Universitat Politècnica de Catalunya

Treball Final de Grau

Bachelor Degree in Informatics Engineering

Plataforma web para la búsqueda y recomendación de artículos científicos

Autores
Ruben Bagan Benavides
Alberto López Sánchez

Fecha de la defensa
28 de Junio del 2018

Director del proyecto
Ricard Gavaldà Mestre

Departamento del director del proyecto
Computación

Especialidad
Computación

Q2 Curso 2017-2018

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Agradecimientos

Dar las gracias a nuestro director el Cu. Ricard Gavaldà Mestre por dirigir el proyecto y motivarnos a seguir hacia adelante cuando más se necesitaba.

Abstract

Català

Aquest projecte proposa la realització d'una plataforma web sota llicència de software lliure per a la cerca i recomanació d'articles científics. La idea d'aquest projecte és presentar un prototipus per veure la seva viabilitat utilitzant únicament els articles de la branca de Computer Science. Consisteix a crear una base de dades de papers a partir de les dades publicades en alguns dels llocs webs més importants: ArXiv, DBLP i Google Scholar, entre d'altres.

L'obtenció de les dades es fa mitjançant un procés de web crawling. Les dades recopilades són processades per generar informació per al recomanador. Per a la recomanació ens basem en la similitud del contingut dels articles i en el graf social generat per la comunitat acadèmica extret de les dades que hem recopilat.

La interacció amb el sistema es fa a través d'un client web.

Castellano

Este proyecto propone la realización de una plataforma web bajo licencia de software libre para la búsqueda y recomendación de artículos científicos. La idea de este proyecto es presentar un prototipo de su viabilidad usando únicamente la rama de *Computer Science*. Consiste en crear una base de datos de *papers* a partir de los datos publicados en los sitios web más importantes: ArXiv, DBLP y Google Scholar, entre otras.

La obtención de los datos se hace mediante un proceso de *web crawling* y explicamos que dificultades hemos tenido. Los datos recopilados son procesados para generar información para el recomendador. Para la recomendación nos basamos en la similitud del contenido de los artículos y el grafo social generado por la comunidad académica extraído de los datos que hemos recopilado.

La interacción con el sistema se hace a través de un cliente web.

English

This project proposes the realization of a web platform under free software licence for the search and recommendation of scientific papers. The main idea of this project is to show a prototype to find its viability using the articles in Computer Science only. Consists in create a database from the publications of some of the most important websites like ArXiv, DBLP and Google Scholar. Obtaining the data is made through a web crawling process.

We also explain the difficulties we had during that process and how we solve it. The data collected is processed for generate the required information for the recommender system. To do recommendations we use the similarity between the articles and the social graph generated by the academic community extracted from the data we have get.

The interaction with the system is through a web client.

keywords: crawler, recomender, web, platform, graph, social, papers, authors, journals, conferences, python, mongodb, neo4j, opensource, free, pagerank

Índice general

Índice de tablas

Índice de figuras

Introducción

Este documento está estructurado en tres partes. Primero describimos la concepción del proyecto, donde explicamos el alcance, hacemos la planificación temporal y económica, analizamos la sostenibilidad en los tres ámbitos (económico, ambiental y social) y comentamos las leyes y normativas a las que está sometido.

En la segunda parte se encuentra todo lo relacionado con el buscador. En primer lugar, el crawler que se ha creado para la recopilación de los *papers* y la creación de la base de datos, y como hemos superado todos los obstáculos que hemos encontrado. Por último, se comentan los módulos que generan el grafo social y como se extrae información de este.

En la tercera parte se encuentra todo lo relacionado con la creación del software necesario para ofrecer una interfaz de usuario cómoda para interactuar con el sistema y realizar las recomendaciones. Esto incluye la especificación, diseño e implementación de los diferentes módulos que componen esta parte del sistema.

Por último, en las conclusiones, hablamos de los resultados obtenidos del proyecto y de las posibles acciones futuras que se podrían llevar a cabo.

Parte I

Gestión del proyecto

Capítulo 1

Definición del alcance y contextualización

1.1. Contextualización

La comunidad científica estructura su comunicación en forma de documentos que aparecen como artículos y editoriales en revistas, comunicaciones y ponencias en actas de congresos, o informes técnicos en repositorios especializados. Hay también presentaciones en vídeo, presentaciones orales que no quedan por escrito, etc. pero estas no serán objeto de este proyecto. Nos concentraremos en las contribuciones que quedan en forma de documento, y concretamente de documento digital accesible en la web.

Por simplificar los distintos tipos (artículo, ponencia, comunicación, charla invitada, editorial, ...) llamaremos a todos estos documentos *papers*, usando el anglicismo que coloquialmente usa incluso en la comunidad científica de habla española.

Según cifras del 2009[18], cada año se publican aproximadamente 2,5 millones de *papers* en todo el mundo académico. El repositorio DBLP[40], que indexa una parte de los *papers* de *computer science*,

podemos ver como de grande es el problema:

- Conferencias: 5.356
- Autores: 2.052.722
- Papers: 4.086.504
- Revistas: 1.567

Aunque redujéramos el área de información a una subárea especifica de *Computer Science* como podría ser la inteligencia artificial, el número puede ser muy grande.

Un investigador profesional, que tiene normalmente una área mucho más reducida y especializada, podría encontrar interesantes (o incluso cruciales) para su investigación varios cientos de artículos por año. El problema es doble: por un lado, localizarlos; seguramente seguirá algunas de las revistas y conferencias más relevantes para su trabajo, pero muchos de los artículos relevantes serán publicados en sitios que no comprueba regularmente.

Por otro lado, lo más probable es que no tenga tiempo para leerlos todos; por este motivo, y que tenga que priorizar. La priorización (antes de leerlos) puede hacerse en base a información proporcionada directa o indirectamente por sus colegas, que confieren prestigio o reputación a artículos, por ejemplo, citándolos. Ha aparecido toda una industria de indicadores de impacto científico (de personas, de revistas, ...) que, por desgracia no terminan de convencer a los investigadores mismos, y que acaban teniendo más la función de evaluarlos que de orientarlos.

Este problema genera una necesidad para los investigadores. Encontrar una herramienta que según sus criterios encontrar entre todos los *papers* que se publican cuales son los más relevantes para él y notificarle cuales son. Esta noción de relevancia debe tener en cuenta la proximidad a sus intereses (su campo de trabajo) y la reputación que, de distintas formas, la comunidad da a los trabajos.

Existen plataformas que intentan dar solución como puede ser: ResearchGate[31] y Google Scholar[37], pero no tienen en cuenta de manera suficiente los criterios del usuario, como explicaremos después.

1.2. Formulación del problema

Ofrecer al usuario una plataforma web que proporcione las herramientas necesarias para encontrar de forma rápida y sencilla los *papers* que sean más relevantes para el usuario. El usuario podrá dar *feedback* al sistema a partir de un sistema de valoraciones y filtrar su búsqueda usando diversos criterios como, fechas de publicación, autores relacionados, importancia dentro de una comunidad, etc.

Para ello la plataforma tiene que:

- Recopilar toda la información necesaria a través de diferentes páginas web o bases de datos con un *Crawler* (Capítulo 6) con el objetivo de obtener información sobre: *papers*, autores, revistas y conferencias.
- Organizar y procesar toda la información, para que otros módulos puedan hacer uso de ella. La información tiene que quedar estructurada de manera que la búsqueda y recomendación posteriores sean eficientes y ajustadas. Capítulos: 7 (Base de datos), 8 (*Graph Builder*) y 9 (*Graph Analyzer*)
- Permitir al usuario interactuar con el sistema mediante un *front-end* que es el cliente web. Capítulo 13.
- Recoger el *feedback* del usuario. Para ello la plataforma:
  - Poner a disposición del usuario un sistema de valoraciones del estilo "Me gusta/No me gusta".
  - Un sistema de *keywords*, donde las *keywords* seleccionadas en el perfil del usuario contribuirán positivamente a las recomendaciones.
  - Recordar los papers que ha visitado el usuario.

1.3. Actores

1.3.1. Desarrolladores del proyecto

Este proyecto está formado por dos desarrolladores: Ruben Bagan Benavides y Alberto López Sánchez.

En un proyecto normalmente participan diferentes roles, por ejemplo: *Project Engineer [25], Software Engineer [26], Developer Engineer y QA Engineer* [52]. En nuestro caso tenemos que ser multidisciplinarios y ejercer la función de diferentes roles.

Al ser dos desarrolladores se ha hecho una división clara de las tareas que hace cada uno especificadas en la sección 5. En el caso de Ruben Bagan Benavides se encarga de los módulos que corresponden a la búsqueda y Alberto López Sánchez al módulo de recomendación y el cliente web. Ambos miembros han acordado la especificación y el diseño de todos los módulos de la plataforma.

1.3.2. Director del proyecto

El director del proyecto es el Cu. Ricard Gavaldà Mestre [38] profesor del departamento de Computer Science (UPC-CS) [29] de la Universitat Politècnica de Catalunya - BarcelonaTech (UPC) [41] y miembro de LARCA (Laboratory for Relational Algorithmics, Complexity and Learning) research group [43]. Tiene el rol de guiar y supervisar el desarrollo correcto del proyecto.

1.3.3. Actores beneficiarios

La plataforma web puede ser usada por cualquier tipo de persona, desde empresas a una persona en particular para aprendizaje autónomo, pero de todos ellos destacamos los que creemos que son los usuarios donde tenemos que centrar nuestros esfuerzos:

- *Universidades*: Esto incluye estudiantes de grado, máster y doctorado, así como también investigadores. Las personas pertenecientes a esta entidad los *papers* son su principal fuente de información, por ejemplo, justificar el funcionamiento de un algoritmo usando como justificación un *paper* o para construir nuevas investigaciones en base a los resultados de otros *papers*.
- *Empresas tecnológicas*: Es común que las grandes empresas tengan departamentos de R+D o incluso de investigación pura. Estos departamentos también tienen la necesidad de consultar las investigaciones realizadas por otros para construir nuevas tecnologías.

1.3.4. Usuarios

Consideramos que los mayores beneficiados de esta plataforma son los investigadores, ya que les proporcionamos una nueva herramienta para buscar y mantenerse informado de los *papers* más relevantes para ellos, lo que les permite mejorar su productividad y no quedarse atrás en sus campos.

1.4. Estado del arte

Tal y como se comenta en la contextualización, Sección 1.1, actualmente hay en funcionamiento diversos servicios como ResearchGate, Mendeley [58] o Google Scholar, que prestan un servicio similar al nuestro, nosotros intentamos mejorar las recomendaciones en base a la historia y las preferencias ya mostradas por el usuario.

Para realizar recomendaciones existen técnicas que tienen en cuenta las valoraciones directas o indirectas que un usuario puede dar sobre un elemento de la web, como, por ejemplo, un vídeo de la plataforma Youtube. Valoraciones directas, por ejemplo, un sistema (me gusta, no me gusta). Valoraciones indirectas, por ejemplo, una compra de un producto (en general, si alguien compra algo es que cree que le va a gustar), comentarios que pueden analizarse para detectar su sentimiento [74], preferencias de los contactos de uno mismo, etc.

Las técnicas que usan los sistemas recomendadores pueden clasificarse en:

- Basadas en contenido: Son técnicas que tienen en cuenta las características de los objetos a recomendar, como por ejemplo, el género o el director de una película o, las especificaciones técnicas de un ordenador, o, como en nuestro caso con los *papers*, el contenido de los documentos.
- Collaborative filtering: Son técnicas que usan las valoraciones que ha dado el usuario hasta el momento, buscan usuarios que hayan dado valoraciones similares, y recomendar entonces objetos que hayan gustado a estos pero que el primero no haya visto.
- Hibridas: Combinan las dos técnicas anteriores.

Estas técnicas se usan para recomendar para recomendar cualquier tipo de objetos, por ejemplo, papers. Para poder usar una técnica basada en contenido necesitamos definir como podemos comparar dos *papers*, Para ello existen técnicas que usan las palabras que forman los documentos para calcular la similitud entre estos.

Se han propuesto un buen número de técnicas para esto.

- TF-IDF (Term Frquency - Inverse Document Frequency) [27]. Esta técnica clásica, a partir de un conjunto de documentos calcula un peso para las palabras de cada documento. Es un modelo *bag-of-words* porque ignora el orden y posición de las palabras y solo considera cuantas veces aparece cada palabra en el documento. De esta manera, cada documento puede verse como un vector indexado por las palabras del conjunto y cada componente es el peso de la palabra en este documento. A partir de aquí, puede utilizarse cualquier noción de distancia entre vectores para decidir cuan similares son dos documentos. La distancia dada por el coseno del ángulo es una elección muy frecuente.
- Doc2Vec [10] es una técnica mucho más reciente pero que esta dando muy buenos resultados. A diferencia de la técnica anterior donde la posición que ocupan las palabras en el texto no es relevante, esta técnica tiene en cuenta el orden en el que aparecen, intentando determinar la semántica de las palabras teniendo en cuenta qué otras palabras están en su entorno inmediato.

Para refinar los resultados de los métodos anteriores, sobre todo para TF-IDF, existen técnicas de procesamiento del lenguaje natural [23] como la lematización [15] y el *Stemming* [16] que nos permiten mejorar las comparaciones limpiando y reduciendo las palabras de los documentos.

Los buscadores (y recomendadores) basados únicamente en contenido tienen grandes limitaciones para encontrar contenido interesante en los grandes volúmenes de datos que contiene Internet hoy en día. Si solo buscamos las palabras que el usuario introduce en su consulta, casi siempre habrá cientos o millones de documentos que las contienen. Algunas de ellos procederán de fuentes genuinas

y con reputación, y muchos serán basura, spam, o copias de otros documentos. Una de las grandes genialidades de Google cuando apareció en 1996 es introducir una manera de asignar un grado de "reputación" a cada documento, con la famosa medida Pagerank [24].

Como la reputación es un concepto social, el PageRank se construye a partir de los enlaces que (otros) generadores de contenidos de internet han puesto hasta este documento, de quienes a su vez apuntan a estos usuarios, etc. Es natural pensar entonces en la Web como un *grafo* en qué los nodos son documentos, páginas o usuarios, y los arcos son los enlaces que llevan desde uno a otro. En redes sociales particulares, los arcos pueden representar otras relaciones como likes, recomendaciones, menciones, seguimientos, ...

A partir de la idea del buscador de Google y de Pagerank, se ha convertido en habitual construir un grafo entre documentos (o páginas, usuarios...) para representar relaciones de tipo social, reputación o confianza que en principio no pueden deducirse del contenido. Una vez construido un grafo, hay información de muchos tipos que pueden usarse para mejorar búsqueda y reputación, además, de Pagerank. Por ejemplo, podemos usar técnicas de detección de comunidades propuestas en artículos como [4].

Para mejorar las recomendaciones de un sistema basado en collaborative filtering podemos usar la información social sacada con una de las técnicas anteriores y aplicar los métodos que propone este articulo [5]. Por otra parte, podemos hacer recomendaciones haciendo clustering de las comunidades formadas en los grafos como se propone en [12]. Para detectar las comunidades que forman los autores sabemos que existen varios algoritmos y metodologías. La mayoría tiene un coste de cómputo muy alto. Por ejemplo, el algoritmo de Girvan–Newman [2]. En [14] podemos ver una comparación de diversos algoritmos.

Hay que notar que no hay un solo tipo de grafo social que pueda emplearse. En el caso de papers, uno puede construir un grafo en base a qué papers citan a cuales. Pero también puede usarse el grafo de coautorías: si dos autores han sido coautores de un paper, es probable que los papers del segundo autor sean interesantes para quien busca los papers del primer autor. Este enfoque se usa en [63].

El estado del arte de los mejores algoritmos para la detección de comunidades parece que no está del todo claro [73], pero hay un cierto consenso en que el algoritmo de Louvain [54] es un buen método para encontrar comunidades que no se superpongan. En [7] muestran una generalización del algoritmo de Louvain para redes complejas con gran cantidad de nodos y relaciones.

Para decidir como almacenar el grafo generado por los datos se ha mirado el estado del arte de las bases de datos una buena solución tanto por velocidad como por facilidad de uso es Neo4j [60].

1.4.1. Decisión

Hemos decidido implementar una solución nueva basándonos en los trabajos citados anteriormente ya que los proyectos existentes son de código cerrado, pero muchos de los algoritmos y las técnicas sí están descritos públicamente.

1.4.2. Obstáculos

- Trabajo en grupo. Lo más común en un TFG es que el proyecto lo desarrolle un único alumno y rara vez en grupo de dos. Esto incrementa la dificultad del proyecto porque el proyecto está dividió en dos TFG's distintos sobre un mismo tema. Además debemos de tener una buena comunicación entre nosotros para que el proyecto avance sin problemas.
- Falta de papers. Si queremos que la plataforma tenga viabilidad necesitamos que los autores de los *papers* nos cedan el derecho a publicar sus investigaciones en nuestro portal. Esto nos permite mejorar la experiencia de los usuarios, y ofrecer la comodidad de consultar dichas investigaciones en la misma plataforma sin salirse.

- Derechos de autor. Este desafío está relacionado con el anterior y es que no podemos agenciarnos con los *papers* publicados en otras plataformas ya que vulnera los derechos de autor. Como consecuencia, la experiencia del usuario se verá afectada negativamente al tener que acceder a dichos *papers* a través de otros portales. Si que podemos hacer una mención del portal ofreciendo un vínculo directo, y también podemos proporcionar el abstract [39] de dicho paper.
- La falta de papers obliga a que tengamos que descargar el contenido de otras plataformas, pero no hacer público dicho contenido por el anterior obstáculo. Esto implica que tenemos que desarrollar *crawlers* [28], para que de forma autónoma recopilen toda la información. Esto ha llevado un tiempo considerable en el desarrollo del proyecto.
- Desconocimiento en profundidad del Data Mining [20]. Existe una asignatura optativa de Data Mining [42], pero ninguno de los integrantes ha podido cursarla. Tenemos que aprender para poder defendernos.
- Manejamos una gran cantidad de datos, aproximadamente se calcula que hay más de 60 millones de papers, y cada año se publican más de 2,5 millones según cifras del 2009 [18]. Para almacenar toda esta información contando que todos los papers tengan un peso de 1 MB (que es muy poco), necesitaríamos aproximadamente 57 TB de espacio, cosa totalmente inviable con nuestros recursos. Incluso aun reduciendo la dimensión del problema solo a una área como la Computer Science [19] el volumen de información es muy grande (mínimo 4 millones de papers). Por tanto, pensar bien que estructuras de datos usaremos y como almacenarlos es clave para poder gestionar toda esta información de forma eficiente.
- El coste computacional de manipular los papers para aplicar diferentes algoritmos tiene un coste muy elevado. Con algunas técnicas es prohibitivo. Esto tiene un efecto negativo y es que una vez empezamos los cálculos no sabremos si son correctos hasta que terminen, y en caso de error tendríamos una penalización de tiempo muy alto.
- Existen algunas plataformas web que ofrecen un servicio similar como es el caso de ResearchGate, pero no podemos competir contra ellas, ya que cuentan con una ventaja en tiempo muy elevado y más recursos humanos que nosotros. Nuestra decisión es innovar en la experiencia de usuario.

1.5. Metodología y rigor

1.5.1. Métodos de trabajo

Existen muchos métodos de trabajo en la ingeniería del software y cada uno se adapta mejor según la necesidad del proyecto. En nuestro caso consideramos que la metodología Scrum [17] es la opción que más se ajusta a nuestra necesidad por las siguientes razones:

1. El proyecto está formado por un grupo, aunque solo este compuesto por dos personas.
2. Es ideal para proyectos en entornos complejos, donde se necesita obtener resultados pronto, debido al periodo tan corto de tiempo que disponemos.
3. Flexibilidad en los requisitos.
4. Productividad.

La clave de esta metodología son los ciclos cortos (suelen ser entre 2 o 4 semanas), en las cuales se desarrolla una parte pequeña, pero dando como resultado un incremento del producto final. La siguiente figura ilustra muy bien el funcionamiento.

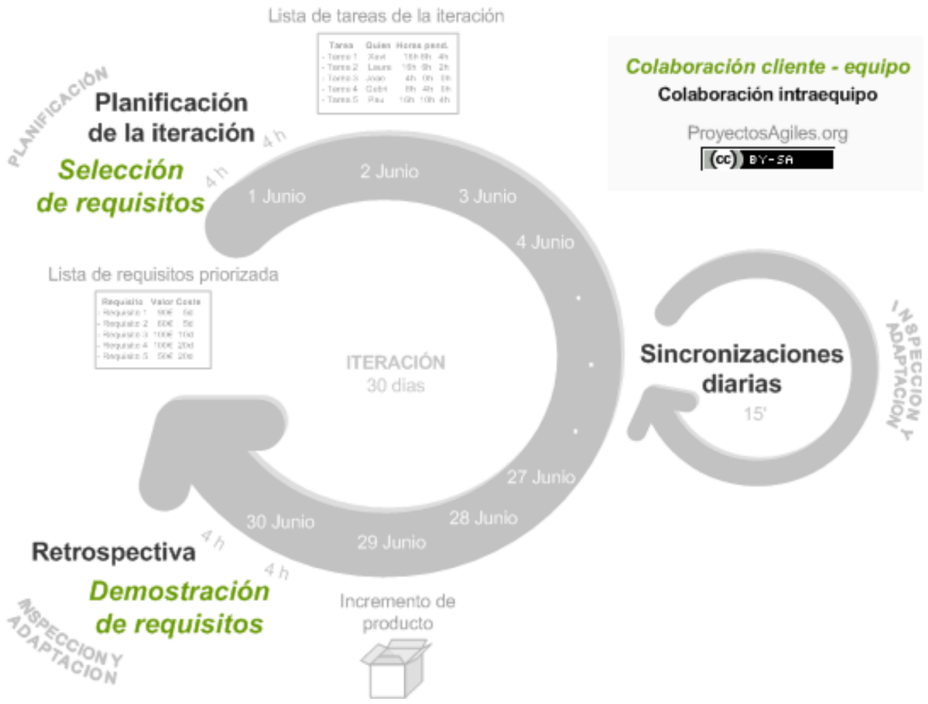

Figura 1.1: Gráfico de la metodología scrum

1.5.2. Herramientas de seguimiento

Para mantener el contacto con el director del proyecto, usaremos el correo electrónico para hacer consultas puntuales o enviar informes. Además, se concretarán días para presentar los progresos que vamos haciendo.

En cuanto al seguimiento del proyecto usaremos lo plataforma de Github [30]. Aparte de tener una plataforma donde tener un control de versiones del código, incluye herramientas útiles para poder hacer un seguimiento de proyecto bastante exhaustivo:

- Control de errores: En caso de detectar un fallo, podemos añadir una notificación para avisar de que algo necesitar ser arreglado.
- Milestones [22]: Una opción muy poco explotada generalmente, pero nos permite incrementar el rendimiento del grupo, porque se adapta muy bien con la metodología Scrum. Nos permite fijar hitos para cada ciclo.
- Projects: Otra característica importante y poco usada, es la clásica pizarra con las diferentes columnas: por hacer, en proceso, terminado, etc... Nos ayudará a organizar mejor las tareas.

1.5.3. Métodos de validación

Durante las reuniones de seguimiento con el director del proyecto, se mostrará el estado del proyecto enseñando las nuevas funcionalidades implementadas en cada ciclo. Esto nos permite poder detectar a tiempo posibles errores que han pasado desapercibidos antes de iniciar otra iteración. También puede ser un buen momento para incluir mejoras que no se tenían previstas.

1.5.4. Rigor

Para comprobar la calidad de las recomendaciones lo ideal sería someter el proyecto a varias rondas de test por diferentes grupos de investigadores, pero eso es imposible en un proyecto de este tipo dado nuestros recursos.

Los experimentos estarían formados por grupos de investigadores cada grupo especializado en diferentes áreas. Donde se buscarían papers con los que los probadores estuvieran familiarizados y mostrarles las recomendaciones que da el sistema de estos para que evaluasen la calidad.

En un futuro podrían implementarse técnicas que intentasen medir la satisfacción de los usuarios de la plataforma.

Capítulo 2

Planificación Inicial

La planificación busca plasmar que secuencia de pasos y tareas se harán para lograr nuestro objetivo. Intentando en todo momento, que las dependencias entre las tareas no afecten negativamente para así lograr una mayor paralización entre ellas y mejor aprovechamiento del tiempo.

2.1. Recursos

Los integrantes del proyecto son los recursos humanos, que asumen el papel de diferentes roles:

- Senior Project Engineer: Rol que simboliza el jefe de proyecto.
- Junior Software Engineer: Rol que simboliza el analista.
- Junior Developer Engineer: Rol que simboliza el programador.
- QA Engineer: Se encarga de verificar que todo funciona correctamente.

2.2. Planificación temporal

El diagrama de Gantt ilustrado en la Figura 2.2, muestra la división por etapas de la planificación, y por cada una cual es su distribución de tareas y quien las hace. En la Figura 2.1 están ilustradas todas las tareas del proyecto, la duración, el recurso asignado, su dependencia y finalmente, quien del grupo hace cada parte.

	Nombre	Duración	Inicio	Fin	Predecesoras	Recursos	Persona
1	**GEP**	**30días**	**26/02/2018**	**06/04/2018**		**Senior Project Engineer**	**R, A**
2	GEP: Definición del alcance y contextualización	6días	26/02/2018	05/03/2018		Senior Project Engineer	R, A
3	GEP: Planificación temporal	5días	06/03/2018	12/03/2018	2	Senior Project Engineer	R, A
4	GEP: Gestión económica y sostenibilidad	5días	13/03/2018	19/03/2018	3	Senior Project Engineer	R, A
5	GEP: Presentación preliminar	4días	20/03/2018	23/03/2018	4	Senior Project Engineer	R, A
6	GEP: Bloque de especialidad, documento final y presentación oral	10días	26/03/2018	06/04/2018	5	Senior Project Engineer	R, A
7	**Analisis y diseño**	**15días**	**19/03/2018**	**06/04/2018**		**Junior Software Engineer**	**R, A**
8	Analisis de requisitos	5días	19/03/2018	23/03/2018		Junior Software Engineer	R, A
9	Especificación	5días	26/03/2018	30/03/2018	8	Junior Software Engineer	R, A
10	Diseño y arquitectura	5días	02/04/2018	06/04/2018	9	Junior Software Engineer	R, A
11	**Desarrollo**	**40días**	**09/04/2018**	**01/06/2018**	**1,7**	**Junior Developer Engineer**	
12	Módulo del Crawler	5días	09/04/2018	13/04/2018		Junior Developer Engineer	R
13	Módulo de la BBDD	5días	16/04/2018	20/04/2018	12	Junior Developer Engineer	R
14	Módulo del Grafo	5días	23/04/2018	27/04/2018	13	Junior Developer Engineer	R
15	Módulo del Recommendador	15días	09/04/2018	27/04/2018		Junior Developer Engineer	A
16	Primer Prototipo	1día	27/04/2018	27/04/2018		Junior Developer Engineer	R, A
17	Desarrollo de la plataforma web	5días	30/04/2018	04/05/2018	16	Junior Developer Engineer	R, A
18	Ampliación: módulo del Crawler	5días	07/05/2018	11/05/2018	17	Junior Developer Engineer	R
19	Ampliación: módulo del Grafo	5días	14/05/2018	18/05/2018	18	Junior Developer Engineer	R
20	Ampliación: módulo de la BBDD	5días	21/05/2018	25/05/2018	19	Junior Developer Engineer	R
21	Ampliación: de Recomendador	15días	07/05/2018	25/05/2018		Junior Developer Engineer	A
22	Segundo Prototipo	1día	25/05/2018	25/05/2018		Junior Developer Engineer	R, A
23	Ampliación: mejoras de la plataforma web	5días	28/05/2018	01/06/2018	22	Junior Developer Engineer	R, A
24	**Release Candidate**	**10días**	**04/06/2018**	**15/06/2018**	**11,28**	**Senior Project Engineer,Junior Developer Engineer,QA Engineer**	**R,A**
25	Control de calidad y corrección de errores	5días	04/06/2018	08/06/2018		Junior Developer Engineer,QA Engineer	R,A
26	Preparación de la presentación	5días	11/06/2018	15/06/2018	25	Senior Project Engineer	R,A
27	Preparación de la exposición oral	5días	11/06/2018	15/06/2018	25	Senior Project Engineer	R,A
28	Memoria	55días	19/03/2018	01/06/2018		Senior Project Engineer	R,A

PNG Generated On: 8/4/2018 15:18:23

Figura 2.1: Tareas diagrama de Gantt

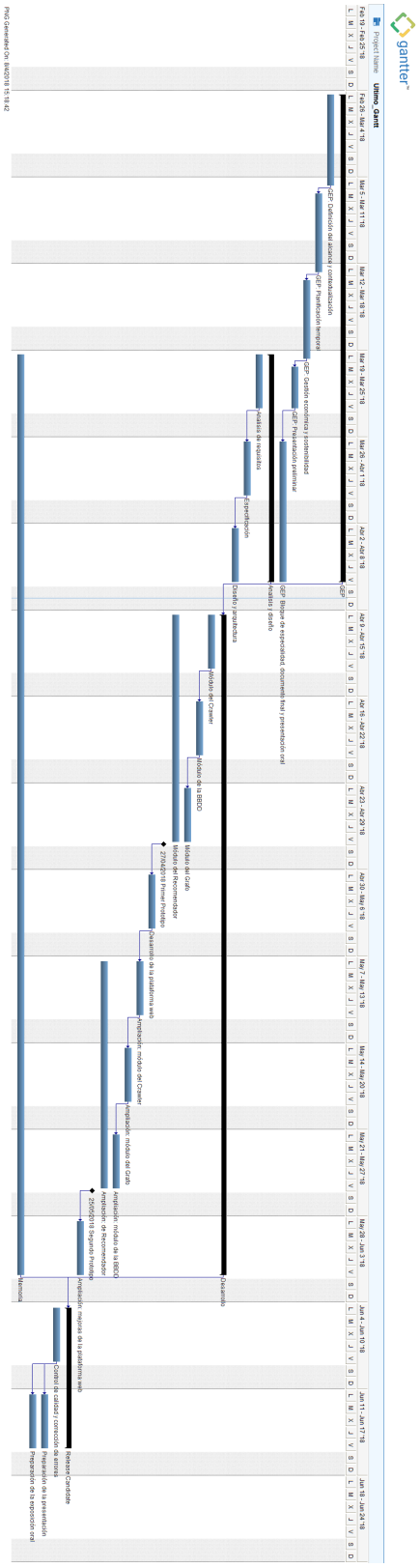

Figura 2.2: Grafico diagrama de Gantt

2.2.1. Ciclos de desarrollo

Todos los ciclos tienen un denominador común:

- *Test*: para comprobar el correcto funcionamiento de las implementaciones que se llevan a cabo se destinará siempre un día en exclusividad donde se pasará una batería de pruebas para comprobar el rendimiento y si los resultados son los esperados.
- *Documentación de la memoria*: mientras se desarrolla la plataforma se irá documentando los diferentes avances que se realizan.
- *Seguimiento del proyecto*: al igual que para la etapa de test, también se reserva un día para reunirnos con el director del proyecto y hacer un seguimiento de los progresos. En este proceso es donde se verán errores si los hay y que no se ha tenido en cuenta y posibles mejoras.

2.2.2. Etapa inicial (26/02 - 06/04)

En esta etapa se desarrollan dos actividades: elaborar la GEP y el Análisis y diseño de la plataforma que se inicia en la cuarta semana.

Por parte de GEP tenemos las siguientes entregas:

- Planificación temporal.
- Gestión económica y sostenibilidad.
- Presentación preliminar.
- Apartado especialidad de computación.

2.2.3. Etapa intermedia (09/04 - 01/06)

Esta es la etapa más larga, su duración va des de la semana 7 hasta la 14, y su función es implementar los principales módulos del proyecto, de la parte de búsqueda y de recomendación, y en paralelo hacer la memoria del proyecto.

Durante esta etapa hay dos hitos, el primer prototipo (27/04) y el segundo prototipo (25/05).

Dentro de esta etapa se desarrollan los siguientes módulos:

2.2.3.1. Módulo: Crawler (09/04 - 13/04)

Es la primera tarea y consiste en desarrollar un software que nos permita alimentar nuestra base de datos de forma automática, extrayendo toda la información necesaria de sitios web como Google Scholar, arXiv, dblp, entre otros.

2.2.3.2. Módulo: Base de datos (16/04 - 20/04)

Durante el tiempo de desarrollo de este módulo nos dedicaremos a recopilar los metadatos y los artículos con el *crawler* desarrollado por el módulo anterior, y hacer un primer preprocesado de datos para insertarlos en la BBDD.

2.2.3.3. Módulo: Graph Builder (23/04 - 27/04)

Desarrollar el módulo encargado de leer los datos almacenados en la BBDD y generar los grafos necesarios para realizar las búsquedas y recomendaciones.

2.2.3.4. Módulo: Recomendador (09/04 - 27/04)

A lo largo de toda la etapa se desarrolla en paralelo el Modulo Recomendador del proyecto, para que pueda estar listo con los datos finales para el *primer prototipo*.

2.2.3.5. Primer Prototipo (27/04)

Con todos los módulos anteriores desarrollados, se llega al hito de conseguir el primer prototipo, que consiste en una versión básica y funcional, este prototipo carecerá de interfaz gráfica. Es el punto de partida hacia un sistema más grande.

2.2.3.6. Módulo Cliente Web (30/04 - 04/05)

Con todos los módulos anteriores ya desarrollados y un primer prototipo sobre el que trabajar, desarrollaremos la primera versión del Cliente web que hará de *front-end* con el usuario.

2.2.3.7. Ampliaciones de los módulos (04/05 - 25/05)

Durante el resto de esta etapa se irán ampliando y mejorando los módulos ya existentes para lograr el segundo prototipo.

2.2.3.8. Segundo Prototipo (25/05)

En este momento tendríamos que tener prototipo con todas las características implementadas y cerradas.

2.2.3.9. Mejoras modulo Cliente Web (28/05 - 01/06)

Durante la última semana de esta etapa, acabaremos de implementar las características restantes del Cliente web.

2.2.4. Etapa final (04/06 - 15/06)

En este punto no se añaden nuevas características por cuestión de tiempo. Esta etapa se centra en la depuración de errores y la calidad del producto final. El motivo de dejar tres semanas de margen con poca carga de trabajo es para tener más flexibilidad en la planificación en caso de error, se comentará en la sección de alternativas.

2.2.5. Alternativas

- ¿Cuándo se corrigen los errores detectados en el control de seguimiento?

  Todos los ciclos se tienen en cuenta los resultados de la revisión del anterior ciclo, y como consecuencia se reserva un breve tiempo dentro del ciclo para arreglar dichos errores si lo hay. En caso contrario se sigue con la planificación establecida.

- ¿Qué pasa si no hay tiempo suficiente para implementar una característica?

  El incumplimiento del desarrollo de una característica es algo que puede darse con una probabilidad relativamente alta debido al desconocimiento del problema, el motivo puede ser por falta de previsión de tiempo (poco tiempo asignado), conocimientos demasiado básicos, etc.

  Para evitar que esto pueda afectar negativamente, si en la primera semana del desarrollo de dicha característica surgen diferentes problemas que imposibilitan su realización, automáticamente se reemplaza por una nueva más asequible debido a que se dispone de menos tiempo.

  De hecho, se tiene en cuenta dicho problema y se ha dejado un margen de tres semanas en la etapa final las cuales podemos usar en caso de necesidad.

- ¿Se desarrollará más características si se termina antes de tiempo?

  Si se da el caso de que todas las características a implementar se completan antes de tiempo y existe un exceso de tiempo, se buscara añadir de nuevas que se adecuen al tiempo excedente.

Capítulo 3

Gestión económica

En esta etapa se hace un análisis de los costes directos e indirectos del proyecto. Es importante ya que antes de destinar los recursos a su producción podemos saber con antelación si es viable económicamente. Aun tratándose de un proyecto de fin de grado, saber si tu idea es rentable o no, es un indicador, en caso negativo de cambiar de idea, y en caso positivo, una motivación más para iniciar dicho proyecto.

En este informe se plasmarán los costes directos como son el: Hardware, software y los recursos humanos. También los indirectos que donde se tiene en cuenta la electricidad y conexión a Internet. Finalmente, el coste total y que plan de contingencia tenemos.

3.1. Costes directos

3.1.1. Recursos Hardware

En el desarrollo intervienen múltiples ordenadores, cada uno con un fin concreto. Disponemos de dos ordenadores principales, que llamaremos PC Sobremesa 1 y PC Sobremesa 2, en estos se desarrolla el proyecto, además, de proporcionar gran capacidad de cálculo. Un servidor cuya única funcionalidad es recopilar información y enviarla a los dispositivos de almacenamiento para después ser tratada.

Finalmente, un portátil que se usa para poder mostrar el funcionamiento de la plataforma en las sesiones de control con el director, también está como soporte en caso de tener que añadir más potencia de cálculo. En el coste de los ordenadores de sobremesa está incluido: la torre, pantalla y periféricos. El servidor solo está compuesto por una torre.

Uno de los elementos a tener en cuenta referente al uso de los recursos materiales es la amortización del equipo. Hacienda permite amortizar el hardware de 3 a 4 años, después se han de renovar por obsolescencia. Para calcular la amortización hay que saber cuantas horas reales trabajara este hardware, es decir, todos los días laborables que dure el proyecto, trabajando 8h diarias. Finalmente, solo hay que dividir el coste del hardware por el total de horas reales previamente calculadas.

3.1.1.1. Especificaciones del hardware

- PC Sobremesa 1
  - CPU: i5-4690k, 8GB RAM, 1050Ti GTX y 512GB de HDD.
- PC Sobremesa 2
  - CPU: i7-4790k, 16GB RAM, 1060 GTX y 3TB de HDD.

- Portátil DELL XPS 15
  - CPU: i7-7700HQ, 16GB RAM, 1050 GTX, 512 GB de HDD.
- Servidor
  - CPU: Core2Duo E8400, 8 GB RAM, 460 GTX, 1 TB de HDD.

3.1.1.2. Estimación de costes del hardware

Producto	Vida Útil (Años)	Coste en €	Amortización (€/h)
PC Sobremesa 1	4	1.810	1,89
PC Sobremesa 2	4	2.150	2,24
Portátil	3	1.800	1,88
Servidor	4	800	0,83
	Total	6.560	6,64

Tabla 3.1: Tabla con la estimación de costes del hardware

Nota: El total de horas reales del proyecto son: 956h

3.1.2. Recursos Software

Contamos con la ventaja de que únicamente usamos software bajo licencia FLOSS (Free/Libre and Open Source Software), por ende, no hay que realizar ninguna inversión en este recurso. Además, tanto las Descripciones o parches, así como añadir nuevas funcionalidades no tiene ningún coste.

Para el desarrollo se utiliza una gran variedad de software ya incluido en la propia distribución (Ubuntu 16.04 LTS), pero consideramos los siguientes programas los imprescindibles para realizar el proyecto: *Python 3.7, Visual Studio Code 1.21.1, MongoDB 3.4.9, Angular 4.1.3, Node.js 8.10.0, Scrapy 1.5, Git 2.16.2, Kraken 3.4.1, StarUML 2.8.1, Neo4J 3.3.0 y LaTeX, .*

3.1.2.1. Servicios

Para el control de versiones utilizamos la herramienta Git, pero el almacenamiento del código se hace mediante repositorios privados en la plataforma GitHub. Al ser estudiantes de la UPC, tenemos un número ilimitado de repositorios privados, además, de tener acceso a todas las características de la plataforma sin ningún coste.

3.1.3. Recursos Humanos

Para elaborar la tabla de recursos humanos se ha consultado los salarios medios en el área de Barcelona en el portal web Glassdoor[32]: Senior Project Engineer[36], QA Engineer[35], Junior Software Engineer[34] y Junior Software Developer[33]

Hay que remarcar, que los salarios publicados en dicho portal son informes enviados de forma anónima por trabajadores de dichas empresas o creados mediante estadística. En ningún momento se tiene intención de representar el salario medio real, sino ser una mera orientación. Debido a que cada empresa de forma individual negocia el contrato y es información sensible.

Cada uno de los desarrolladores del proyecto debe ser multidisciplinar y asumir el rol de todos los actores implicados.

Actor	Letra	Horas	€ / Hora	Coste en €
Senior Project Engineer	P	196	30	5.880
Junior Software Engineer	S	120	13	1.560
Junior Developer Engineer	D	600	10	6.800
QA Engineer	Q	40	13	512
Total		956		14.752

Tabla 3.2: Tabla de costes de los actores

Tarea	Días	Horas	Recurso	Coste en €
GEP	30	81	P	2.430
—— GEP: Entrega 1	6	25	P	750
—— GEP: Entrega 2	5	10	P	300
—— GEP: Entrega 3	5	10	P	300
—— GEP: Entrega 4	4	8	P	240
—— GEP: Entrega 5 y 6	10	28	P	840
Análisis y diseño	15	120	S	1.560
—— Análisis de requisitos	5	40	S	520
—— Especificación	5	40	S	520
—— Diseño y arquitectura	5	40	S	520
Desarrollo	40	560	D	6.400
—— Módulo del Crawler	5	40	D	400
—— Módulo de la BBDD	5	40	D	400
—— Módulo del Grafo	5	40	D	400
—— Módulo del Recomendador	15	120	D	1.200
—— Desarrollo de la plat. Web	5	40	D	800
—— Ampliación módulo del Crawler	5	40	D	400
—— Ampl. módulo del Grafo	5	40	D	400
—— Ampl. módulo de la BBDD	5	40	D	400
—— Ampl. del recomendador	15	120	D	1.200
—— Ampl. de la plat. web	5	40	D	800
Release Candidate	10	45	Q,D,P	2.120
—— Control de calidad y corrección de errores	5	40	Q,D	920
—— Preparación de la presentación y exposición oral	5	40	P	1.200
Memoria	55	110	P	3.300

Tabla 3.3: Costes de recursos humanos en función del diagrama de Gantt

3.2. Costes indirectos

3.2.1. Conexión a internet

Internet es una herramienta imprescindible hoy en día para buscar información, comunicación entre los dos desarrolladores, o la parte más importante la recolección de datos. Como el proyecto no se desarrolla en exclusividad en un único lugar, se pondrá el coste de la tarifa de ambos integrantes. Tenemos dos costes: 95 € y 50 € mensuales, ambos con fibra óptica. Tiene un coste total de 725 €. El coste es calculado des de febrero hasta junio.

3.2.2. Electricidad

La electricidad es necesaria para alimentar todos los equipos que componen el proyecto. El consumo va en función de una jornada laboral de 8h diarias. Siempre teniendo en cuenta 956h de proyecto y un coste de luz de 0,12 €/kWh. La tabla 3.4 refleja los costes del consumo por cada equipo.

Producto	**Consumo (kWh)**	**kWh Totales**	**Total** en €
PC Sobremesa 1	0,1	95.6	11,47
PC Sobremesa 2	0,07	66.92	8,03
Portátil	0,01	9.56	1,15
Servidor	0,07	66.92	8,03
Total		239	28,68

Tabla 3.4: Tabla con los costes en electricidad del equipo

3.3. Plan de contingencia

El plan de contingencia solo tiene en consideración los recursos humanos, ya que no esperamos necesitar hardware ni software adicional para realizar el proyecto. Creemos que el único imprevisto puede venir de que alguna de las características a implementar requiera más tiempo del previsto para implementarla.

No tenemos experiencias anteriores en las que podamos basarnos para poder estimar un porcentaje, pero tomaremos la precaución de asumir un 15 % de los recursos humanos y otro 15 % de los costes indirectos. Es decir 2212,8 € adicionales para RRHH y 113,07 € para los costes indirectos.

3.4. Imprevistos

No prevemos que pueda haber ningún imprevisto, exceptuando de que falle alguno de los componentes que se usan para realizar el proyecto.

Todos los equipos menos el servidor están en garantía. Si en el caso peor, alguno de los equipos necesarios para realizar el proyecto fallase, podríamos considerar el portátil como un ordenador principal de forma provisional, ya que está contabilizado como herramienta de soporte, mientras el equipo dañado está en reparación.

Si el servidor fallase no nos quedaría más remedio que usar alguno de los otros equipos como servidor.

3.5. Presupuesto

Concepto	**Coste** en €
Costes directos	21.312
Costes indirectos	753,68
Contingencia	2.325,87
Imprevistos	0
Total	24.391,55

Tabla 3.5: Tabla con los costes del presupuesto final

3.6. Modelo de control de costes

Los recursos de hardware son una inversión inicial y no se les puede hacer un control constante, solo se les puede hacer una comparación del coste final con el coste calculado en el presupuesto cuando finalice el proyecto.

Los recursos humanos si que se les puede hacer un control del presupuesto, al final de cada tarea se actualizara el número de horas efectivamente realizadas, el coste de los recursos usados y los gastos de las contingencias ocurridas. Con los nuevos cálculos podremos ver cuanto nos hemos desviado de nuestra planificación inicial.

Si en alguna etapa se produce una diferencia notable, se realizará un estudio de porque se ha producido esta diferencia y se intentara corregir para el siguiente ciclo, creando un presupuesto más exacto.

Para calcular las desviaciones se hará de la siguiente manera:

- Desvío de mano de obra en precio = (coste estándar – coste real) * consumo de horas real
- Desvío en la realización de una tarea en precio = (coste estándar – coste real) * consumo horas reales
- Desvío de un recurso en precio = (coste estándar – coste real) * consumo real
- Desvío en la realización de una tarea en consumo = (consumo estándar – consumo real) * coste real
- Desvío total en la realización de tareas = coste total estándar tarea – coste total real tarea
- Desvío total en recursos = coste total estándar recursos – coste total real recursos
- Desvío total costes fijos = coste total coste fijos presupuestado – coste total fijo real

Capítulo 4

Sostenibilidad

En esta sección vamos a evaluar el impacto de nuestro proyecto en los tres ámbitos de la sostenibilidad: ambiental, económica, social. Después elaboraremos la matriz de sostenibilidad a partir de las apreciaciones hechas en los siguientes apartados.

4.1. Ambiental

4.1.1. Proyecto puesto en Producción

Para la huella ecológica de la puesta en producción del proyecto, consideramos el número de horas total para el desarrollo del proyecto y los recursos de hardware mencionados en el apartado de gestión económica. En los cálculos también tendremos en cuenta que nuestro proyecto consta de una parte de computo adicional que hay que realizar periódicamente con los nuevos datos que hayan entrado en el sistema.

4.1.1.1. Adquisición de equipos

Durante el desarrollo del proyecto no vamos a comprar ningún ordenador nuevo y aprovechamos todos los equipos que ya tenemos. Para minimizar el impacto ambiental, podríamos prescindir del servidor durante la etapa de desarrollo y usar uno de los dos ordenadores personales como tal, pero consumiríamos recursos que se podrían invertir en potencia de cálculo.

4.1.1.2. Consumo eléctrico

Durante el desarrollo, el servidor estará encendido todo el día porque siempre tendrá asignada una tarea a realizar. Considerando el hardware especificado de la máquina sabemos que son aproximadamente 70W en media. Desde el 26 de febrero hasta el 15 de junio. Asumiendo 0,385 Toneladas de CO2 por kWh según la *Generalitat de Catalunya* [13].

0,07KWh * 24 horas * 14 semanas * 7 días = 164,68KWh. el equivalente a 63,5 Toneladas de CO2 emitidas a la atmósfera.

El resto de ordenadores están dentro del uso habitual de una jornada laboral. Con excepciones de dos veces por semana, aproximadamente, donde se tiene la necesidad de hacer un recalculo de los datos necesarios de nuestro proyecto. Esta parte puede llegar a durar un día completo con un consumo de hasta 400W por máquina .

0,4KWh * 24 horas * 16 semanas * 2 días = 307,2KWh. el equivalente a 118,272 Toneladas de CO2.

4.1.2. Vida Útil

Nuestro proyecto es una plataforma web, necesitamos tener el número de servidores necesario para poder atender a todas las peticiones. En un principio la instalación de la plataforma se hará en Google Cloud Computing para poder dimensionar con comodidad la plataforma, pero creemos que al inicio con una instancia para el servidor web (Intel Haswell 4 Virtual CPU, 8GB RAM) y otra un poco más potente (Intel Haswell 8 Virtual CPU) para la API REST y las bases de datos será suficiente. En un futuro pasaríamos a usar una arquitectura elástica, capaz de aumentar o disminuir los recursos en función de la demanda.

Al menos durante la vida útil del proyecto los dos desarrolladores originales tendremos que seguir manteniendo el proyecto y incorporando nuevas características lo que añade otras dos estaciones de trabajo a tener en cuenta. (200W de media teniendo en cuenta torre y dos monitores por estación de trabajo).

También necesitaremos una instancia adicional (Intel Haswell 4 Virtual CPU, 8GB RAM) donde poder poner el servidor de pruebas. Este mismo servidor se usará también para hacer los precalculos mensuales con la información nueva añadida a la base de datos.

4.1.3. Riesgos

Durante la vida útil del proyecto el recalculo de los datos será cada vez más costoso porque el volumen de este se irá incrementando, lo que en un principio podría ser un día, podría acabar siendo una semana o más al mes, hasta que no se encuentre una forma más eficiente de añadir nuevo contenido al proyecto. Esto podría implicar el uso de una instancia más al 100 % pare realizar dicho cálculo. Esto también nos afecta económicamente porque más instancias para calcular implica más coste mensual de nuestro proveedor de la nube.

4.1.3.1. Conclusión

Aunque las instancias de Google Cloud Computing sean virtuales estimaremos el consumo teniendo como referencia un hardware similar.

- Dos instancias sirviendo la web,
  - (Intel Haswell 8 Virtual CPU) 150W al 50 % de carga, 0,15KWh * 24 horas * 31 días = 111,6KWh/mes el equivalente a 42,966 Toneladas de CO2 al mes.
  - (Intel Haswell 4 Virtual CPU) 100W al 50 % de carga, 0,1KWh * 24 horas * 31 días = 74,4KWh/mes el equivalente a 28,644 Toneladas de CO2 al mes.
- Dos estaciones de trabajo. 0,2KWh * 8 horas * 5 días * 4 semanas = 32KWh/mes el equivalente a 12,32 Toneladas de CO2 al mes.
- Instancia de pruebas. 0,1KWh * 10 horas * 31 días (Teniendo en cuenta que se mantendrá encendido algunas veces después de la jornada laboral) = 31KWh/mes

Lo que hace un total de 249KWh/mes o 95 Toneladas de CO2 al mes. Todo esto teniendo en cuenta que la situación de la vida útil es la misma que durante el desarrollo del proyecto. Es decir, continuando desarrollando el proyecto desde nuestras casas.

4.2. Económica

4.2.1. Proyecto puesto en Producción

En los apartados anteriores, se ha realizado una evaluación de costes tanto materiales como de recursos humanos, teniendo en cuenta desviaciones de calendario y posibles imprevistos. También se han establecido medidas de control para ajustar el presupuesto durante la realización del proyecto.

4.2.2. Vida útil

Durante la vida útil del proyecto tendremos que tener en cuenta los siguientes costes:

4.2.2.1. Cloud Computing

Utilizaremos una arquitectura escalable en un proveedor en la nube, para, en un futuro, poder abastecer a un número mayor de usuarios. Esto también significa que el coste variara en función de los usuarios que estén usando la plataforma. Lo que nos permite un ahorro en la fase inicial, mientras tenemos pocos usuarios.

Teniendo en cuenta las instancias mencionadas en el apartado Ambiental,calculamos que el coste aproximadamente es:

- Dos instancias sirviendo la web,
  - (Intel Haswell 8 Virtual CPU) 230€/750h = 230€
  - (Intel Haswell 4 Virtual CPU) 170€/750h = 170€
- Instancia de pruebas (Intel Haswell 4 Virtual CPU) 170€/750h * (10 horas * 31 días) = 0.22€/h * 310h = 68.2€

Lo que hace un total de 468,2€/mes en Servidores Cloud.

4.2.2.2. Resto

El coste estimado del proyecto durante su primer año se ha considerado una partida de 10.000€ en imprevistos, donde se incluyen posibles reparaciones, actualizaciones y cualquier imprevisto o desviamiento en el desarrollo del proyecto. También hay un gasto importante en marketing para darnos a conocer. El desglose de gastos se puede ver en la siguiente tabla:

Tabla 4.1: Gastos durante el primer año

Concepto	**Coste en €**
Marketing	20.000
Mobiliario Oficina	5.000
Hardware y PC	4000
Servicios en la nube	5.625,6
Sueldo desarrolladores	56.000
Alquiler oficina	15.000
Servicios (agua,luz,...)	10.000
Imprevistos	10.000
Total	**125.625,6**

4.2.2.2.1 Alternativas

Es cierto que al inicio de la vida útil podríamos arreglárnoslas usando nuestros propios ordenadores y trabajar desde casa. Eso reduciría bastante el presupuesto quedando solo Marketing, Servicios en la nube y Imprevistos dejando un total de 35.625,6€.

Al ser un proyecto software libre, conseguir que RedHat nos financie nos permitiría ahorrar los servicios en la nube, ya que nos los proporcionarían ellos. Eso dejaría el total en aproximadamente 30.000€

4.2.3. Riesgos

Que se tarde más tiempo del previsto o requiera más desarrolladores tener una plataforma capaz de rivalizar con la competencia.

Que el volumen de datos a procesar requiera más recursos de los que se han previsto, eso implica más gasto en potencia de cálculo.

4.3. Social

4.3.1. Proyecto puesto en Producción

Este proyecto nos va a ayudar a adentrarnos un poco más en la parte del Aprendizaje automático dedicada al procesamiento del lenguaje natural. Ver como funciona el proceso de investigación dado que este proyecto tiene una fuerte componente de investigación en ver que técnicas dan el mejor resultado.

4.3.2. Vida útil

Creemos que la mayoría de la comunidad científica podría beneficiarse gracias al proyecto incrementando la calidad del material que encuentren y que tarden menos en encontrarlo. Esperamos que eso repercuta en mejorar su comodidad en el trabajo y por tanto mejorar su calidad de vida. Por otro lado, sabemos que quizá algunos autores se vean perjudicados si de alguna manera sus artículos acaban considerados de poca calidad reiteradamente.

Creemos que si el proyecto saliese adelante podría llegar a suponer un cambio en la forma de investigar de todo el mundo y ayudaría también a evitar la aparición de conferencias y revistas de dudosa reputación donde aceptan cualquier cosa. Esperamos también que junto a la ayuda de la comunidad científica cada vez haya más publicaciones abiertas. Esto ayudaría a nuestro proyecto puesto que sería posible acceder a los contenidos reales de los artículos y no solo a la información general (título, autores, lugar de publicación).

Somos conscientes de que actualmente hay varios grupos de investigadores tanto en empresa privada como pública que están buscando lo mismo que nosotros, por tanto, nosotros queremos poner la primera piedra liberando esta plataforma, para que a partir de aquí se construya un proyecto mucho más grande que dé la solución real que se quería cuando se propuso este proyecto.

4.3.3. Riesgos

En el ámbito social no se contempla ningún riesgo que pueda perjudicar a ningún segmento de la población.

4.4. Matriz de sostenibilidad

Por los motivos comentados en los apartados anteriores calificamos con las notas de la tabla 4.2 las diferentes secciones de la sostenibilidad.

	PPP	**Vida Util**	**Riesgos**	**Nota**
Ambiental	8	6	9	-
Economico	8	6	5	-
Social	8	9	8	-
Nota	24/30	21/30	22/30	67/90

Tabla 4.2: Notas de la sostenibilidad

4.5. Leyes y normativas

Por el momento no guardamos datos personales de los usuarios que se registran en nuestra plataforma, así que no tenemos que preocuparnos por la LOPD (Ley Orgánica de Protección de Datos) o actualmente la (General Data Protección Regulation) europea.

Para la realización de nuestro proyecto el usuario podrá descargar el PDF del *paper* directamente a través de un enlace que le proporcionará la plataforma sin pasar por la web de la revista donde está publicada, pero sabemos que no podemos publicar este contenido dado el copyright del que disponen las editoriales, solo ofrecer enlaces a las webs de las revistas donde están publicados. Esto nos permite dar la sensación al usuario de que el PDF está alojado o publicado en nuestra plataforma. Además, para no infringir ninguna ley también se dará expondrá la autoría y derechos de la publicación.

4.5.1. Licencia

Nuestra única opción para satisfacer estas necesidades es hacer que nuestro proyecto sea totalmente libre. Esto es lo único que puede llevar a un mayor uso que los grandes repositorios actuales, que son de código cerrado.

La licencia elegida es GPLv3 [3]. La principal razón de esta decisión es que nos garantiza que todas las modificaciones sobre el software tienen que ser liberadas obligatoriamente y ser libres bajo licencia GPLv3. Esta licencia es conocida como *copyleft* o vírica.

Además, nos garantiza las cuatro libertades del software libre:

- La comunidad puede aprender como está hecho el software.
- Libertad para distribución de copias para uso privado o público sin restricción.
- Libertad para modificación del código para adaptarlo a tus necesidades.
- Permite que otras personas participen en el proyecto y aporten mejoras. Tienen la libertad de modificar el software.

Libertad en este contexto no denota obligatoriedad. Es un proyecto de por y para la comunidad, el símil más parecido sería la Wikipedia.

4.6. Visión

Nuestro proyecto quiere contribuir a resolver un problema real. Nuestra *visión* es que evolucione con el tiempo a una gran plataforma para la comunidad científica que permita:

- Publicar sus *papers* sin coste.
- Compartir libremente y sin restricciones los conocimientos con la comunidad.
- Ser punto de encuentro de diferentes comunidades.
- Permitir informarse de las últimas novedades.
- Ofrecer un gran archivo de todas los *papers* que se han ido publicando.

4.6.1. Marketing

Nuestra principal misión sería darnos a conocer en los lugares donde se concentren el mayor número de académicos posible, eso suele ser en congresos, conferencias, seminarios y ferias de TIC en universidades, pero de momento nos centraremos en la difusión en internet por redes sociales.

4.6.2. Patrocinadores

Al ser un proyecto de software libre, un posible método de financiación podrían ser los patrocinadores, empresas como RedHat [67] o la Mozilla Fundation [59] que no solo podrían ayudar financiando el proyecto económicamente si no también con recursos, como servicios en la nube, cloud computing, etc.

Capítulo 5

División de trabajo

En este capítulo se especificarán las diferentes partes que realizará cada integrante, dejando clara la división de trabajo.

5.1. Parte Buscador: Ruben Bagan Benavides

5.1.1. Módulo: Base de datos

Desempeña dos funciones:

1. Servir las peticiones de datos de los diferentes módulos.
2. Responsable de generar la base de datos a partir de los datos recopilados por el módulo del *crawler*.

5.1.2. Módulo: Crawler

Extraer los datos de sitios web mediante *spiders* y enviarlos a través de una *pipeline* al módulo de Base de Datos.

5.1.3. Módulo: Graph Builder

Su función es transformar los datos almacenados en el módulo de base de datos en un grafo y almacenar el resultado en una base de datos orientada a grafos (BDOG).

5.1.4. Módulo: Graph Analyzer

Extraer los metadatos del grafo generado por el módulo *Graph Builder* que son necesarios por el módulo *recomendador* para mejorar la experiencia de usuario.

5.2. Parte Recomendador: Alberto López Sánchez

Usaremos estas tres técnicas combinadas para hacer las recomendaciones:

1. Emplear técnicas de Procesamiento del Lenguaje Natural (NLP) para encontrar papers parecidos en contenido.
2. Emplear la información social extraída del grafo para mejorar las recomendaciones.
3. Usar las preferencias del usuario para mejorar las recomendaciones.

5.2.1. Cliente web

Es la vista del sistema ofrece al usuario un *front-end* con el que interactuar. El usuario podrá: buscar, filtrar y explorar los papers, autores, conferencias y revistas disponibles. La plataforma también le hará recomendaciones en función de: criterios del usuario esto también incluye su *feedback* y la información que extraemos sobre los datos.

Como tecnología para el desarrollo web usaremos Angular [44], ya que necesitamos que sea bien modular y extensible a los cambios que ocurran en el proyecto.

5.2.2. API REST/Controlador

Es el controlador del sistema que ofrece una API REST para que los clientes puedan interactuar, en concreto nuestro cliente web.

5.2.3. Motor del recomendador

Parte del controlador del sistema. Se encarga de recomendar papers en base a un paper y usuario dado, se conecta con el resto de módulos para obtener la información necesaria para ser capaz de hacer las recomendaciones.

5.2.4. Modelo

Es el modelo del sistema y está dividido en tres partes:

- La que accede a los papers, journals, revistas, conferencias.
- La que accede a la información del grafo. (Pageranks, Shortest Path, Comunidades).
- La que accede a la *cache* del recomendador.

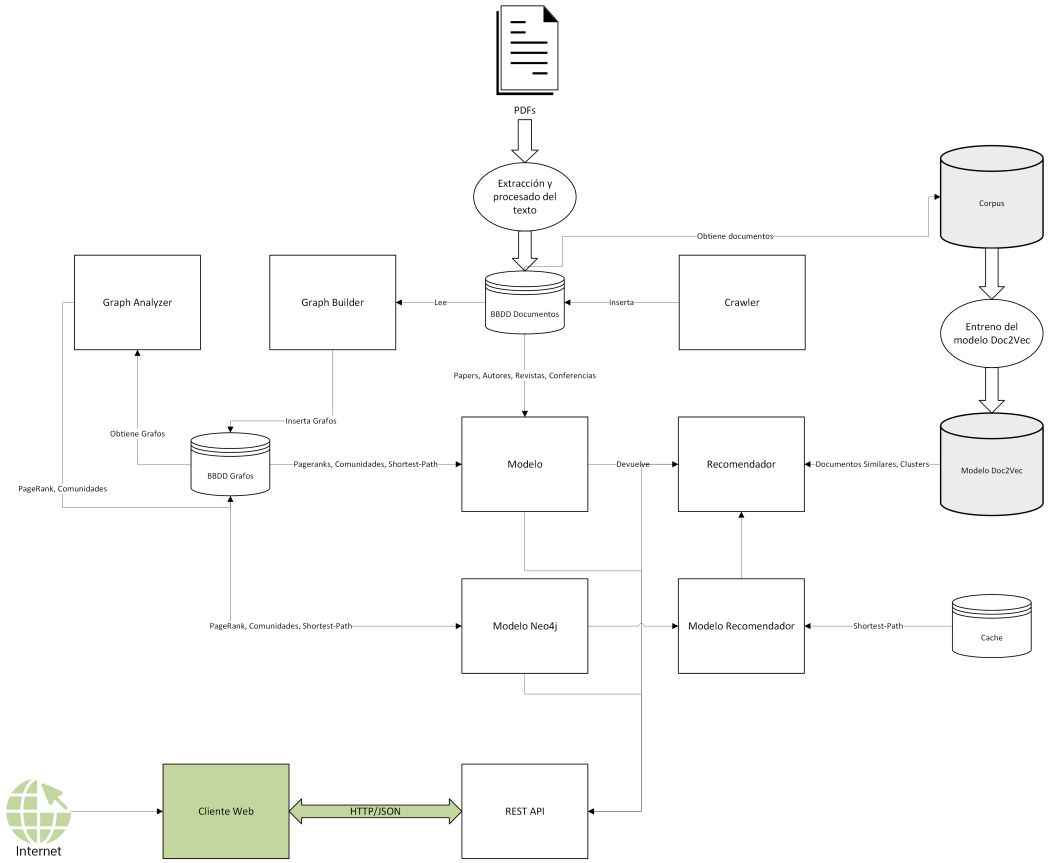

Figura 5.1: Esquema de la arquitectura completa.

Parte II

Buscador

Capítulo 6

Módulo: Crawler

6.1. Objetivo

Extraer los datos de sitios web mediante *spiders* y enviarlos a través de una *pipeline* al módulo de Base de Datos.

6.2. Introducción

Para introducir al lector los conceptos importantes para poderlos usar libremente en el documento, definiremos que:

- *crawler*: Nos referimos al programa completo que contiene todas las *spiders*.
- *spider*: Es un hilo de ejecución del *crawler* que tiene como objetivo recopilar información de una página o sitio web iterando por el árbol de etiquetas HTML y aplicar métodos de extracción de datos. Estos métodos están definidos como XPATH [45] y CSS [49].
- *web crawling*: Hace referencia al proceso que hace el programa cuando recopila datos de un sitio o página web.

El módulo de *Crawler* es el encargado de importar datos de Internet y almacenarlos en el módulo de base de datos. La recopilación de datos se llama proceso de *web crawling* y es un proceso complejo, y nuestra limitación de recursos materiales añade un nivel más de dificultad.

Antes de empezar un proceso de *web crawling* primero hay que hacer una investigación en profundidad buscando como están relacionados los datos entre las diferentes páginas que componen el sitio web. Consiste en:

- Observar como están organizados los diferentes elementos.
- Identificar los elementos objetivo.
- Ver donde se posicionan habitualmente en la estructura web.
- Que identificador tienen asociado habitualmente.

Este proceso es necesario porque queremos programar una *spider* para que sepa como extraer la información.

Los sitios web generalmente no almacenan la información de forma consistente. Podemos encontrarnos que exista un caso, en que una página web del sitio que no cumpla con el patrón usual. Este es el factor que hace que este proceso sea complejo porque no podemos saber si una página cambiará la forma de como están estructurados los elementos. Lógicamente no es viable visitar todas y cada una de las páginas y ver todos los posibles casos.

Por tanto, una vez hemos hecho la investigación, toca pensar como hacer que nuestras *spiders* se adapten a estos cambios perdiendo la mínima información posible. Durante el desarrollo del módulo nos hemos enfrentado a este problema, y hemos encontrado métodos que solucionan el problema. En la sección 6.8 se explicará más a fondo.

No es el único problema que nos enfrentamos al hacer un proceso de *web crawling*. Este proceso genera una gran cantidad de peticiones contra el servidor, que se traduce en mucho tráfico y implica un coste para el sitio web porque consumimos recursos extra para unos falsos usuarios, nuestras *spiders*.

Para defenderse de estas *spiders*, los sitios web tienen contramedidas para evitar este tipo de "ataques". Existen muchas medidas, pero las más comunes son:

1. Detectar si se produce un cierto número de peticiones seguidas de una misma IP, si es así, se puede deducir que un *crawler* está recopilando información. La medida por excelencia para evitar este tipo de "ataques" es bloquear esa IP durante un largo periodo de tiempo.
2. Hacer que la página web su contenido se cargue de forma dinámica.
3. Añadir tiempos de carga aleatorios en el contenido.
4. Bloquear la IP que no pertenezcan a entidades universitarias.

Para evitar este tipo de "ataques", algunos sitios webs ofrecen la opción de cobrarte una cuota a cambio de que tus *spiders* no sean bloqueadas.

Es aquí la dificultad añadida, no disponemos de recursos económicos para pagar la cuota. Para demostrar la viabilidad del proyecto necesitamos la información almacenada en los servidores. Esto implica pensar en métodos para superar las barreras, sabiendo que no es una solución a largo plazo para la plataforma completa que aspiramos a desarrollar (más allá de nuestro TFG).

Todos los métodos más importantes están explicados en la sección 6.8. Lógicamente este tipo de acciones no son éticas y se ha intentado en la medida de lo posible evitar que las acciones sean muy agresivas. El paso más lógico sería buscar una empresa interesada en nuestro proyecto y que nos esponsoricen y eliminar esta parte del problema.

Queremos exponer los problemas que nos hemos encontrado y que decisiones hemos tomado. El criterio siempre ha sido buscar la opción que equilibre la balanza entre tiempo y optimalidad. Las soluciones propuestas no son únicas, sino una de muchas. Durante el largo proceso de desarrollo se ha invertido un periodo de tiempo en:

- Aprender a como utilizar las tecnologías para desarrollar el módulo.
- Pensar en métodos para superar los obstáculos.

Al final este proyecto, el módulo es capaz de recopilar información de las siguientes páginas:

- https://dblp.uni-trier.de/
- https://arxiv.org/
- https://aisnet.org/
- https://www.dagstuhl.de/
- https://scholar.google.es/
- http://www.mdpi.com/
- https://www.sciencedirect.com/
- https://ria.ua.pt/
- https://link.springer.com/
- https://ieeexplore.ieee.org/
- https://www.sciencedirect.com/
- https://www.igi-global.com/

Hemos aprendido que incorporar una nueva fuente de información no es mecánico, ya que hay que estudiar su formato y contenido para que el *crawler* sepa procesarla.

6.3. Especificación

6.3.1. Entrada de una *spider*

Las *spiders* que necesitan argumentos para establecer una conexión con el módulo de Base de datos requieren de los siguientes parámetros.

- *host*: La dirección donde esta ubicada la base de datos.
- *port*: Puerto de la conexión con la base de datos.
- *database*: Nombre de la base de datos que almacena las colecciones.
- *collection*: La colección donde se almacena el campo con la URL.
- *field*: Campo que contiene la dirección URL.

6.3.2. Salida de una *spider*

6.3.2.1. Estructura *Journal*

- *papers*: es una lista de identificadores.

Journal	
Name	**Type**
url:	String
name:	String
year:	Number
volume:	Number
number:	Number
month:	Number
papers:	List<String>

Tabla 6.1: Estructura de datos de una *spider* que almacena revistas

6.3.2.2. Estructura *Edition*

- *conferences*: es una lista de identificadores.

Edition	
Name	**Type**
name:	String
code_name:	String
year:	Number
conferences:	List<String>

Tabla 6.2: Estructura de datos de una *spider* que almacena ediciones

6.3.2.3. Estructura *Conference*

- *papers*: es una lista de identificadores.

Conference	
Name	**Type**
name:	String
url:	String
paper:	List<String>

Tabla 6.3: Estructura de datos de una *spider* que almacena conferencias

6.3.2.4. Estructura *Paper*

- *doc_type*: Especifica si el *paper* fue publicado en una conferencia o en una revista.
- *google_scholar*: Es un enlace directo a Google Scholar.
- *citations*: Cuantos *papers* lo han citado.

Paper	
Name	**Type**
title:	String
doc_type:	String
abstract:	String
google_scholar:	String
category:	String
categories:	List<String>
authors:	List<String>
url_pdf:	String
url:	String
pages:	String
year:	Number
keywords:	List<String>
release_date:	String
citations:	Number
DOI:	String

Tabla 6.4: Estructura de datos de una *spider* que almacena *papers*

6.4. Diseño inicial

El módulo estará compuesto por dos componentes.

- *Spiders*: Son procesos especializadas en extraer la información de un único sitio web, capaces de extraer elementos: autores, conferencias, *papers* y revistas.
- *Servidor*: Encargado de gestionar todas las *spiders*: crear las instancias necesarias, gestionar su entrada y salida, etc.

La interacción con el módulo se hará a través de un cliente que hará de capa de extracción con el servidor y nos permitirá pedir: procesar sitios web o páginas específicas.

Como requisito se quiere conseguir que el módulo trabaje con independencia a las estructuras de los sitios web, para añadir nuevas *spiders* sin modificar el núcleo.

Se definirá una estructura de datos con todos los campos que consideremos necesarios para cada elemento y permitir que los campos puedan no contener ningún valor por si en una página o sitio no existe.

Finalmente, toda la información que es obtenida por parte de las *spiders* se almacena en archivos con la siguiente *template*: timestamp_nombreSpider_recurso_sitioWeb

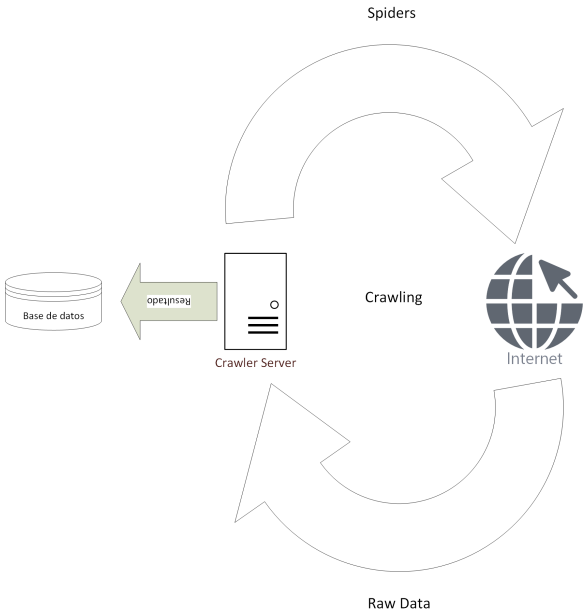

Figura 6.1: Diseño inicial de la estructura del módulo

6.5. Diseño final

Desde el diseño inicial el módulo ha recibido constantes cambios hasta llegar el estado actual. Se describirán todos los cambios hechos y que soluciones se proponen siempre con el objetivo de buscar un equilibrio entre inversión de tiempo y beneficios al proyecto.

6.5.1. Prescindir del servidor

Después de hacer un estudio de cuanto tiempo requería desarrollarlo se tomó la decisión de cancelarlo. La razón es que el resultado del estudio indicó que el impacto sobre el resultado final del proyecto era muy bajo a cambio de destinar menos tiempo a otros módulos donde si se podía sacar más beneficio.

El resultado final de esta decisión es una arquitectura donde solo hay una única *spider* trabajando sobre un mismo sitio web. Las consecuencias negativas de esta decisión tendrán una repercusión al rendimiento del módulo y obligará a buscar soluciones para reducir su impacto:

1. Perder la opción de paralelizar el proceso de *web crawling* y reducir el tiempo. El servidor se encargaba de gestionar que un número determinado de *spiders* "ataquen" a un mismo sitio web de forma coordinada.
2. No poder pausar el proceso. Si hay un corte de luz o un error hay que empezar de nuevo, la penalización de tiempo es muy alta.

6.5.2. Prescindir del cliente

Al eliminar el servidor ya no existe la necesidad de desarrollar un cliente con el que comunicarnos. Esta funcionalidad se ha redefinido en comunicarse directamente con las *spiders* a través de argumentos. En el diseño final existen dos tipos de *spiders*:

1. Especializadas en extraer información únicamente de una página de un sitio web.
2. Especializadas en extraer toda la información de un sitio web.

Este nuevo diseño ofrece flexibilidad total a la hora de extraer información, porque permite segmentar un sitio web en objetivos específicos.

6.5.3. Prescindir de usar ficheros

La siguiente modificación de diseño a sido reemplazar el sistema de trabajar con ficheros por una *pipeline*. Este nuevo método elimina la desventaja de trabajar con un sistema de ficheros y eliminar la posibilidad de equivocarnos al no trabajar sobre el fichero correcto. El funcionamiento de la *pipeline* es el siguiente:

- Una *spider* recopila un bloque de información de una página web.
- Empaqueta la información y la envía a la entrada de la *pipeline*.
- La información se codifica y es enviada al módulo de base de datos a través de una petición que puede ser: añadir un nuevo o actualizar.
- La base de datos recibe la información codificada, la descodifica y genera un nuevo registro o actualiza según la petición.

6.5.4. Modificaciones en las estructuras de salida

El último cambio es modificar las estructuras de datos con las que trabaja las *spiders*. Se ha tomado la decisión de prescindir de recopilar información de autores. La razón de esta decisión se debe al problema conocido como: *Author name disambiguation* [46]. Para ponernos en contexto en el problema, supongamos que tenemos dos *papers* y uno de los autores es común en ambos, pero su nombre está escrito diferente en los artículos, por ejemplo: *Lana Yeganova* en uno y *L. Yeganova* en otro.

El problema se puede complicar más si en los sitios web ya sea por error humano o intencionado podemos encontrarnos que el nombre sea almacene como *Lana. Y*. La complejidad se puede incrementar si queremos juntar además diferentes bases de datos de diferentes sitios web.

Estas dos investigaciones [72] y [11] hablan de una posible solución usando diferentes técnicas de Machine Learning [21] y usar como datos las citaciones. Resolver el problema tiene una complejidad elevada y no pretendemos en este TFG darle solución por una cuestión de limitación de tiempo.

Para finalizar, el último cambio aplicado sobre las estructuras es que la edición de la conferencia ahora es una entidad independiente. El motivo de esta elección es porque en una edición se pueden celebrar diferentes conferencias y nos puede ser útil para extraer más metadatos. También se han añadido más campos en las estructuras de datos para albergar más información.

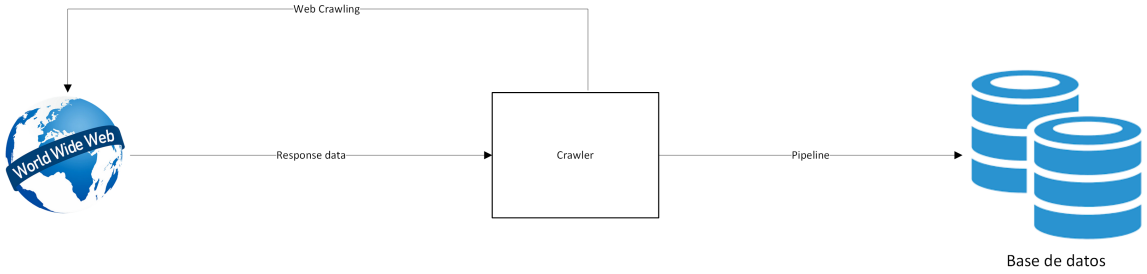

Figura 6.2: Diseño final de la estructura del módulo

6.6. Tecnologías

- *Scrapy* : Es uno de los *frameworks* más conocidos para hacer *web crawling*. Es de código abierto bajo licencia MIT. Está completamente programado en Python y con compatibilidad tanto para Python 2.7 y Python 3.6.
- *Pymongo* : Es el driver que nos permite comunicarnos con la base de datos MongoDB y establecer la *pipeline* entre los módulos. Es de código abierto bajo licencia MIT. Al igual que Scrapy tiene compatibilidad con Python 2.7 y Python 3.6.
- *Python 3.6* : Es el lenguaje para desarrollar todo el módulo.

6.7. Estructura de llamadas de las *spiders*

La implementación de las *spiders* consiste en ejecutar una cadena de llamadas a funciones ordenadas y en cada una hacer la operación necesaria. Hay dos esquemas porque existen dos tipos de *spider*:

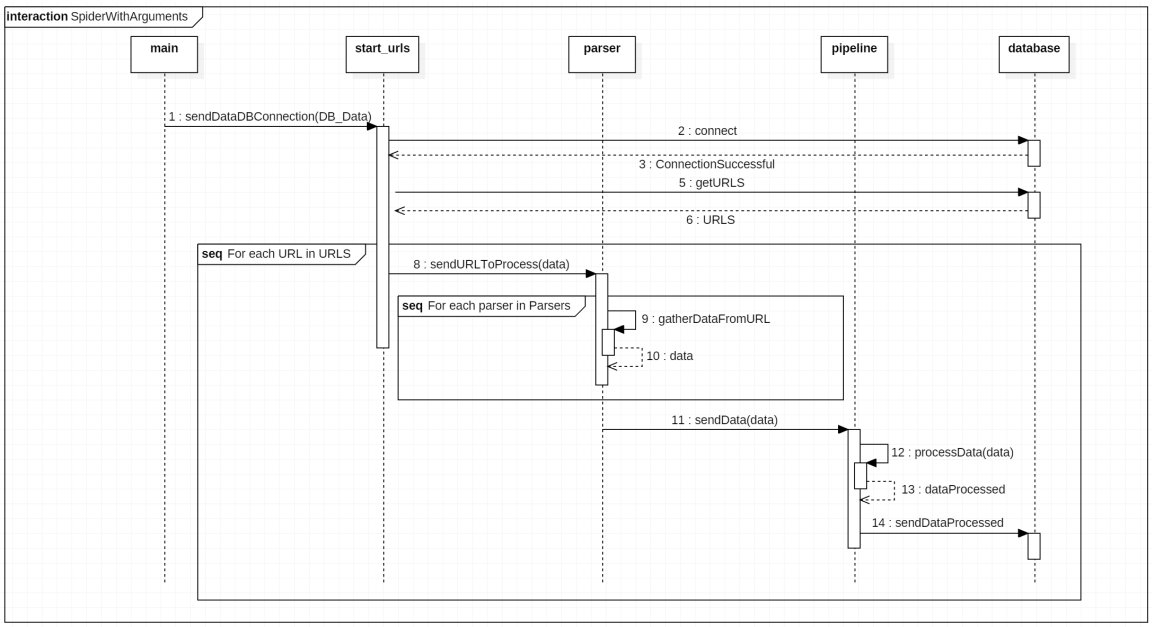

Figura 6.3: Diagrama de secuencia del proceso de una *spider* con argumentos.

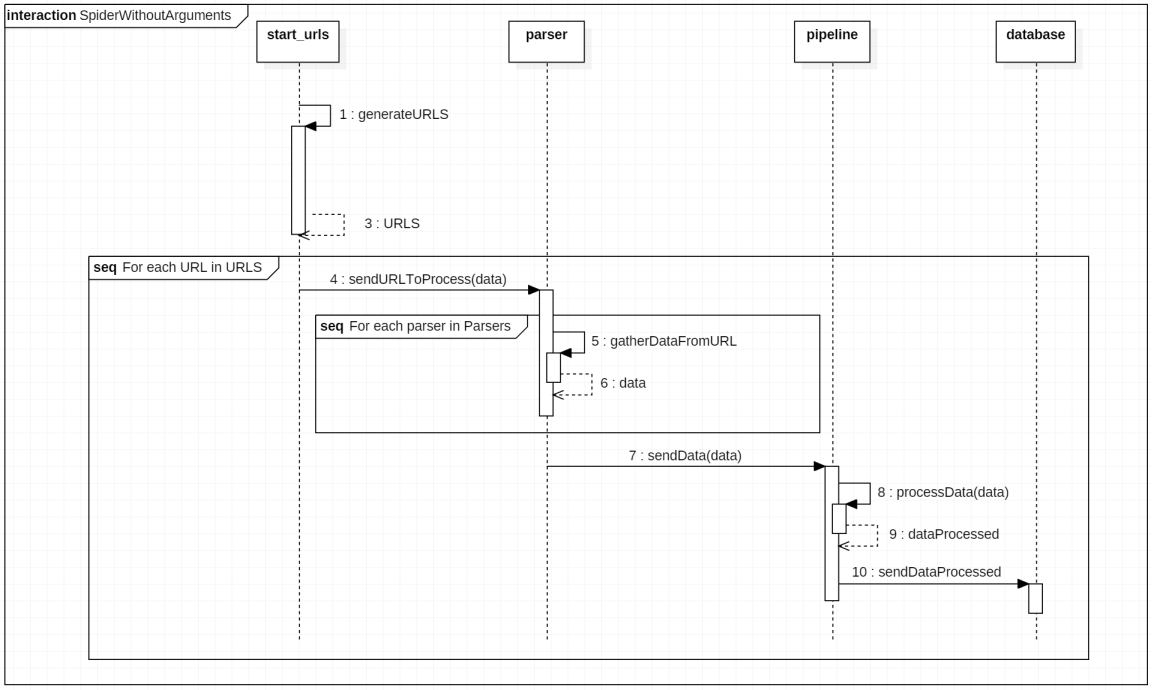

Figura 6.4: Diagrama de secuencia del proceso de una *spider* sin argumentos.

6.8. Problemas y soluciones

En esta sección hablaremos de todos los problemas que consideramos más relevantes que han surgido el desarrollo y cuales han sido sus soluciones.

6.8.1. El problema del volumen de datos

Actualmente hemos decidido trabajar con dos grandes bases de datos que son los sitios web: Arxiv y DBLP. Durante el desarrollo necesitaremos extraer información extra porque en estas bases de datos no disponemos de toda la información que nos gustaría. Debemos extraer datos de páginas de sitios web específicos.

Tanto Arxiv como DBLP son portales que publican una gran cantidad de información del domino de *Computer Science*. En DBLP [40] almacenan datos como: *papers*, autores, conferencias y revistas, y en Arxiv únicamente *papers* y autores, por tanto, tenemos acceso a un volumen de datos más que suficiente para hacer una demostración de la viabilidad de la plataforma.

Una primera idea consistía en extraer toda la información de ambos portales, esto hace un total de: 4.4M+ de *papers*, 2M+ de autores y 1M de datos entre conferencias y revistas.

La ventaja de disponer de este volumen de datos, es permitirnos en un futuro disponer de un histórico de las investigaciones publicadas. Poder hacer recomendaciones de otros *papers* a partir de sus citaciones.

Antes de empezar el proceso, se decidió hacer un primer análisis de cuanto tiempo tardaría el proceso de *web crawling* con este volumen de datos. El análisis consistía en dejar en funcionamiento el servidor durante un día entero, procesando el máximo número de peticiones posibles.

Una vez hecha la prueba teníamos las métricas de cuantas páginas por minuto es capaz de procesar: 102 pag/min, ahora podemos calcular una estimación de cuanto tiempo tardará en el caso peor.

El resultado fue que el proceso nos llevaría como mínimo un mes. De todos los factores que intervienen en el proceso, no disponer de un servidor hace que el tiempo de procesado se incremente considerablemente, además de otras consecuencias descritas en el diseño final.

Después de este primer análisis, vemos que es totalmente inviable seguir con este método, porque significa no poder progresar sobre los demás módulos ya que todos dependen de tener estos datos.

Una alternativa que se estudió fue la de extraer la información de los XML que ponen a disposición tanto Arxiv como DBLP. Estos XML son una instantánea con todos los datos que se han publicado en el sitio web hasta el momento.

Esta opción fue descartada por no contener los enlaces a sitios web externos como Google Scholar, que son necesarios para saber: el número de citaciones, que artículos lo han citado, página de publicación, enlace a un PDF, etc.

Para solucionar este problema se buscó una solución que consistía en: maximizar el número de *papers* posible para cumplir nuestro objetivo con el TFG y minimizar el tiempo.

La solución consiste por hacer un análisis sobre la totalidad de los datos que se dispone y ver que cantidad de publicaciones se hace cada año. La DBLP pone a disposición de forma pública estadísticas de las publicaciones anuales, tal como muestra en la Figura 6.2.

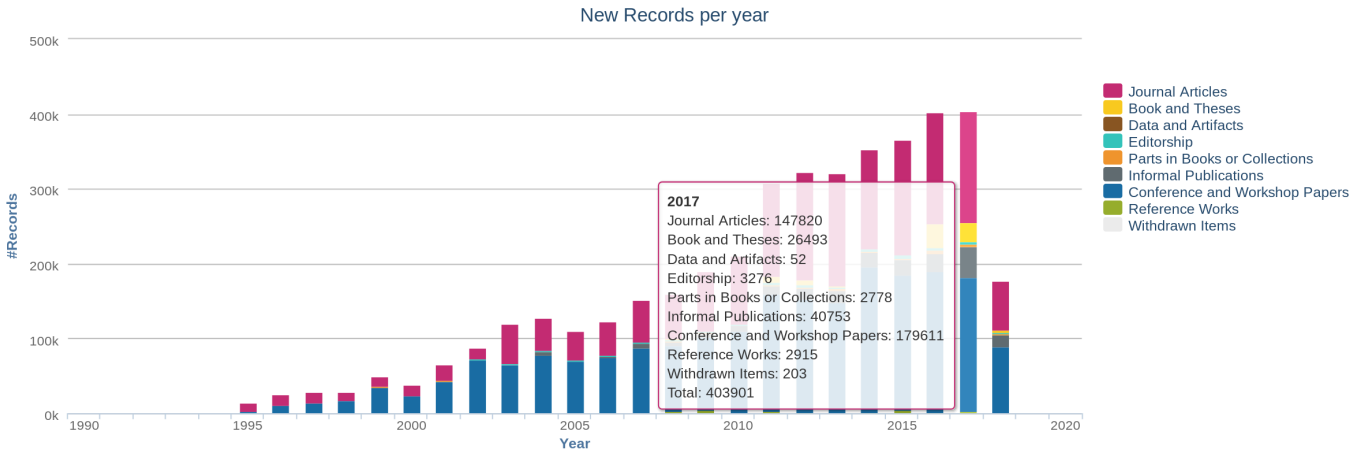

Figura 6.5: Estadísticas de las publicaciones anuales en la DBLP

Después del análisis se tomó la decisión de solo trabajar con los *papers* publicados a partir del año 2017. Esta decisión tiene las siguientes ventajas:

1. La gran mayoría de las publicaciones hechas a partir de este año incorporan el sistema de identificación DOI.
2. Seguimos cumpliendo con el objetivo del TFG que es disponer de una cantidad considerable de *papers* para poder hacer recomendaciones.
3. Satisface la necesidad de un investigador de encontrar las publicaciones que le son más relevantes para él. El interés principal de un investigador son las publicaciones más recientes para informarse.
4. El tiempo de recopilación se reduce considerablemente.

No obstante, esta decisión tiene una desventaja: Perder la opción de disponer de un historial de publicaciones. Esto sucede porque habitualmente en la bibliografía de un *paper* se hace referencia a publicaciones muy anteriores a su fecha de publicación.

El resultado de esta nueva solución hace que el proceso de *web crawling* tarde cerca de tres a cuatro días con las limitaciones que tenemos descritas en el diseño final. Más adelante veremos que existen más mejoras para hacer que el tiempo se pueda reducir a solo dos horas.

6.8.2. El problema de la incoherencia en los datos

El mayor de los problemas que te enfrentas cuando haces *web crawling* es recopilar datos incoherentes. Este tipo de problemas son difíciles de detectar cuando trabajas con grandes volúmenes de datos. Presentaremos dos ejemplos muy típicos para ilustrar bien como es el tipo de problema.

6.8.2.1. Ejemplo 1

Supongamos que estamos extrayendo todos los títulos de los *papers* y en el análisis previo observaste que todos están almacenados como texto plano. El problema surge cuando en la web existen títulos con como este:

Special Issue on High-Speed *Vision-Based* Autonomous **Navigation** of UAVs.

En la web no está almacenado como texto plano sino como texto enriquecido:

<title>Special Issue on High-Speed Vision-Based Autonomous Navigation of UAVs.</title>

El resultado final será este:

Special Issue on High-Speed Autonomous of UAVs.

6.8.2.2. Ejemplo 2

El segundo ejemplo es el más común y el más difícil de solucionar. El problema consiste en que eventualmente los datos pueden variar de posición o de nombre. El resultado no es erróneo sintácticamente pero su contenido no es el correcto.

Supongamos que queremos extraer el abstract de un *paper* y que habitualmente está dentro de la etiqueta:

<div @id="abstract"> ....

Puede ocurrir que en una página se añada un nuevo elemento y fuerza que la etiqueta anterior pase a ser:

<div @id="content" @name="abstract">

Pueden existir indefinidas alternativas, por ejemplo: que en vez de abstract pase a llamarse content.

6.8.2.3. Solución al problema

La inexperiencia con el proceso de *web crawling* hace que estos problemas nos pasasen inadvertidos hasta que completamos la primera recopilación de datos. Fue entonces que por casualidad observemos que teníamos registros con datos incoherentes. Como consecuencia nos forzó rehacer otra vez el proceso de recopilación de datos.

Encontrar una solución al problema es sencillo, la parte más compleja es saber de su existencia. Esto es porque no podemos comprobar si en todas las páginas del sitio web existe algún cambio en el diseño. Tampoco podemos saber si todos los datos de los registros son correctos, porque tendríamos que comparar los datos originales uno a uno con el generado. Existen varias formas de solucionar estos problemas.

La solución al problema de no capturar el texto completo, consiste en forzar a capturar todo el texto. Estoy incluye capturar las etiquetas que haya tanto dentro como fuera y con expresiones regulares eliminarlas.

Para extraer los datos y que sea independiente a posición y al nombre asociado, también hay que usar expresiones regulares, combinarlas con XPath [45] y trabajar con rutas absolutas y relativas.

Por ejemplo, queremos extraer los abstract de un sitio web, una forma sería:

xpath('//h2[text()[re:match(.,"Abstract")]]/../p[@class="Para"]/text()').extract()

La anterior sentencia funciona de la siguiente forma:

1. Busca todas las cabeceras de tipo H2 de forma absoluta.
2. De estas etiquetas, filtramos todas aquellas que el texto no sea igual a "Abstract".
3. Las resultantes, visitamos la etiqueta hermana de tipo *p* con nombre de clase "Para".
4. Finalmente extraemos el texto que contiene la etiqueta donde está el abstract.

Una mejora es evitar que sea sensible a mayúsculas especificando a la expresión regular que ignore el *case sensitive*.

La única limitación a esta solución es que hay que fijar al menos un elemento común entre todas las páginas web del sitio, en este caso H2. Si hay una variación de H2 a H4, se puede hacer que pruebe con diferentes números hasta dar con el correcto.

6.8.3. Reducir el tiempo de *web crawling*

Durante el desarrollo del módulo se aprendieron nuevas técnicas para refinar este proceso y tienen como objetivo reducir el tiempo. La ventaja de reducir el tiempo implica reducir el coste económico necesario porque no se consumen tantos recursos del servidor del sitio web. Existen diversas formas, pero en esta sección se explicarán dos procesos que ayudan en gran medida a reducirlo.

6.8.3.1. Evitar el procesado de datos

Un problema habitual cuando iniciamos un proceso de *web crawling* es que no siempre podemos extraer el contenido exacto libre de palabras o caracteres que no queremos. Para comprender mejor el problema es ver un ejemplo real.

Cuando consultamos las conferencias en la DBLP y queremos ver que edición es y donde se celebró tenemos este formato de salida:

31. ASE 2016: Singapore y **ASE 2017: Urbana, IL, USA**

La diferencia principal entre las dos conferencias es la forma de describir la ubicación.

Cuando se hace una conferencia en USA hay que poner el estado y después la ciudad, mientras que en los países que no siguen este sistema, únicamente se pone país y ciudad. Otra diferencia, aunque significativa, es que no todas las entradas tienen porque tener como prefijo un número antes de la edición, en el primer caso hay un "31.".

Si quisiéramos extraer la información de la edición y el lugar donde se celebró implica pensar en una forma que sea independiente a posibles alteraciones que puedan surgir en el texto. Se usan casos especiales para representar todas las posibles alteraciones que puede sufrir una entrada. Esto es así porque sabemos del anterior apartado que no siempre los datos están almacenados de la misma forma.

Este pequeño procesado aunque trivial a nivel de coste computacional, hay que tener en cuenta que se ejecutará millones de veces y puede llegar a tener un impacto en el tiempo total del proceso de *web crawling*.

La decisión que se tomó es delegar el procesado de los datos al módulo de base de datos, en la sección de implementación del módulo se entrará en más detalle.

6.8.3.2. Buscar las dependencias entre los elementos

El segundo método es analizar como están relacionados los elementos para identificar las dependencias.

Entendemos como dependencia que si un elemento A es requisito de B, entonces para llegar a B antes debemos pasar por A. Estas dependencias varían según el sitio web, y hay que hacer un análisis específico para cada sitio.

En el caso de la DBLP para acceder al contenido de un *paper* tenemos que pasar antes por la conferencia o revista donde se publico. Es importante darse cuenta que en la DBLP no existe una dependencia entre conferencia y revista, son dos elementos independientes. Esto nos permite que podamos extraer las revistas y conferencias en paralelo.

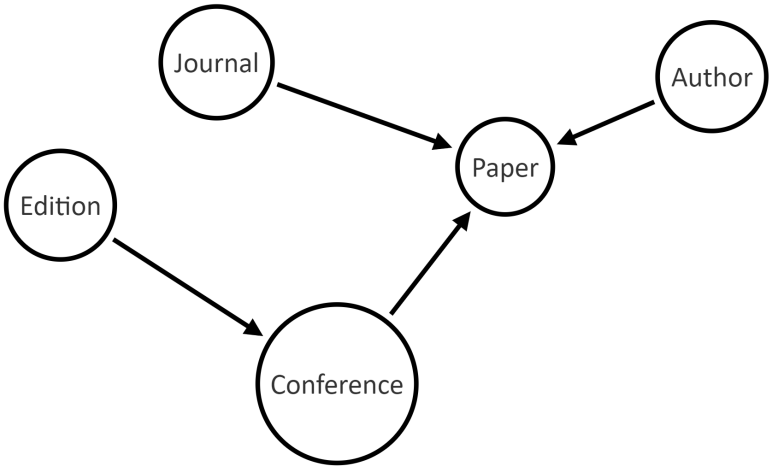

Figura 6.6: Dependencias de la DBLP

Una vez sabemos como están relacionados los elementos hay que cambiar el paradigma de como funciona una *spider*. Hasta ahora la idea es tener una única *spider* especializada en extraer toda la información de un sitio o página web.

Si queremos explotar esta segmentación, la solución pasa por hacer que una *spider* esté especializada en extraer la información de un elemento de un sitio web en concreto. Esto permite poder lanzar varias *spiders* simultáneamente logrando una paralelización del proceso y reducir el tiempo considerablemente. No obstante, esta solución genera una nueva desventaja.

Anteriormente cuando una única *spider* recopilaba la información de un sitio web, era fácil poder asociar elementos entre si. Por ejemplo para consultar un *paper* hemos tenido que pasar fuerza por una conferencia o revista. Por tanto, sabemos que este *paper* pertenece a la conferencia o revista anterior. Con esta solución no disponemos de esa información porque todos los elementos se procesan de forma individual.

Para solucionar este problema se modificó el diseño inicial de como tratar las *spiders* dando como resultado el diseño final. Ahora una *spider* puede recibir como argumentos parámetros de conexión de una base de datos y a través de un pequeño proceso podemos volver a conseguir asociar los elementos. La consecuencia es pagar un coste pequeño de tiempo de conexión y recopilación de datos. El proceso está dividido en dos fases:

1. Procesamos todos los elementos que no tienen dependencia, que llamaremos padres.
2. Por cada elemento padre haremos el siguiente proceso:
   a) Procesamos el elemento.
   b) Le preguntamos al elemento si tiene asociados otros objetos él. En caso afirmativo, debemos modificar la estructura de datos de salida para añadir un nuevo campo para almacenar las URL de los elementos asociados a él. Podemos aprovecharnos de la unicidad de las URL como identificador.
3. Con los elementos padre procesados, si estos tenían elementos asociados a él debemos procesarlos. Para procesarlos la única forma de tener la información del elemento padre es empezar por las URL almacenadas en dichos elementos padre.
4. Por cada elemento asociado haremos el siguiente proceso:
   a) La *spider* debe conectarse a la base de datos y obtener todas las URLs almacenadas en el elemento padre.
   b) Procesamos el elemento.
   c) Almacenamos la URL que hemos usado para llegar hasta él en un nuevo campo en su estructura. Esto nos permitirá en el futuro poder hacer una asociación entre el nuevo elemento y su predecesor a través de la URL.
   d) Le preguntamos al elemento si tiene asociados otros objetos él. En caso afirmativo, debemos modificar la estructura de datos de salida para añadir un nuevo campo para almacenar las URL de los elementos asociados a él. Podemos aprovecharnos de la unicidad de las URL como identificador.
5. Repetimos los procesos hasta que no quedan más elementos que recopilar.

Una vez finalizado todo el proceso, podremos asociar los elementos a través de la URL de forma provisional ya que se reemplazará por el ID generado por la base de datos. Pero para ahorrar tiempo de procesado esta tarea se delega al módulo de base de datos que se explicará en su implementación.

Para no hacer repetidas consultas en la base de datos, una opción es almacenar todos los datos en un diccionario y que sea accesible para toda la *spider*.

Finalmente, esta solución tiene como única desventaja el coste de tiempo de carga de todos los elementos de la base de datos, pero nos recompensa con diferencia poder paralelizar el proceso.

6.8.4. Extraer datos de Google Scholar

Con recopilar la información de Arxiv y DBLP no es suficiente. Porque los datos que tenemos son limitados: no tenemos keywords, solo tenemos abstracts por parte de Arxiv, tampoco acceso a los PDF, etc. Una alternativa sería si el *paper* tiene DOI, consultar la página que tiene asociada.

Consultar el DOI no nos garantiza tener acceso a la información que buscamos, porque generalmente tienen bloqueado el acceso al contenido mediante un pago. Pero existe otra fuente de información que si dispone de todos estos datos, es Google Scholar.

En la introducción uno de los motivos de rechazar usar los XML es no tener acceso a Google Scholar. Para introducir al lector que es Google Scholar, se define como usar el motor de búsqueda de Google pero enfocado a encontrar *papers* de forma precisa.

Además de un resultado preciso también nos ofrece: la página web donde está alojado el *paper*, número de citaciones y quienes lo han citado, generalmente un enlace al PDF con acceso gratuito, *papers* relacionados, un enlace a los autores con su página personal, etc.

Sin embargo, existen dos problemas al usar Google Scholar.

6.8.4.1. Superar las barreras de Google

Para evitar "ataques" constantes de *spiders* Google bloquea el acceso de esa IP al recibir un número exacto de peticiones seguidas en un intervalo muy pequeño.

Para saber el número máximo de peticiones se hizo una prueba que consistía en medir hasta cuantas peticiones seguidas se pueden hacer hasta el bloqueo de IP. El número exacto es 50. El bloqueo de IP es indefinido hasta que confirmes que no eres un robot resolviendo un *captcha* en su web.

Aquí al igual que en el problema del volumen de datos, se calculó cuanto tiempo se tardaría en recopilar toda la información. El método consistía en tener una máquina que cada cierto tiempo procesara un bloque de 50 URL hacia Google. Para evitar el bloqueo de la IP pública había que esperar un tiempo entre cada bloque.

El resultado de la prueba fue que para tan solo 200.000 URLS tardaríamos más de tres meses en recopilar toda la información.

6.8.4.2. Búsquedas con enlaces erróneos

El segundo problema que nos encontramos es que los enlaces que tenemos son simples búsquedas directas al buscador. De todos los resultados que nos devuelve Google Scholar, tenemos que iterar entre todas las respuestas y buscar cual coincide con el título del *paper*.

Decidimos que únicamente buscaríamos la respuesta en la primera página y si no esta, con mucha probabilidad es que no existe o el título no es adecuado. La decisión de decidir si una entrada coincide con el título de un *paper* depende de cuanto riesgo estas dispuesto a asumir.

Una opción es aceptar a partir de cierto % de similitud, pero corremos el riesgo de aceptar entradas que posiblemente son totalmente diferentes al tema del *paper*. Nuestra decisión fue solo aceptar las respuestas que tengan una coincidencia del 100 %, porque queremos la fiabilidad total de que el contenido que extraemos y el tema del *paper* sea el mismo. Esta decisión tiene como consecuencia limitar el número de resultados totales.

6.8.4.3. Solución

La solución ideal sería pedir que nuestro ISP renovase la IP pública cada 50 peticiones a Google, entonces el tiempo se reduce a cuestión de días, pero lógicamente esto no es posible. Pero la solución final pasa por usar esta idea como base y está compuesta por dos piezas.

La primera es usar los servicios de Amazon. Los clientes de Amazon Prime disponen cada mes 750h de calculo y 30 GB de espacio, la potencia de las máquinas es limitada pero el proceso de *web crawling* no requiere de cálculos complejos.

La característica más importante de este servicio es que Amazon tiene contratado un rango de IP públicas amplio. Ofrece además la posibilidad de tener máquinas en diferentes países ampliando el banco de IP públicas.

El siguiente problema es como enviar las URL a las *spiders* de las máquinas de Amazon. La segunda pieza corresponde a desarrollar un servidor que tiene como objetivo subministrar los bloques de URL pendientes a las *spiders*. El servidor actuará como intermediario entre el módulo de base de datos y las *spiders*.

A continuación, se explicaran los pasos de comunicación entre el servidor y Amazon. Para facilitar la lectura, llamaremos al servidor como LOCAL SERVER.

1. Una *spider* se conecta al LOCAL SERVER y pide un bloque de 50 URL de Google Scholar
2. LOCAL SERVER consulta a la base de datos y filtra las URL que aún no han sido procesadas. Recopila las 50 primeras y las envía.
3. Durante las 50 peticiones:
   a) La *spider* procesa una de las URL y obtiene un resultado satisfactorio o no.
   b) Enviar los resultados al LOCAL SERVER.
   c) LOCAL SERVER procesa el resultado, si es correcto marca como procesada esa URL y en caso contrario sigue estando en pendiente. El resultado lo envía a la base de datos.

Con estas dos piezas ya se puede formar la solución del problema:

1. Generar un número determinado de instancias de Amazon, en nuestro caso ocho.
2. Cargar en cada instancia una imagen con todo el software necesario y ejecutar un *script* que de forma autónoma inicié un proceso de *web crawling*.
3. El proceso de *web crawling* ejecuta el proceso de interacción con LOCAL SERVER descrito anteriormente.
4. Una vez procesadas las 50 peticiones, se reinicia la instancia para renovar la IP pública.

Todo este proceso está automatizado mediante *scripts* y por tanto, solo basta con iniciarlo una única vez. El proceso se repite hasta que no queda ninguna petición sin procesar.

El resultado final es que un 90 % de las peticiones se han procesado correctamente debido a la restricción impuesta sobre buscar entradas y títulos iguales. El 10 % restante no se consiguió procesar porque no se encontraba el título entre los resultados.

6.8.5. El problema de las páginas dinámicas

Las páginas dinámicas son la forma más fácil y eficiente de evitar que las *spiders* "ataquen" un sitio web. Un ejemplo es https://ieeexplore.ieee.org/.

Para entender el problema, primero hay que entender que hace Scrapy. Cuando Scrapy recibe una página web para procesarla lo que hace es descargarla en local y lanzar las peticiones en la copia.

El problema de este tipo de páginas, reside en que el contenido es cargado después de un lapso de tiempo variable y además de que el contenido se genera de forma dinámica. Por tanto, la copia que recibe Scrapy suele ser una página en blanco con metadatos.

No existe una solución sencilla para sortear este problema. Una solución propuesta por los desarrolladores es ejecutar un navegador embebido dentro de la propia *spider* y descargar el contenido. El programa necesario es PhantomJS. La complejidad de implementar este *framework* es alta y dada nuestra limitacióno de tiempo no fue posible.

Aún así, muchos de los *papers* que disponemos están en la IEEE y necesitamos la información que nos proporciona: *keywords* y abstracts para el recomendador. Por tanto, debemos buscar una alternativa fácil para extraer los datos.

La solución paso por analizar los metadatos de la copia que genera Scrapy. El contenido del fichero son bloques de texto comprimidos sin formato y sin estilo para dificultar encontrar la información.

Buscando patrones en los ficheros de la IEEE, esta siempre guarda al menos una parte del contenido principal embebido entre los metadatos para que la carga sea más rápida. Además, por casualidad el contenido coincide con el que buscamos: las *keywords* y abstracts.

Por ejemplo, para extraer el abstract hay que hacer los siguientes pasos:

1. Buscar el bloque de texto donde la palabra abstract está embebida.
2. Buscar la posición donde empieza la palabra y sumarle 3 a la posición porque está representado como un vector tal que así: “abstract:[””
3. Recorrer letra a letra hasta llegar a una ”, que significa fin del abstract.

Este sería el código:

```
data = response.xpath('//script[@type="text/javascript"][3]')[1].extract()
pos = data.find('abstract') + len('abstract') + 3
abstract = ""
while (data[pos] != '"'):
    abstract += str(data[pos])
    pos = pos + 1
```

Figura 6.7: Ejemplo de código de extracción de un abstract en IEEE

Con este proceso todos los *papers* con DOI a IEEE tienen *keywords* y abstract, en nuestro caso son casi el 50 % de nuestros registros.

6.9. Futuro

- Añadir la opción de que una *spider* sea agnóstica al contenido de la web. Es decir sepa extraer información de cualquier página web buscando palabras clave como: *download pdf, view pdf, abstract, authors*, etc.
- Añadir la capacidad de saber extraer contenido de páginas dinámicas.
- Crear un servicio *deamon* para Linux. Permitir mediante configuración que cada cierto tiempo haga una nueva búsqueda por si hay nuevas publicaciones.
- Crear un servidor para gestionar las *spiders*.
- Un cliente web para monitorizar el estado del proceso de *web crawling*.
- Añadir más sitios web.

Capítulo 7

Módulo: Base de datos

7.1. Objetivo

Desempeña dos funciones:

1. Servir las peticiones de datos de los diferentes módulos.
2. Responsable de generar la base de datos a partir de los datos recopilados por el módulo del *crawler*.

7.2. Introducción

El módulo en un inicio se pensó que solo sirviera peticiones a los demás módulos, pero con la necesidad de optimizar el proceso de *web crawling*, nos obligó a redefinir su funcionalidad.

La función añadida es que ahora también es la responsable de construir la base de datos a partir de los datos generados por el módulo del *Crawler*. Partimos de un estado inicial donde disponemos de diferentes conjuntos de datos pero que no están relacionados entre si y algunos de los datos necesitan un postprocesado.

No obstante, sabemos de antemano que todos los elementos que tengan asociado otro elemento, existe un campo común entre ellos que identifica la asociación de forma unívoca y nos permite crear la relación. Para la construcción de la base de datos, el módulo tendrá que ir completando las siguientes etapas:

Postprocesado de los datos

Como ya hemos dicho esta etapa surge de la necesidad de optimizar el proceso de *web crawling*. En un inicio esta etapa contemplaba dos procesos:

- Eliminar todas las palabras o caracteres no válidos de un registro, por ejemplo, saltos de línea.
- Extraer los metadatos de un registro, por ejemplo, extraer la ubicación donde se celebró una conferencia.

Pero debido a cambios explicados en el diseño final, la extracción de metadatos como las ubicaciones de las conferencias no se ha hecho. La función actual de esta fase es hacer que todos los datos tengan la misma estructura, tal y como se define en el primer punto de la lista. Para ello se usan expresiones regulares para el filtrado.

Generar las asociaciones entre los elementos

La segunda etapa es la más sencilla de todas y consiste en crear todas las relaciones entre los elementos. Aprovecharemos que el módulo del *Crawler* ya nos ha hecho gran parte del trabajo y solamente tenemos que unir las diferentes colecciones.

Crear la colección de autores

El problema de la ambigüedad de los nombres de los autores ya explicado en el módulo del *Crawler* sigue sin poder resolverse en este módulo. Aún así necesitamos los autores para poder hacer recomendaciones por ejemplo, en base a una coautoría. Por tanto, necesitamos crear la colección de autores teniendo en consideración este problema.

Fusionar las bases de datos de Arxiv y DBLP

Está etapa es en la que haremos más hincapié porque es el proceso más complejo. Fusionar diferentes bases de datos de distintos sitios web es un proceso complejo. Para visualizar el problema, hay que ver las bases de datos como conjuntos y nuestro objetivo es encontrar la intersección.

El problema de encontrar la intersección es definir con qué atributo relacionamos los elementos de distintos conjuntos. El atributo elegido en este caso ha sido usar el título. Pero como ya se explicará en detalle en la sección, este campo tiene muchas limitaciones e inconvenientes que no usar otro campo que identifique de forma univoca un elemento, como podría ser el DOI.

Una solución propuesta ha sido buscar la similitud entre ambos títulos, y considerar que dos títulos son el mismo a partir de cierto umbral de similitud. Pero esta solución también tiene inconvenientes. Explicaremos todas las decisiones hemos tomado para optimizar el máximo beneficio.

Extraer la información de los *papers*

Una parte clave para el funcionamiento del módulo del recomendador es que tenga a su disposición diferentes métricas para poder mejorar las recomendaciones. La última etapa del módulo es la encargada de extraer la información de los *papers*.

El problema principal que tuvimos fue pensar como extraer la información de un PDF. En un inicio, se probó con diferentes programas que tenían un proceso muy sencillo de conversión de PDF a fichero de texto plano. Los inconvenientes de estos métodos es que se pierde mucha información y metadatos como los nombres de las secciones.

Para resolver el problema fue necesario plantearse las siguientes preguntas: ¿Cómo podemos reconocer una sección?, En caso de encontrarla, ¿Cómo podemos identificar dónde empieza y dónde acaba?, etc.

Una vez conseguimos resolver este problema surgieron dos nuevos: coste temporal y espacial. El coste en disco para una muestra de tan solo 110.000 PDF era de aproximadamente 0,75 TB y un coste temporal de un mes de cálculo. La solución encontrada transformo estos costes a tan solo un coste espacial de 3 GB y un coste temporal de tan solo 3 días.

Finalmente, uno de los objetivos adicionales del módulo es intentar automatizar el máximo posible todo este proceso mediante *scripts*. Deben, además, ser fáciles de ampliar porque en un futuro podamos incluir nuevas bases de datos sin modificar la estructura ya creada.

Al igual que el anterior módulo, una parte del tiempo se destinó al aprendizaje de como funcionan las tecnologías que usamos. El resto del tiempo se destinó a investigar o pensar en las soluciones más óptimas con los recursos que tenemos.

7.3. Especificación

7.3.1. Grobid

7.3.1.1. Entrada

- *host* : La URI para conectarse a la base de datos.
- *port* : Puerto para conectarse a la base de datos.
- *database_input* : Base de datos donde está la colección con las URL a PDF.
- *database_output* : Base de datos donde escribirá los datos generados. La colección está fijada como *grobid_output*.
- *pdf_field* : El campo de la colección donde buscar las URL.
- *input_collection* : Colección donde se almacenan los datos.
- *parallel* : Número de hilos para paralelizar el proceso.

7.3.1.2. Salida

- Escribe todo el contenido de los XML que ha podido procesar a la colección especificada en la entrada.
- En la carpeta *xmls* están todos los XML procesados.
- En la carpeta *pdfs* están todos los PDF que no han podido ser procesados por algún error.

Todos los ficheros usan el MongoDB Id del *paper* para identificar un fichero.

7.3.2. *Scripts* para generar la base de datos

La salida y la entrada de los *scripts* para generar la base de datos son iguales solo varía el nombre de las colecciones.

7.3.2.1. Entrada

- *host* : La URI para conectarse a la base de datos.
- *port* : Puerto para conectarse a la base de datos.
- *database_input* : La base de datos donde se almacenan las colecciones.
- *database_output* : La base de datos donde se almacenará la nueva colección.
- *collection_input_A* y *collection_input_B* : Las colecciones que se unirán.

7.3.2.2. Salida

Genera una nueva colección con la fusión de las dos colecciones recibidas como argumento.

7.4. Diseño inicial

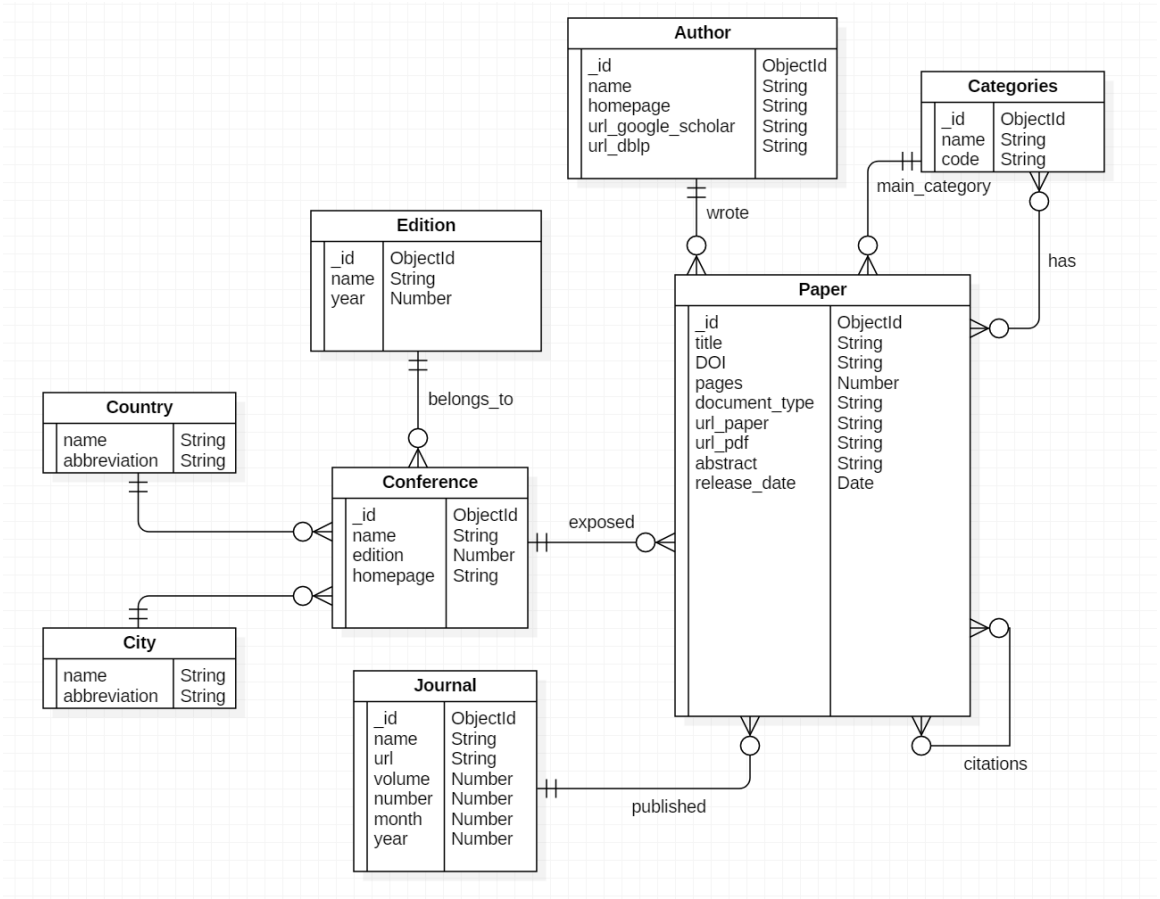

Figura 7.1: Diseño inicial de la base de datos

7.5. Diseño final

Se han hecho varios cambios respecto el diseño inicial. El primer cambio consiste en modificar los contenidos de algunas tablas:

1. En el caso de los *papers* y ediciones se han añadido más campos para albergar más información.
2. En la tabla de conferencias y autores se han eliminado los campos que no le dábamos ningún uso.

En un principio se quería tener la localización donde se celebraban las conferencias ya que esta información podría ser útil a la hora de mejorar las recomendaciones. Se ha tomado la decisión de no recopilar esta información por dos motivos:

1. Existe una gran cantidad de conferencias donde no se ha publicado la ubicación.
2. Errores humanos al escribir ciudades, por ejemplo, confundir San Sebastián como un país.

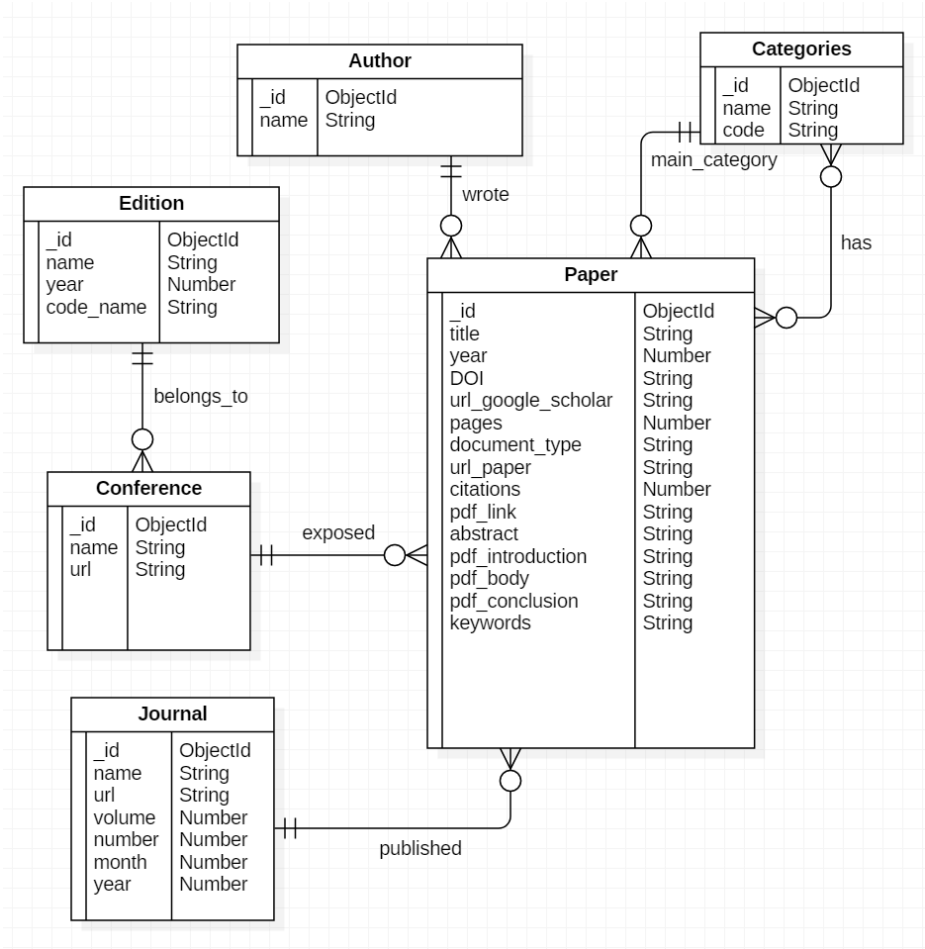

Figura 7.2: Diseño final de la base de datos

7.6. Tecnologías

- *Pymongo* : Es el driver que nos permite comunicarnos con la base de datos MongoDB. Es de código abierto bajo licencia MIT. Compatibilidad con Python 2.7 y Python 3.6.
- *MongoDB* : Software encargado de gestionar los datos. Licencia GNU AGPL.
- *Python* : Lenguaje principal para programar todas las funcionalidades. Licencia PSFL similar a la GPL pero sin *copyleft*.
- *Fuzzywuzzy* : Paquete para Python que incorpora el algoritmo Levenshtein Distance necesario para calcular la similitud entre palabras. Licencia GPL v2.0
- *Grobid* : Software que utiliza técnicas de Machine Learning para extraer las secciones de un PDF y guardar el resultado en un XML. Licencia Apache 2.0.

7.7. Implementación

7.7.1. Conceptos básicos de MongoDB

Es necesario introducir conceptos básicos de como trabaja MongoDB para poder usarlos sin restricción. MongoDB es un sistema de base de datos NoSQL. Trabaja con dos elementos principales:

- *Documento* : Es donde se almacenan los datos mediante diccionarios. Las claves son los nombres de los campos.
- *Colección* : Es el contenedor donde se almacenan todos los documentos.

Ejemplo de documento:

```
{
    "Human":
    {
        "name" : "",
        "age"  : ""
    }
}
```

Aunque MongoDB sea NoSQL hemos decidido añadir identificadores entre los documentos con el fin de relacionarlos y así evitar duplicados a la hora de generar el grafo con el módulo del Graph Builder.

7.7.2. Generar las relaciones entre los documentos

El proceso consiste en crear relaciones entre los documentos de las diferentes colecciones. Previamente el módulo del *Crawler* ya nos ha hecho gran parte del trabajo creando un campo común entre dos documentos que tienen que estar relacionados. El valor que se usó para el campo fue la URL porque identifica de forma univoca un elemento en un sitio web.

Solo hay que tener en consideración seguir el orden de las dependencias descritas por el sitio web. El método es el siguiente:

1. Leer los datos de la colección A y guardarlos en un diccionario. Usar como clave el ID de MongoDB.
2. Repetir para la colección B, pero esta vez hay que elegir como clave el campo común en este caso la URL.
3. Por cada documento (A') de la colección A:
   a) Obtener el documento B' de la colección B que tenga como clave el valor del campo común de A'.
   b) Añadir al documento B' un nuevo campo para guardar el ID de A'.
   c) Añadir al documento A' un nuevo campo para guardar el ID de B'.
   d) Ahora B' y A' están asociados.
4. Escribir las dos colecciones nuevas A y B en una nueva base de datos.

Se tomó la decisión de escribir siempre el resultado de modificar las colecciones en una nueva base de datos. El motivo es evitar sobrescribir sobre la original y poder hacer una comprobación de que el proceso es correcto.

La comprobación consiste en que existe el mismo número de documentos las nuevas colecciones que en sus respectivas originales. Cada documento de la colección tiene que tener un campo con el ID de los documentos asociados a él. Además, debe tener la misma cantidad de identificadores que de URL en el campo provisional.

Para reducir el coste del algoritmo escribimos las colecciones enteras al final en vez de ir escribiendo documento a documento. Los desarrolladores de MongoDB recomiendan siempre que se pueda, hacer escrituras en bloque. Porqué el sistema interno sabe aplicar las optimizaciones necesarias para que vaya más rápido.

7.7.3. Crear la colección de autores

Debido a el problema de la ambigüedad de nombres se decidió no recopilar los autores en el proceso de *web crawling*. Sin embargo, el recomendador precisa de los autores para poder hacer recomendaciones de coautoría.

Se tomaron dos decisiones:

1. Crear la colección de autores a partir de la colección de *papers* ya que cada documento tiene un campo especifico con los nombres de los autores.
2. Interpretar a todos los autores como personas diferentes aunque sean la misma.

7.7.4. Fusionar Arxiv con DBLP

La parte más compleja del módulo y la vez más interesante es tener la opción de poder fusionar diferentes bases de datos. No solamente poder fusionarlas, sino que también lograr el máximo número de coincidencias. Ya veremos más adelante que es difícil conseguirlo y porque lo es.

En este caso trabajamos únicamente con dos bases de datos: DBLP y Arxiv, pero el proceso serviría para cualquier cantidad. Antes de hacer la fusión es importante recordar que disponemos de dos ventajas:

1. No hay elementos desconocidos entre las dos bases de datos. Porque todas están generadas a partir de la información extraída por el módulo del *Crawler* y los elementos son conocidos.
2. Sabemos que siempre existirá al menos un campo en común entre dos colecciones de distintas bases de datos. Por ejemplo, el título de un *paper* seguro que está en ambos documentos de la colección de *papers*.

Cuando queremos fusionar dos bases de datos una forma de visualizar este proceso es imaginarnos las colecciones como conjuntos y nuestro objetivo es buscar la intersección entre ellas. El siguiente paso es pensar como comparamos los documentos que forman parte de los conjuntos y es aquí donde reside la dificultad. La forma que hemos utilizado en este proyecto es comparar los campos comunes en busca de una similitud.

En este caso, la fusión consiste en unificar el elemento *paper* de ambas bases de datos porque en Arxiv no hay más elementos.

El proceso empieza cuando buscamos que campos hay en común entre dos documentos de la colección *papers*. No siempre tenemos campos que identifiquen de forma univoca un documento y este es el problema que nos enfrentaremos cuando intentamos fusionar Arxiv y DBLP.

Sabemos de antemano que como mínimo ambos documentos tienen el campo título, pero elegir este campo no es la mejor de las opciones si tenemos alternativas como el DOI. Esto es por el DOI identifica de forma univoca. Pero en Arxiv aproximadamente solo el 5 % de los *papers* tienen asociado este atributo. Por tanto, no podemos usar este campo.

Otra alternativa hubiera sido usar los autores. Suponiendo que no existiera el problema de la ambigüedad de nombres de autores, se podría comparar a través de la fecha de publicación, que autores participaron y el medio donde se publicó. Si dos documentos comparten esta información con una alta probabilidad podemos decir que son el mismo *paper*.

Las razones de porque el título no es un buen campo entre otros factores, el principal es porque no tenemos la garantía de que ambos sitios web escriban el título igual. Por lo tanto, no nos queda más opción que usar el título y esto implica buscar una solución para poder hacer la comparación.

La solución explorada es fusionar a través de la similitud de un título. El razonamiento detrás de esta idea también serviría para otros campos de texto, pero los algoritmos no tienen porque ser válidos. En este caso usamos un algoritmo que está pensado para textos con pocas palabras.

Esta solución es una buena aproximación a una solución razonable, pero por cuestión de tiempo no hemos podido implementar mejores opciones. Empezaremos introduciendo cual es esta solución y finalmente, como se podría mejorar.

7.7.4.1. Buscar la similitud entre dos títulos

Es lógico pensar que no existen dos *papers* con el mismo título y que podría ser una buena forma de comparar. El problema de los campos de texto como es el caso del título, es que no existe ningún procedimiento estándar de como los sitios web deben almacenar los datos. Por tanto, tenemos que ponernos en la peor de la situaciones.

Antes de hacer la fusión hicimos pruebas de como están almacenados los títulos en ambas bases de datos y encontramos los siguientes hechos:

1. Para un mismo *paper* puede ocurrir que el título almacenado en BDLP tenga alguna palabra diferente respecto el título almacenado en Arxiv.
2. Un caso más extremo. El mismo *paper* pero con títulos totalmente diferentes entre si.
3. Es habitual que un *paper* tenga revisiones. Casualmente puede ocurrir que el título final sea diferente a una versión anterior y esta es la que está almacenada en la otra base de datos.
4. El más común de todos: error humano al introducir el título.

Con estos hechos queda descartada la solución de comparar 1:1 ambos títulos. El resultado sería bastante pobre ya que es difícil que estén escritos exactamente. Una persona si es capaz de extraer metadatos leyendo únicamente los títulos aunque tengan las diferencias descritas en esa lista. Y puede saber con cierta probabilidad si están relacionados o son totalmente distintos. Lógicamente no podemos revisar uno a uno todos los títulos.

La solución que se propone es calcular la similitud entre dos textos. Como los títulos son textos que generalmente tienen pocas palabras, para calcular la similitud se tomó la decisión de usar el algoritmo de Levenshtein ya visto en la asignatura de Algoritmia. El coste del algoritmo es $\mathcal{O}(nm)$ donde n y m es la longitud de los textos.

El algoritmo calcula el número mínimo de operaciones necesarias para transformar una cadena a otra y que sean iguales. El número resultante lo podemos transformar en un porcentaje de similitud que nos indicará con que probabilidad de que sean comunes. La decisión más importante es decidir un *threshold* que nos servirá para decidir a partir de que porcentaje se considera que son iguales o no.

Para ahorrarnos tiempo de programarlo usaremos un paquete para Python llamado *FuzzyWuzzy*. Este paquete usa el algoritmo de Levenshtein que está embebido en dos funciones que automáticamente ya nos devuelven la similitud de dos textos expresada en porcentaje.

Una parte del algoritmo consiste en usar una de las dos funciones que incluye el paquete anterior. Las dos funciones no son excluyentes una de la otra, ambas se usarán para calcular la similitud, se explicará el porqué más adelante. Haremos un breve resumen de como funcionan ambas funciones.

7.7.4.1.1 FuzzyWuzzy: Partial String Similarity

Supongamos que queremos comparar dos cadenas de texto que hablan de dos equipos de fútbol americano. Queremos ver cual es la similitud aplicando el algoritmo de Levenshtein directamente:

fuzz.ratio("YANKEES", "NEW YORK YANKEES") → 60 %
fuzz.ratio("NEW YORK METS", "NEW YORK YANKEES") → 75 %

Para nosotros es fácil ver que la primera opción con seguridad habla del mismo equipo, mientras que la segunda hablan de dos equipos diferentes. Aún así la distancia de edición es menor, por tanto, para el algoritmo es más similar. Tenemos el problema de que si dos frases no tienen la misma longitud el resultado puede dar falsos positivos.

Una solución consiste en utilizar un heurístico que el autor del paquete llama *best partial*. Se usa cuando dos cadenas de texto tienen longitud diferente. Sea M la longitud de la cadena más pequeña y N para la contraria. Nos interesa ver la mejor puntuación posible comparando M con una sub-cadena de N de longitud M.

fuzz.ratio("YANKEES", "NEW YOR") → 14
fuzz.ratio("YANKEES", "EW YORK") → 28
fuzz.ratio("YANKEES", "W YORK ") → 28
fuzz.ratio("YANKEES", " YORK Y") → 28
...
fuzz.ratio("YANKEES", "YANKEES") → 100

La última comparación nos da que son exactos y con el resultado del heurístico podemos deducir que ambas cadenas son iguales. Para simplificar el proceso la función de *partial ratio* ya nos hace todo el trabajo y el resultado es:

fuzz.partial_ratio("YANKEES", "NEW YORK YANKEES") → 100
fuzz.partial_ratio("NEW YORK METS", "NEW YORK YANKEES") → 69

7.7.4.1.2 FuzzyWuzzy: Token sort ratio

La longitud de las cadenas no es el único problema que podemos tener, también hay que tener en consideración el orden de las palabras.

fuzz.ratio("New York Mets vs Atlanta Braves", "Atlanta Braves vs New York Mets") → 45
fuzz.partial_ratio("New York Mets vs Atlanta Braves", "Atlanta Braves vs New York Mets") → 45

Las cadenas son las mismas sin embargo la puntuación es la mitad porqué no tienen el mismo orden. Para solucionar el problema se aplica una ordenación alfabética de los *tokens* para que no influya en la puntuación.

fuzz.token_sort_ratio("New York Mets vs Atlanta Braves", "Atlanta Braves vs New York Mets") → 100

7.7.4.1.3 Los índices de MongoDB

Otra parte del algoritmo es usar los índices de MongoDB para acelerar el rendimiento. Tenemos una colección A de *papers* y queremos hacer la intersección con la otra colección. Una primera aproximación es intentar hacer el producto cartesiano para asegurarnos que hemos explorado todas las soluciones. Pero veremos que esta solución es muy costosa.

- El producto cartesiano tiene un coste en este coste asintótico de $\mathcal{O}(nm)$ donde n y m son el número de elementos de cada colección.
- Por cada documento tenemos que calcular la similitud con las dos funciones un total de m veces.
- El coste del algoritmo de Levenshtein es $\mathcal{O}(kp)$ donde k y p son la longitud de los textos.
- El coste de la ordenación de la función *token_sort_ratio* es menospreciado por el coste más grande que es Levenshtein.

El coste total es $\mathcal{O}(n * (kp * m))$. Es totalmente inviable.

La única parte que podemos optimizar es el producto cartesiano. Hay que buscar una forma de substituirlo por otro método menos costoso. La solución pasa por hacer que MongoDB nos haga parte del trabajo.

Al igual que muchos SGBD, MongoDB también tiene una opción de trabajar con índices. Estos índices les permiten a los SGBD acceder a un registro en tiempo $\mathcal{O}(\log(n))$. Un índice es una estructura arbórea, de tipo *B-Tree* [48] (existen muchas variantes y optimizaciones), que hace que todas las operaciones comunes tengan un coste en el caso peor de $\mathcal{O}(\log(n))$, donde n es la altura del árbol. En espacio tiene un coste de $\mathcal{O}(n)$.

El siguiente paso es crear dos índices sobre el campo del título:

1. Un índice donde el campo título este ordenado ascendiente.
2. Un índice para hacer comparaciones de texto por similitud.

Con estos índices de MongoDB se puede educir el coste del algoritmo considerablemente de la siguiente forma. En vez de buscar comparar un documento contra todos los demás, una mejor opción es pedir que MongoDB usando los índices devuelva los diez registros que según él se parecen más.

La decisión de pedir que devuelva estos diez registros en vez de uno, es porque MongoDB aplica diferentes métodos para buscar la similitud, concretamente aplica TD-IDF [27]. Nosotros queremos aplicar nuestras propias métricas sobre los registros y decidir cual es el más similar en base a unos *threshold*. Consideramos que con los diez más similares es suficiente.

El coste final del algoritmo es $\mathcal{O}(n * kp)$, donde n es el número de registros de la colección a compararnos y k y p son la longitud de los textos respectivamente.

7.7.4.1.4 Algoritmo

Vistas todas las piezas que participan en la solución, este es el algoritmo principal para hacer la fusión:

1. Recopilar todos los *papers* de la colección A.
2. Por cada elemento A' de la colección A:
   a) Pedir a la base de datos los diez *papers* que más se parezcan A'.
   b) Calcular la similitud con los diez documentos.
   c) Guardar como resultado la opción con mayor similitud.
3. Filtrar todos los resultados cuya similitud sea inferior a un *threshold*. Este paso genera el conjunto C.

4. Por cada elemento C' del conjunto C.
    a) Buscar en la base de datos el documento equivalente a C'
    b) Añadir los campos que sean diferentes al documento original.

7.7.4.1.5 Experimentación

Durante el desarrollo se contemplaba usar únicamente una de las dos funciones del paquete. Porque son dos maneras de aplicar el algoritmo de Levenshtein.

Por tanto, ser quería ver cual dos daba mejores resultados para hacer la fusión. Para la realización de las pruebas se ejecutó el anterior algoritmo y los resultados se escribieron en dos ficheros diferentes indicando que función se utilizó. En el momento de la prueba, cada fichero tenía cerca de 35.000 entradas.

Para saber cual de las dos funciones es mejor debemos encontrar para ambas funciones el *threshold* mínimo que determina si aceptamos o no un resultado. El método que hemos seguido para encontrarlo ha sido:

1. Ordenamos los ficheros por porcentaje de similitud descendiente.
2. Fijamos un porcentaje mínimo que consideramos aceptable, en ambos casos empezamos a partir de un 98 % de similitud.
3. Ampliar el intervalo, reduciendo en 1 % la similitud mínima aceptada.
4. Repetir el paso anterior hasta encontrar que el porcentaje de falsos positivos es mayor que un mínimo.

Es importante entrar en detalle en el paso 4, para ello hay que explicar a fondo que son los falsos positivos. Cuando se calcula la similitud de un título con los diez más similares tal y como hace el algoritmo, siempre escogemos como respuesta el que tiene mayor porcentaje de similitud. Pero esto no implica que sea el correcto.

Durante el análisis de los resultados almacenados en los ficheros se observó un comportamiento. Si pudiéramos representar los resultados de las similitudes gráficamente en colores, siendo el verde el 100 % y rojo el 0 %, lo que observaríamos es un degradado de verde a rojo. No existe en ningún momento un espacio en la gráfica donde haya un salto de color de forma brusca.

Un ejemplo de este comportamiento es el siguiente. Supongamos que con una similitud del 88 %, uno de cada diez entradas es incorrecta, si reducimos hasta el 80 % podemos suponer que ahora es dos de cada diez.

Si aceptamos estas entradas como válidas a partir de ahora debemos considerar que existe la posibilidad de tener entradas falsas positivas entre los resultados finales.

Este hecho fuerza a tomar una decisión entre dos opciones.

1. Limitar el intervalo a solo aceptar con seguridad que no exista ningún falso positivo.
2. Asumir que un porcentaje de nuestros datos pueden ser falsos positivos.

Nuestra decisión fue aceptar aproximadamente que como máximo el 10 % de los datos fueran falsos positivos. La razón de esta decisión es poder fusionar el máximo número de registros teniendo constancia de este hecho.

El resultado del estudio para buscar los *thresholds* mínimos es que para la función *Partial Ratio* es >= 96 % y para la función *Token Sort* es >= 88 %.

La última observación más curiosa fue ver que la función *Partial Ratio* puntúa con una similitud muy alta en algunas respuestas a diferencia de la función *Token Sort*. Esto afecta a los títulos que tienen una longitud muy pequeña y además, incluyen caracteres especiales como $, , etc,.

Al ver este hecho se quiso ver cual es el total de entradas que aceptará cada función aplicando el filtro del *threshold*. El resultado fue que había una diferencia de 200 entradas aproximadamente entre ambas. Para no perder estos títulos se tomó la decisión de aceptar ambas funciones para maximizar el número de registros para fusionar.

7.7.4.1.6 Resultados

Los resultados de aplicar esta solución son bastante satisfactorios, ya conseguimos ver que 9.000 de 35.000 registros son comunes entre ambas bases de datos. La siguiente decisión es, ¿qué hacer con los registros restantes?

La decisión que se tomó fue solo mezclar los registros comunes y descartar los demás. La razón principal es porque queremos que la base de datos sea homogénea y añadir elementos todos los elementos implicaría:

- Tal y como se dijo en la experimentación, la similitud actúa como un degradado. Entonces los documentos que se encuentran debajo de los *thresholds* impuestos cabe la posibilidad de tener duplicados. Esta decisión implicaría tener *papers* duplicados y sería perjudicial para el recomendador.
- No tienen conferencias o revistas asociadas. Esto obligaría a crear un elemento fantasma donde todos estos *papers* estén asociados a él.
- Añadirlos implicaría posiblemente añadir nuevos autores, empeorando la situación de ambigüedad de los nombres de autores.

7.7.4.1.7 ¿Existe una solución óptima?

Si en ambas colecciones existiera un identificador como el DOI, está sería la solución óptima. Por el contrario, hemos tenido que averiguar una forma de unirlos sin este campo. Existen diversas formas de solucionar el problema, pero son más complejas y por limitaciones de tiempo no hemos podido implementarlas. Hemos pensado dos propuestas que ayudarían a mejorar la solución actual:

1. La comparación con el título no es suficiente para decidir si son el mismo elemento o no, necesitamos más métricas para decidir si dos documentos son iguales o no. Seguimos sin poder contar con usar los autores por el problema antes mencionado. Una nueva métrica sería acceder a los PDF de ambos documentos y comparar su contenido.

   La similitud entre documentos permitiría poder flexibilizar el *thresholds*. Porque podemos definir pesos sobre las métricas en base al porcentaje de la similitud. Por ejemplo, a partir de un 80 % de similitud, el peso de la métrica de comparar el contenido del PDF es mayor porque nos permite saber con más exactitud si son iguales.

   Esta pequeña mejora ayudaría a poder ampliar el tamaño de la intersección de las colecciones. Para calcular la similitud de dos PDF usaríamos la tecnología de Doc2Vec que está pensada específicamente para está tarea. Está explicada con más detalle en la sección del recomendador.

2. Sabemos que las personas somos capaces de extraer metadatos de un texto y poder relacionar dos textos totalmente diferentes solo por el contexto. En vez de una persona, se podría pensar en usar técnicas de Machine Learning para automatizar la decisión de si dos documentos son similares o no.

7.7.4.2. Extracción de datos de un PDF

Una vez tenemos la base de datos construida la última fase consiste en extraer la información de los PDF asociados a los *papers*. Esta fase es necesaria porque el módulo del recomendador necesita el contenido de los PDF para hacer comparaciones y poder recomendar.

Las secciones que queremos extraer de cada PDF son:

- Abstract.
- Keywords.
- Introducción.
- Desarrollo.
- Conclusiones.

Con el módulo del *Crawler* disponemos de una gran colección de PDF donde extraer información. El objetivo de esta fase es construir un algoritmo que:

1. Acceda a la colección donde se almacenan los PDF.
2. Descargue el contenido del servidor donde se almacena.
3. Seleccionar las secciones descritas anteriormente.
4. Almacenar las secciones al documento pertinente.

7.7.4.2.1 Segmentar el contenido

La parte más compleja de todo este procedimiento es segmentar las secciones del PDF. La primera aproximación fue usar una herramienta muy sencilla que transforma los PDF a texto plano, pero gran parte del contenido se perdía o no era posible identificar las secciones.

Como el anterior método no funciona, debemos pensar como lo haría una persona si lo hiciera a mano:

1. Identificar la cabecera de una sección.
2. Marcar esa posición como el inicio.
3. Seguir el contenido hasta llegar a otra sección o al fin del documento.
4. Marcar esa posición como final de sección.
5. Repetir hasta llegar al final del documento.

Este problema requiere de técnicas de dos áreas muy importantes en la IA: Machine Learning y Visión por Computador. Por suerte para nosotros el problema de extraer contenido de *papers* es común.

El proyecto Grobid [57] fue concebido con esta finalidad. Utiliza técnicas de Machine Learning para extraer las secciones de un PDF y guardar el resultado en un XML. Además, está pensada específicamente para documentos científicos. Grobid no es un sistema infalible y no siempre sabe extraer las secciones de todos los PDF.

7.7.4.2.2 Algoritmo

1. Conectarse a la base de datos.
2. Acceder a la colección donde se almacenan las URL de los PDF y almacenarlas en un diccionario.
3. Por cada elemento del diccionario:
   a) Comprobar que no se haya descargado anteriormente. Para saber de su existencia solo falta comprobar si hay su versión en XML. Para identificar un XML y un PDF se usa el ID del *paper*.
   b) Si ya está descargado saltar al siguiente elemento.
   c) En caso contrario descargarlo.
   d) Generar el XML del PDF usándolo como entrada para el Grobid.
4. Por cada XML almacenado:
   a) Recorrer el XML identificando las secciones.
   b) Almacenar el contenido de las secciones en variables.
   c) Aplicar un postprocesado sobre los datos de las variables. Este paso es muy importante para el Doc2Vec del módulo del recomendador. Porque necesita que los textos estén libres de caracteres o palabras no adecuadas. El proceso consiste en borrar:
      - Etiquetas web.
      - Caracteres especiales, como el salto de linea, &, #, $, %, etc.
      - Paréntesis, llaves y corchetes que no tengan texto entre ellos.
      - Borrar el exceso de espacios entre las palabras.
      - Dígitos.
      - Concatenaciones de palabras mediante '- ', por ejemplo: desc- argar.
      - Signos de puntuación.
      - Eliminar formulas, figuras, tablas y bibliografía.
   d) Si después del postprocesado ninguna de las variables tiene texto se considera que ese XML no es válido. Este problema sucede mayoritariamente porque Grobid no ha sido capaz de identificar las secciones del PDF.
   e) En caso contrario se considera éxito y hay que añadir las secciones al documento donde esta el *paper*.

7.7.4.2.3 Optimizaciones

El algoritmo tiene dos inconvenientes: espacio y tiempo.

El coste en espacio es muy alto. Cuando se ejecutó el algoritmo teníamos una colección de 110.000 PDF. El peso medio aproximadamente de cada uno es de 7 Mb, esto hace un total de 770.000 Mb que son 752 Gb a cambio.

No se tienen en cuenta los XML generados en el total porque tienen un peso medio de 30 Kb. Por limitaciones de espacio de disco en el servidor no podíamos descargar y almacenar todos los PDF.

La optimización consiste en eliminar los PDF una vez se ha generado el XML. Este cambio hizo que el coste en espacio pasase a ser aproximadamente de tan solo 3 Gb.

El segundo problema es el tiempo que se tardaría en procesar 110.000 PDF. Para calcular cuanto tiempo nos llevaría procesar todos los datos, se hizo una prueba que durante un día entero procesase el mayor número posible. El resultado fue que con el algoritmo actual se tardaría más de un mes.

Esto se debe a dos factores:

1. La velocidad de descarga que permiten los servidores donde están alojados los PDF es muy baja, cerca de 200 kbps.
2. El programa es secuencial, por tanto, hasta que no termina de descargar no ejecuta el siguiente paso.

La optimización que se pensó fue permitir paralelización del proceso:

1. Una vez se han copiado de la colección todas las URL, dividirlas en bloques iguales y distribuirlos entre todos los hilos de la CPU. El número va especificado como entrada en el programa.
2. Hacer que tanto la conversión a XML como la descarga del PDF sean procesos independientes. Descargar sin pausa todos los documentos y un proceso externo que avise al Grobid cuando tiene un nuevo PDF que transformar.

Este cambio ayudo que el proceso solo tardase tres días ejecutándose en un procesador con 8 hilos.

7.7.4.2.4 Problemas y soluciones

Durante el proceso nos hemos encontrado con diferentes problemas, pero los más destacables son estos dos:

1. Servidores que bloquean peticiones de descarga de PDF.
2. Recorrer el árbol XML.

En el momento que como usuarios a través del navegador solicitamos descargar un PDF a través de un enlace publicado en un sitio web, automáticamente se nos redirecciona a una nueva URL. Esta URL es una petición al servidor de que queremos descargar o visualizar el documento.

Este mecanismo genera un problema a la hora de automatizar la descarga. Necesitamos saber que partes se tienen que modificar en la URL original para adaptarla a una petición que entienda el servidor. Como cada sitio web aplica genera una petición al servidor diferente, nos obliga a tener que ir por cada sitio web y buscar cual es la modificación.

Por ejemplo, queremos descargar este documento de la IEEE:

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8370631

Si utilizamos los comandos de Linux como WGET y CURL sobre ella, el servidor de la IEEE no es capaz de responder porque no identifica la URL como una petición. En todas las URL hay que modificar alguna palabra para forzar la petición. En este caso encontramos una pequeña ayuda en una publicación hecha en un blog de como descargar mediante WGET sobre el sistema STAMP[64]. La modificación consiste en hacer una petición GET sobre STAMP, dando como resultado:

https://ieeexplore.ieee.org/stampPDF/getPDF.jsp?arnumber=8370631

El segundo problema es recorrer los árboles XML generados. Esto se debe a que no todos los *papers* tienen la misma estructura ni tampoco usan los mismos nombres para identificar las secciones.

Para ver el problema más fácilmente un ejemplo. Para extraer las conclusiones hay autores que ponen como título de sección: conclusions, concluding, conclusion, summary, etc. Este hecho se da también en la introducción y abstract.

La solución al problema pasa por crear diferentes conjuntos de palabras clave que puedan identificar una sección. Cuando recorremos el árbol comparamos las etiquetas con el conjunto para saber a que sección pertenece.

7.7.4.2.5 Resultados

Los resultados del proceso han sido muy satisfactorios. De los 110.000 PDF hemos podido extraer la información de 60.000. Los PDF restantes no se han podido por una cuestión de tiempo, por dos razones:

1. Necesitamos arreglar las peticiones de descarga de PDF de varios sitios web.
2. Como ya hemos dicho, Grobid no es un sistema infalible, así que hay un conjunto pequeño de PDF que no es capaz de transformar a XML.

Con este proceso hemos incrementado en gran medida el potencial del recomendador porque ahora hay un gran número de *papers* que tiene: abstract, introducción, desarrollo, keywords, conclusión. Esta estructura facilitará la recomendación al poder dar distinta importancia a distintas partes del documento.

Muy útiles para que el sistema recomendador pueda recomendar por similitud de estas secciones o por keywords.

7.8. Futuro

Estos son los cambios que querríamos hacer en un futuro para mejorar el módulo:

- Hacer que se puedan descargar todos los PDF de los sitios web restantes.
- Aplicar métodos de Machine Learning para resolver el problema de ambigüedad de nombres.
- Extraer metadatos como las ubicaciones de las conferencias.
- Trabajar con nuevas métricas a la hora de hacer la fusión de bases de datos.
- Automatizar el proceso de crear las relaciones entre documentos para cualquier tipo de base de datos.
- Mejorar la segmentación de los documentos para detectar secciones más específicas.

Capítulo 8

Módulo: Graph Builder

8.1. Objetivo

La función es transformar los datos almacenados en el módulo de base de datos en un grafo y almacenar el resultado en una base de datos orientada a grafos (BDOG) [50].

8.2. Introducción

Es el módulo donde menos decisiones se han tenido que tomar. Su única funcionalidad es la que está descrita como objetivo,

actúa como puente entre los dos sistemas de bases de datos: BDOG y MongoDB. Lo que se hará más hincapié qué criterios se siguieron para diseñar el grafo y que tecnologías hemos usado y qué ventajas nos aportan.

8.3. Especificación

8.3.1. Entrada

La entrada para el fichero: *mongo_neo4j_driver*

- *host* : La URI para conectarse a la base de datos.
- *port* : Puerto para conectarse a la base de datos.
- *database_input* : La base de datos donde se almacenan las colecciones.

8.3.2. Salida

La salida del módulo son ficheros con extensión CSV. Se clasifican en dos tipos según su contenido:

1. Información que describe un vértice.
2. Las aristas. Pueden tener información asociada.

Para representar los vértices:

authors.csv	
Contenido	MongoDB Id - Name

Tabla 8.1: Contenido del fichero authors.csv

categories.csv	
Contenido	MongoDB Id - Name - Code

Tabla 8.2: Contenido del fichero categories.csv

conferences.csv	
Contenido	MongoDB Id - Name - Year

Tabla 8.3: Contenido del fichero conferences.csv

editions.csv	
Contenido	MongoDB Id - Name - Code Name - Year

Tabla 8.4: Contenido del fichero editions.csv

journals.csv	
Contenido	MongoDB Id - Name - Year - Volume - Number

Tabla 8.5: Contenido del fichero journals.csv

papers.csv	
Contenido	MongoDB Id - Title - Year

Tabla 8.6: Contenido del fichero papers.csv

Para representar las aristas:

author_paper.csv	
Contenido	Author MongoDB Id - Paper MongoDB Id - Year

Tabla 8.7: Contenido del fichero author_paper.csv

conference_edition.csv	
Contenido	Conference MongoDB Id - Edition MongoDB Id - Year

Tabla 8.8: Contenido del fichero conference_edition.csv

paper_category.csv	
Contenido	Paper MongoDB Id - Category MongoDB Id - Year

Tabla 8.9: Contenido del fichero paper_category.csv

paper_conference.csv	
Contenido	Paper MongoDB Id - Conference MongoDB Id - Year

Tabla 8.10: Contenido del fichero paper_conference.csv

paper_journal.csv	
Contenido	Paper MongoDB Id - Journal MongoDB Id - Year

Tabla 8.11: Contenido del fichero paper_journal.csv

8.4. Diseño Inicial

Crear un programa cliente para generar grafos a partir de los datos almacenados usando un *driver* que permita la comunicación con el módulo de base de datos. A través del programa se define el criterio con el que el sistema relacionara los elementos para generar un grafo. Todos los grafos tienen como base está estructura:

Sea $G(v, u)$ un grafo no dirigido, v el conjunto de vértices que representa un recursos: paper, autor, revista o conferencia y u las aristas que relacionan los recursos.

Un ejemplo de criterio, podría ser generar el grafo de coautoría. Los grafos generados se almacenan en ficheros.

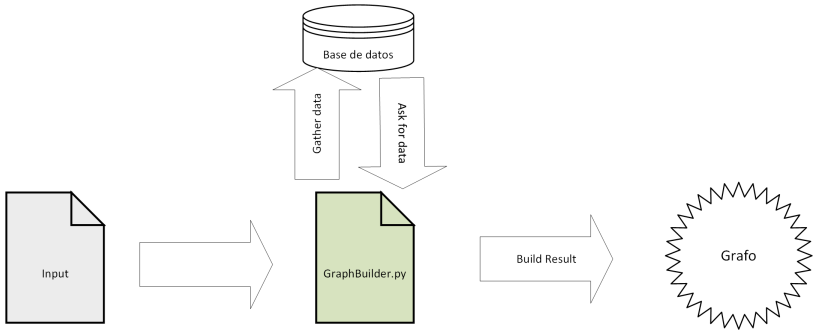

Figura 8.1: Diseño inicial del graph builder

8.5. Diseño Final

El planteamiento respecto al diseño inicial es completamente diferente. La posibilidad de generar grafos indistintamente del criterio tiene una complejidad elevada porque requiere comprobar que se pueda construir y demostrar que los resultados generados son válidos.

Durante el desarrollo se descubrió que existen ya tecnologías pensadas para satisfacer nuestra necesidad y son bases de datos orientadas a grafos.

Usar estos sistemas nos ofrece muchas ventajas, pero la principal es poder construir un grafo y generar subgrafos a partir de él. A diferencia del diseño inicial, todos los grafos son dirigidos.

Esta nuevas tecnologías nos permiten satisfacer la necesidad de crear grafos bajo un criterio tal y como está especificado en el diseño inicial. Otra ventaja implícita es no trabajar con ficheros.

El diseño del grafo es el siguiente:

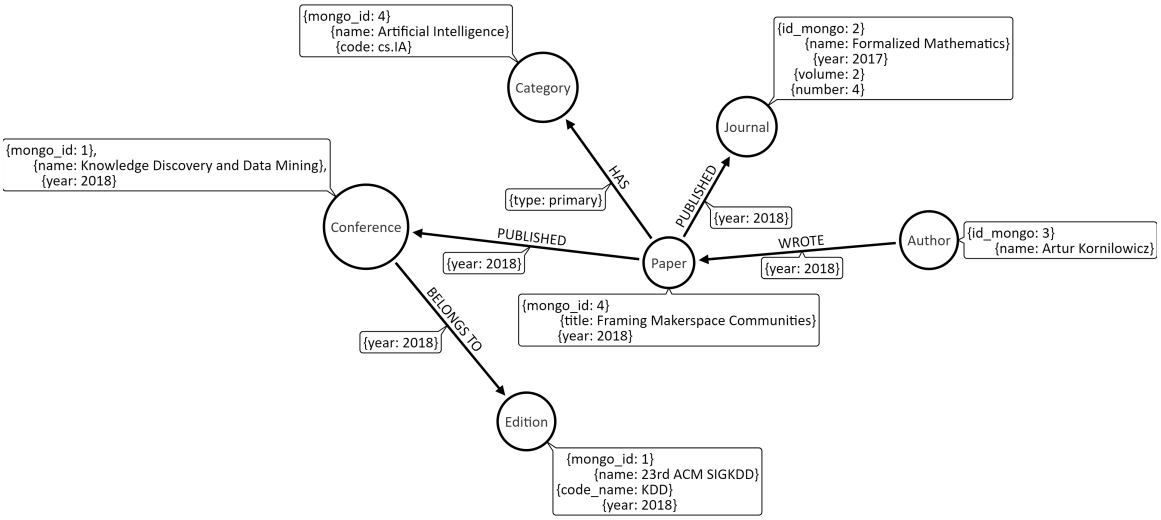

Figura 8.2: Diseño inicial de la base de datos

Criterios del diseño:

- Un *paper* puede tener más de una categoría, pero siempre tendrá una principal. La relación entre *paper* y *category* tiene un atributo type que puede tomar valores: primary o secondary.
- un *paper* puede ser publicado en una revista en un año y más tarde ser presentado en una conferencia.
- Las conferencias pertenecen a una única edición. Pero en una edición se pueden celebrar varias conferencias.

8.6. Tecnologías

- *Cypher* : Es el lenguaje con sintaxis básica pero con mucha potencia de expresión con el que podemos interactuar con Neo4J y permite actualizar, insertar, borrar y consultar datos de un grafo. Es un lenguaje declarativo. Está licenciado bajo GPLv3 y AGPLv3.
- *Pymongo* : Es el driver que nos permite comunicarnos con la base de datos MongoDB. Es de código abierto bajo licencia MIT. Compatibilidad con Python 2.7 y Python 3.6.
- *Neo4j* : Neo4J es un programa de software libre pensado para ser una base de datos orientada a grafos. Está licenciado bajo GPLv3 y AGPLv3.

8.7. Implementación

Para la construcción del grafo usaremos el software Neo4J. El proceso consta de tres fases:

1. Generar los CSV donde almacenar la información del grafo.
2. Cargar los ficheros en Neo4J
3. Crear el grafo mediante un *script* Cypher.

8.7.1. Neo4J

Antes de explicar todo el proceso de crear el grafo es importante introducir qué es y como trabaja Neo4J. Neo4J es un programa de software libre y es una base de datos orientada a grafos.

Tiene las mismas funcionalidades que un SGBD común como MariaDB. Podemos: insertar, actualizar, borrar y consultar grafos. El potencial de este software es poder aplicar los algoritmos de grafos que conocemos.

Los grafos están compuestos por nodos y relaciones y ambos pueden tener propiedades:

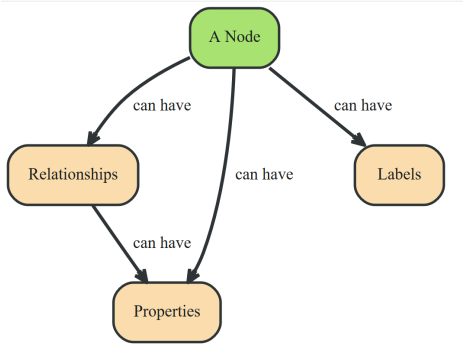

Figura 8.3: Elementos que componen un grafo en Neo4J

Nuestro objetivo es transformar los datos en un grafo. Por tanto, necesitamos un lenguaje con una sintaxis sencilla pero que nos permita tener mucha potencia de expresión. Para esta ocasión los desarrolladores crearon el lenguaje *Cypher*.

Para ver el potencial y como funciona, introduciremos un ejemplo. Supongamos que queremos modelar este enunciado:

Dos personas, John y Sally son amigos. Ambos han leído un libro titulado Graph Databases.

Aplicando las definiciones de la Figura 8.3, podemos representar el anterior enunciado de está forma:

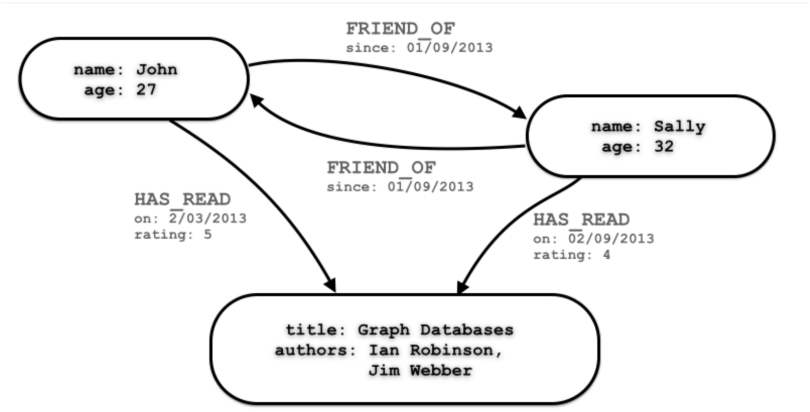

Figura 8.4: Representación en grafo del enunciado

Una vez tenemos un grafo es ahora cuando podemos ver el auténtico potencial de *Cypher* haciendo preguntas como:

1. ¿Desde cuándo John y Sally son amigos? 01/09/2013

```
MATCH (sally:Person { name: 'Sally' })
MATCH (john:Person { name: 'John' })
MATCH (sally)-[r:FRIEND_OF]-(john)
RETURN r.since AS friends_since
```

Figura 8.5: Ejemplo 1 de pregunta con *Cypher*

2. ¿Quien leyó primero el libro, Sally o John? John

```
MATCH (people:Person)
WHERE people.name = 'John' OR people.name = 'Sally'
MATCH (people)-[r:HAS_READ]->(gdb:Book { title: 'Graph Databases' })
RETURN people.name AS first_reader
ORDER BY r.on
LIMIT 1
```

Figura 8.6: Ejemplo 2 de pregunta con *Cypher*

8.7.2. Generar los CSV

Neo4J admite diferentes métodos para introducir datos, pero la más sencilla y recomendada por los desarrolladores es usar ficheros CSV. Cada fichero representa una entidad del grafo: nodo o relación.

El procedimiento para generar los CSV es:

1. Conectarse a la base de datos con los argumentos recibidos.
2. Copiar las colecciones: autores, *papers*, revistas, conferencias y categorías en diccionarios. Usar como clave el id de MongoDB.
3. Generar los nodos autores.
4. Generar las relaciones entre autores y *papers*.
5. Generar los nodos ediciones.
6. Generar los nodos conferencias.
7. Generar las relaciones entre conferencias y ediciones.
8. Generar los nodos categorías.
9. Generar los nodos *papers*.
10. Generar las relaciones entre categorías y *papers*.
11. Generar las relaciones entre conferencias y *papers*.
12. Generar los nodos revistas.
13. Generar las relaciones entre revistas y *papers*.

Una vez se ha generado todos los ficheros, los desarrolladores de Neo4J aconsejan importarlos directamente a la aplicación para que a la hora de generar el grafo sea más rápido.

8.7.3. Generar el grafo

Para generar el grafo debemos usar el lenguaje *Cypher*. Es importante mantener un orden a la hora de construirlo:

1. Crear los nodos.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///authors.csv" AS row
FIELDTERMINATOR ';'
CREATE (:Author {mongo_id: row.mongo_id, name: row.name});
```

Figura 8.7: Ejemplo de creación de un nodo

2. Crear los índices para los identificadores Mongo Id y nombre de un nodo porque reduce considerablemente el tiempo de búsqueda.

```
CREATE INDEX ON :Author(mongo_id);
CREATE INDEX ON :Author(name);
```

Figura 8.8: Ejemplo de creación de un índice

3. Crear las relaciones.

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM "file:///author_paper.csv" AS row
FIELDTERMINATOR ';'
MATCH (author:Author {mongo_id: row.mongo_id_author})
MATCH (paper:Paper {mongo_id: row.mongo_id_paper})
MERGE (author)-[wrote:WROTE]->(paper)
SET wrote.year = toInt(row.year);
```

Figura 8.9: Ejemplo de creación de una relación

8.8. Futuro

- Automatizar el proceso de carga de datos a Neo4J.

Capítulo 9

Módulo: Graph Analyzer

9.1. Objetivo

Extraer los metadatos del grafo generado por el módulo *Graph Builder* que son necesarios por el módulo *recomendador* para mejorar la experiencia de usuario.

9.2. Introducción

Es el modulo mas simple de todos ya que la implementación de los algoritmos ya existe en el motor de base de datos que hemos utilizado. Para ejecutar los algoritmos hemos usado el lenguaje de consultas que el propio SGBD proporciona para ello.

9.3. Especificación

El módulo no requiere de especificación porque no tiene entrada ni salida. El resultado se guarda en la propia base de datos.

En el caso del pagerank añade un campo pagerank a los nodos que lo tengan calculado.

En el caso de la comunidad añade un campo con el id de la comunidad al que pertenece el nodo.

9.4. Diseño inicial y final

Recibe como entrada un grafo generado por el módulo *Graph Builder* y se le aplica un algoritmo para extraer metadatos, por ejemplo: calcular el *pagerank*. El resultado es sobreescrito sobre el mismo grafo.

9.5. Tecnologías

- *Cypher* : Es el lenguaje con sintaxis básica pero con mucha potencia de expresión con el que podemos interactuar con Neo4J y permite actualizar, insertar, borrar y consultar datos de un grafo. Es un lenguaje declarativo. Está licenciado bajo GPLv3 y AGPLv3.
- *Neo4j* : Neo4J es un programa de software libre pensado para ser una base de datos orientada a grafos. Está licenciado bajo GPLv3 y AGPLv3.

9.6. Implementación

9.6.1. PageRank

El PageRank [24] originalmente se pensó para medir la importancia de una página web en internet. Funciona contando el número y la calidad de enlaces a una página para determinar aproximadamente como de importante ese sitio es. Se asume que los sitios web más importantes son más propensos a recibir más enlaces de otros sitios, especialmente enlaces que a su vez son de calidad.

El PageRank es el resultado de un algoritmo matemático utilizado sobre el grafo que forman las páginas webs de todo internet al enlazarse unas a otras, siendo cada página un nodo y cada enlace entre dos paginas un arco. La puntuación resultante indica la importancia de una página en concreto. Un hiperenlace a una página cuenta como un voto de soporte. El PageRank de una página es definido recursivamente y depende del PageRank de los nodos que enlencen a esa página. Una página que es enlazada por paginas con PageRank alto recibe un PageRank.

En nuestro caso concreto usaremos el PageRank para determinar la importancia de un autor dentro del grafo de coautoría generado por las relaciones de si un autor ha trabajado con otro en un mismo paper. Son relaciones no dirigidas, así que será como si se hiciera un arco en cada dirección.

Para calcular el PageRank sobre el grafo de coautorias en nuestra base de datos usamos la consulta mostrada en la figura 9.1

```
CALL algo.pageRank('MATCH (a:Author) RETURN id(a) as id',
                   'MATCH (a1:Author)-[:WROTE]->(p:Paper)<-[:WROTE]-(a2:Author)
                   RETURN id(a1) as source, id(a2) as target',
                   {graph:'cypher', iterations:5, write:true});
```

Figura 9.1: Codigo *Cypher* necesario para calcular el PageRank sobre el grafo generado de coautoría.

Un ejemplo del PageRank aplicado a un grafo de ejemplo puede verse en la figura 9.2

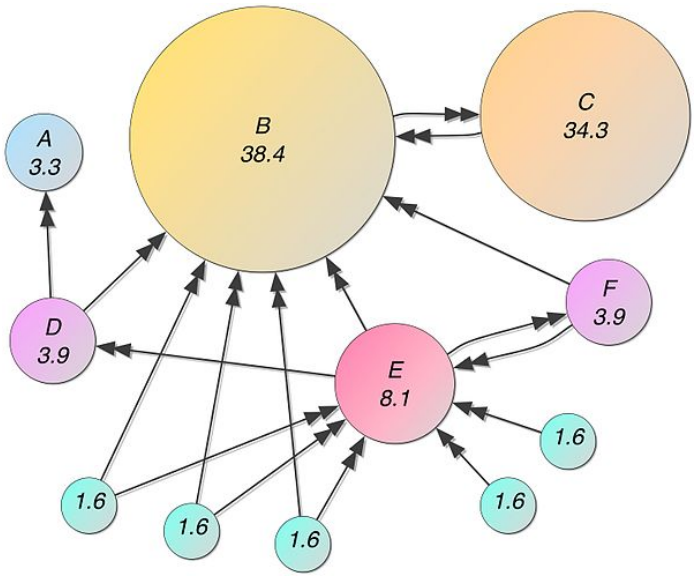

Figura 9.2: Ejemplo de PageRank aplicado sobre un grafo. Fuente: [24]

9.6.2. Comunidades

En el estudio de redes complejas, se dice que una red tiene estructura de comunidad si los nodos de la red pueden ser fácilmente agrupables en conjuntos de nodos donde cada conjunto de nodos está densamente conectado internamente. En el caso de querer encontrar las comunidades que no se *sobreponen* implica que el grafo se divide naturalmente en grupos de nodos con una conexión densa internamente y escasa entre grupos.

La definición más general basada en el principio de pares de nodos es más posible estar conectado si los dos miembros son de la misma comunidad y menos posible de estar conectado si ellos no comparten comunidad.

En el caso que nos ocupa queremos encontrar las comunidades que forman los autores de los *papers* cuando colaboran para elaborar uno, utilizando para ello el grafo de coautoría.

Decidimos usar para ello el algoritmo de Louvain [6], como explicamos en el estado del arte, es un algoritmo que se puede paralelizar y funciona muy bien con redes con muchos nodos y conexiones.

```
CALL algo.louvain('MATCH (a:Author) RETURN id(a) as id',
                  'MATCH (a1:Author)-[:WROTE]->(p:Paper)<-[:WROTE]-(a2:Author)
                  RETURN id(a1) as source, id(a2) as target',
                  {graph:'cypher', iterations:5, write:true});
```

Figura 9.3: Codigo *Cypher* necesario para detectar las comunidades sobre el grafo generado de coautoría.

Por limitaciones del visualizador de la base de datos Neo4j no podemos visualizar las comunidades obtenidas por la query anterior sin recurrir a otro software. Mostramos un ejemplo de un grafo con comunidades en la figura 9.4.

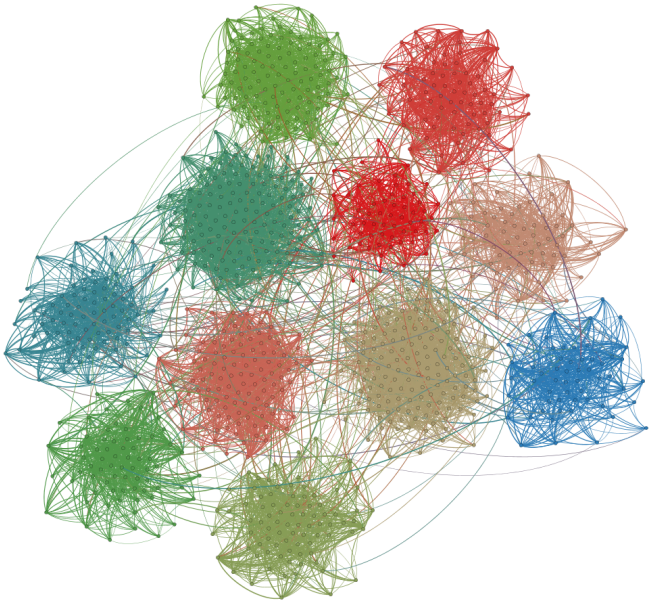

Figura 9.4: Ejemplo de visualización de comunidades en un grafo. Fuente: [55]

9.7. Futuro

- Considerar otros subgrafos a analizar:
  - Citaciones de los papers: los nodos serian papers y existiría un arco entre dos papers si uno cita a otro (dirigido).
  - Conferencias: Dos conferencias están unidas por un arco si un autor ha ido a las dos.
  - Journals: Dos journals están unidos por un arco si un autor ha publicado en las dos.
- Utilizar el algoritmo de comunidades para detectar clusters de papers usando el grafo de citaciones.
- Aplicar el algoritmo de pagerank sobre los subgrafos anteriores para puntuar los elementos que conforman los subgrafos.
- Calcular otras posibles métricas para los grafos como *betweeneess*, *edge betweeneess*, *closeness*.

Parte III

Recomendador

La parte recomendador esta pensada para ser una aplicación hecha con arquitectura MVC (Modelo vista controlador) [51]. El módulo de recomendación y la API REST forman el controlador, mientras que el módulo modelo actúa como modelo y el cliente web como la vista. Se puede ver en la figura 9.5.

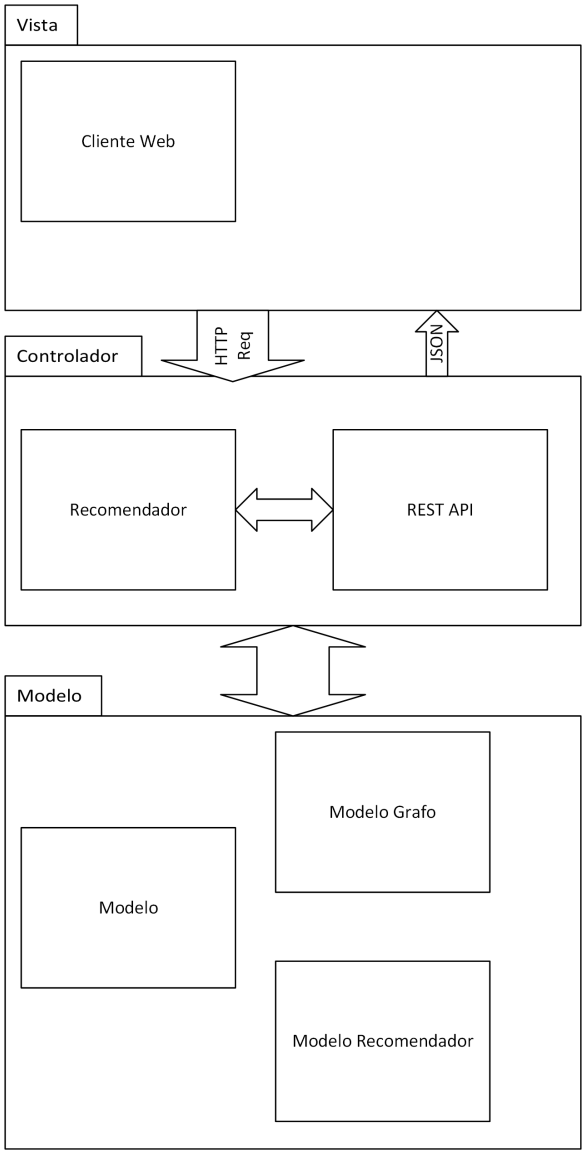

Figura 9.5: Visión General de la arquitectura MVC del recomendador.

Capítulo 10

Módulo: Recomendador

Este módulo es parte del controlador dentro del esquema MVC (Modelo Vista Controllador) [51] que sigue todo el sistema como se puede ver en la figura 9.5.

10.1. Introducción

Para poder realizar las recomendaciones vamos a usar las técnicas explicadas a continuación. Algunas de ellas son introductorias para facilitar el entendimiento de las siguientes.

10.1.1. Similitud entre papers

El problema que tenemos que afrontar es que para un paper dado necesitamos que el sistema nos de los n más parecidos a este según un criterio. Una posible forma de hacerlo es representando los documentos como vectores y calculando la distancia entre ellos, Para ello valoramos principalmente dos opciones: Bag of Words (BOW) junto TF-IDF (Term Frequency - Inverse Document Frequency) y Paragraph2Vec [65] también conocido como Doc2Vec. Como hemos explicado parcialmente en el estado del arte, Paragraph2Vec es un framework que no necesita ser supervisado que aprende el significado semántico de las palabras y las representa como vectores. Además, a diferencia de Word2Vec también representa los documentos como vectores, lo que nos permite calcular una distancia entre esos dos documentos, por ejemplo, la distancia coseno. Al disponer de una distancia también podemos agruparlos utilizando técnicas de clustering y detectar grupos que hablen del mismo tema.

10.1.1.1. Bag of Words

Bag of Words (BOW) [47] es un método para representar un documento como un vector. Cada documento es un subconjunto de palabras del conjunto total de palabras contenidas en todos los documentos (corpus). Para representar cada documento se necesita un vector de dimensión p, donde p es el número de palabras distintas contenidas en todos los documentos. En un corpus tan grande p puede llegar a tener un valor superior a 70.000, además, este vector puede a llegar ser muy disperso si el documento es de pocas palabras o con muchas repetidas.

El vector representa en cada componente una palabra del corpus, el valor de cada componente cuenta cuantas veces ha aparecido esa palabra en el documento. Ejemplo de un vector:

```
[3,0,...,0,1,2,,0,...,0,1,...]
```

Para poder hacer la comparación entre estos vectores hay que normalizarlos, una buena forma de

normalizarlos es usando TF-IDF (Term Frequency - Inverse Document Frequency) [27]. TF-IDF es una medida numérica que expresa cuan relevante es una palabra para un documento en un corpus. El valor tf-idf aumenta proporcionalmente al número de veces que una palabra aparece en el documento, pero es compensada por la frecuencia de la palabra en la colección de documentos, lo que permite compensar el hecho de que algunas palabras son generalmente más comunes que otras.

Para reducir el número de palabras y mejorar la similitud cuando se usa TF-IDF se eliminan las stopwords de los documentos y se aplica lematización [15] a las palabras que consiste en, dada una forma flexionada (es decir, en plural, en femenino, conjugada, etc), hallar el lema correspondiente. El lema es la forma que por convenio se acepta como representante de todas las formas flexionadas de una misma palabra.

Al normalizar dos documentos del mismo corpus con esta técnica, se puede calcular el ángulo que forman los vectores de estos documentos y calcular así la distancia coseno que evalúa en un rango de 0 a 1 la similitud de los documentos, siendo 1 si son iguales o, 0 si son totalmente diferentes.

Está técnica ha dado muy buenos resultados y todavía hoy en día es muy usada, pero, este método tiene como principal desventaja que ignora el orden de como aparecen las palabras en el texto o las que esta tiene alrededor. Paragraph2Vec, también conocido como Doc2Vec, sí que lo tiene en cuenta y constituyó un nuevo estado del arte en 2013. Por eso es el que hemos decidido usar para nuestro proyecto.

10.1.1.2. Word2Vec

Doc2Vec, que está fuertemente basado en Word2Vec, representa los documentos como vectores. Para entender como funciona Doc2Vec primero hay que entender como funciona Word2Vec.

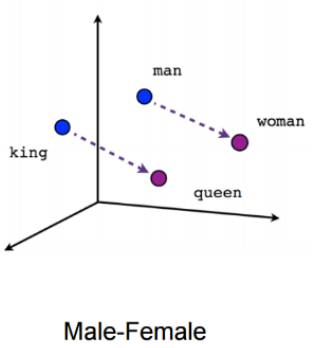

Figura 10.1: Ilustración de como Word2Vec representa las palabras como vectores, se puede ver como las palabras king y man están en una misma coordenada en el espacio y queen y woman en otra, idealmente la resta de vectores king - man = queen. Fuente: [70]

La representación de Word2vec es creada usando dos algoritmos Continous Bag-of-Words model (CBOW) y el modelo Skip-Gram.

10.1.1.3. Continous Bag of Words

Crea una ventana alrededor de la palabra actual, para predecirla a partir del contexto de las palabras que la rodean. Cada palabra se representa como un vector de características, después del entreno estos vectores se convierten en los vectores de palabras. Esquema del funcionamiento en la figura 10.2.

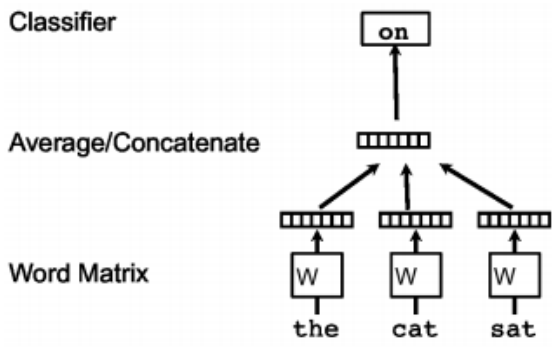

Figura 10.2: Esquema del Algoritmo CBOW : Las palabras “the” “cat” “sat” se usan para predecir “on”. Fuente: [65]

10.1.1.4. Skip gram

Es el algoritmo contrario a CBOW, en vez de predecir una palabra, se usa una palabra para predecir todas las de su alrededor, es mucho más lento que CBOW, pero se considera más preciso con palabras poco frecuentes.

10.1.1.5. Doc2Vec

El objetivo de Doc2Vec es crear una representación numérica de un documento independientemente de su longitud. Para ello Doc2Vec usa el mismo modelo que Word2Vec pero añadiendo otro vector con todos los identificadores de los documentos, de esta manera el algoritmo CBOW queda como se ve en la figura 10.3

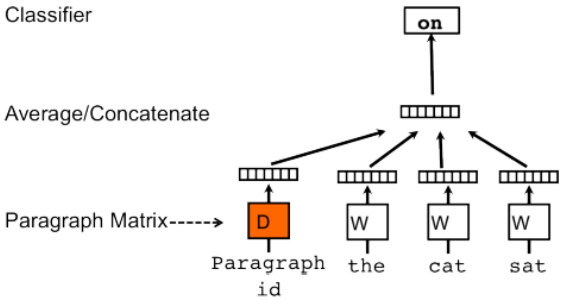

Figura 10.3: Esquema algoritmo CBOW adaptado a documentos. Fuente: [65]

Así que cuando se entrenan los vectores W también se entrena el vector D, al final del entrenamiento del modelo contienen una representación del documento.

La versión del algoritmo Skip Gram para Doc2Vec también es similar como puede verse en la figura 10.4

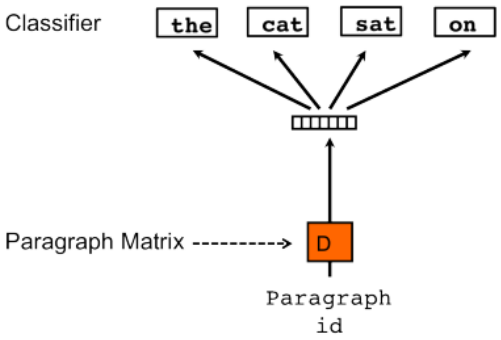

Figura 10.4: Distributed Bag of Words version of Paragraph Vector (PV-DBOW). Fuente: [65]

La explicación completa de como funcionan estos algoritmos está en los papers citados de este tema.

10.1.2. Pagerank

Una buena forma de ver la importancia de un nodo en un grafo es utilizando el algoritmo de Pagerank [1]. En concreto lo usaremos sobre el subgrafo de coautoría, generado por el módulo Graph Analyzer, para darle un peso a los autores y así puntuar los papers según el peso de estos.

10.1.3. Shortest Path

Calcular el shortest path entre dos papers sobre el grafo generado por el módulo graph builder (seccion 8.7.1) nos permite saber si estos están cerca por alguna colaboración entre alguna comunidad de autores o incluso si lo ha escrito el mismo autor si la distancia es 2.

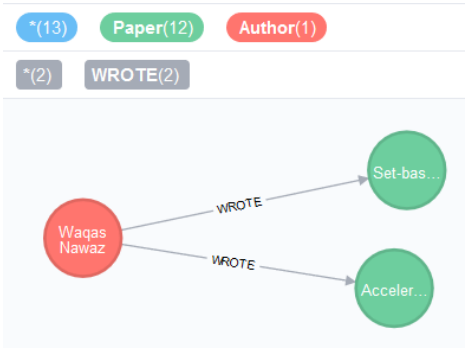

Figura 10.5: Ejemplo de distancia 2 en el algoritmo shortest path (ignorando la dirección del arco).

Para ejecutar este algoritmo correctamente solo podemos usar las relaciones (arcos) del tipo *:WROTE* que relacionan si un autor ha escrito un paper, si no podríamos obtener falsos resultados con caminos más cortos, por ejemplo, usando la relación *:HAS* con papers que comparten la misma categoría obtendríamos siempre distancia dos. Se ignora que las relaciones de tipo *:WROTE* son dirigidas *(autor)-[:WROTE]-(paper)*, para poder llegar también de un paper a un autor.

En la figura 10.5 se puede ver una parte de la base de datos sobre la cual se ejecuta el algoritmo de shortest path entre dos de estos nodos y en la figura 10.7 el resultado.

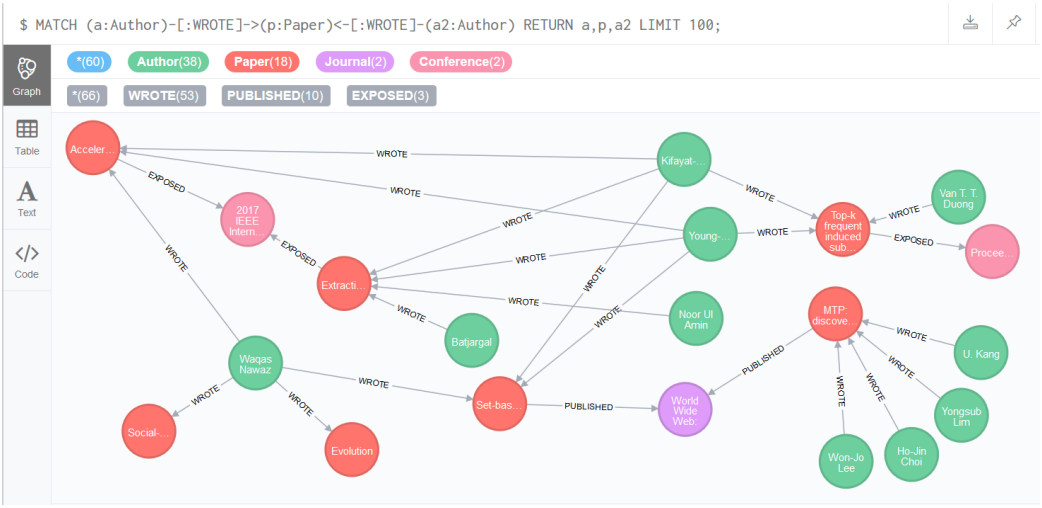

Figura 10.6: Pequeño fragmento de la base de datos representada como grafo

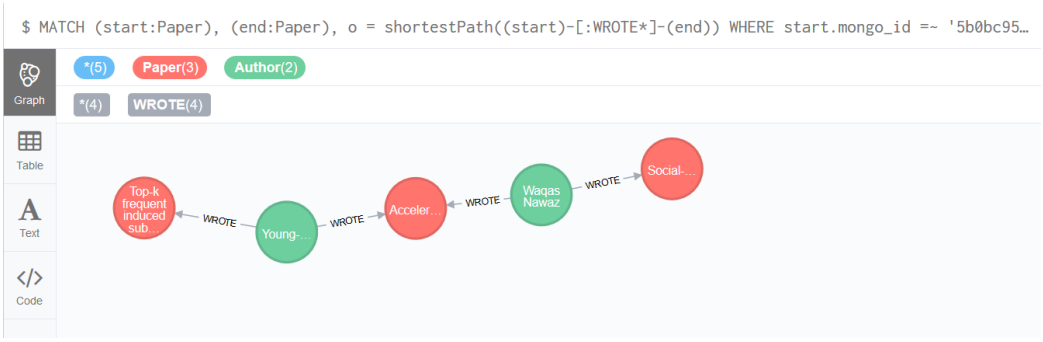

Figura 10.7: Ejemplo de camino encontrado por el algoritmo de shortest path sobre el grafo de ejemplo de la figura 10.5 entre dos papers usando solo la relación :WROTE.

10.1.4. Clustering

Una vez que podemos representar los documentos como vectores, estos también representan puntos en el espacio, así que podemos formar grupos con las nubes de puntos que más cerca estén unos de otros. Para ello valoramos usar varios algoritmos [68], basados en medidas geométricas de distancia, como por ejemplo, DBScan, Mean-Shift, K-means o su generalización Gaussian mixture.

Para evaluar la calidad de los grupos formados por estos existen diferentes métricas que veremos más adelante dependiendo de los algoritmos usados.

10.1.5. Comunidades

Una técnica más que tenemos en consideración son las comunidades formadas por los autores en el grafo social generado por el módulo Graph Analyzer del capítulo 9.

10.2. Especificación

Recomendar papers al usuario, para ello el sistema tendrá en cuenta:

- La similitud del contenido de estos.
- Las valoraciones que el usuario haya hecho sobre los papers (me gusta/no me gusta).
- El historial de papers visitado en el sitio.
- El PageRank de los autores.
- La distancia en el grafo a la que se encuentra un paper de otro.
- Si están en el mismo cluster.

Requisito funcional es que el siguiente grafo exista: *Autores relacionados por papers*: Los vértices representan los autores, y existirá un arco entre dos autores, si han escrito un paper en común. Este grafo permite encontrar los autores que más colaboraciones tienen, y encontrar las comunidades de autores.

10.3. Diseño

El diseño del módulo recomendador es complejo porque está relacionado con casi todas las otras partes del sistema. A continuación, desglosamos las partes que lo componen.

- Generador Corpus
- Corpus (Colección de documentos).
- Entrenador Doc2Vec.
- Modelo Doc2Vec.
- Generador Clusters.
- Descriptor Clusters.
- Calidad Clusters.
- Interfaz final.

10.3.1. Generación del corpus

Para la generación del corpus se utiliza el contenido de los papers extraído y preprocesado por el módulo de bases de datos (sección 7.7.4.2), este contenido consta de abstract, introducción, cuerpo, conclusión. Si alguna de las partes no está disponible se ignora, pero, como mínimo tiene que existir una. para que ese paper pase a formar parte del corpus, si no, no se pueden hacer recomendaciones basadas en la similitud, que es el componente sobre el que hemos basado el recomendador.

Todo el contenido se filtra con la expresión regular: [A-Za-z0-9]+, eliminando así todos los símbolos extraños, puntuación, etc... Esto se hace pensando en el entreno de Doc2Vec con este corpus. Para un modelo diferente podríamos haber necesitado un filtrado diferente.

10.3.1.1. Formato final del corpus

Lista de tuplas donde el primer componente de la tupla es el identificador dentro de la base de datos y el segundo componente es el contenido.

Representación: [(id, contenido_filtrado),...,(id,contenido_filtrado)]

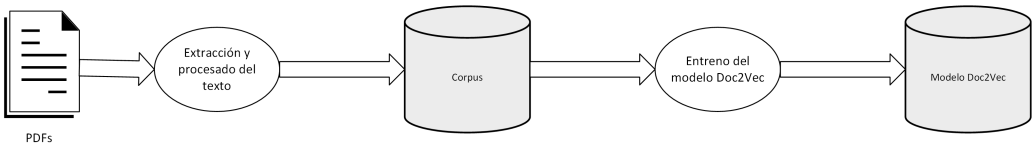

Figura 10.8: Esquema del proceso que se sigue desde que se obtienen los PDFs hasta que se consigue el modelo entrenado.

10.3.2. Puntuación

Hay muchas funciones posibles para combinar estos distintos factores. Para empezar, hemos probado esta:

Queremos que el coeficiente de similitud sea la valoración principal sobre la que basar la recomendación y el resto de factores la modifiquen positiva o negativamente dependiendo de como ayuden.

$Puntuacion = S * K * U * \frac{1}{G} * C$

- S Es el coeficiente de similitud con un rango entre 0 y 1, donde, valdrá 1 si un paper es idéntico a otro, 0 si son totalmente diferentes. Es el factor más importante.
- K Es el número de keywords que coinciden con el paper que se esta recomendando. Consideramos que cada keyword que coincida contribuye muy positivamente a ser similares. Por eso no lo hacemos dividiendo sobre el total de keywords entre los dos papers.
- U La valoración del usuario sobre un paper. Puede ser igual a uno de estos valores:
  - 0.5: El usuario ha decidido dar una puntuación negativa a ese elemento.
  - 1: El usuario no ha dado puntuación alguna sobre ese elemento.
  - 2: El elemento se ha valorado positivamente.
- G Es el número de nodos de en el camino devuelto por el algoritmo de shortest path. Se hace la inversa porque contra más largo sea ese camino peor. Por ejemplo, si los papers estuvieran escritos por el mismo autor la distancia sería 2. Si no están conectados en absoluto decimos que como máximo la distancia es 100.
- C Si están en el mismo cluster, muy probablemente estarán hablando del mismo tema:
  - 2: Si el paper recomendado está en el mismo cluster
  - 1: Si el paper recomendado no está en el mismo cluster.

Somos conscientes de que en el futuro habría que probar muchas otras fórmulas, pero por ahora eso está fuera del tiempo disponible para el TFG.

10.4. Implementación

La implementación de este módulo se ha hecho como el resto del sistema en python, ya que se integra bien con el resto de módulos y además, tiene un amplio abanico de librerías para el *Machine Learning* y el procesamiento del lenguaje natural.

Para construir el módulo de recomendación hemos implementado unos scripts en python para la generación del corpus, entreno del modelo, evaluación del modelo, calcular los clusters, evaluar la calidad de los clusters y generar un descriptor para esos clusters.

Los frameworks que hemos utilizado son:

- Gensim [56]. Nos proporciona una implementación de Doc2Vec eficiente.
- Scikit Learn [69]. Nos proporciona el modelo TF-IDF, el algoritmo k-means y el índice de Calinski-Harabaz para medir la calidad de un clustering.
- NLTK [61] Nos proporciona las stopwords del idioma ingles y varios tipos de lematización.
- Numpi[62] Nos proporciona una implementación de vectores y matrizes eficiente.

,

Para la interacción con el modelo se ha implementado una clase que permite acceder a las recomendaciones.

10.4.1. Generación del corpus

Para la generación del corpus primero se han descargado de internet los PDFs correspondientes a los papers que tenemos en la BBDD y se les ha extraído el contenido. De esta parte se ha encargado la parte buscador (sección 6.1).

La generación del corpus se encuentra en el script corpus_generator.py. Este script realiza las siguientes acciones:

- Conecta con la base de datos para obtener la lista de papers sobre la que generar el corpus.
- Por cada paper extrae el texto disponible. Concatena todas las partes disponibles de los papers (abstract, introducción, body, conclusión). Si un paper no tiene ninguna de estas partes no se añade al corpus.
- A continuación se usa una expresión regular para quedarnos únicamente con las palabras que encajen con [A-Za-z0-9]+, de esta manera eliminamos símbolos extraños.
- Se genera un fichero en texto plano que es una lista de de pares [(id_pdf, çontenido extraído")] capaz de ser interpretada por Python que será el corpus con el que entrenaremos el modelo Doc2Vec. El motivo de este formato es que Doc2Vec necesita etiquetar los documentos para saber que vector está entrenando.

10.4.2. Entrenamiento del modelo

Se encuentra en el script corpus_train.py. este script carga el corpus generado por el script anterior y entrena el modelo Doc2Vec proporcionado por el framework Gensim.

El entrenamiento del modelo se realiza usando el algoritmo Distributed Bag of Words (PV-DBOW). La ventana es de tamaño 5 (distancia máxima de una palabra a otra en una frase).

Cada iteración se ha bajado la ratio de aprendizaje del modelo. Haciendo así que en las ultimas iteraciones haya pocos cambios.

El entreno del modelo es un proceso muy costoso computacionalmente ya que necesita almacenar todos los vectores en RAM mientras los entrena. Entrenar el modelo con 115K papers ha necesitado de 1h en un procesador de 8 núcleos y 13GB de RAM.

10.4.3. Comprobación de la calidad del modelo

Una vez entrenado el modelo hay que verificar la calidad de este, si fuese un modelo de Machine Learning de predicción de una variable sería más sencillo, solo tendríamos que calcular el MMSE

(Minimum mean square error) en caso de ser un problema de regresión o calcular la matriz de confusión en un problema de clasificación.

cómo hemos visto en la explicación de Word2Vec es un modelo que se entrena para predecir palabras cercanas a una palabra dada, Pero el uso que se le suele dar es para calcular la similitud entre dos palabras o en el caso concreto de Doc2Vec la similitud entre documentos.

Para comprobar la calidad de un modelo Doc2Vec, ya que el modelo de palabras incluido lo permite [9], se usa una técnica llamada *Analogical reasoning* [70], consiste en usar el modelo para predecir relaciones semánticas y sintácticas. como por ejemplo: *Un reí es a una reina lo mismo que un padre a?*.

Nosotros hemos decidido probar con algunas operaciones para evaluar el modelo. Se encuentran en el script corpus_test.py.

Palabras similares a machine, aquí esperamos palabras que estén relacionadas con machine learning. Podemos ver que está dando los resultados esperados

```
Similars to machine:
[('learning', 0.7003984451293945), ('reinforcement', 0.677763819694519),
('neural', 0.6622331738471985), ('nlp', 0.655903697013855),
('translation', 0.6539851427078247), ('supervised', 0.6479794979095459),
('explainability', 0.6437076926231384), ('developunified', 0.6418248414993286),
('statistical', 0.6414344310760498), ('vision', 0.636380136013031)]
```

Palabras similares a graphics, nos devuelve hardware relacionado para hacer gráficos, conceptos de gráficos y lo más importante que podemos apreciar aquí es que graphic es una palabra muy similar a graphics así que ha captado que tienen un significado parecido.

```
Similars to graphics:
[('graphic', 0.7298784852027893), ('gpu', 0.6818692684173584),
('cpus', 0.6487172245979309), ('opencv', 0.6279831528663635), ('painterly', 0.6255042552947998),
('uniied', 0.6218037605285645), ('gpus', 0.6198499202728271), ('rendering', 0.6155308485031128),
('instanced', 0.6123372316360474), ('coprocessors', 0.6013293266296387)]
```

Palabras similares a algorithm, nos está devolviendo una lista de sinónimos de Algoritmo.

```
Similars to algorithm:
[('method', 0.8446657061576843), ('framework', 0.7954708337783813),
('approach', 0.7890148758888245), ('strategy', 0.7578275203704834),
('procedure', 0.7338170409202576),('scheme', 0.7296162843704224),
('technique', 0.7284539937973022), ('model', 0.6781560182571411),
('mechanism', 0.6719720959663391), ('formulation', 0.6702737808227539)]
```

Palabras similares a cryptography, podemos ver que nos ha devuelto un algoritmo de cifrado como RSA, conceptos de criptografía, y la palabra elliptic que hace referencia a criptografía de curvas elípticas, que es el cifrado más usado hoy en día por los navegadores web.

```
Similars to cryptography:
[('cryptographic', 0.8091205358505249), ('encryption', 0.7870001792907715), ('cipher',
0.7012331485748291), ('rsa', 0.6962414979934692), ('supersingular', 0.6959288716316223),
('rluts',0.6925610303878784), ('cryptosystem', 0.6901023387908936),
('authentication', 0.6863001585006714), ('fhe', 0.6859890222549438), ('elliptic',
0.6803168058395386)]
```

Aprovechando la representación como vectores de las palabras podemos sumar y restar y ver el resultado:

Por ejemplo, si sumamos machine + learning

```
machine + learning:
[('reinforcement', 0.8258897662162781), ('neural', 0.7626439332962036),
('supervised', 0.741295337677002), ('deep', 0.7351095080375671), ('developunified',
0.71881103515625), ('unsupervised', 0.7141449451446533), ('translation', 0.7080942392349243),
('explainability', 0.7022014260292053), ('generative', 0.6797583103179932), ('for',
0.6766707897186279)]
```

Podemos ver que nos devuelve conceptos fuertemente relacionados con el aprendizaje automático, como supervised, unsupervised, neural, deep.

Basándonos en los sinónimos que el modelo nos da para la palabra algoritmo, podemos sumar dos sinónimos que la definan bien y esperar que devuelva la palabra algoritmo, como podemos ver a continuación:

```
method + approach:
[('framework', 0.8630683422088623), ('algorithm', 0.8426607847213745), ('model',
0.8110402822494507), ('technique', 0.7474384307861328), ('procedure', 0.7414101362228394),
('strategy', 0.72 a
37647175788879), ('methodology', 0.7182762622833252), ('proposal',
0.6978362798690796), ('solution', 0.6943092346191406), ('system', 0.6869431734085083)]
```

Podemos intentar quitar la posible relación que hay entre los gráficos por ordenador y la visión por computador en esté ejemplo hemos quitado la palabra opencv. Podemos ver que está devolviendo resultados relacionados con el hardware de gráficos:

```
graphics + gpu - opencv:
[('cpus', 0.7356969118118286), ('gpus', 0.6980059146881104), ('nics', 0.6466595530509949),
('cpu', 0.6409127712249756), ('parallelism', 0.6395571231842041), ('cores',
0.6342976093292236), ('multicore',0.6329442262649536), ('gpgpu', 0.6180956959724426),
('coprocessors', 0.6176900863647461), ('ffts', 0.6089367866516113)]
```

Si sumamos dos conceptos que están muy relacionados con la criptografía, podemos ver que nos devuelve palabras muy relacionadas con la criptografía aparte de la propia palabra.

```
https + elliptic:  [('cryptographic', 0.7103263139724731), ('supersingular',
0.7041210532188416),('ecdsa', 0.6999403238296509), ('encryption', 0.6851842403411865),
('cryptography', 0.6689081192016602),('cryptosystems', 0.6620275378227234),
('rogaway', 0.6589898467063904), ('elgamal', 0.6497411727905273),('searchable',
0.6480428576469421), ('insecure', 0.6424065232276917)]
```

Si al ejemplo anterior le restamos una palabra que no éste muy relacionada, podemos ver que afecta al resultado, pero no desvía excesivamente del tema principal que es la criptografía.

```
https + elliptic - opencv:  [('cryptosystems', 0.5308120250701904), ('deenition',
0.5235574245452881),('ppss', 0.5044746398925781), ('cryptanalysis', 0.5010378360748291),
('elgamal', 0.499813973903656),('searchable', 0.498440682888031), ('distributivity',
0.49677443504333496), ('certificateless',0.49528950452804565), ('ribe',
0.4947996437549591), ('camenisch', 0.49469423294067383)]
```

Por último, un ejemplo para experimentar que pasa si a la palabra machine le restamos learning, habla de algunos otros tipos de máquinas.

```
machine - learning =  [('imminent', 0.36763817071914673), ('defibrillators',
0.36544638872146606),('knocking', 0.35399138927459717), ('machines', 0.3439665734767914),
('awaiting', 0.32678207755088806), ('incorrectly', 0.29431864619255066), ('geotechnical',
0.2911996841430664), ('earliest', 0.28721296787261963), ('facilities', 0.28635045886039734),
('permanent', 0.28505659103393555)]
```

Después de todas estas pruebas podemos concluir que nuestro modelo está bien entrenado y cumplirá para los propósitos de similitud entre papers de computer science.

10.4.4. Generación de los clusters

Se encuentra en el script clusterer.py. Utiliza el algoritmo de k-means para generar los clusters con los vectores que forman el corpus una vez procesados por el modelo Doc2Vec. De 5 a 50 incrementando de 5 en 5, desde 100 a 1200 de 100 en 100. Guarda los clusters en ficheros.

10.4.5. Comprobando calidad de los clusters

Para comprobar la calidad de los clusters realizados con el algoritmo k-means, vamos a usar el índice de Calinski-Harabaz [8] que mide la ratio entre la media de la dispersión entre clusters divido entre la dispersión del cluster. Se encuentra en el script clusterer_calinski.py

Salida del script, k es el número de clusters:

```
Calinski-Harabaz score for k=5 is 3326.2972444331476
Calinski-Harabaz score for k=10 is 2195.5447982343353
Calinski-Harabaz score for k=15 is 1724.3137124417628
Calinski-Harabaz score for k=20 is 1431.0701301639876
Calinski-Harabaz score for k=25 is 1234.305206418308
Calinski-Harabaz score for k=30 is 1091.983964297446
Calinski-Harabaz score for k=35 is 977.979645857199
Calinski-Harabaz score for k=40 is 887.09137713892
Calinski-Harabaz score for k=45 is 816.7672093745874
Calinski-Harabaz score for k=50 is 756.46073488908
Calinski-Harabaz score for k=100 is 446.8244963386127
Calinski-Harabaz score for k=200 is 256.3672396132024
Calinski-Harabaz score for k=300 is 183.5754651543306
Calinski-Harabaz score for k=400 is 144.2298750146995
Calinski-Harabaz score for k=500 is 119.45262249125791
Calinski-Harabaz score for k=600 is 102.39495092147119
Calinski-Harabaz score for k=700 is 89.82689156647639
Calinski-Harabaz score for k=800 is 80.13542688315708
Calinski-Harabaz score for k=900 is 72.54049971291467
Calinski-Harabaz score for k=1000 is 66.1713490302636
Calinski-Harabaz score for k=1100 is 60.92237257443156
Calinski-Harabaz score for k=1200 is 56.54603031773511
```

Si usamos 50 clusters observamos que nos quedan 2200 papers por cluster de media. Esperamos que independientemente de la valoración de Calinski-Harabaz sea una buena agrupación para detectar los temas de los que hablen dentro de las categorías generales.

10.4.6. Descriptor para los clusters

Se encuentra en el script naming_clusters.py

Para darle información adicional al usuario sobre el cluster que está consultando, hemos decidido sacar las 20 mejores palabras que describan el cluster. Para ello aplicaremos TF-IDF para cada grupo de papers que conforman un cluster y sacaremos las palabras más relevantes globalmente según esta métrica.

Para la aplicación del TF-IDF sobre los clusters primero tenemos que preprocesar los documentos eliminando stopwords y realizando la lematización de las palabras. Después generamos el modelo y extraemos las palabras más importantes.

Si no eliminamos las stopwords lo que ocurre es que en las descripciones de los clusters quedan palabras como 'the', 'at', 'is, etc., ya que son muy comunes en los textos.

Al aplicar lematización conseguimos darle más peso a los términos que salen escritos en el texto en diferentes formas.

Finalmente escribimos los clusters en la base de datos.

Ejemplos de descriptores de clusters obtenidos:

```
["irreversible", "Conditions", "WK", "physical", "extremes", "sensing",
   "finished", "care", "Various", "timing", "locations", "thin",
   "explores", "PLPs", "Unfortunatelysometimes", "Allowing",
   "presentnatural", "finer", "coupled", "laws"]

["attributes", "avatar", "authoring", "Scanning", "Wizard", "scanning",
 "setup", "Text", "entry", "Substance", "abuse", "issignificant",
   "intervention", "offerpromising", "substance", "inappropriate",
   "deviant",  "deviancy", "aid", "leverages"]
```

No se tiene en cuenta si las palabras que definen el cluster son las que mas los distinguen unos de otros, pero para ello habría que hacer el TF-IDF sobre los descriptores de los clusters y quedarse otra vez con las palabras mas importantes. Experimentalmente parece que la calidad de los descriptores es suficiente sin esta técnica.

10.4.7. Información extraída de la base de datos de grafos

Para obtener el Pagerank de los autores el sistema lo consulta en la base de datos Neo4j donde están todos los datos del sistema representados en forma de grafo. La explicación de esta parte está en la parte del buscador 9.6.1.

Para obtener el shortest path primero se consulta en la cache del recomendador si ya tiene el shortest path calculado entre esos dos papers. En caso negativo se pide a la base de datos de grafos que lo calcule. esta cache se renovará cada cierto tiempo, cuando se introduzca información nueva al sistema, por ejemplo, una vez al mes.

10.4.8. Valoraciones del usuario

El sistema usa las valoraciones que el usuario realiza sobre los papers de la base de datos a partir de la API que expone el modelo.

10.4.9. Funcionamiento

El sistema pone a disposición una API donde se puede consultar los papers más similares, para un usuario y paper concreto, la similitud entre dos papers y el camino más corto entre dos papers.

Cuando el usuario pide al recomendador los papers más similares ocurre el siguiente proceso:

1. Obtiene del modelo Doc2Vec una lista de los 10 papers más parecidos en contenido. Por cada paper de esta lista:
   a) Pide a la base de datos de grafos el shortest path entre el paper original y cada uno de la lista de similares.
   b) Obtiene de la información del usuario si le ha dado una puntuación negativa, positiva o no puntuado.
   c) Obtiene el Pagerank de los autores y hace la media.
   d) Comprueba si está en el mismo cluster.
   e) Calcula la puntuación final para ese paper.

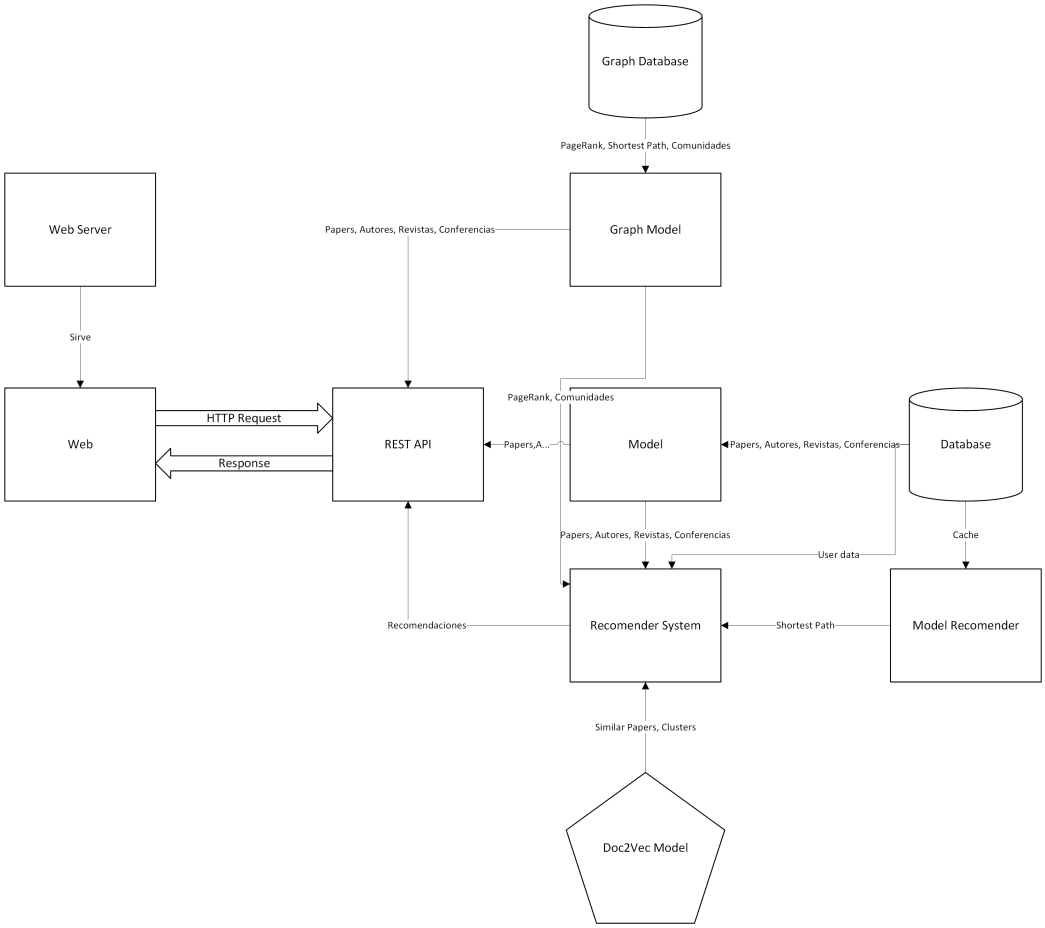

Figura 10.9: Flujo de datos entre las diferentes partes del sistema

10.5. Futuro

En un futuro se podría tener en cuenta:

- Considerar y evaluar otras funciones de puntuación distintas, que combinen los inputs de otros módulos mediante otras fórmulas o con otras ponderaciones.
- Mejorar los descriptores de los clusters con otras técnicas.

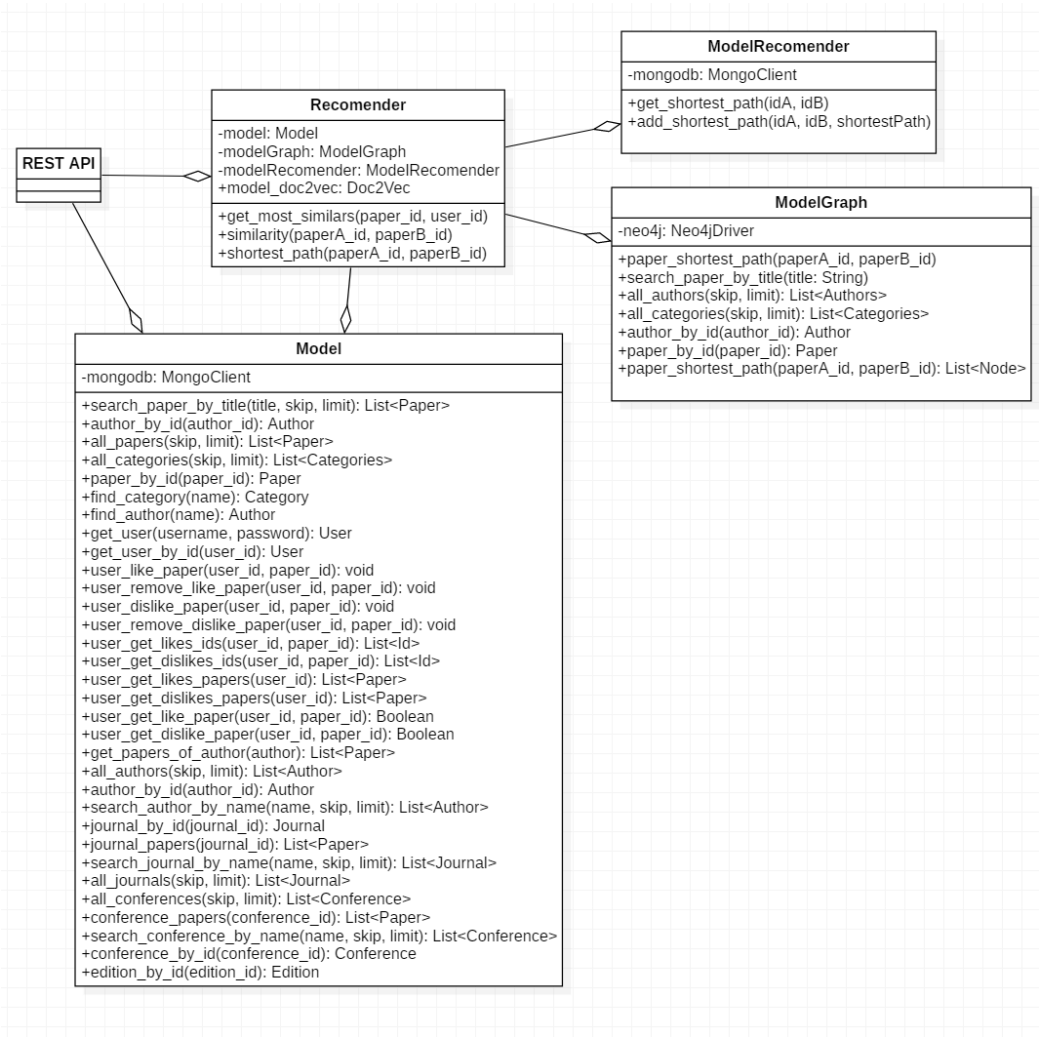

Figura 10.10: Las clases que forman el backend del sistema, incluidas las del recomendador y el modelo.

- El histórico de los papers que ha visitado el usuario para mejorar la experiencia de usuario y las recomendaciones.
- Utilizar el grafo formado por las citaciones que se producen entre los papers para calcular el PageRank de estos y utilizarlo en el sistema para puntuar mejor los papers.
- Utilizar las comunidades formadas por los journals y conferencias.
- Recomendar otros elementos como: autores en base a las puntuaciones que el sistema da a los papers publicados por este.

Capítulo 11

API REST/Controlador

11.1. Descripción

La API REST es un servidor web al cual se le pueden hacer peticiones HTTP, tanto de consulta como de actualización. Es parte del controlador junto al recomendador en el esquema MVC.

Esta parte es bastante estándar y por eso se describe de manera muy directa. Se decidió que todas las respuestas fueran en formato jSON.

11.2. Especificación

- ruta: /
  - Argumentos: No tiene argumentos.
  - Descripción: Devuelve el saludo del servidor.
- ruta: /papers
  - Argumentos: No tiene argumentos
  - Descripción: Devuelve una lista con los últimos 50 papers añadidos a la base de datos.
- ruta: /papers/<paper_id>
  - Argumentos:
    - paper_id: Id del paper que se desea consultar
  - Descripción: Retorna el paper con el id pasado como parámetro.
- ruta: /papers/by_title/<title>
  - Argumentos:
    - title: String con las palabras que contendrá el título de los papers encontrados.
  - Descripción: Devuelva una lista con los papers que en el título tengan alguna de las palabras contenidas en la string title.
- ruta: /papers/similars_metadata/<paper_id>
  - Argumentos:
    - paper_id: Id del paper del que se desean obtener la información de los papers similares a el.

- Descripción: Devuelve una lista con la información de los papers mas similares al paper identificado por paper_id. No los papers en si.

- ruta: /login?username=<username>&password=<password>
- Argumentos:
  - username: Nombre del usuario con el cual se quiere hacer login.
  - password: Contraseña del usuario con el cual se quiere hacer login.
- Descripción: Si tiene éxito, retorna la información del usuario con el que se ha intentado hacer login.

- ruta: /user/<user_id>/paper/<paper_id>/dislike
- Argumentos:
  - user_id: Id del usuario sobre el cual se quiere hacer la Descripción.
  - paper_id: Id del paper sobre el cual se quiere hacer la Descripción.
- Descripción: Actualiza la base de datos, con la información de que al usuario identificado por user_id no le ha gustado el paper identificado por paper_id. Si el usuario previamente había marcado ese paper como like, se desmarcara y pasara a ser dislike.

- ruta: /user/<user_id>/paper/<paper_id>/like
- Argumentos:
  - user_id: Id del usuario sobre el cual se quiere hacer la actualización.
  - paper_id: Id del paper sobre el cual se quiere hacer la actualización.
- Descripción: Actualiza la base de datos, con la información de que al usuario identificado por user_id le ha gustado el paper identificado por paper_id. Si el usuario previamente había marcado ese paper como dislike, se desmarcara y pasara a ser like.

- ruta: /user/<user_id>/paper/<paper_id>/remove_dislike
- Argumentos:
  - user_id: Id del usuario sobre el cual se quiere hacer la actualización.
  - paper_id: Id del paper sobre el cual se quiere hacer la actualización.
- Descripción: Actualiza la base de datos, con la información de que al usuario identificado por user_id ha dejado de no gustarle el paper identificado por paper_id. Si no estaba marcado no ocurrirá nada.

- ruta: /user/<user_id>/paper/<paper_id>/remove_like
- Argumentos:
  - user_id: Id del usuario sobre el cual se quiere hacer la actualización.
  - paper_id: Id del paper sobre el cual se quiere hacer la actualización.
- Descripción: Actualiza la base de datos, con la información de que al usuario identificado por user_id ha dejado de gustarle el paper identificado por paper_id. Si no estaba marcado no ocurrirá nada.

- ruta: /user/<user_id>/paper/dislikes
- Argumentos:
  - user_id: Id del usuario sobre el cual se quiere hacer la consulta.
- Descripción: Devuelve la lista de papers que no le han gustado al usuario.

- ruta: /user/<user_id>/paper/likes
- Argumentos:
  - user_id: Id del usuario sobre el cual se quiere hacer la consulta.

- Descripción: Devuelve la lista de papers que le han gustado al usuario.

- ruta: /authors
- Argumentos: No tiene argumentos
- Descripción: Devuelve una lista con los últimos 50 autores añadidos a la base de datos.

- ruta: /authors/by_name/<name>
- Argumentos:
  - name: String con el nombre por el que se buscara al autor.
- Descripción: Devuelve el autor que encuentre en la base de datos con ese nombre.

- ruta: /authors/<author_id>
- Argumentos:
  - author_id: Identificador del autor en la base de datos.
- Descripción: Devuelve el autor identificado con ese id en la base de datos.

- ruta: /journals
- Argumentos: No tiene argumentos
- Descripción: Devuelve una lista con los últimos 50 journals añadidos a la base de datos.

- ruta: /journals/by_name/<name>
- Argumentos:
  - name: String con el nombre por el que se buscara al journal.
- Descripción: Devuelve el journal con ese nombre en la base de datos.

- ruta: /journals/<journal_id>
- Argumentos:
  - journal_id: Identificador del journal en la base de datos.
- Descripción: Devuelve el journal identificado con ese id en la base de datos.

- ruta: /journals/<journal_id>/papers
- Argumentos:
  - journal_id: Identificador del journal en la base de datos.
- Descripción: Devuelve una lista con los últimos 50 papers de ese journal.

- ruta: /conferences
- Argumentos: No tiene argumentos
- Descripción: Devuelve una lista con los últimos 50 conferences añadidos a la base de datos.

- ruta: /conferences/by_name/<name>
- Argumentos:
  - name: String con el nombre por el que se buscara la conferencia.
- Descripción: Devuelve la conferencia con ese nombre en la base de datos.

- ruta: /conferences/<conference_id>
- Argumentos:
  - conference_id: Identificador de la conferencia en la base de datos.
- Descripción: Devuelve la conferencia identificada con conference_id.

- ruta: /conferences/<conference_id>/papers

- Argumentos:
  - conference_id: Identificador de la conferencia en la base de datos.
- Descripción: Devuelve una lista con los últimos 50 papers de la conferencia.

- ruta: /editions/<edition_id>
- Argumentos:
  - edition_id: Identificador de la edición en la base de datos.
- Descripción: Devuelve la edición identificada con edition_id.

11.3. Implementación

La API se ha implementado exactamente como indica la especificación sobre el microframework Flask y Python.

Se accede a todos los datos a través de los modelos tal y como se ha indicado en los diagramas anteriores. En este sentido el servidor contiene por composición las clases que le permiten acceder a la base de datos,

Un fragmento de codigo del fichero server.py que corresponde a la implementación de la API para las peticiones de los papers:

```
@app.route('/')
def hello_world():
    jn = {"Hello" : "This is the REST API for the sciflix backend."}

    return app.response_class(
        response=json.dumps(jn, separators=(',', " : "), indent=4),
        status=200,
        mimetype='application/json')


@app.route('/papers')
def papers():

    skip = request.args.get('skip', 0)
    limit = request.args.get('limit', 50)

    result = model.all_papers(int(skip), int(limit))

    return app.response_class(
        response=json.dumps(result, separators=(',', " : "), indent=4, cls=MongoJsonEn
        status=200,
        mimetype='application/json')


@app.route('/papers/by_title/<title>')
def search_paper_by_title(title):

    skip = request.args.get('skip', 0)
    limit = request.args.get('limit', 50)
```

```
    result = model.search_paper_by_title(title, int(skip), int(limit))

    return app.response_class(
        response=json.dumps(result, separators=(',', ": "), indent=4, cls=MongoJsonEn
        status=200,
        mimetype='application/json')

@app.route('/papers/<paper_id>/cluster_num')
def cluster_number_by_paper_id(paper_id):

    result = model.cluster_of_paper(paper_id)

    return app.response_class(
        response=json.dumps({'num_cluster': result['num_cluster']}, separators=(',', "
        status=200,
        mimetype='application/json')


@app.route('/papers/<paper_id>/cluster')
def cluster_by_paper_id(paper_id):

    result = model.cluster_of_paper(paper_id)

    return app.response_class(
        response=json.dumps(result, separators=(',', ": "), indent=4, cls=MongoJsonEn
        status=200,
        mimetype='application/json')

@app.route('/papers/<paper_id>')
def paper_by_id(paper_id):

    result = model.paper_by_id(paper_id)
    print(result)
    return app.response_class(
        response=json.dumps(result, separators=(',', ": "), indent=4, cls=MongoJsonEn
        status=200,
        mimetype='application/json')


@app.route('/papers/similars_metadata/<paper_id>')
def similar_papers_metadata(paper_id):

    result = []
    similars = recommender.get_most_similars(paper_id, logged_user['_id'])
    for paper in similars:

        p = model.paper_by_id(paper['id'])
        if p is not None:
            paper['paper'] = p
            result.append(paper)

    return app.response_class(
        response=json.dumps(result, separators=(',', ": "), indent=4, cls=MongoJsonEn
        status=200,
```

```
        mimetype='application/json')
```

Capítulo 12

Modelo

12.1. Introducción

El modelo forma parte de la arquitectura MVC de la plataforma y es el encargado de realizar las operaciones de acceso, actualización, borrado sobre la base de datos generada por la parte buscador.

12.2. Especificación

Hemos decidido dividir el modelo en tres partes dependiendo de que tipo de datos se trate:

- Modelo: Se encarga de los datos mas típicos de la web, que no requieren ningún tipo de procesado. Esto incluye papers, autores, revistas, conferencias y la información del usuario. Permite buscar por nombre, título o id dependiendo del tipo de elemento.
- Modelo Graph Database: Se encarga de acceder a los datos que se han de extraer de la base de datos en forma de grafo. Como el PageRank de los autores o el shortest path.

  Las consultas son a la base de datos Neo4j con el lenguaje de consulta Cypher de la propia base de datos.
- Modelo Recomendador: Es el modelo que se encarga de acceder a la cache del módulo recomendador. Le proporciona los shortest path al módulo recomendador.

  La cache del recomendador esta implementada sobre una colección de la base de datos MongoDB.

12.2.1. Diseño Modelo

El diseño del modelo refleja la misma estructura divisoria propuesta en la especificación, en las figuras 12.1, 12.2 y 12.3 puede verse las métodos que implementan y como se han distribuido.

12.2.2. Implementación

Las clases se han implementado en Python para que puedan ser usadas por el resto del sistema, cada una para acceder a la base de datos que le corresponde, en concreto:

- Model:

Figura 12.1: Diseño de la clase modelo.

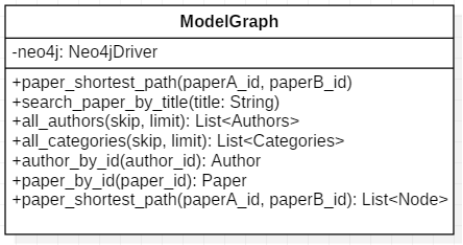

Figura 12.2: Diseño de clase modelo encargada de acceder a la base de datos de grafos.

- Accede a la base de datos MongoDB con el driver oficial de MongoDB para Python.
- Usa la base de datos scrapy_output que es el resultado final de la unión de los datos obtenidos por el buscador.
- Obtiene los datos de las colecciones authors, categories, clusters, journals, conferences, editions, todas ellas definidas en la parte buscador y corpus_papers (son los papers que cumplen los requisitos para ser añadidos al corpus para las recomendaciones. Generada a partir de los scripts que generaban el corpus del recomendador).
- Almacena los usuarios de la plataforma y toda la información relacionada. Tiene la siguiente estructura:
  - username: Nombre de usuario
  - password: Contraseña del usuario para iniciar sesion.
  - idLikedPapers: Contiene una lista de los ids de los papers que han gustado al usuario.
  - idDislikedPapers: Contiene una lista de los ids de los papers que no han gustado al usuario.
- Almacena los clusters obtenidos a partir de ejecutar el algoritmo sobre la base de datos Neo4j. La tabla tiene la siguiente estructura:
  - num_cluster: Identificador del cluster.
  - papers_id: Lista de los identificadores de papers que pertenecen al cluster.
  - name: Descriptor generado para este cluster.

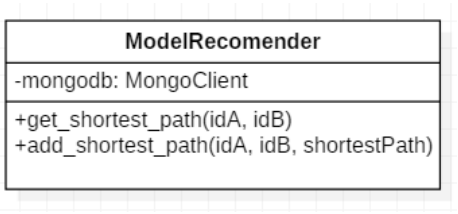

Figura 12.3: Diseño de la clase modelo encargada de acceder a la cache del recomendador.

- Model Graph Database:
  - Accede a la base de datos Neo4j con el driver oficial de Neo4j para Python.
  - usa la base de datos generada por el módulo Graph Builder definida en la parte buscador.
  - Obtiene el PageRank de los autores, las comunidades de los autores y es capaz de calcular el shortest path entre dos papers.
- Modelo Recomendador:
  - Este modelo se usa para almacenar la información que calcula el recomendador. Actúa como cache para no tener que repetir las operaciones costosas como calcular el shortest path cada vez.
  - Usa una base de datos aparte dentro del mismo servidor mongodb llamada *recomender*. Únicamente contiene una colección llamada *shortests_paths* con la siguiente estructura:
    - idA: id del paper de salida
    - idB: id del paper de destino.
    - shortest_path: Lista de pares que contienen los ids de los nodos de neo4j. Se puede enlazar todo el camino uniendo el segundo elemento de cada par con el primero del siguiente.

Capítulo 13

Cliente Web

13.1. Objetivo

Ofrecer al usuario una interfaz gráfica para poder interactuar con el sistema, que se pueda utilizar desde cualquier navegador web.

13.2. Especificación

Ha de permitir al usuario realizar las siguientes acciones:

- Navegar de forma anónima por la página o iniciar sesión con usuario y contraseña (En esta versión de la plataforma todavía no se pueden crear nuevos usuarios).
- Navegar por los papers disponibles en la plataforma en forma de lista, permitiendo filtrar estos introduciendo texto en un campo de búsqueda. El campo de búsqueda filtrará por título, keywords y autores.
- Navegar por los autores disponibles en la plataforma en forma de lista, permitiendo filtrar estos introduciendo texto en un campo de búsqueda. El campo de búsqueda filtrará por el nombre del autor.
- Navegar por las conferencias disponibles en la plataforma en forma de lista, permitiendo filtrar estas introduciendo texto en un campo de búsqueda. El campo de búsqueda filtrará por el nombre de la conferencia.
- Navegar por los journals disponibles en la plataforma en forma de lista, permitiendo filtrar estas introduciendo texto en un campo de búsqueda. El campo de búsqueda filtrará por el nombre del journal.
- Mostrar los detalles de un paper concreto permitiendo ver toda la información disponible de estos y recomendar papers similares a este con los criterios definidos en el sistema recomendador.
- Mostrar los detalles de un autor en concreto permitiendo ver toda la información disponible de este y los papers que ha escrito.
- Mostrar los detalles de una conferencia en concreto permitiendo ver toda la información disponible de esta incluyendo la edición y la lista de papers que se publicaron en esa conferencia.

- Mostrar los detalles de un Journal en concreto permitiendo ver toda la información disponible de este incluyendo la lista de papers que se publicaron en ese Journal.
- Permitir al usuario editar su información personal, papers que ha marcado como que le gustan, papers que ha marcado como que no le gustan.

13.3. Diseño

El cliente web se divide en dos partes. Una es la cabecera, siempre visible, que nos permite acceder a la exploración de papers, autores revistas o conferencias, y acceder a la información de usuario o iniciar sesión.

La segunda parte es donde el usuario puede ver la información que ha solicitado, la exploración de los papers, autores, revistas y conferencias.

13.3.0.1. Diseño Inicial

Esta es la primera versión descartada del prototipo del cliente web, desarrollada en HTML + Javascript sin ningún framework de ayuda. en la sección de implementación están las imágenes del diseño final.

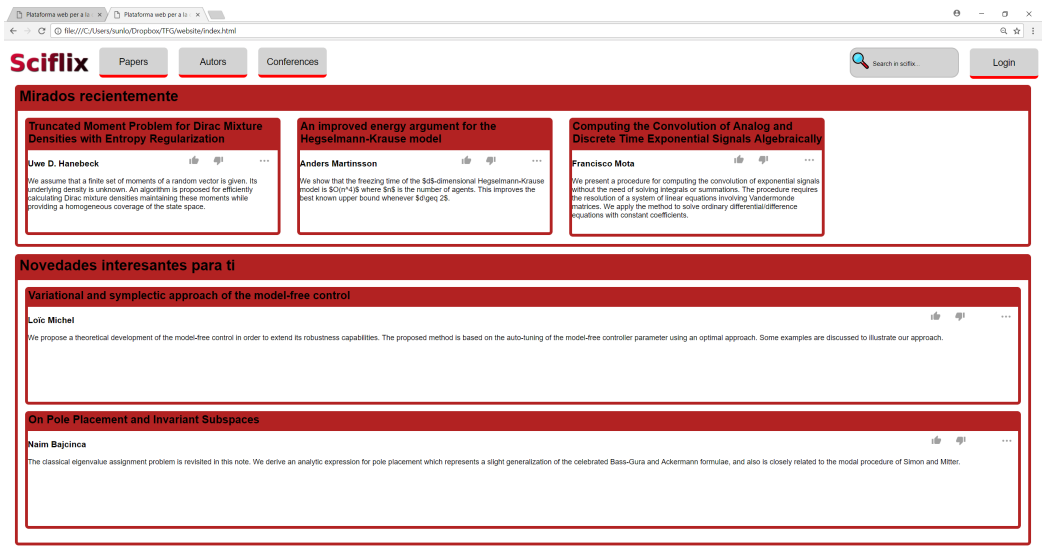

Figura 13.1: Dashboard del diseño inicial

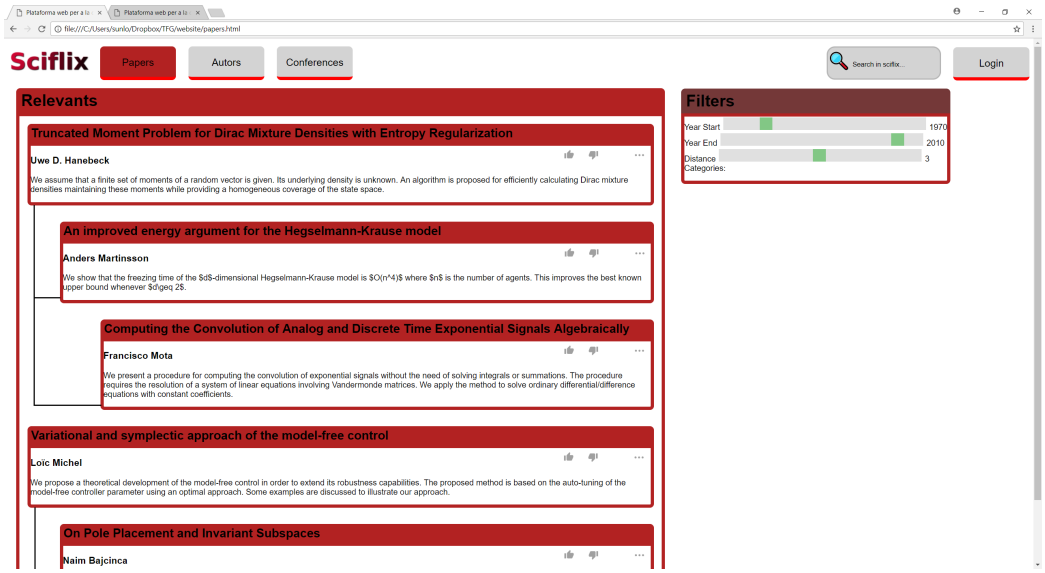

Figura 13.2: Sección papers del diseño inicial

13.4. Implementación

El cliente web está desarrollado sobre el framework Angular[44], es un framework reactivo, programado en TypeScript [71] que permite la programación asíncrona a través de Observables, los explicamos en la sección 13.4.4.

13.4.1. Tipos de Clases/Objetos

Los principales tipos de Objetos que forman un sitio web hecho sobre el framework angular son los siguientes:

- Components: Son los objetos que se visualizan en la web, están formados por tres ficheros, un HTML con la témplate, un CSS con el estilo del componente y un TypeScript donde se definen los métodos y atributos de la clase del componente.

  A continuación, hablaremos de los elementos que muestran los componentes. Entendemos por elemento un Autor, Paper, Revista, Conferencia o Edición.

  La mayoría de componentes siguen la siguiente estructura:

  - Lista de elementos principal: Compuesto por un campo de búsqueda y una lista de los elementos a mostrar.
  - Vista en detalle de un elemento: Muestra toda la información de un elemento del sistema y quizá las recomendaciones posibles.
  - Vista ïnline"del elemento: Muestra únicamente el campo más relevante de ese elemento. Se inicializan con el id del objeto a mostrar. Esto se ha pensado para que otros componentes puedan embeber estos componentes inline para mostrar esa información, por ejemplo, un paper muestra en qué conferencia ha sido presentado con un componente conferencia inline. De esta forma el componente paper que no tiene información de la conferencia solo el id, pueda delegar la carga de esta información a ese componente. Esto aprovecha al máximo la arquitectura de la BBDD MongoDB donde no es posible hacer uniones de tablas como en los SGBDR (Sistema Gestor de Bases de Datos Relacional) SQL. Estos

componentes se inicializan con el identificador del elemento a mostrar y el componente a su vez pedirá al servicio encargado de servir ese elemento que le da la información, que a su vez lanzará una petición HTTP contra la API REST que le devolverá la información, esto al ser un proceso asíncrono fuerza a la API REST a hacer varias consultas en paralelo a la base de datos, resultando en una carga de la página muy rápida. En la figura 13.3 se puede ver la representación de la inicialización de un componente.

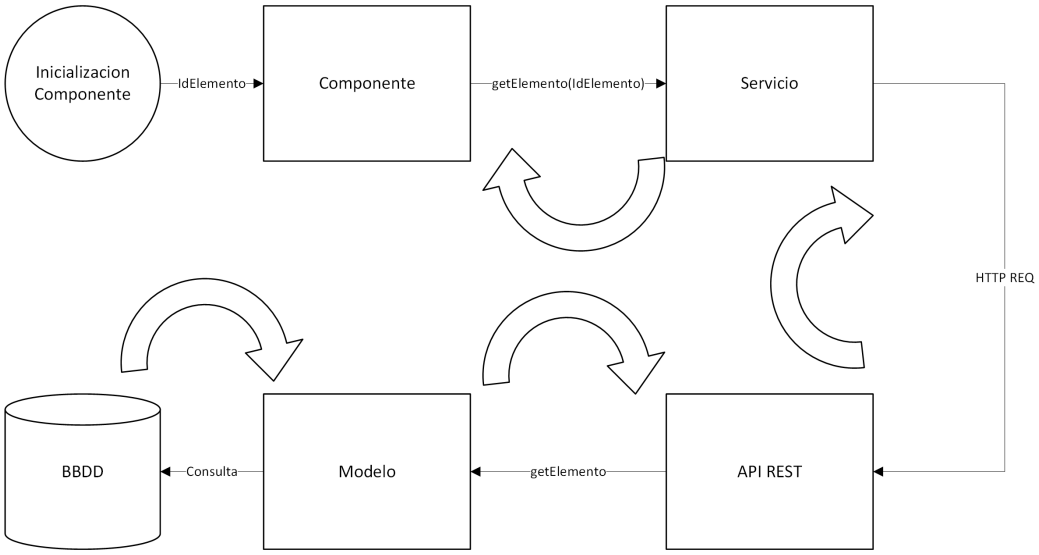

Figura 13.3: Proceso de inicialización de un componente

- Services: Son las clases que se encargan de obtener la información y subministrarla a los componentes. Normalmente solo hay una instancia de cada servicio. En nuestro caso todos los servicios obtienen la información de la API REST con un cliente HTTP.
- POJOs (Plain Old JavaScript Objects): Son las clases que son estructuras de datos simples que contienen la información que devuelven los servicios, por ejemplo, Paper, Author, Conference, Journal.

13.4.2. Componentes

- App-Component: Es la raíz de la página web, todo lo que tenga este componente sera visible siempre. contiene el componente App-Routing. En la figura 13.4 se puede ver la estructura principal de la página que consta de una cabecera que siempre es visible y el lugar que ocupa el componente mostrado por App-Routing.

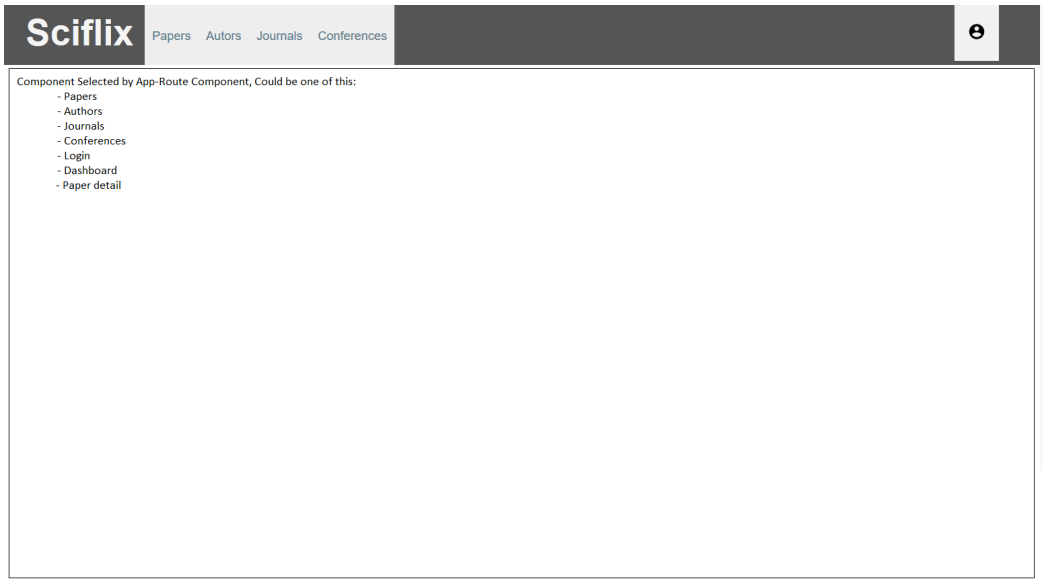

Figura 13.4: Componente App-root

- App-Routing: Se encarga de cargar la página web con el componente principal indicado por la URL. Como se puede ver en la figura 13.4. Los componentes que puede cargar son los siguientes:
  - path: 'papers', component: PaperSearchComponent
  - path: 'dashboard', component: DashboardComponent,
  - path: 'paper/:id', component: PaperDetailComponent
  - path: 'login', component: LoginComponent
  - path: 'autors', component: AuthorsComponent
  - path: 'author/:id', component: AuthorDetailComponent
  - path: 'journals', component: JournalsComponent
  - path: 'conferences', component: ConferencesComponent
  - path: 'journal/:id', component: JournalDetailComponent
  - path: 'conference/:id', component: ConferenceDetailComponent
  - path: '', redirectTo: '/dashboard', pathMatch: 'full'

  Los componentes en los que su URL contiene :id significa que ese componente recibirá como parámetro en la variable id el id del componente que cargara y mostrara.
- Login: Componente con los campos de usuario y contraseña para iniciar sesión en la web como se puede ver en la figura 13.5

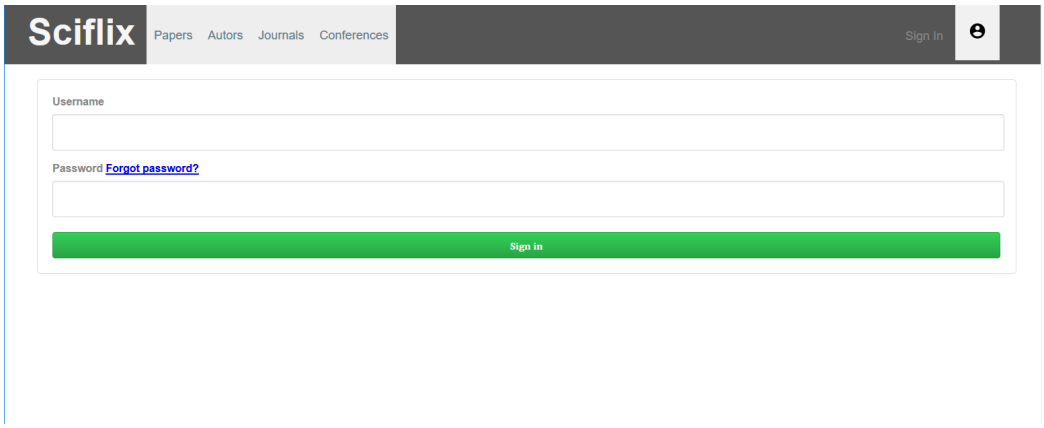

Figura 13.5: Componente Login

- Dashboard: Componente que muestra un pequeño resumen del perfil del usuario. Las valoraciones que ha hecho el usuario sobre los papers.

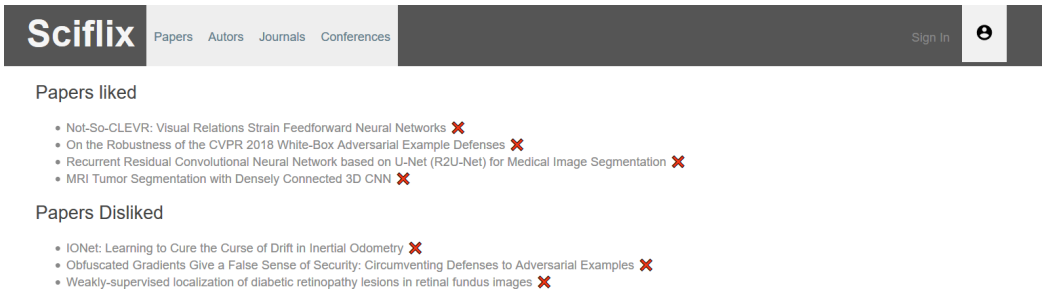

Figura 13.6: Componente Dashboard

- Paper-Detail: Muestra toda la información del paper que recibe como entrada. La lista de papers relacionados está generada por el componente Similar-Paper-List que recibe como parámetro el paper que está mostrando este componente.

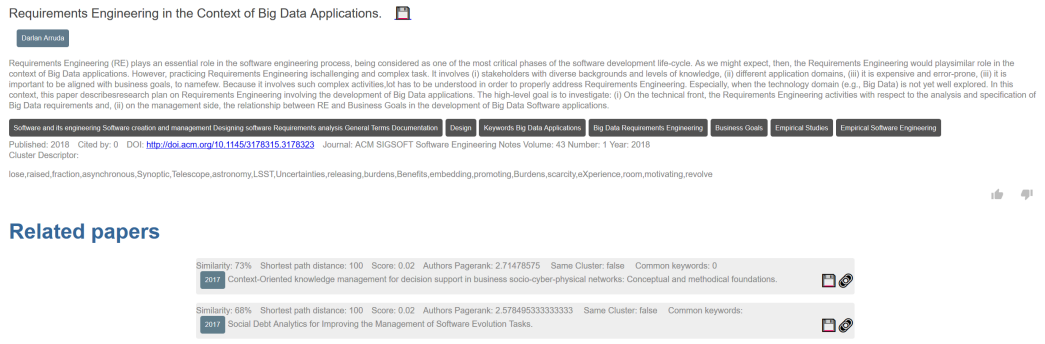

Figura 13.7: Componente Paper-Detail: Permite ver la información de un paper en detalle

- Paper-Search: Contiene un campo de búsqueda para buscar un paper por título y una lista de papers con el resultado. La lista se actualiza cada 300ms desde la última pulsación del usuario dando un efecto de búsqueda en tiempo real mientras se escribe en el campo. Se puede ver un ejemplo del funcionamiento y los componentes que lo componen en la figura 13.9

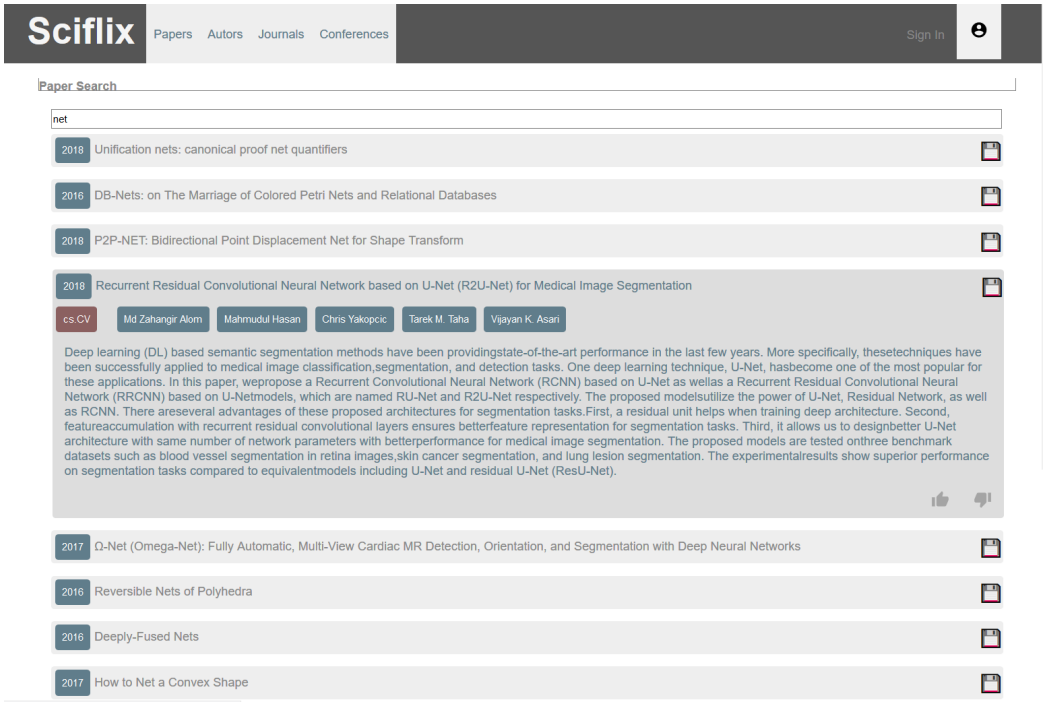

Figura 13.8: Sección principal Papers, buscando el término "net".

- Similar-Paper-List: Muestra los papers más similares dado un paper como entrada, muestra además información con las valoraciones que ha calculado el sistema, con los valores que se han calculado para la fórmula de recomendación, como, por ejemplo: Shortest-Path, Similitud, etc. Forma parte del componente paper-detail como se puede ver en la figura 13.7

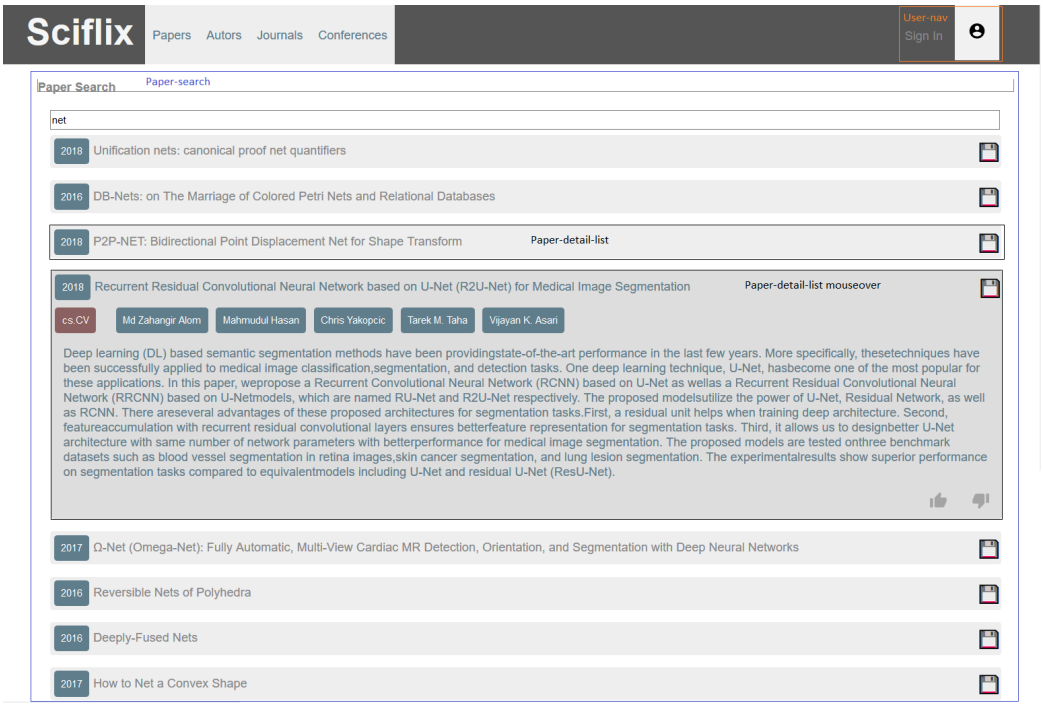

Figura 13.9: Componentes que componen el componente Paper-Search.

- Paper-Detail-List: Representa un elemento de la lista del componente Paper-Search, Muestra un pequeño resumen de los campos más importantes del paper que le ha suministrado como parámetro. Se puede ver en la figura 13.9.
- User-Nav: Es el menú donde el usuario puede hacer login. Forma parte del header es visible en la mayoría de figuras.

Figura 13.10: User-Nav Component: Muestra el nombre de usuario y la foto de perfil.

- authors: Muestra una lista de los autores almacenados en el sistema, incluye la barra de búsqueda con la que podemos filtrar por nombre.

Figura 13.11: Componente que muestra una lista de autores y una barra de búsqueda con la que permite filtrar.

- author-detail: Muestra toda la información disponible sobre un autor, como nombre y Pagerank, incluyendo los papers que este ha escrito.

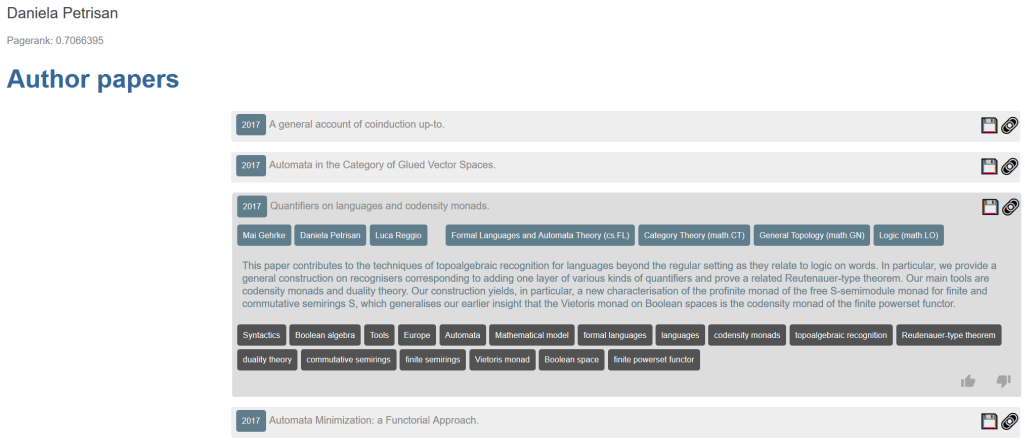

Figura 13.12: Componente que muestra una lista de autores y una barra de búsqueda con la que permite filtrar.

- conferences: Muestra una lista de las conferencias almacenadas en el sistema, incluye la barra de búsqueda con la que podemos filtrar por los datos de la conferencia.

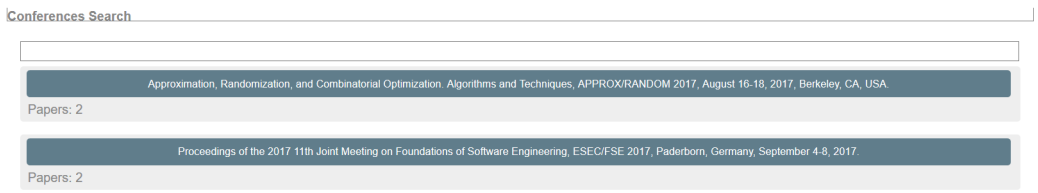

Figura 13.13: Componente que muestra una lista de conferencias y una barra de búsqueda con la que permite filtrar.

- conference-detail: Muestra toda la información disponible sobre una conferencia incluyendo los papers que se han presentado en esta.

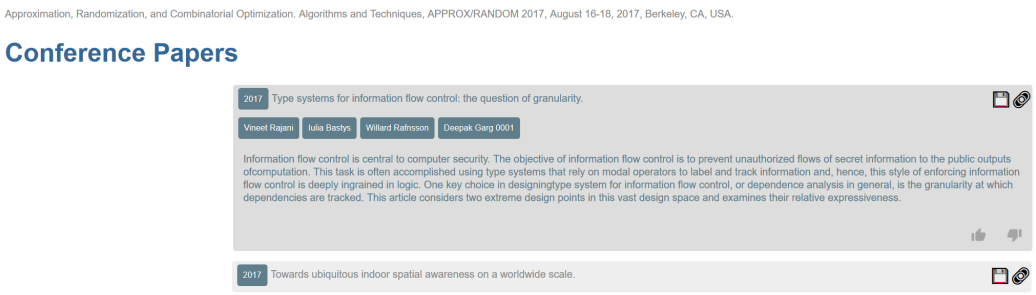

Figura 13.14: Componente que muestra una lista de conferencias y una barra de búsqueda con la que permite filtrar.

- conference-mini: Muestra solo el nombre de la conferencia, es un componente con una funcionalidad ïnline". para poderlo insertar en otras partes de la web, por ejemplo, en el paper detail.

Figura 13.15: Conference-mini: Componente ïnline".

- journal-mini: Muestra solo el nombre del journal y la edición, es un componente con una funcionalidad ïnline". para poderlo insertar en otras partes de la web, por ejemplo, en el paper detail.

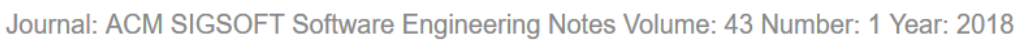

Figura 13.16: Journal-mini: Componente ïnline".

13.4.3. Servicios

Services: Los servicios son los encargados de obtener, actualizar y eliminar los elementos que se muestran en la web. En nuestro caso son los encargados de interactuar con la API REST a través de peticiones HTTP para realizar dichas operaciones sobre los elementos.

- Paper-service:
  - Obtiene los últimos papers añadidos a la base de datos.
  - Obtiene un paper a partir de un id
  - Obtiene los papers similares a otro.
  - Busca por título los papers en la base de datos.
- User-service:
  - Obtiene el usuario anónimo del sistema.
  - Obtiene el usuario identificado por nombre de usuario y contraseña.
  - Obtiene la lista de papers que le han gustado al usuario.
  - Obtiene la lista de papers que no le han gustado al usuario.

    - Actualiza la lista de papers que han gustado al usuario cuando este indica que le ha gustado uno.
    - Actualiza la lista de papers que no han gustado al usuario cuando este indica que no le ha gustado uno.
    - Actualiza la lista de papers que han gustado al usuario cuando este indica que le ha dejado de gustar uno.
    - Actualiza la lista de papers que no han gustado al usuario cuando este indica que le ha dejado de no gustar uno.
- Author-service:
    - Obtiene los últimos autores añadidos a la base de datos.
    - Obtiene un autor a partir de un id.
    - Busca por nombre los autores en la base de datos.
- Conference-service:
    - Obtiene las últimas conferencias añadidas a la base de datos.
    - Obtiene una conferencia a partir de un id.
    - Busca por nombre las conferencias en la base de datos.
    - Obtiene los papers presentados en una conferencia.
    - Obtiene una edición a partir de un id.
- Journal-service:
    - Obtiene las últimas revistas añadidas a la base de datos.
    - Obtiene una revista a partir de un id.
    - Obtiene los papers publicados en una revista.
    - Busca por nombre las revistas en la base de datos.

POJOs (Plain Old Javascript Objects): Son la representación de los siguientes componentes, no tienen por qué almacenarse igual en la base de datos, pero de cara al cliente web contienen los campos que hacen falta para acceder a la información necesaria desde los componentes.

- Paper: Representa al elemento Paper. Contiene una lista con los autores, categorías, keywords.
- Author: Representa al elemento Author. Contiene una lista con los ids de los papers que ha escrito.
- Journal: Representa al elemento Journal. Contiene una lista con los ids de los papers que se han publicado.
- Conference: Representa al elemento Conference. Contiene una lista con los ids de los papers que se han presentado.
- User: Contiene toda la información del usuario, las valoraciones de los papers, usuario, contraseña.
- Edition: Contiene la información de una conferencia concreta en un año y su código identificador.
- Recomendation: Es una clase auxiliar usada para ver las puntuaciones que ha asignado el recomendador a cada variable. Contiene el id del paper que recomienda.

- BDResult: Representa la respuesta que genera la API REST al hacer consultas de actualización a la base de datos, devuelve si la consulta ha ido bien, no solo los datos de respuesta.

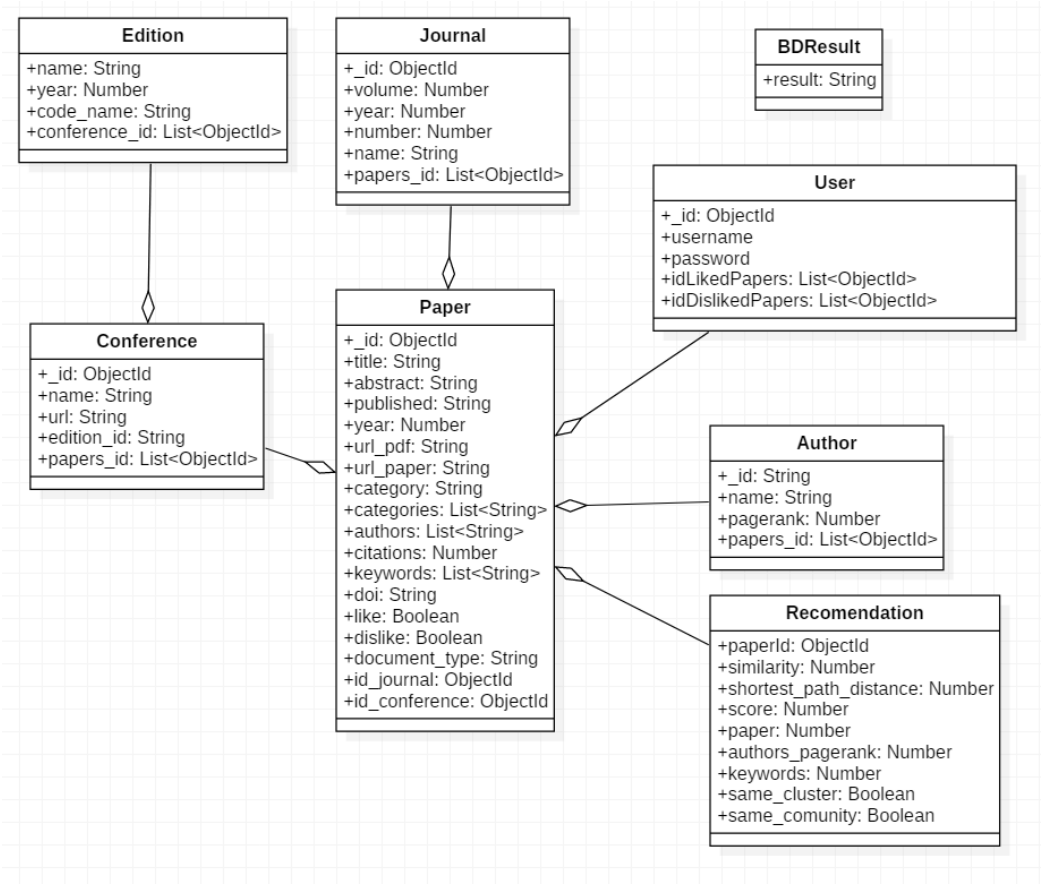

Figura 13.17: POJOs del cliente Web

13.4.4. Observables

La programación reactiva [53] es un paradigma de programación asíncrono relacionado con flujos de datos y la propagación del cambio.

Los observables son un componente de la librería RxJS [66] que permiten la programación reactiva a través de proveer soporte para pasarse mensajes entre publicadores y subscriptores. Esto significa que los métodos de los servicios que devuelven información que han de obtenerla de la API REST no tienen por qué ser bloqueantes, permitiendo la carga o ejecución del resto de la página. Los servicios en vez de devolver un POJO directamente devuelven un Observable del POJO. El objeto que recibe ese observable le subscribe una función que se ejecutara en un momento futuro asíncronamente. Cuando se reciban los datos provocara un cambio en el componente que los recibe pasando a mostrar el nuevo contenido.

13.5. Futuro

Este quizá sea el modulo con mas características posibles a implementar, por citar algunas:

- Mejorar el dashboard.
- Añadir dar de alta un usuario.
- Mejorar la forma de filtrar de los resultados de las búsquedas.

Parte IV

Conclusiones

Capítulo 14

Conclusión

Durante la realización del proyecto hemos tocado sistemas operativos, redes, algoritmos de grafos, técnicas de procesamiento del lenguaje natural. Hemos tenido que hacer computación en la nube, configuración de firewalls, servicios, redes, proxys, contenedores, bases de datos y también hemos utilizado frameworks y lenguajes que no habíamos usado nunca, como Angular y TypeScript con una metodología reactiva. Ha sido un proyecto en el que hemos estado en constante aprendizaje.

El resultado final es que hemos construido un prototipo que cumple con el objetivo que nos hemos propuesto. Lógicamente, la plataforma tiene mucho donde mejorar, pero es un punto de inicio para que la comunidad se pueda aprovechar de ella.

Cuando empezamos el proyecto nuestro campo de visión sobre el dominio era nulo, no sabíamos por donde empezar, como debería ser la estructura, que programas usar, etc. Una prueba de ello es que durante el transcurso del proyecto el diseño inicial y final son diferentes en algunos módulos. Ahora que hemos terminado con este prototipo tenemos una visión mucho más amplia que la que teníamos en un inicio. El diseño actual de la estructura del sistema es un buen inicio, después de estar durante un quatrimestre haciendo el proyecto sabemos cuales son sus puntos débiles y cuales sus fuertes.

Ahora debemos pensar cuales serán nuestros siguientes pasos. Por limitaciones de tiempo no podíamos modificar el diseño hasta llegar a una solución óptima. Con este prototipo nuestros siguientes pasos son empezarían por:

- Dar a conocer nuestro proyecto y empezar a construir una comunidad alrededor de él.
- Completar los objetivos a futuro de cada módulo.
- Buscar financiación para poder pagar los servicios que necesitamos, por ejemplo, pagar las cuotas a los sitios web para hacer *web crawling*

Tenemos ilusión en que el proyecto siga hacia delante porque creemos que es beneficioso socialmente para la comunidad científica en un futuro.

En esta conclusión también nos gustaría hablar sobre la planificación final. El desarrollo del proyecto ha sido bastante diferente de lo que habíamos previsto. Durante la realización nos encontramos bastantes problemas para obtener la base de datos sobre la que completar todo el sistema. Además de la carga de trabajo de la universidad, se decidió finalmente hacer un único prototipo final y mover la ampliación de las características a una fase más temprana donde se fuese implementando todo seguido hasta llegar al prototipo final.

Dado el desconocimiento del problema las fases que se plantearon en un inicio se han prolongado más de lo que se planifico en un inicio. El principal foco de las constantes demoras es solventar todos los problemas que van surgiendo, pero el más remarcable es la recolección de datos para trabajar en el sistema. Hay tres grandes focos donde se destinó casi la totalidad del tiempo:

- Investigación de como arreglar los problemas o que métodos debemos usar en los módulos, por ejemplo, que método debemos usar para calcular la similitud de textos o que algoritmo de comunidades es más relevante.
- Aprendizaje de como usar las herramientas.
- El módulo del *Crawler* fue muy complejo dada las condiciones que se describieron, solventar los problemas que fueron surgiendo hasta que se obtuvo una primera colección de datos nos llevó más tiempo de lo esperado y tiene como consecuencia que los demás módulos tienen menos tiempo dedicado.

También nos enfrentamos con distintos problemas en los demás módulos, pero dada la limitación de tiempo del proyecto se decidió hacer simplificaciones hasta que pudiéramos completarlo con el tiempo que teníamos. En reparto de tiempo del proyecto es:

- Por parte de la FIB se espera que un alumno cumpla:
  - 18 Créditos x 30h/crédito = 540Horas cada miembro, de los cuales 3 créditos son de GEP, 450Horas de TFG. En total como equipo tenemos que dedicar 900Horas.
- Por nuestra parte:
  - En la parte de GEP hemos superado las 90h.
  - La dedicación al TFG se comprende entre principios de abril y a mediados de junio, un total de 67 días. En media hemos dedicado 7h al día. 469H *2 = 938H en total con los dos.

La planificación alternativa de GEP ha cumplido su cometido y nos ha dado el espacio necesario para poder completar el proyecto con un poco de margen.

14.1. El proyecto en números

Originalmente se consiguieron recopilar los 4 millones de papers y dos millones de autores de la DBLP, pero de estos, gran parte resulto inaccesible por el crawler y después de los 200.000 restantes solo se pudo obtener la mitad. Después de todo esto el resultado es el siguiente:

La base de datos contiene:

- 148.001 Papers Activos y 255.262 Papers en total
- 426.689 autores
- 2.696 conferencias
- 6.584 revistas
- Extraemos de 12 dominios diferentes la información de los papers.
- La base de datos ocupa 10GB.
- La base de datos neo4j 625MB.
- Tenemos 2378054 relaciones.
- Tenemos 694866 nodos.
- El corpus ocupa 130MB.
- El modelo doc2vec ocupa 60MB + 57MB de los vectores.
- En total 700GB de disco duro para poder trabajar con los datos de forma cómoda.

Bibliografía

[1] Lawrence Page y col. *The PageRank Citation Ranking: Bringing Order to the Web.* Technical Report 1999-66. Previous number = SIDL-WP-1999-0120. Stanford InfoLab, nov. de 1999. URL: http://ilpubs.stanford.edu:8090/422/.

[2] M. Girvan y M. E. J. Newman. «Community structure in social and biological networks». En: *Proceedings of the National Academy of Sciences* 99.12 (2002), págs. 7821-7826. ISSN: 0027-8424. DOI: 10.1073/pnas.122653799. eprint: http://www.pnas.org/content/99/12/7821.full.pdf. URL: http://www.pnas.org/content/99/12/7821.

[3] Free Software Foundation. *GNU GENERAL PUBLIC LICENSE Version 3.* 2007. URL: https://www.gnu.org/licenses/gpl-3.0.txt.

[4] Santo Fortunato. «Community detection in graphs». En: *CoRR* abs/0906.0612 (2009). arXiv: 0906.0612. URL: http://arxiv.org/abs/0906.0612.

[5] Ioannis Konstas, Vassilios Stathopoulos y Joemon M. Jose. «On Social Networks and Collaborative Recommendation». En: *Proceedings of the 32Nd International ACM SIGIR Conference on Research and Development in Information Retrieval.* SIGIR '09. Boston, MA, USA: ACM, 2009, págs. 195-202. ISBN: 978-1-60558-483-6. DOI: 10.1145/1571941.1571977. URL: http://doi.acm.org/10.1145/1571941.1571977.

[6] Vincent Blondel. *The Louvain method for community detection in large networks.* [Online; accessed 21-June-2018]. 2011. URL: https://perso.uclouvain.be/vincent.blondel/research/louvain.html.

[7] Pasquale De Meo y col. «Generalized Louvain Method for Community Detection in Large Networks». En: *CoRR* abs/1108.1502 (2011). arXiv: 1108.1502. URL: http://arxiv.org/abs/1108.1502.

[8] Bernard Desgraupes. *Clustering Indices.* 2013. URL: https://cran.r-project.org/web/packages/clusterCrit/vignettes/clusterCrit.pdf.

[9] Tomas Mikolov. «Distributed Representations of Words and Phrases and their Compositionality». En: (2013). URL: https://arxiv.org/pdf/1310.4546.pdf.

[10] Tomas Mikolov y col. «Distributed Representations of Words and Phrases and their Compositionality». En: *CoRR* abs/1310.4546 (2013). arXiv: 1310.4546. URL: http://arxiv.org/abs/1310.4546.

[11] Stasa Milojevic. «Accuracy of simple, initials-based methods for author name disambiguation». En: *CoRR* abs/1308.0749 (2013). arXiv: 1308.0749. URL: http://arxiv.org/abs/1308.0749.

[12] Xiang Ren y col. «ClusCite: effective citation recommendation by information network-based clustering». En: *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014.* Ed. por Sofus A. Macskassy y col. ACM, 2014, págs. 821-830. ISBN: 978-1-4503-2956-9. DOI: 10.1145/2623330.2623630. URL: http://doi.acm.org/10.1145/2623330.2623630.

[13] Generalitat de Catalunya. *The electrical mix.* [Online; acceded 2018-03-20]. 2016. URL: http://canviclimatic.gencat.cat/en/redueix_emissions/factors_demissio_associats_a_lenergia/index.html.

[14] B. S. Khan y M. A. Niazi. «Network Community Detection: A Review and Visual Survey». En: *ArXiv e-prints* (ago. de 2017). arXiv: 1708.00977.

[15] Wikipedia. *Lematización — Wikipedia, La enciclopedia libre*. [Internet; descargado 5-marzo-2018]. 2017. URL: https://es.wikipedia.org/w/index.php?title=Lematizaci%C3%B3n&oldid=97771321.

[16] Wikipedia. *Stemming — Wikipedia, La enciclopedia libre*. [Internet; descargado 5-marzo-2018]. 2017. URL: https://es.wikipedia.org/w/index.php?title=Stemming&oldid=104146180.

[17] Proyectos Agiles. *Qué es SCRUM?* [Internet; descargado 5-marzo-2018]. 2018. URL: https://proyectosagiles.org/que-es-scrum/.

[18] Sarah Boon. *21st Century Science Overload, Canadian Science Publishing*. [Online; accessed 5-March-2018]. 2018. URL: http://www.cdnsciencepub.com/blog/21st-century-science-overload.aspx.

[19] Wikipedia contributors. *Computer science — Wikipedia, The Free Encyclopedia*. [Online; accessed 5-March-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Computer_science&oldid=828574530.

[20] Wikipedia contributors. *Data mining — Wikipedia, The Free Encyclopedia*. [Online; accessed 5-March-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Data_mining&oldid=828950114.

[21] Wikipedia contributors. *Machine learning — Wikipedia, The Free Encyclopedia*. [Online; accessed 9-April-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Machine_learning&oldid=834274701.

[22] Wikipedia contributors. *Milestone (project management) — Wikipedia, The Free Encyclopedia*. [Online; accessed 5-March-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Milestone_(project_management)&oldid=826464032.

[23] Wikipedia contributors. *Natural-language processing — Wikipedia, The Free Encyclopedia*. [Online; accessed 6-March-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Natural-language_processing&oldid=827019040.

[24] Wikipedia contributors. *PageRank — Wikipedia, The Free Encyclopedia*. [Online; accessed 6-March-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=PageRank&oldid=826163035.

[25] Wikipedia contributors. *Project engineering — Wikipedia, The Free Encyclopedia*. [Online; accessed 9-April-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Project_engineering&oldid=832963833.

[26] Wikipedia contributors. *Software engineer — Wikipedia, The Free Encyclopedia*. [Online; accessed 9-April-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Software_engineer&oldid=834544813.

[27] Wikipedia contributors. *Tf–idf — Wikipedia, The Free Encyclopedia*. [Online; accessed 6-March-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Tf%E2%80%93idf&oldid=827210390.

[28] Wikipedia contributors. *Web crawler — Wikipedia, The Free Encyclopedia*. [Online; accessed 5-March-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Web_crawler&oldid=827217149.

[29] UPC-CS Department. *Computer Science, web page*. [Online; accessed 5-March-2018]. 2018. URL: https://www.cs.upc.edu/.

[30] Inc. GitHub. *GitHub, Inc, web page*. [Online; accessed 5-marzo-2018]. 2018. URL: https://github.com/.

[31] ResearchGate GmbH. *ResearchGate Official Web*. [Online; accessed 5-March-2018]. 2018. URL: https://www.researchgate.net/.

[32] GlassDoor Inc. *Find The Job That Fits Your Life*. [Online; acceded 18-march-2018]. 2018. URL: https://www.glassdoor.com/index.htm.

[33] GlassDoor Inc. *Junior Software Developer Salaries in Barcelona, Spain Area*. [Online; acceded 18-march-2018]. 2018. URL: https://www.glassdoor.com/Salaries/barcelona-junior-software-developer-salary-SRCH_IL.0,9_IM1015_KO10,35.htm.

[34] GlassDoor Inc. *Junior Software Engineer Salaries in Barcelona, Spain Area*. [Online; acceded 18-march-2018]. 2018. URL: https://www.glassdoor.com/Salaries/barcelona-junior-software-engineer-salary-SRCH_IL.0,9_IM1015_KO10,34.htm.

[35] GlassDoor Inc. *Qa Engineer Salaries in Barcelona, Spain Area*. [Online; acceded 18-march-2018]. 2018. URL: https://www.glassdoor.com/Salaries/barcelona-qa-engineer-salary-SRCH_IL.0,9_IM1015_KO10,21.htm.

[36] GlassDoor Inc. *Senior Project Engineer Salaries in Barcelona, Spain Area*. [Online; acceded 18-march-2018]. 2018. URL: https://www.glassdoor.com/Salaries/barcelona-senior-project-engineer-salary-SRCH_IL.0,9_IM1015_KO10,33.htm.

[37] Google Inc. *Google Official Web*. [Online; accessed 5-March-2018]. 2018. URL: https://scholar.google.com/.

[38] Dr. Ricard Gavaldà Mestre. *Dr. Ricard Gavaldà Mestre, web page*. [Online; accessed 5-March-2018]. 2018. URL: https://www.lsi.upc.edu/~gavalda/.

[39] Scitable by Nature Education. *Scientific Papers*. [Online; accessed 5-March-2018]. 2018. URL: https://www.nature.com/scitable/topicpage/scientific-papers-13815490.

[40] DBLP Team. *DBLP Official Web, Computer Science bibliography*. [Online; accessed 5-March-2018]. 2018. URL: http://dblp.uni-trier.de/.

[41] UPC. *Universitat Politècnica de Catalunya - BarcelonaTech (UPC), web page*. [Online; accessed 5-March-2018]. 2018. URL: http://www.upc.edu/ca.

[42] UPC-FIB. *Data mining, GRAU-MD*. [Online; accessed 5-March-2018]. 2018. URL: https://www.fib.upc.edu/en/studies/bachelors-degrees/bachelor-degree-informatics-engineering/curriculum/syllabus/MD.

[43] UPC-LARCA. *Laboratory for Relational Algorithmics, Complexity and Learning, web page*. [Online; accessed 5-March-2018]. 2018. URL: https://recerca.upc.edu/larca/en.

[44] Wikipedia. *Angular (framework) — Wikipedia, La enciclopedia libre*. [Internet; descargado 4-marzo-2018]. 2018. URL: https://es.wikipedia.org/w/index.php?title=Angular_(framework)&oldid=105732867.

[45] Wikipedia. *XPath — Wikipedia, La enciclopedia libre*. [Internet; descargado 21-junio-2018]. 2018. URL: https://es.wikipedia.org/w/index.php?title=XPath&oldid=108190428.

[46] Wikipedia contributors. *Author name disambiguation — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-June-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Author_name_disambiguation&oldid=845180399.

[47] Wikipedia contributors. *Bag-of-words model — Wikipedia, The Free Encyclopedia*. [Online; accessed 12-June-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Bag-of-words_model&oldid=832576977.

[48] Wikipedia contributors. *B-tree — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-June-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=B-tree&oldid=845377536.

[49] Wikipedia contributors. *Cascading Style Sheets — Wikipedia, The Free Encyclopedia*. https://en.wikipedia.org/w/index.php?title=Cascading_Style_Sheets&oldid=846176014. [Online; accessed 21-June-2018]. 2018.

[50] Wikipedia contributors. *Graph database — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-June-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Graph_database&oldid=846219260.

[51] Wikipedia contributors. *Model–view–controller — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-June-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Model%E2%80%93view%E2%80%93controller&oldid=846717749.

[52] Wikipedia contributors. *Quality assurance — Wikipedia, The Free Encyclopedia*. [Online; accessed 20-June-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Quality_assurance&oldid=846306822.

[53] Wikipedia contributors. *Reactive programming — Wikipedia, The Free Encyclopedia*. [Online; accessed 21-June-2018]. 2018. URL: https://en.wikipedia.org/w/index.php?title=Reactive_programming&oldid=846748588.

[54] Vincent Blondel. *The Louvain method for community detection in large networks*. [Online; accessed 20-June-2018]. URL: https://perso.uclouvain.be/vincent.blondel/research/louvain.html.

[55] Arnaud Browet. *Fast community detection using local neighbourhood search*. [Online; accessed 20-June-2018]. URL: https://perso.uclouvain.be/arnaud.browet/.

[56] *Gensim*. URL: https://radimrehurek.com/gensim/.

[57] *GROBID*. URL: https://grobid.readthedocs.io/en/latest/Introduction/.

[58] *Mendely*. URL: https://www.mendeley.com/.

[59] *Mozilla Fundation*. URL: https://www.mozilla.org/es-ES/foundation/.

[60] *Neo4j*. URL: https://neo4j.com/.

[61] *NLTK - Natural Language Toolkit*. URL: https://www.nltk.org/.

[62] *NumPy*. URL: http://www.numpy.org/.

[63] Laurel Orr y Jennifer Ortiz. «Clustering with the DBLP Bibliography to Measure External Impact of a Computer Science Research Area». En: (). URL: https://homes.cs.washington.edu/~jortiz16/images/MLProjectPaper.pdf.

[64] *Quick download IEEE paper*. URL: https://techqe.blogspot.com/2009/09/quick-download-ieee-paper.html.

[65] Tomas Mikolov Quoc Le. *Distributed Representations of Sentences and Documents*. URL: https://cs.stanford.edu/~quocle/paragraph_vector.pdf.

[66] *Reactive Extensions Library for JavaScript*. URL: https://rxjs-dev.firebaseapp.com/.

[67] *RedHat*. URL: https://www.redhat.com/es.

[68] *Scikit-learn Clustering algorithms*. [Online; accessed 21-June-2018]. URL: http://scikit-learn.org/stable/modules/clustering.html.

[69] *scikit-learn Machine Learning in Python*. URL: http://scikit-learn.org/stable/.

[70] *TensorFlow Word2Vec tutorial*. URL: https://www.tensorflow.org/tutorials/word2vec.

[71] *TypeScript Language*. URL: https://www.typescriptlang.org/.

[72] Liu Wanli y col. «Author name disambiguation for PubMed». En: *Journal of the Association for Information Science and Technology* 65.4 (), págs. 765-781. DOI: 10.1002/asi.23063. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/asi.23063. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/asi.23063.

[73] *What are the state of the art community detection algorithms for very large graphs?* [Online; accessed 20-June-2018]. URL: https://www.researchgate.net/post/What_are_the_state_of_the_art_community_detection_algorithms_for_very_large_graphs.

[74] *YouTube*. URL: https://www.youtube.com/.