelve 0, a la inversa, [j,i].

**getDocSimilarityTop** Recibe como parámetros la URL de un documento y un número de corte, y devuelve un **ranking** con los documentos más similares a este, hasta ese número de corte. Esto se realiza insertando en un Heap, creado en la clase **HeapRanking**, todas las similitudes y automáticamente el heap las irá ordenando de mayor a menor.

Para el ranking se ha usado una implementación basada en el patrón iterador, usando un diseño de interfaces, **Ranking**, **RankingDoc** y **RankingIterator**, y así cualquier implementación futura que cumpla esta interfaz será compatible, en este caso la única es la de **HeapRanking**, que ordena los elementos como un **heap**, y **PostRankingDoc**, que contiene información de cada elemento del ranking como el id del documento y una puntuación asociado a este.

Como queremos una cierta independencia entre módulos, el principal desconoce qué ids usa el índice para los posts, por tanto, no podemos obtener similitudes por ids, sino que el módulo principal pasará URLs, que se consideran únicas para cada post. Con lo cual, ahora LuceneIndexBuilder crea un archivo de traducciones, llamado **pathtoid.id**, según se van insertando los posts en el índice y LuceneIndex carga ese archivo en un diccionario y tiene una función que se encarga de traducir URLs en ids, pues Lucene trabaja con estos últimos.

El segundo bloque de este módulo corresponde con la funcionalidad de un servicio RESTful, usando el framework de DropWizard, para que el módulo principal pueda tanto controlar este como solicitar información, en concreto, similitudes. Este bloque cuenta con **LuceneRestApi**, una clase que extiende de Application e inicia todo el sistema, además, la configuración que acepta proviene de una clase

personalizada, **ServerConfiguration**, la cual carga y extrae el valor de los atributos para insertarlos en **IndexConfiguration**, explicada anteriormente.

El último elemento es el recurso **LuceneManagerResource**, se encarga de establecer los métodos de la API que estarán disponibles, así como aporta la funcionalidad de cada uno de ellos y devolver una respuesta. A continuación, se explicarán cada uno de los métodos.

**/index/create** Usa GET. Se encarga tanto de crear el índice como de cargarlo en el atributo de la clase **ilucene**. Devuelve un código de respuesta 201 CREATED si todo salió correctamente, o 500 INTERNAL SERVER ERROR en caso de algún error.

**/index/read** Usa GET. Simplemente carga en el atributo **ilucene** el último índice creado. Devuelve el mismo resultado que en el caso anterior.

**/index/sim** Usa POST y tiene que recibir en formato **x-www-form-urlencoded** dos parámetros, **urli** y **urlj**. Mediante el uso del método **getDocSimilarity** del índice devuelve un JSON con los campos: doci, con el valor de urli; docj, con el valor de urlj; y sim, con el valor de la similitud de ambos. En caso de error devuelve un 404 NOT FOUND, si ilucene no está instanciada, o 400 BAD REQUEST si hay algún problema con los parámetros.

**index/sim_top** Usa POST y tiene que recibir en formato **x-www-form-urlencoded** dos parámetros, **url** y **cutoff**. Mediante el uso del método **getDocSimilarityTop** del índice devuelve una lista JSON donde cada elemento contiene: pos, con la posición que ha quedado en el ranking; y doc, con la URL del documento similar. Se decidió ocultar el valor de la similitud pues es una información que no se usa y por tanto no era necesario compartirla. En caso de error sigue el mismo procedimiento que el método anterior.

## 3.3. Diagramas

### Diagrama de secuencia

La figura 3.5 representa el diagrama de secuencia principal del proyecto, donde se ven reflejadas las comunicaciones necesarias para que, partiendo de una petición a una web, un usuario pueda recibir recomendaciones. Existen más tipos de comunicación que se pueden encontrar en los anexos: diagrama de secuencia para la obtención del log, en figura A.1; para iniciar el crawling, figura A.2; y para entrenar el sistema, figura A.3.

### Diagrama de casos de uso

La figura 3.6 muestra las distintas formas de interacción de cada actor con el sistema, que en este caso son dos: usuario y administrador. De esta forma, podemos ver cómo el usuario se comunica para enviar sus logs y recibir recomendaciones, mientras que el administrador lo hace para iniciar el entrenamiento e iniciar el crawling. Además, tenemos dos actores especiales, base de datos e índice, cuya labor con el sistema se basa simplemente en almacenar cierta información para proveerla cuando sea necesario.

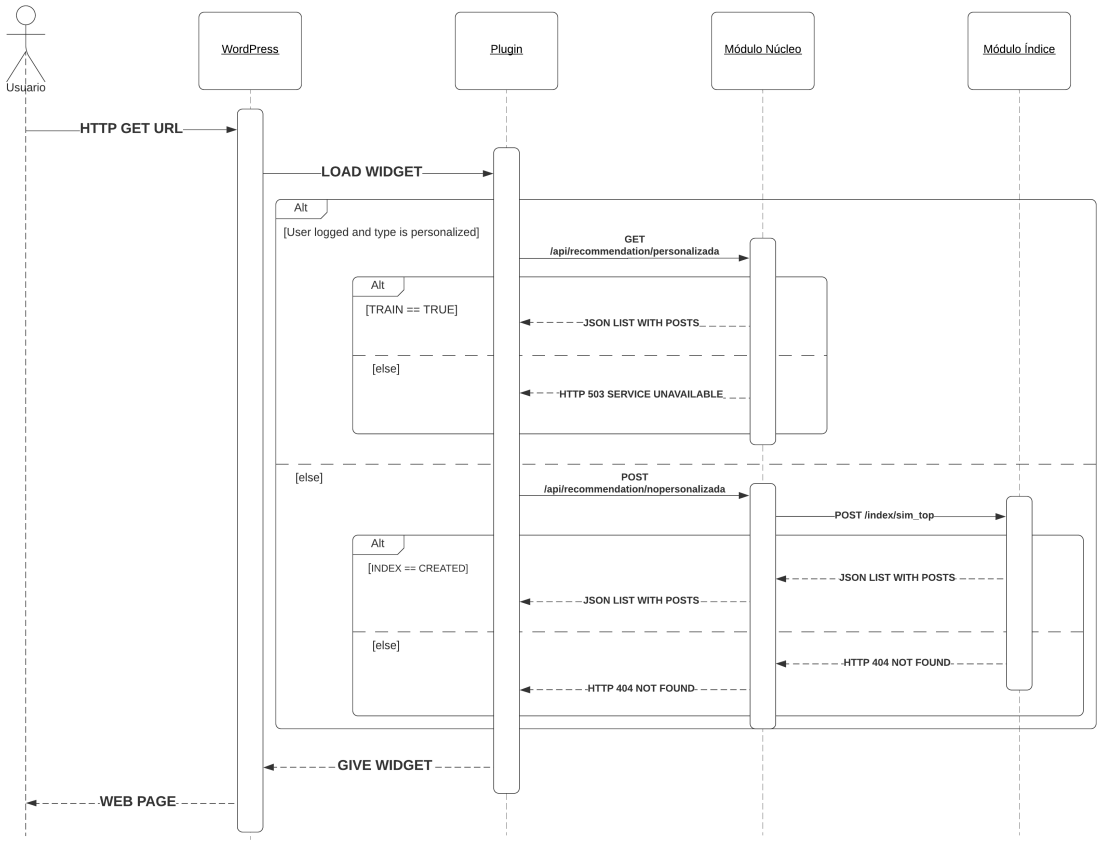


**Figura 3.5:** Diagrama de secuencia para la obtención de recomendaciones.

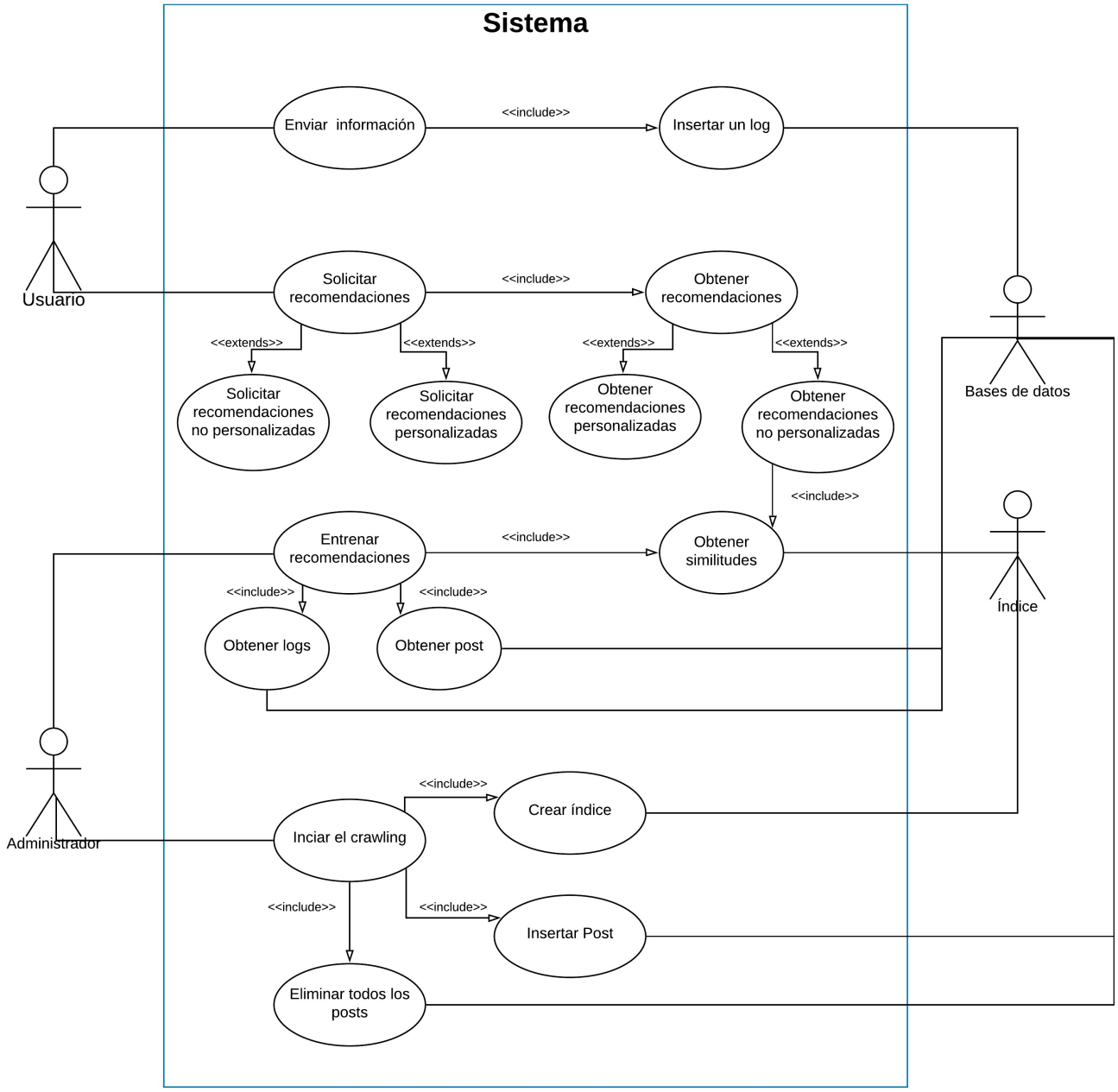


**Figura 3.6:** Casos de uso del sistema.

# 4 Pruebas y resultados

Para una correcta comprobación de los distintos componentes del proyecto y para tener un acercamiento lo más real posible a una instalación en una hipotética fase de producción se ha creado una web de ejemplo, usando obviamente WordPress, sobre la que probar este sistema de recomendación.

En concreto, debido a que el objetivo principal es la de realizar una simulación realista, se ha copiado la revista científica `https://www.investigacionyciencia.es/`, de donde se han cogido un total de 60 entradas, distribuidas en distintas categorías como por ejemplo: *Solos en la Vía Láctea* o *Everests de otros mundos*, de la categoría Vida extraterrestre; *Condicionales y probabilidades* o *En la senda de Jesús Mosterín*, de Lógica.

## 4.1. Pruebas y resultados

Una vez está todo instalado, configurado, el plugin activado, sus widgets creados y el módulo núcleo e índices ejecutados (ver anexo B) se puede dar comienzo a la prueba. Primero, el sistema necesita saber el contenido que hay que recomendar a los usuarios, por lo que es necesario activar el crawler.

Como ya sabemos, al activar el crawler primero solicitará la eliminación de todos los post actuales, tras esto, recorrerá todo el sitio web donde desechará las páginas que no correspondan con un post y estos último los mandará al módulo principal. Una vez ha finalizado, solicitará la creación del índice mediante una petición puente, a través del núcleo, que a su vez llamará al índice. Una vez ha finalizado el proceso, todos los post han sido insertados en la base de datos del núcleo.

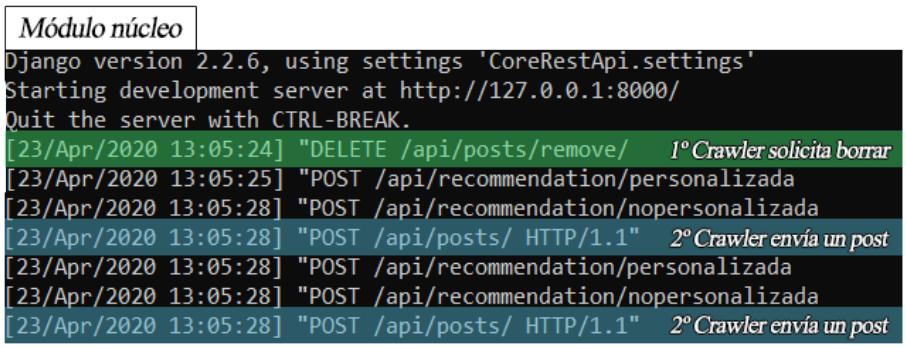


**Figura 4.1:** Peticiones que recibe el núcleo tras iniciar el crawler.

Como se puede observar en la figura 4.1, en las peticiones no subrayadas, el crawler actúa como un cliente y activa el plugin que solicita las recomendaciones y envía los logs. Además, como muestra la figura 4.2, cuando el núcleo recibe recomendaciones no personalizadas solicita al índice similitudes,

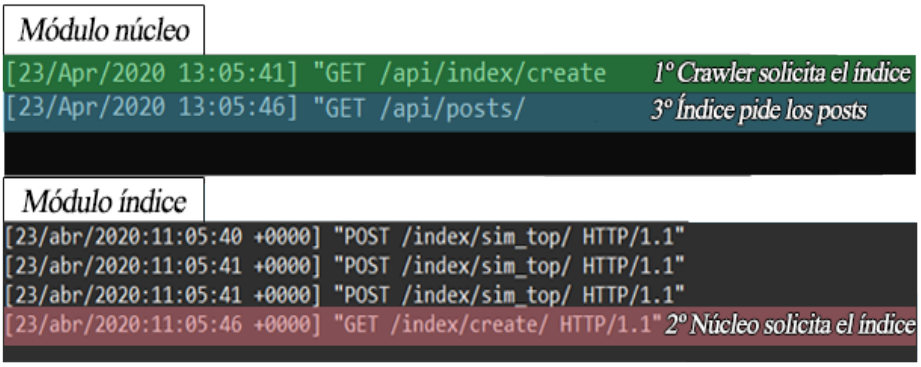


**Figura 4.2:** Arriba: el núcleo recibe la petición de crear el índice por parte del crawler y a su vez la petición del índice de obtener los post, abajo el índice recibe la petición de crearse.

pero si este no está creado devolverán error.

A partir de este momento, las recomendaciones personalizadas ya estarán disponibles en la web, sin embargo, para ello habrá que entrenar enviando al núcleo la petición: `GET /api/recommendation/train`. En ese caso, cuantos más logs haya, más información tendrá el sistema y podrá afinar mejor sus recomendaciones.

Previo a comentar los resultados de las recomendaciones personalizadas vamos a comenzar analizando las no personalizadas, como se muestra en la figura B.6, se le ha dado una prioridad del 40 % tanto a la categoría como al contenido extraído de cada post y solo un 20 % a su título, esta preferencia se debe a que el título puede ser un poco fantasioso o sensacionalista y no representar al contenido, además, la categoría es lo que mejor describe al post pero tampoco queremos conseguir que la similitud se centre solo en este campo y encasillar las recomendaciones, por ello, le damos la misma importancia que el propio contenido y así el sistema es capaz de descubrir items similares fuera de la misma categoría.

El campo categoría es algo especial, como solo está formado por una palabra, la similitud al final se asemeja a un booleano, es decir, si dos post comparten categoría adquiere valor de 1, en caso contrario de 0. Esto hace que coincidir en la categoría, la cual tiene un peso del 40 %, aporta 0.4 puntos extra a la similitud, por lo que es muy probable que primero recomiende posts con la misma categoría y, de hecho, como podemos ver en la figura 4.3 estas recomendaciones sacan bastante ventaja en puntuación a las que no comparten.

En realidad, esto es algo bueno, pues este campo aporta una información muy importante acerca del post, sin embargo, vamos a probar qué resultado devuelve el sistema si reducimos la importancia de la categoría al 5 % y se lo damos al contenido, que acumularía un 75 %.

Sorprendentemente, como muestra la figura 4.4, obtenemos el mismo ranking, esto se debe a que sin ese aumento de puntuación del 0.4, ahora del 0.05, que ofrecía la categoría, el sistema sigue siendo capaz de detectar solo con el contenido y con un poco de ayuda del título posts que hablan acerca de los mismos temas y posicionarlos entre los primeros, aunque está vez sus puntuaciones son muchos más similares y no destacan tanto.

Este es uno de los problemas que se comentaron acerca de la recomendación no personalizada, que no es capaz de innovar mucho en cuanto al tema pues trata de devolver resultados lo más similares posibles al del post actual.

En cuanto a la recomendación personalizada, se ha realizado una prueba cerrada donde tres usua-

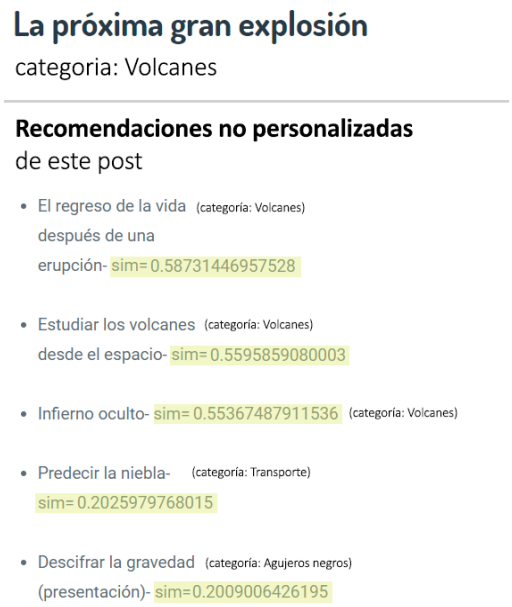


**Figura 4.3:** Un post de la web con las recomendaciones no personalizadas que devuelve para una similitud con pesos del 0.2 para el título, 0.4 a la categoría y 0.4 al contenido.

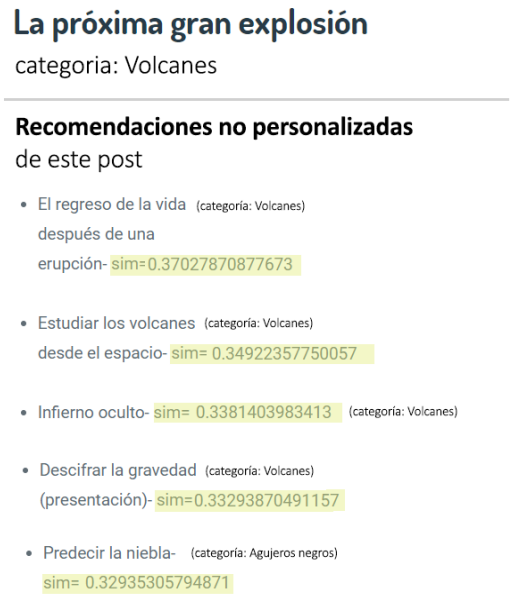


**Figura 4.4:** Un post de la web con las recomendaciones no personalizadas que devuelve para una similitud con pesos del 0.2 para el título, 0.05 a la categoría y 0.75 al contenido.

rios: user1, user2 y user3, visitan un total de 9 post distintos, como podemos ver en la figura 4.5. En esta prueba, el usuario 1 y el usuario 2 son muy similares, sin embargo, el usuario 3 tiene gustos completamente distintos.

| | | Items | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | everest de otros. | astrogusanos | descf. los sentidos | exo. Vecino | detectives elec. | geometria trop. | belleza mat. | cond y prob. | homeo. peligrosa |
| Users | user1 | 2 | 1 | 1 | | | | | | |
| | user2 | 2 | 2 | | 1 | 2 | | | | |
| | user3 | | | | | | 5 | 2 | 1 | 4 |
| | | vida extraterrestre | | | planetas | transporte | matemáticas | | lógica | |
| | | Categorías | | | | | | | | |

**Figura 4.5:** Ejemplo de matriz usuarios-items de recomendaciones personalizadas. Para ello se ha hecho uso de 3 usuarios los cuales han visitado un total de 9 post, el resto de post no se muestran pues su puntuación es 0 con todos los usuarios.

Existen más categorías en la web de prueba, sin embargo, debido a que no hay ningún usuario que las haya visitado es como si no existiesen para este tipo de recomendación, es por eso que la recomendación personalizada funciona muy bien una vez el sistema lleva tiempo en funcionamiento y se han registrado una gran cantidad de logs de multitud de usuarios. En consecuencia, al principio, es un sistema muy torpe pero que va aprendiendo poco a poco.

Esto también ocurre con los usuarios nuevos, hasta que no se obtenga suficiente información de estos, el sistema será incapaz de devolver resultados de calidad. Por ejemplo, el usuario 3, aunque tiene algunos logs dentro de la web estos no coinciden con ningún otro usuario, por lo tanto, hasta que llegue ese momento la puntuación basada en usuarios será nula debido a que no se le ha encontrado ningún vecino.

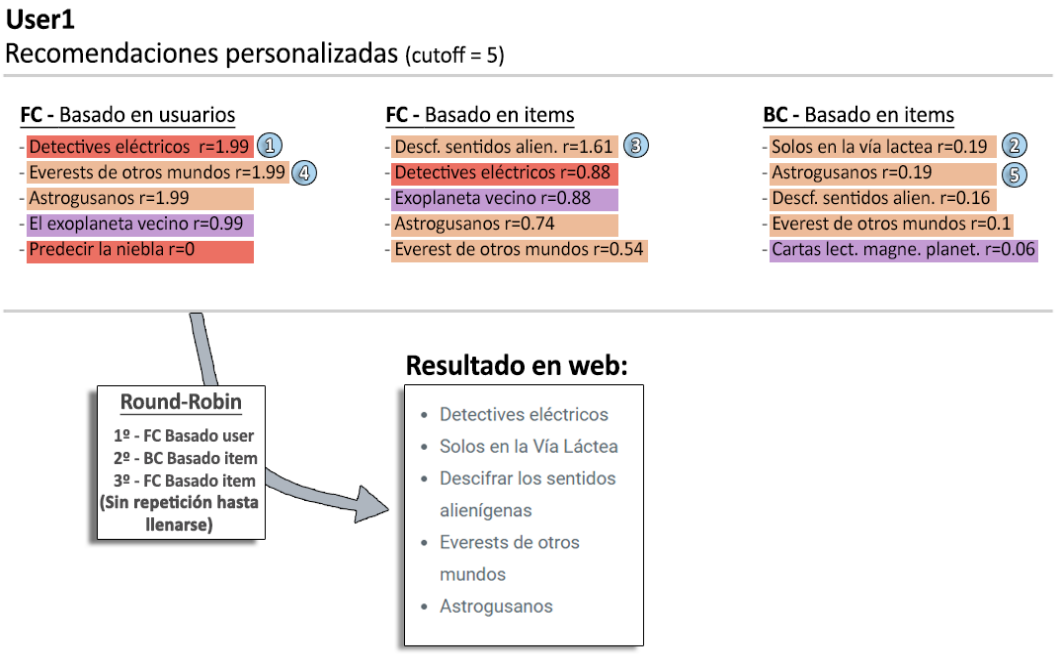


**Figura 4.6:** Ejemplo de recomendaciones personalizadas para el usuario 1. Se muestra lo que devuelve cada tipo de recomendación personalizada a partir de los datos de la matriz 4.5 y el resultado final en web tras realizar el algoritmo de Round-Robin. Los colores representan las categorías de esos post, su significado se puede encontrar en la matriz anterior.

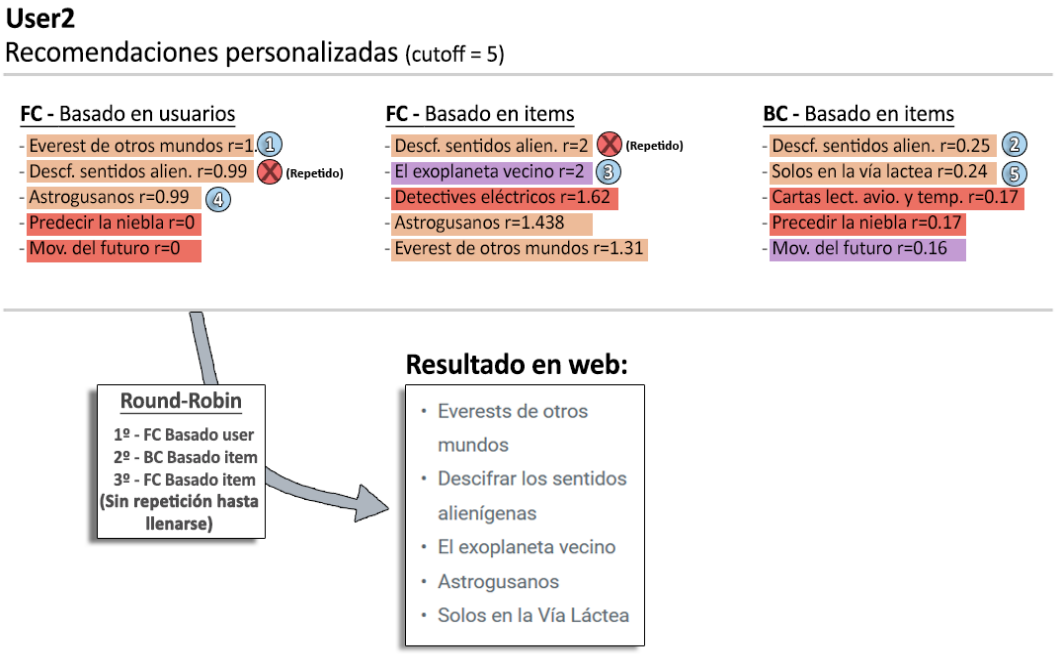


**Figura 4.7:** Ejemplo de recomendaciones personalizadas para el usuario 2. Toda la explicación y notación es igual a la de la figura 4.6.

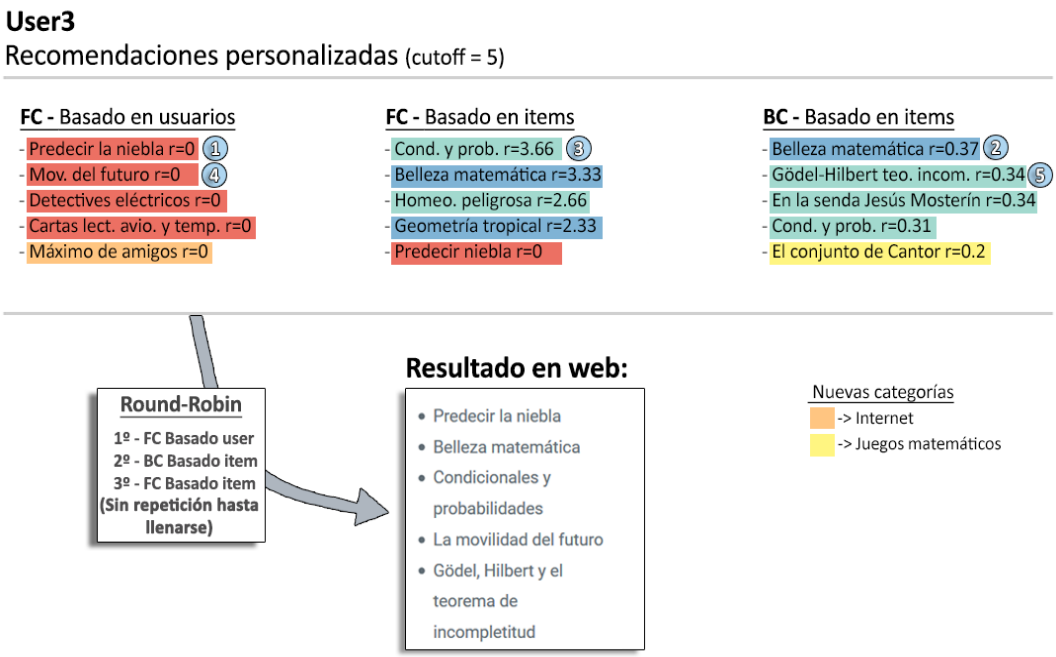


**Figura 4.8:** Ejemplo de recomendaciones personalizadas para el usuario 3. Toda la explicación y notación es igual a la de la figura 4.6.

Cabe mencionar que el sistema permite devolver como recomendaciones posts que el usuario ya ha visitado, es decir, se ha decidido no bloquear estos elementos.

Los índices numéricos que se muestran en las figuras 4.6, 4.7 y 4.8 representan el procedimiento que hace el algoritmo Round-Robin implementado para extraer documentos de cada tipo de recomendación. El recorrido que sigue es: primero filtrado colaborativo basado en usuarios, segundo basado en contenido y tercero filtrado colaborativo basado en items. Como se aprecia en la figura 4.7, si hay repetición se coge el siguiente elemento de la lista de recomendación que tocaba.

Como se puede ver en la figura 4.6 o figura 4.7, basado en usuarios es el algoritmo que más consigue innovar. De hecho, premia aquellos post que el usuario no ha visitado pero si lo ha hecho su vecino, cuanta más similitud se tiene mayor será la puntuación. Por ejemplo, en este caso, aunque se trata de una prueba pequeña y controlada, el usuario 1 y el usuario 2 se parecen bastante pero el segundo ha visitado un nuevo post, **detectives eléctricos**, desconocido por el primero, volviéndose el más recomendado para el usuario 1. Sucede algo similar con **El exoplaneta vecino** pero con menor puntuación, lo que provoca que el Round-Robin no lo coja para el resultado final. En el caso inverso, el usuario 1 aporta únicamente un post con pobre puntuación al usuario 2, **Descifrar los sentidos alienígenas**. Concluyendo, la recomendación basado en usuarios consigue que todo usuario, si dispone de varios vecinos muy similares, obtiene como resultado muchos post nuevos, los cuales han visitado sus vecinos.

Por otra parte, como el usuario 3 no tiene similitud con ningún otro vecino, figura 4.5, la puntuación basada en usuario no aportan nada, como muestra la figura 4.8, sin embargo, la seguimos utilizando entre el resto de recomendaciones, pues, aunque sea la menos personalizada, tal vez consigue que el usuario expanda sus fronteras y así encontrar algún vecino.

En la recomendación basada en items de filtrado colaborativo, la similitud se calcula mediante la similitud coseno, siendo los vectores las columnas de estos items de la matriz 4.5, es por eso que cuantos más usuarios haya, mejor se puede ajustar el resultado. Por ejemplo, dos items los cuales han sido visitados únicamente por un mismo usuario tendrán la máxima similitud, 1. Esto es lo que ocurre entre los post del usuario 3, aunque en este caso se produce por tener poca información. Además, la similitud entre dos items sin usuarios en común será de 0, es por eso que la similitud de los post del usuario 3 con la de los usuarios 1 y 2 son nulas, y es imposible recomendar al usuario 3 otros post que no sean los que ya ha visitado mediante este tipo de recomendación.

A diferencia de la anterior, esta recomendación no funciona correctamente solo con tres usuarios aunque hayan registrado muchas acciones, sino que requiere que existan muchos de estos, es decir, necesita de más participación para evitar los resultados pobres.

En cuanto a la recomendación basada en contenido, su funcionamiento se basa en ir cogiendo los post que han visitado los usuarios y devolver los más similares a estos según la similitud calculada en el índice. De esta forma, es el método que más consigue personalizar y acertar en la recomendación, pero, a cambio, bloquea al usuario en los mismos temas.

# 5 Conclusiones y trabajo futuro

A continuación, se va a realizar un análisis final del proyecto estableciendo los puntos fuertes y débiles y definiendo el sentido del mismo en el momento actual. Además, se van a definir algunas pautas que se pueden seguir para mejorar este proyecto e incluso para llegar a hacer de él un producto final utilizable. El código desarrollado en esta práctica, explicado en la sección anterior, se puede encontrar en el repositorio GitHub: `https://github.com/srfonso/wp-recommendation-system`

## 5.1. Conclusiones

Los sistemas de recomendación se han convertido en una de las herramientas principales de Internet y han cambiado la forma con la que los usuarios interaccionan con los servicios web. Desde el mismo momento que alguien usa un buscador por primera vez, comienza un proceso de creación de su perfil virtual donde poco a poco el sistema va aprendiendo acerca de sus gustos. La finalidad es conseguir ofrecer al usuario las mejores recomendaciones posibles para facilitar su trabajo, evitar que este se aburra, y maximizar las posibles ganancias que podrían conseguir con él, ya sea gracias al tiempo que este permanece en un sitio web o la capacidad para hacerle comprar productos.

Por tanto, había que buscar una forma de extraer información acerca del usuario además de aportarle recomendaciones. Primero, había que conocer el público y así elegir la tecnología sobre la que construir el sistema, teniendo en cuenta que se buscaba un gran alcance y que cualquier web pudiese implementar este proyecto. Tras analizar las distintas herramientas existentes en el desarrollo de webs, se tomó la decisión de montar el proyecto sobre WordPress, el mayor gestor de páginas web del mundo [23], y, a partir de las extensiones que admite, diseñar en una todo el sistema de comunicación cliente – sistema.

Sin embargo, la obtención de información acerca de los usuarios es inútil si se desconoce el terreno donde se encuentran, es decir, no se puede recomendar si no se sabe qué recomendar. Había que encontrar una forma rápida, fácil y automática de que todo el proyecto fuera capaz de conocer la página web sobre la que trabaja, así pues, surge la necesidad de usar un crawler y, por consiguiente, un índice que procese toda la información desestructurada extraída del crawler.

En la actualidad existen grandes empresas que tienen sus propios sistemas de recomendación y las pequeñas tienen que adaptarse y hacer uso de herramientas que les ofrecen las primeras, limitando las posibilidades de ajustarse a sus necesidades, además de no tener nunca el control total del sistema, simplemente lo que la herramienta les permite. Por ello, este proyecto busca definir una arquitectura, sentar las bases y poner los primeros cimientos hacia un sistema de recomendación libre y abierto donde cualquier persona pueda configurar y personalizar a su gusto para su sitio web. El sistema modular busca precisamente adaptarse en cada momento a las capacidades de ese sitio web, tales como, comenzando todo en una misma máquina y, según la página web va creciendo en

capacidad y número de usuarios, el sistema de recomendación es capaz de ajustarse a sus necesidades, por ejemplo, colocando el núcleo e índice en máquinas con mejor hardware. Además, permite una independencia entre módulos que facilita una posible evolución del proyecto, es decir, se puede mejorar el crawler sin necesidad de tocar nada del plugin o del núcleo.

Otro punto importante es la accesibilidad que logra este proyecto, mediante el CMS más usado de internet [23], WordPress, y usando simplemente un plugin propio, es capaz de aportar, a casi cualquier web, un motor totalmente funcional de recomendación. Además, permite un alto nivel de personalización sin requerir de altos conocimientos informáticos, esto se consigue gracias a todos los archivos de configuración que provee cada módulo; por ejemplo, desde la configuración del crawler, se puede especificar dónde encontrar cierta información de la página web para que este pueda extraerla; en el índice se pueden modificar los campos y pesos que este debe usar al crear el índice y calcular las similitudes; y también, el propio plugin incluye un widget el cual se puede añadir y configurar desde el propio menú de WordPress.

Para finalizar, los resultados obtenidos se ajustan con los esperados, realizan los cálculos en cada tipo de recomendación de forma correcta y, como se ha podido comprobar en el apartado de pruebas, se obtienen recomendaciones que ayudan al usuario a seguir leyendo contenido de interés para él, y por otro lado obtiene otras que le ayudan a innovar. La combinación de tres tipos de recomendación personalizada consigue una mezcla mucho más sólida y reduce los defectos de cada una, además, si una falla por falta de información las otras dos son capaces de mantener la coherencia. Y finalmente, la recomendación no personalizada es capaz de encontrar parecidos en los posts y ofrecer similares a estos con una precisión bastante buena.

## 5.2. Trabajo futuro

Uno de los principales añadidos que se podrían realizar en una posible futura actualización, sería la implementación de más tipos de recomendación en el widget. Es decir, a parte de la recomendación personalizada y no personalizada sería sencillo hacer uso de la matriz de usuarios-items, figura 2.2, para añadir otros tipos como: un ranking con los posts más populares, con los posts más desconocidos de la web, o con los posts más nuevos.

Otra variación interesante para mejorar la calidad de las recomendaciones es la de eliminar de los resultados aquellas que el usuario ya haya visitado con antelación, dando valor a la aparición de novedades. Además, se podría estudiar hacer uso de la IP del usuario, que ya se recoge actualmente, para identificarle en el caso de que no hubiera iniciado sesión.

En relación con la matriz, se podría realizar una modificación más profunda y añadir más información acerca de la interacción del usuario, como podrían ser el número de clics o tiempo de permanencia en la página, por ejemplo.

Una nueva funcionalidad, que sería sencilla de realizar una vez que se tiene la arquitectura montada, es la creación de un buscador, el cual se podrá explotar de diversas formas, ya sea con un sencillo buscador en la web o mediante un bot conversacional. Esto no supondría mucha complejidad adicional puesto que bastaría con añadir una clase al índice que hiciese uso de este para hacer consultas y añadir el método en ambas API REST del proyecto, no obstante, el intérprete de lenguaje natural para el bot sí tendría que hacerse desde cero o utilizar herramientas o librerías externas.

Finalmente, un elemento muy importante si se quiere usar en producción, aunque no tiene relación directa con el tema del proyecto, es la implementación de seguridad en las comunicaciones, tanto de autenticación a la hora de usar los métodos de ambas API REST, las del núcleo e índice, como enviar la información encriptada entre módulos, es decir, hacer uso de HTTPS.

# Bibliografía


[1] F. Ricci, L. Rokach, and B. Shapira, eds., *Recommender Systems Handbook*. Springer, 2015.

[2] B. Liu, *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data. Second Edition*. Data-Centric Systems and Applications, Springer, 2011.

[3] S. C. Fuentes Reyes and M. Ruiz Lobaina, "Minería Web: un recurso insoslayable para el profesional de la información," *ACIMED*, vol. 16, pp. 0 – 0, 10 2007.

[4] Redacción, "Ee.uu. teme que se expongan a luz pública sus programas de ciberespionaje," *La vanguardia*, Jun 2013.

[5] A. Serena, "Francia buceará en las redes sociales para combatir el fraude fiscal," *La voz de Galicia*, Oct 2019.

[6] C. G. y. A. R.-G. Ernestita Menasalvas, "Big data en salud: Retos y oportunidades," *Universidad Politécnica de Madrid*, 2017.

[7] C. M. Álvarez and L. P. M. José, *Derecho al olvido y a la intimidad en Internet: el nuevo paradigma de la privacidad en la era digital*. Reus, 2015.

[8] "What is a web crawler and how does it work?." https://en.ryte.com/wiki/Crawler.

[9] "W3schools about xpath." https://www.w3schools.com/xml/xpath_intro.asp.

[10] "Official jsoup documentation." https://jsoup.org/apidocs/.

[11] "Official pyspider documentation." http://docs.pyspider.org.

[12] "Official scrapy documentation." https://docs.scrapy.org.

[13] R. Baeza-Yates and B. A. Ribeiro-Neto, *Modern Information Retrieval - the concepts and technology behind search, Second edition*. Pearson Education Ltd., Harlow, England, 2011.

[14] "Official apache lucene documentation." https://lucene.apache.org/core/documentation.html.

[15] "Official elasticsearch documentation." https://www.elastic.co/guide/.

[16] R. Marín, "Cms vs framework. diferencias, ventajas y desventajas.," *Revistadigital INESEM*, May 2018.

[17] "Official wordpress website." https://es.wordpress.com/.

[18] "Official joomla website." https://www.joomla.org/.

[19] "Official blogger website." https://www.blogger.com.

[20] "Official symfony website." https://symfony.com/.

[21] "Official angularjs website." https://angularjs.org/.

[22] "Official django website." https://www.djangoproject.com/.

[23] "¿qué porcentaje de sitios web son wordpress en 2019?." https://adictec.com/estadisticas-sitios-web-wordpress/.

[24] "Comparison of the usage statistics of wordpress vs. joomla for websites." https://w3techs.com/technologies/comparison/cm-joomla,cm-wordpress, Apr 2020.

[25] "Official woocommerce website." https://woocommerce.com/blog/.

[26] "Datanyze: Woocommerce stats." https://www.datanyze.com/market-share/e-commerce-platforms--6/woocommerce-market-share.

[27] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker, "Constraint-based recommender systems," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 161–190, Springer, 2015.

[28] X. Ning, C. Desrosiers, and G. Karypis, "A comprehensive survey of neighborhood-based recommendation methods," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 37–76, Springer, 2015.

[29] Y. Koren and R. M. Bell, "Advances in collaborative filtering," in *Recommender Systems Handbook* (F. Ricci, L. Rokach, and B. Shapira, eds.), pp. 77–118, Springer, 2015.

[30] "Official curl on php documentation." https://www.php.net/manual/es/book.curl.php.

[31] J. M. Rodríguez, "Normalización del texto: Normalización de palabras y stemming." http://pdln.blogspot.com/2012/10/normalizacion-del-texto-normalizacion.html, Oct 2012.

[32] D. Simonovic, "Un comienzo con microservicios: Un tutorial de dropwizard." https://www.toptal.com/java/un-comienzo-con-microservicios-un-tutorial-de-dropwizard, Apr 2017.

[33] I. García and A. Bellogín, "Towards an open, collaborative REST API for recommender systems," in *Proceedings of the 12th ACM Conference on Recommender Systems, RecSys 2018, Vancouver, BC, Canada, October 2-7, 2018* (S. Pera, M. D. Ekstrand, X. Amatriain, and J. O'Donovan, eds.), pp. 504–505, ACM, 2018.

# Apéndices

# A

# Diagramas

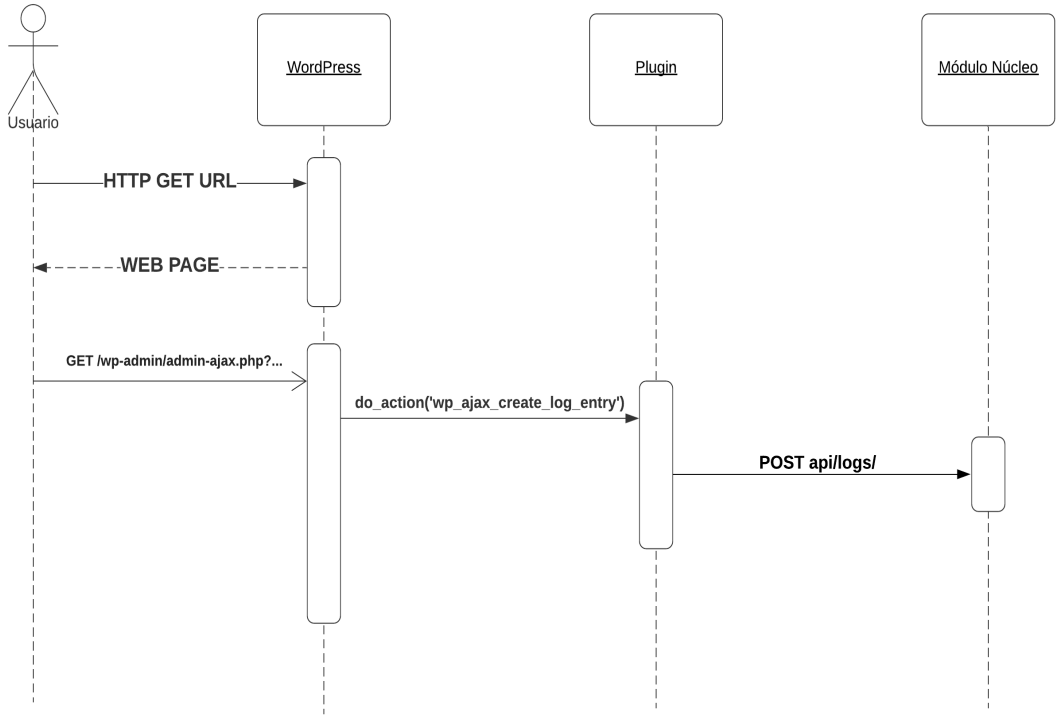


**Figura A.1:** Diagrama de secuencia para la obtención de un log.

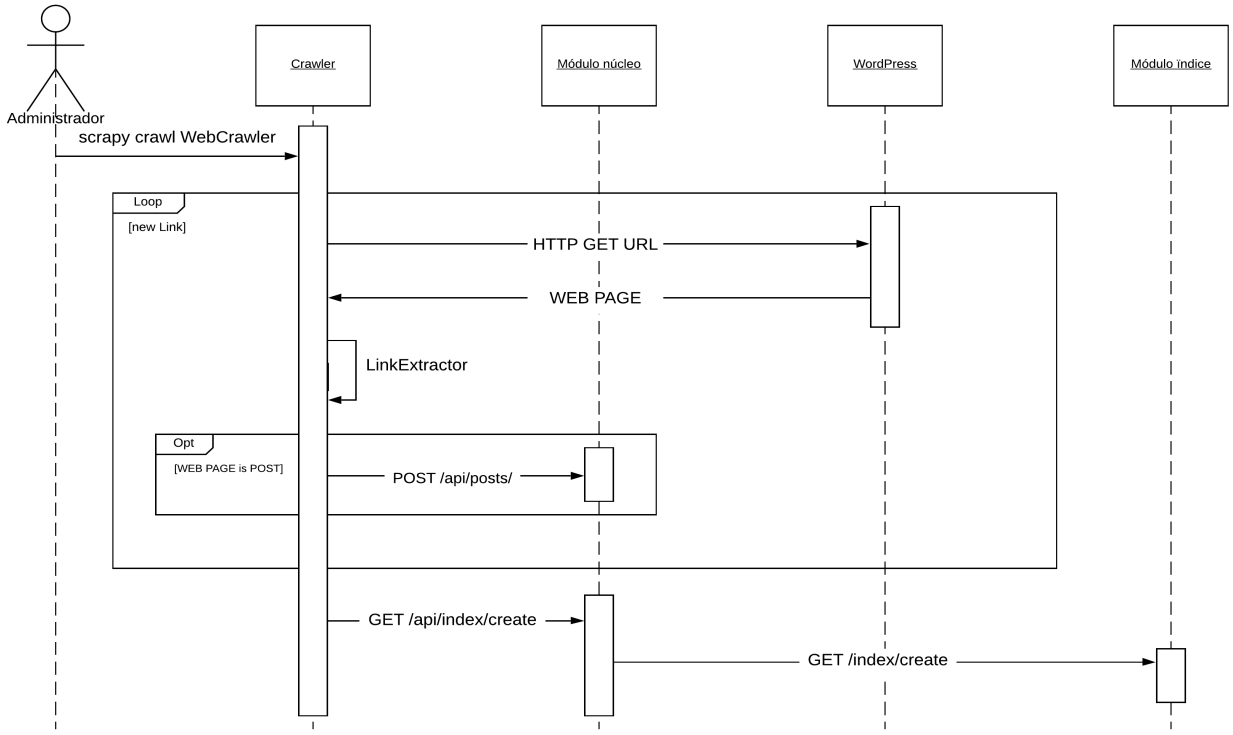


**Figura A.2:** Diagrama de secuencia para el inicio del crawler, y además, la creación del índice.

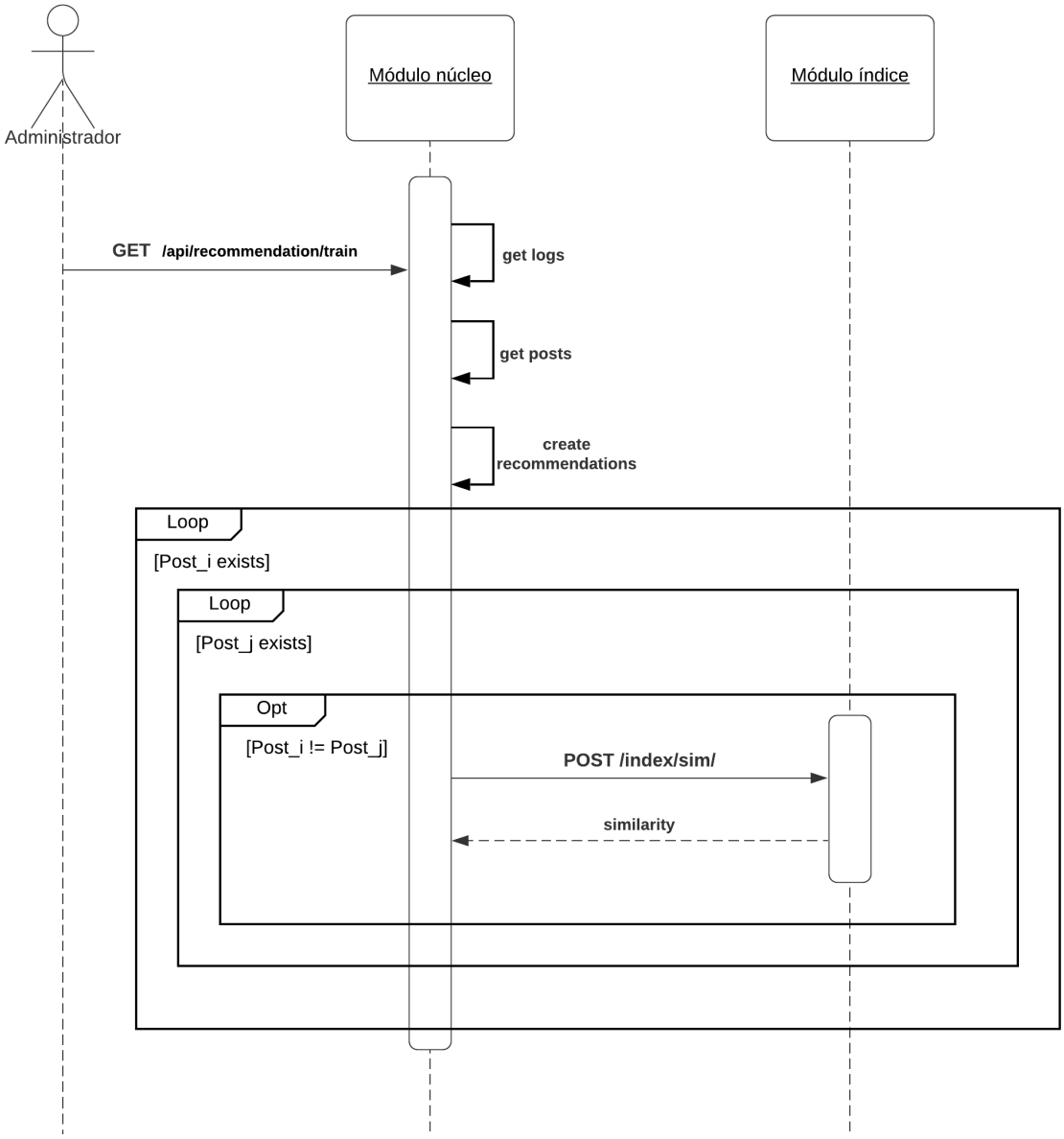


**Figura A.3:** Diagrama de secuencia sobre el proceso de entrenamiento.

# B

# Instalación y configuración

## B.1. Software requerido

A continuación se detallará el software, ya sean librerías, programas o herramientas requeridas para hacer funcionar cada componente. En cuanto al hardware, mientras este sea compatible con el software requerido, solo va a influir en la cantidad de tráfico que podrá soportar el servidor.

### Módulo núcleo

La versión de python usada para el desarrollo ha sido **Python 3.7**, aunque también se ha probado en **Python 3.6** y funciona correctamente. Sin embargo, la versión 2.X o versiones iguales o inferiores a la 3.5 no son compatibles.

Los paquetes de python necesarios son:

pip install djangorestframework.

pip install requests (en principio ya viene con la instalación de python).

pip install numpy (para la matriz de recomendaciones).

Run: **$>** `python manage.py runserver`

### Módulo crawler

Al igual que en el caso anterior ha sido desarrollado usando **Python 3.7** y comparte la misma compatibilidad de versiones.

Los paquetes de python necesarios son:

pip install scrapy.

pip install requests (en principio ya viene con la instalación de python).

pip install numpy (para la matriz de recomendaciones).

Run: **$>** `scrapy crawl WebCrawler`

### Módulo índice

Este módulo ha sido desarrollado y ejecutado bajo Java 8 y para la ejecución del mismo se ha hecho uso del entorno de desarrollo **Eclipse**.

Al ejecutarlo hay que añadir el argumento: `server [archivo de configuracion]`

En este caso: `server server.yml`

En cuanto a las bibliotecas o librerías usadas, como funciona bajo **Maven**, no es necesario descargar ni instalar nada. **Maven** es una herramienta enfocada en la gestión y construcción de proyectos Java, que, mediante una configuración XML donde se establecen las bibliotecas a usar, se encarga de descargarlas e implementarlas en el proyecto para una correcta compilación del mismo.

### Plugin

El plugin ha sido probado bajo WordPress 5.2.5 y usando la versión 7.3.9 de PHP.

## B.2. Configuración

Primero, antes de empezar a probar el sistema, hay que completar los archivos de configuración correctamente. En principio, no habrá que tocar los elementos relacionados con las APIs, pues a menos que se modifique el código eso no va a cambiar, pero si será necesario adaptar las IPs, puertos o la web de prueba.

En este caso, como todos los módulos del sistema se van a ejecutar bajo el mismo ordenador, las IPs serán las mismas, localhost o 127.0.0.1. Sin embargo, deberemos aportar puertos distintos.

Comenzamos con el plugin, para insertarlo hay que copiar la carpeta en el directorio de `.../WordPress/wp-content/plugins`. Después, podemos abrir el código fuente PHP y editar las variables de configuración, que se encuentran al principio: con la IP, puerto y métodos de la API del módulo núcleo (estos valores deberán coincidir con los establecidos en dicho módulo).

```
/**************************************************************************
 **                                                                      **
 **                    VARIABLES DE CONFIGURACIÓN                        **
 **        (IP del servidor de recomendacion, metodos de este...)        **
 **************************************************************************/
$RECOMENDATION_SERVER_IP = 'http://127.0.0.1:8000/';
$LOG_METHOD = 'api/logs/';
$REC_NOTPERS_METHOD = 'api/recommendation/nopersonalizada';
$REC_PERS_METHOD = 'api/recommendation/personalizada';
```

**Figura B.1:** Variables de configuración del plugin. Se encuentra en las primeras líneas de *recommendation_plugin.php*.

Una vez configurado, podremos activarlo en la sección de plugins del panel de control de WordPress, y tras esto, ya podremos añadir un nuevo Widget a partir del cual se mostrarán las recomendaciones. En el panel de control del widget se puede configurar el tipo de recomendación a usar y los resultados a devolver, en este caso se van a añadir dos (uno para cada tipo de recomendación), para probar correctamente el sistema.

A continuación, tocar seguir con el módulo núcleo, abrir el archivo **settings.py** y configurar las primeras variables: IP, puerto y método, necesarios por la API del módulo índice (de nuevo, tendrá que coincidir con los valores configurados en el índice).

Después, para ejecutar el módulo basta con ejecutar en la raiz del proyecto lo siguiente: `python manage.py runserver`, por defecto el puerto es el 8000, aunque, si queremos modificar el puerto habrá que añadir una pequeña extensión al comando: `python manage.py runserver 0.0.0.0:[PUERTO]`.

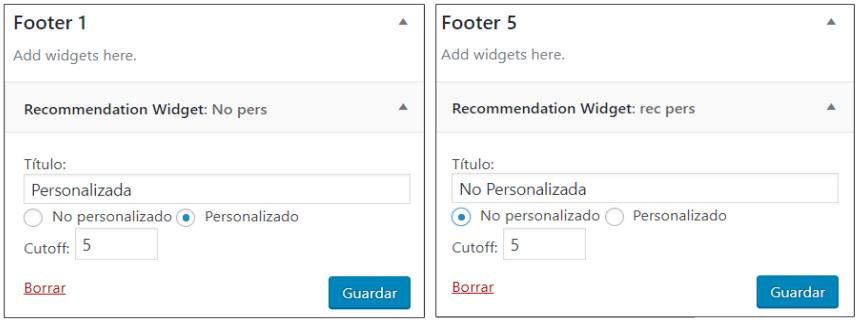


**Figura B.2:** Menú de configuración del widget de recomendaciones.

```
# Project configuration connections
INDEX_IP = "127.0.0.1"
INDEX_PORT = "5000"
INDEX_CREATE_METHOD = "/index/create"
INDEX_READ_METHOD = "/index/read"
INDEX_SIM_METHOD = "/index/sim"
INDEX_TOPSIM_METHOD =  "/index/sim_top"
```

**Figura B.3:** Variables de configuración del módulo núcleo, *settings.py*.

Una vez iniciado se quedará a la espera de peticiones, y si es la primera vez, con la base de datos de logs y posts vacía.

```
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
April 22, 2020 - 18:17:18
Django version 2.2.6, using settings 'CoreRestApi.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

**Figura B.4:** Módulo núcleo iniciado a la espera de peticiones.

En el módulo índice, hay que modificar el archivo **server.yml**, tanto IP y puerto, como una variable, **indexfields**, la cual contiene una lista con los campos que va a usar el índice junto con los pesos de esos campos, tanto para calcular la similitud de cada uno como la total. Hay que decir que el valor dado a los pesos no tiene que sumar 1, el propio índice se encarga de normalizarlo en la franja **[0-1]**. Es obligatorio que los elementos de esta listas formen parte de los atributos de los que están formados los posts en el módulo núcleo. La variable **jsonPathField** debe contener el nombre del atributo del post donde se almacena su URL, este campo debe ser fijo obligatoriamente pues actúa de identificador.

Para ejecutar, se puede realizar desde el IDE de Eclipse o compilar y obtener el ejecutable. En este último caso, **server.yml** debe estar junto al programa. Una vez ejecutado el módulo, al igual que el núcleo queda a la espera de peticiones.

Finalmente, falta el crawler, al igual que en caso anterior hay que modificar un archivo en la raíz del proyecto, **conf.json**. Además de establecer IP, puerto y métodos del módulo núcleo, hay que indicar el


**Figura B.5:** Respuesta JSON al obtener logs y posts de la base de datos. Como es la primera vez está vacía.

```yaml
# Configuracion del servidor donde se encuentran los posts (CoreRestApi)
restApiIp: 127.0.0.1
restApiPort: 8000
restApiMethod: api/posts/

# Nombre del campo donde se almacena la ruta de los posts en el json
jsonPathField: path

# Nombre del resto de campos (con sus pesos) que se van a usar para indexar
# El valor de los pesos se normaliza por tanto se puede poner cualquier valor
indexFields:
  - field: title
    weight: 0.2
  - field: content
    weight: 0.4
  - field: categories
    weight: 0.4

server:
  applicationConnectors:
  - type: http
    port: 5000
  adminConnectors:
  - type: http
    port: 5001
```

**Figura B.6:** Archivo de configuración del módulo índice, *server.yml*.

```
16:30:10,039] org.eclipse.jetty.server.handler.ContextHandler: Started i.d.j.M
16:30:10,232] org.eclipse.jetty.server.AbstractConnector: Started application@
16:30:10,236] org.eclipse.jetty.server.AbstractConnector: Started admin@27cf31
16:30:10,236] org.eclipse.jetty.server.Server: Started @2172ms
```

**Figura B.7:** Módulo índice iniciado a la espera de peticiones.

dominio de la web, una semilla con su página principal, por ejemplo, y las tres direcciones XPath, ya explicadas en su apartado 3.2.3, para que la araña sea capaz de extraer ciertos elementos de la web.

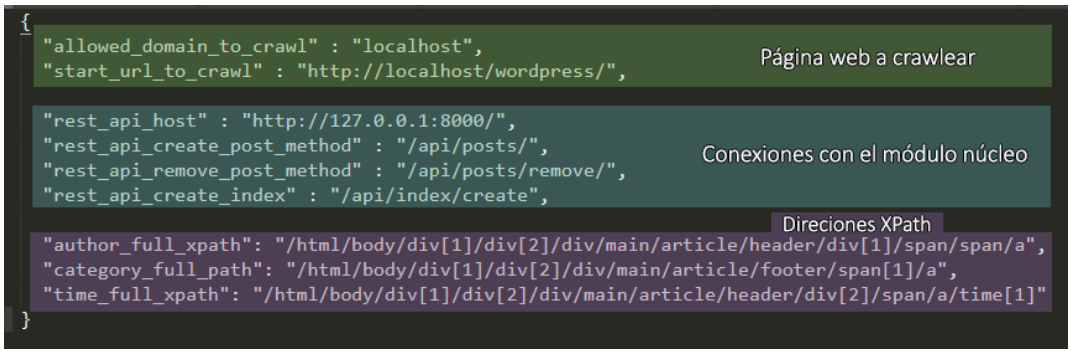


**Figura B.8:** Archivo de configuración del crawler, *conf.json*.

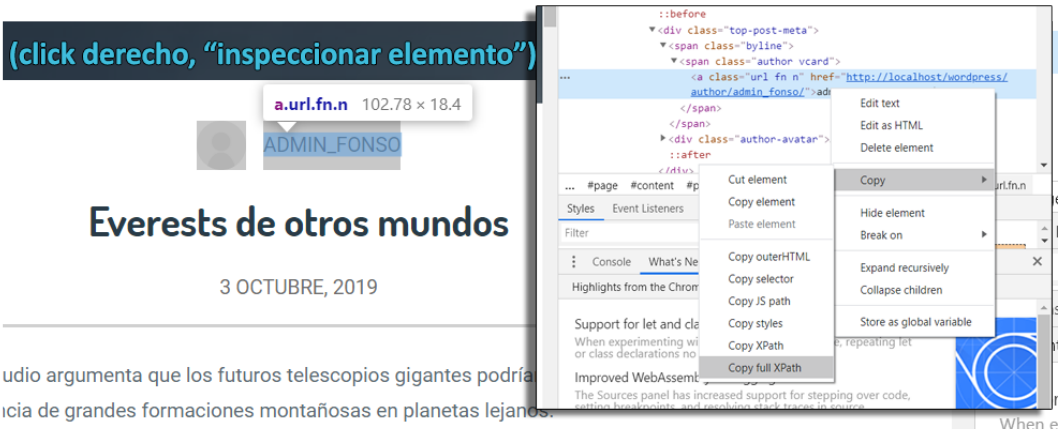


**Figura B.9:** Ejemplo de obtención de una dirección XPath.


UAM
UNIVERSIDAD AUTONOMA
DE MADRID