Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓN EN TECNOLOXÍAS DA INFORMACIÓN

Metabuscador de documentos científicos

Estudiante: Miguel Oliveira Carballo

Dirección: Diego Fernández Iglesias

Fidel Cacheda Seijo

A Coruña, Septiembre de 2023

A mis familiares y amigos

Agradecimientos

Me gustaría agradecer a toda mi familia y amigos por apoyarme siempre, en especial a mi madre y padre, y a mis tutores por ayudarme a sacar adelante este trabajo. También agradecer a todos aquellos que, de una forma u otra, hicieron estos cuatro años más llevaderos. Gracias a todos.

Resumen

En la actualidad existen una gran cantidad de documentos científicos provenientes de diferentes puntos geográficos. Estos son publicados por diferentes editoriales y, a su vez, en revistas en relación con el contenido a tratar o área en la que se trabaja. Empresas como *Elsevier* o *Clarivate* exponen dichos artículos en bases de datos accesibles para los usuarios, permitiendo realizar distintas consultas de diferentes complejidades para filtrar y obtener los documentos deseados.

Con este proyecto se busca desarrollar un metabuscador que, a partir del uso de los servicios de APIs proporcionados por los proveedores previamente mencionados, permita al usuario acceder a dichas bases de datos mediante una aplicación web con una interfaz de usuario simple y diversas opciones para concretar la consulta deseada. Adicionalmente, se pretende añadir una herramienta de notificación que avisará al usuario de nuevas publicaciones que coincidan con las consultas que considere de interés. Para ello, se buscará emplear tecnologías a la orden del día en lo que se refiere al desarrollo de aplicaciones web en conjunto con los conocimientos adquiridos a lo largo de toda la estancia en la carrera.

A lo largo de este escrito se detallaran todos los aspectos relativos al desarrollo del proyecto, desde su concepción hasta el producto final junto con los pasos y decisiones tomadas para llegar hasta tal punto. Finalmente, se concluirá con una recapitulación de los hechos y las mejoras que el autor considera oportunas de cara al futuro de la aplicación.

Abstract

Nowadays, there are a large number of scientific documents from different locations arround the world. These are released by different publicators and, at the same time, in journals in relation to the content to be discussed or the ambit in which they work. Companies such as *Elsevier* or *Clarivate* expose these articles in databases accessible to users, allowing users to make different queries with various complexities, and get what they desire.

This proyect seeks to develop a metasearcher that, bassed on the use of API services provided by the previously mentioned providers, allows the user to access these databases through a web application with a simple user interface and different options to specify the most appropiated query. Additionally, it is intended to add a notification tool that will inform the user about new publications matching queries saved by herself. To do this, the objective will be to use current technologies regarding the development of web applications in conjunction with the knowledge acquired throughout the stay in the degree.

Throughout this writing, all aspects related to the development of the project will be detailed, from its conception to the final product along with the steps and decisions taken to get to that point. Finally, it will conclude with a recapitulation of the facts and the improvements that the autor considers appropiate for the future of the application.

Palabras clave:

- Metabuscador
- API
- Documentos científicos
- Aplicación web
- Sistema de notificaciones
- Código libre
- *Scopus*
- *Web of Science*

Keywords:

- Metasearcher
- API
- Scientific articles
- Web application
- Notification system
- Open Source
- *Scopus*
- *Web of Science*

Índice general

Índice de figuras

Índice de tablas

Capítulo 1
Introducción

Esta memoria trata sobre el desarrollo de una aplicación web que, mediante el uso de APIs, permite conectar y acceder a documentos publicados en varias revistas de diferentes editoriales. A lo largo del escrito se explicará el proceso seguido, los problemas encontrados así como las decisiones tomadas a lo largo del proyecto. Durante este capítulo se realizará una breve introducción al tema a tratar así como un pequeño resumen de como será la estructura del resto del documento.

1.1 Motivación

Internet se ha convertido en una fuente de datos inmensa donde cualquiera puede examinar gran variedad de información. En la actualidad, es habitual utilizar diferentes herramientas informáticas para extraer de forma rápida y sencilla esos contenidos. Uno de los ejemplos más habituales de este tipo son los buscadores, herramientas en línea que permiten a los usuarios obtener resultados relevantes en función de las consultas que ellos mismos generen. Un gran número de dispositivos disponen de navegadores que permiten conectar con estas herramientas, convirtiendo así la web en algo similar a una gran enciclopedia.

Este tipo de herramientas están presentes también en el ámbito de la documentación científica, aunque con menor número de posibilidades que otros campos. *Elsevier* o *Clarivate* ofrecen buscadores con numerosas posibilidades de filtrado para que así los usuarios encuentren la publicación que más se aproxime a sus inquietudes. Aún así, es posible que, por falta de acuerdos con otras empresas o por cualquier otro motivo, ciertos documentos no se muestren en una u otra alternativa. Así pues, se pretende solventar este problema mediante la implementación de un metabuscador que englobe las herramientas mencionadas, facilitando así la búsqueda.

1.2 Alcance y objetivos

El objetivo principal de este proyecto es combinar dos conocidos buscadores a los cuales la Universidad de A Coruña está suscrito (*Scopus* y *Web of Science*) para proporcionar una página web que contenga un metabuscador que permita al usuario realizar consultas que se traduzcan de forma automática al lenguaje empleado por sendas bases de datos. Las funcionalidades deseables son las siguientes:

- Una aplicación que permita realizar consultas por palabras clave de forma genérica a las bases de datos sin la necesidad de conocer la sintaxis empleada por estas. Los resultados deben contener información básica como título y autor, además de otros a mayores que sería conveniente añadir para mayor completitud, como número de veces que ha sido citado, tipo de publicación o revista, libro o conferencia de la que proviene.
- Una aplicación que permita realizar consultas por palabras clave en campos concretos como autor, título o lengua empleada entre otras posibilidades. De nuevo, el usuario no debe tener la necesidad de conocer la sintaxis empleada por las bases de datos. Los resultados deben contener información básica tal y como se planteo en el punto anterior.
- Una aplicación para guardar consultas. Adicionalmente, se debe disponer de un método de mensajería que permita notificar al usuario. Se podrá acceder a las consultas guardadas así como a los resultados correspondientes y también se podrá eliminar los elementos guardados.

Con el objetivo de que cada individuo que use el sistema pueda acceder a las funcionalidades completas de la aplicación para guardar consultas, debe definirse a mayores un sistema de gestión de usuarios que permita tanto el registro como el inicio de sesión.

1.3 Estructura de la memoria

La memoria se divide en diez capítulos, marcados por el proceso de investigación y desarrollo llevados a cabo para obtener el producto esbozado previamente. A continuación, se describe brevemente el contenido:

1. **Introducción**. Capítulo actual. Se introduce el trabajo, su alcance y motivaciones.
2. **Herramientas y tecnologías empleadas**. En él se expondrá todo aquello empleado durante el transcurso del proyecto.
3. **Estado del arte**. En este capítulo se analizarán otras piezas software ya existentes empleadas para resolver el mismo problema.

4. **Metodología de desarrollo**. En él se definirá como se desarrollará el proyecto, indicando la metodología empleada y cómo se aplica.
5. **Análisis de las APIs empleadas**. En él se analizarán las distintas APIs ofrecidas por *Clarivate* y *Elsevier*, haciendo especial hincapié en las más convenientes.
6. **Iteración 1**. Durante este capítulo se explicará el proceso realizado para la producción del buscador simple.
7. **Iteración 2**. Partiendo del resultado del capítulo previo, se explicará el proceso seguido para desarrollar el buscador avanzado junto con los problemas encontrados.
8. **Iteración 3**. Durante este capítulo se explicarán los pasos dados durante esta tercera etapa del proyecto, la cual consistió en la combinación de ambos buscadores, así como las complicaciones y reconsideraciones que se hicieron sobre ella.
9. **Iteración 4**. Último capítulo centrado en el desarrollo del proyecto, en el que se desarrollan las funcionalidades de guardar consultas.
10. **Conclusiones**. Capítulo final en el que se analizará el trabajo realizado y qué pautas o mejoras se deberían hacer en caso de dar continuidad al proyecto.

A mayores, para todo aquel lector interesado en la aplicación final, se dispone de un repositorio de GitHub de el cual se puede descargar la aplicación y montarla en local. Recordar que para el correcto funcionamiento, como se verá más adelante, son necesarias claves para el uso de las APIs, así como estar conectado a una red de una institución que disponga de la suscripción a los recursos empleados.

El enlace al repositorio es `https://github.com/moliveirac/metasearcher_proyect.git`.

Capítulo 2

Herramientas y tecnologías empleadas

En este capítulo se explicarán de forma resumida las diferentes herramientas empleadas en el proyecto.

2.1 Lenguajes de programación

A la hora de tomar la decisión sobre que lenguajes emplear durante el desarrollo de un proyecto, en cierta forma, se determina en gran parte cual va a ser su resultado. Dada la casuística de la aplicación a desarrollar, una página web, algunas posibilidades son inevitables.

A continuación se mencionarán y describirán brevemente los lenguajes empleados, así como los motivos detrás de su uso.

Python

Python [1] es un lenguaje de alto nivel de programación interpretado, multiparadigma (programación orientada a objetos, imperativa y funcional) y de código abierto. Este se basa en usar una sintaxis lo más clara posible. Al ser interpretado, se puede ejecutar en diversos sistemas operativos.

Es un lenguaje ampliamente usado en campos como el desarrollo de aplicaciones, análisis de datos o *machine learning*, entre otros. Es por ello que encaja perfectamente con el proyecto a desarrollar: una aplicación web con cierto enfoque al análisis de datos.

HTML

HTML (de las siglas en inglés *HyperText Markup Language*) [2] es un lenguaje basado en etiquetas. Se emplea para definir el significado y estructura del contenido web. Este se suele

apoyar en otras tecnologías con el fin de otorgar al contenido una apariencia más cuidada (CSS) o con el fin de proporcionar mayor funcionalidad al recurso (JavaScript). Resulta esencial para el desarrollo de una aplicación web.

Es un estándar a cargo del *World Wide Web Consortium* (o *W3C*), comité encargado de implementar tecnologías uniformes en el uso y desarrollo de Internet.

JavaScript

JavaScript [3] es un lenguaje de programación de alto nivel, multiparadigma e interpretado, usado comúnmente para otorgar la posibilidad, a las páginas web estáticas en lenguaje HTML, un funcionamiento dinámico, permitiendo así mejorar la interfaz cliente.

Es también usado en aplicaciones externas a la web. No obstante, no es la aplicación que se le dará en este caso.

CSS

CSS (de sus siglas en inglés *Cascading Style Sheets*) [4] es el lenguaje de estilos utilizado para determinar la presentación de documentos definidos en lenguajes como XML (y otros basados en él, como SVG) o HTML, describiendo cómo un elemento debe ser renderizado. Posee una especificación estandarizada por parte del W3C.

2.2 *Frameworks* y librerías

En la actualidad se dispone de gran cantidad de *software* con diferentes posibilidades que, a su vez, se usa para generar más *software*, dando lugar así a comunidades colaborativas como las de tecnologías de código abierto. En este caso, se emplearán tecnologías que siguen dicho esquema, las cuales se mencionan a continuación.

Django

Django [5] es un *framework* de alto nivel cuyo objetivo es facilitar al desarrollador el diseño de páginas web. Destaca por la rapidez con la que se puede crear una aplicación completamente funcional desde prácticamente cero, acompañado por una robusta seguridad ante ataques comunes a este tipo de software y su gran escalabilidad. Grandes empresas como Instagram o Mozilla han desarrollado sus aplicaciones empleando esta herramienta.

Con el objetivo de obtener un resultado lo más cercano a una interfaz simple y sencilla de usar, además de similar a los ejemplos consultados, se añadirán distintas librerías que facilitarán, en gran medida, el desarrollo de esta. Estas son:

- **django-crispy-forms**. Esta librería [6] permite, al usuario de Django, personalizar los formularios de forma sencilla mediante el uso de clases de Python, evitando así definirlos directamente en cada plantilla que se vayan a usar.
- **crispy-bootstrap5**. *Django crispy forms* ofrece una serie de elementos básicos para personalizar el formulario. Bootstrap 5 (véase 2.2) define nuevos *widgets*, como los *floating labels*, que, combinando HTML, JavaScript y CSS, proporcionan una interfaz más cuidada. *Crispy-bootstrap5* [6] permite al usuario incluirlos a los ya existentes de *Django crispy forms*.
- **django-crontab**. Con esta librería [7] se permite combinar Cron con Django, habilitando así la posibilidad de programar tareas periódicas a realizar por el servidor. Se usará principalmente para el envío de notificaciones.

Bootstrap

Bootstrap [8] es un *framework* de código abierto desarrollado por Mark Otto y Jacob Thornton, miembros del equipo de Twitter (ahora llamada *X*) cuando hicieron público su trabajo. Ofrece un conjunto de documentos CSS y Javascript para el desarrollo de páginas web adaptables. Es una solución sencilla para el problema del diseño *front-end* de este proyecto. En concreto, usaremos la versión Bootstrap 5, que a día de comienzo del proyecto es la más reciente.

Pandas

Pandas [9] es la librería que emplearemos para trabajar con los resultados obtenidos de las APIs de Scopus y Web of Science. Permite al usuario manipular datos de forma sencilla, rápida y eficiente, trabajar con una gran cantidad de formatos y ver de distintas maneras los resultados obtenidos, mediante tablas o incluso gráficos. Ciertas funcionalidades están implementadas en C para proporcionar resultados más optimizados. Es una de las herramientas más utilizadas para el análisis de datos y es de código abierto.

Psycopg2

Psycopg [10] es un adaptador de la base de datos PostgreSQL al lenguaje Python. Está implementado en C, complementando a la librería Libpq, la cual se usa para enviar consultas a la base de datos y recibir los resultados.

Psycopg será necesario para conectar Django (2.2) con la base de datos en la que se almacenarán los usuarios y demás datos relacionados con las consultas guardadas.

Requests

Requests [11] es una librería que permite realizar conexiones HTTP desde Python de forma rápida y sencilla. Será la librería a emplear para conectar la aplicación con las APIs que tanto Scopus como Web of Science proporcionan.

Elsapy

Elsapy [12] es un SDK mediante el cual podemos hacer conexiones a la base de datos de Scopus sin necesidad de elaborar las peticiones HTTP desde cero. No está garantizado su mantenimiento ni su correcto funcionamiento en todas las plataformas, pero para este caso concreto esto no supone mayor problema que prestar atención a posibles cambios de la API.

smtplib

SMTP, de sus siglas en inglés *Simple Mail Transfer Protocol*, es un protocolo TCP/IP empleado para el intercambio de mensajes de correo electrónico [13]. La librería smtplib permite utilizarlo de forma programática desde Python.

2.3 Herramientas de desarrollo

Otro punto esencial de cualquier proyecto son las herramientas usadas para el desarrollo del mismo. En la actualidad existen gran variedad de opciones, con sus diferentes ventajas e inconvenientes y que, de nuevo, definen en gran parte el resultado.

Durante esta sección se detallarán las herramientas empleadas.

Visual Studio Code

Visual Studio Code [14] es un editor de código disponible en plataformas como Windows, Linux o macOS diseñado inicialmente para trabajar con JavaScript, TypeScript y Node.js. Incluye soporte para depuración, control integrado en Git, autocompletado de código y refactorización de código entre otras ventajas. Además, permite añadir extensiones para trabajar en otros lenguajes, como Python; con otras tecnologías, como Docker; o con otras aplicaciones, como GitHub.

Se empleará esta herramienta apoyada en la extensión ***Dev Containers*** para desplegar de forma automática el entorno de producción en Docker.

Docker

Docker [15] es una herramienta multiplataforma que permite ejecutar aplicaciones en distintos entornos sin la necesidad de reconfigurarlas. Para ello, se basa en contenedores,

máquinas virtuales muy ligeras que se ejecutan sobre el *hipervisor* que Docker proporciona. Permitirá el despliegue de un entorno de pruebas de forma muy sencilla y rápida.

2.4 Sistemas de Gestión de Bases de Datos

Si se desea almacenar información de forma permanente, es necesario emplear algún sistema de almacenamiento de la información. Las bases de datos son una herramienta empleada comúnmente para este tipo de casos. A continuación se expone la empleada en este proyecto.

PostgreSQL

PostgresSQL [16] es una base de datos relacional de código abierto. Es una de las más usadas y fiables, con más de 35 años en desarrollo activo. Trata de seguir el estándar SQL, con 170 de las 179 características definidas en el *SQL:2016 Core*. Se empleará esta herramienta para alojar toda la información relacionada con usuarios y consultas guardadas.

2.5 APIs empleadas

Por último, se procede a explicar brevemente los servicios empleados en el proyecto. Sus diferentes posibilidades serán ampliamente explicadas en el capítulo 5.

Scopus

Elsevier es una editorial académica con sede en Ámsterdam, Países Bajos, especializada en contenido científico, técnico y médico. Incluye diversos productos como revistas, como *The Lancet* o la colección electrónica de ScienceDirect; y servicios, como la base de datos bibliográfica de resúmenes y citas de artículos Scopus o la herramienta para medir el rendimiento en investigación SciVal [17].

Elsevier dispone de un portal web con un conjunto de APIs que permiten a sus clientes acceder de forma programática a los datos en función del plan de subscripción contratado [18]. Algunos ejemplos disponibles son:

1. Datos relacionados con artículos publicados por revistas científicas como citas, metadatos y resúmenes, disponibles también en Scopus.
2. Acceso completo a documentos y libros publicados por Elsevier (en la plataforma de ScienceDirect).
3. Métricas sobre investigadores y sus publicaciones disponibles en SciVal.

En concreto, nos centraremos en los datos ofrecidos por Scopus, con la posibilidad de apoyarnos en las otras dos APIs mencionadas.

Web of Science

Clarivate es una empresa orientada al análisis de datos, incluyendo ámbitos como la investigación científica y académica, inteligencia farmacéutica y biotecnológica o la gestión de propiedad intelectual. Dispone de varios servicios dedicados al análisis de los campos previamente mencionados, como Web of Science, EndNote o Derwent Innovation, entre otros muchos [19].

Clarivate ofrece servicios como los mencionados anteriormente a través de APIs para proporcionar acceso a datos de forma programática a través de la página web *Clarivate Developer Portal* [20]. En nuestro caso resultan de especial interés las relacionadas con Web of Science, en concreto aquellas que permitan la búsqueda de artículos y obtención de metadatos relacionados.

Capítulo 3
Estado del arte

A continuación, se analizarán las principales fuentes de datos y sus buscadores junto con ejemplos de otras aplicaciones con propósitos similares al del proyecto que se considerasen de interés o se hayan tenido en cuenta a lo largo del desarrollo.

3.1 Herramientas ofrecidas por los servicios empleados

Las fuentes de datos proporcionadas por Elsevier y Clarivate son Scopus y Web of Science. Ambas constan de su respectiva aplicación web donde el usuario, combinando una serie de campos con las palabras clave que desea, puede filtrar los contenidos disponibles y obtener resultados acordes a sus necesidades.

3.1.1 Scopus

Scopus es la aplicación web proporcionada por la editorial Elsevier para acceder a documentación de carácter científico. Incluye más de 84 millones de archivos de alrededor de 7000 editoriales [21]. Dispone de una interfaz en la que el usuario, siempre y cuando pertenezca a algún plan de subscripción que se lo permita, puede realizar consultas sobre documentos, autores e instituciones. En concreto, para el desarrollo de esta aplicación, nos interesa especialmente el apartado de la interfaz de búsqueda de documentos, ya que en el enfoque inicial no se plantea permitir buscar autores o instituciones, tan solo filtrar por estos si fuese necesario. También resulta de interés los campos por los cuales se pueden realizar las consultas y las limitaciones aplicadas a estas, dado que serán las mismas o muy similares a las disponibles en las APIs, recogidas en la correspondiente sección de análisis en capítulos posteriores.

En la figura 3.1 vemos una captura de la página de Scopus después de autenticar un usuario en esta y conectarse vía la Universidad de A Coruña, como se indica en la esquina superior izquierda. Esta aplicación web dispone de un menú horizontal (recuadro 1) donde el usuario puede seleccionar el tema de su búsqueda. Estos son: documentos, autores, descubrir inves-

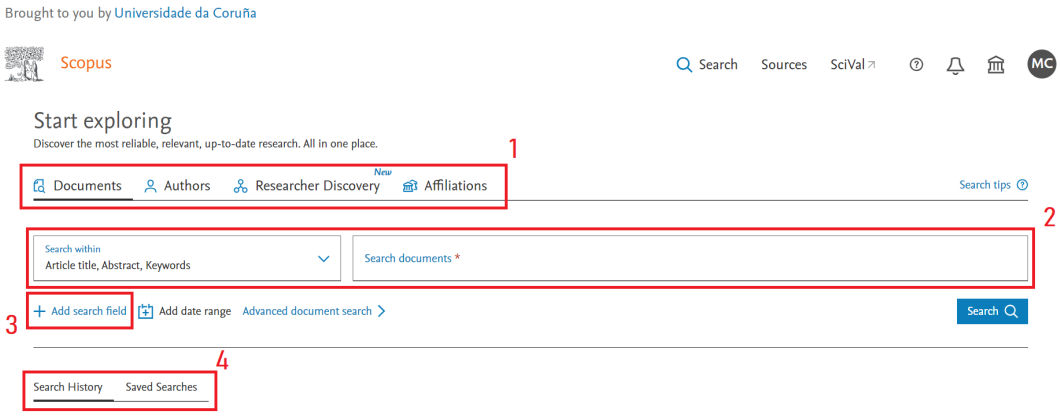


Figura 3.1: Página de Scopus en la sección de documentos.

Figura 3.2: Ejemplo de búsqueda de más de un campo.

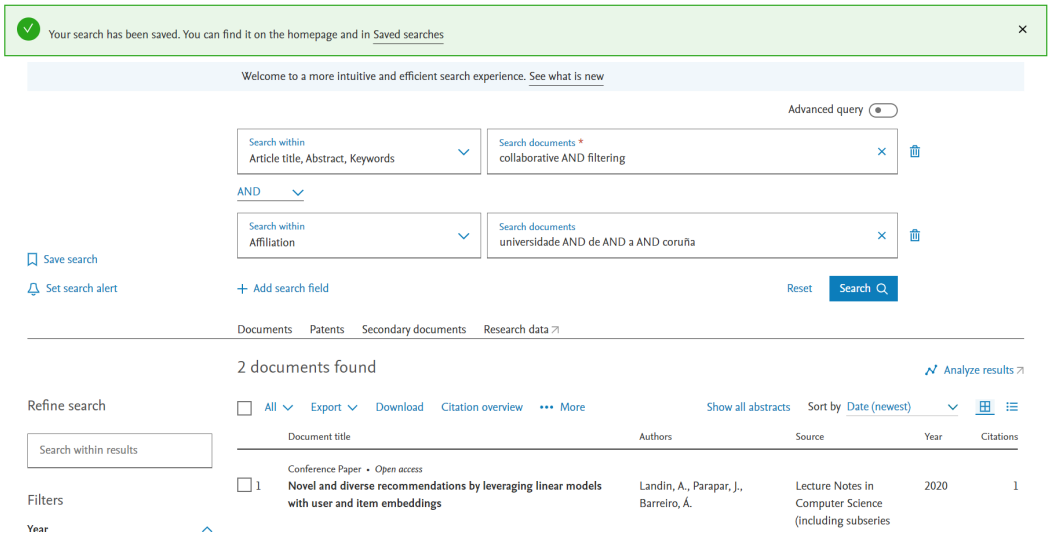


Figura 3.3: Captura de Scopus mostrando resultados de búsqueda tras guardar la consulta.

tigadores e instituciones. Dentro del menú de documentos vemos dos cuadros de entrada (recuadro 2): un desplegable donde se puede seleccionar el campo en el que se quiere buscar y un cuadro de texto donde se introducen las palabras clave relacionadas con el texto. Justo debajo, hay disponible un botón de añadir campos de búsqueda, lo cual añade dinámicamente otra fila de 2 cuadros de entrada separada de la anterior por un desplegable en el que se puede seleccionar el enlace booleano mediante el cual se combinan los campos. Para mayor claridad, se incluye un ejemplo de una búsqueda en la figura 3.2. Por último, vemos otro menú donde el usuario puede seleccionar entre su historial de búsqueda y las consultas guardadas y un botón de búsqueda para realizar la consulta contra la base de datos. Esta estructura no es exclusiva de Scopus. Más adelante se detallará la interfaz de Web of Science, la cual puede variar ligeramente en la presentación pero el funcionamiento es prácticamente el mismo.

En la figura 3.3 vemos lo que un usuario vería tras enviar una consulta, recibir una serie de resultados del servidor y guardarla. El apartado de búsqueda es muy similar al de 3.1. Este incluye una serie de botones adicionales para guardar la consulta, recibir notificaciones en caso de nuevas publicaciones y un botón para restablecer la búsqueda. La página consta de una segunda sección donde se muestran los resultados. Estos se muestran, bajo un criterio de ordenación modificable, con cierta información relevante, como título del documento, autores, fuente, año de publicación y veces que el documento ha sido citado. A mayores, presenta una columna a la izquierda donde el usuario puede añadir filtros. Por otra parte, el recuadro verde que se muestra en la zona superior de la captura confirma que el usuario ha guardado la consulta.

3.1.2 Web of Science

Web of Science (WoS) es la aplicación web proporcionada por Clarivate Analytics con un objetivo similar al de Scopus: facilitar el acceso a documentación científica. Consta de gran cantidad de documentos provenientes de diversas revistas científicas, libros y conferencias. El conjunto de documentos proporcionados por WoS se denomina *Web of Science Core Collection* e incluye los contenidos de las siguientes bases de datos [22]:

- Science Citation Index (SCI)
- Social Sciences Citation Index (SSCI)
- Arts & Humanities Citation Index (A&HCI)
- Index Chemicus
- Current Chemical Reactions
- Conference Proceedings Citation Index

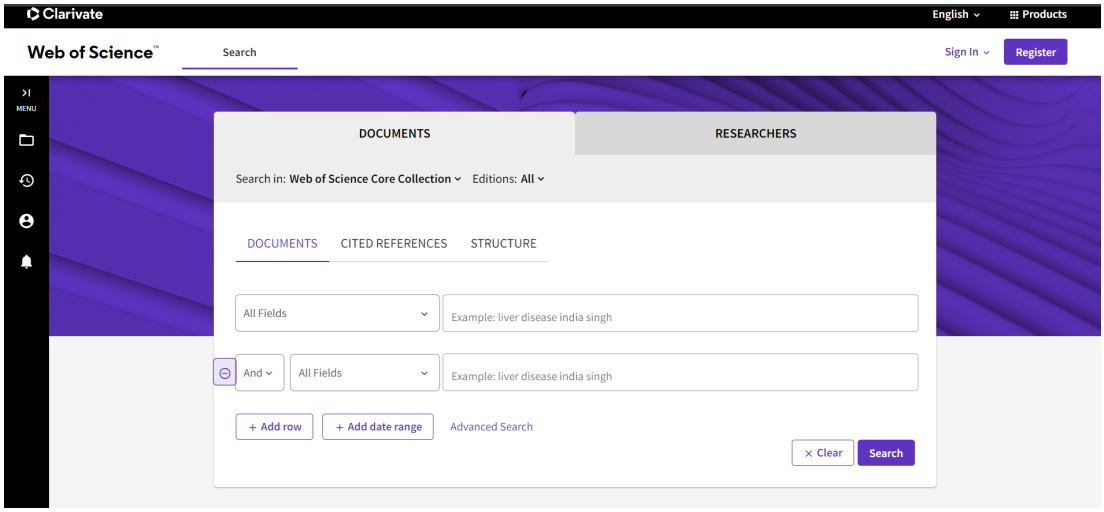


Figura 3.4: Página de Web of Science en la sección de documentos

De igual forma que con Scopus, es de especial interés el diseño de la interfaz empleado en esta aplicación web así como los campos disponibles y limitaciones en las consultas, que veremos más adelante.

El diseño es similar al de Scopus, como se puede comprobar en la figura 3.4. De nuevo hay un menú, aunque esta vez consta de dos opciones: documentos e investigadores. Solo se investigará la de documentos por lo mencionado anteriormente. En dicha sección, la primera fila está formada por un desplegable y una entrada de texto a los cuales, en las siguientes filas, se les añade otro desplegable para conectar los distintos campos empleados en la búsqueda por medio de un booleano. A continuación, se muestran 3 botones en la esquina inferior izquierda para añadir más filas, un rango de fechas o usar la búsqueda avanzada y otros 2 en la esquina inferior derecha para restablecer la búsqueda o enviar la consulta a la base de datos.

En la figura 3.5 vemos cómo la consulta se muestra como texto, aunque el usuario puede modificar la consulta haciendo clic sobre el texto, lo que abrirá un desplegable con la misma estructura que la expuesta en la figura 3.4. En los resultados se muestran tanto el título del documento como sus autores, fecha de publicación, fuente, páginas e identificadores relacionados y las primeras líneas del resumen, así como las citas y referencias al articulo y una serie de enlaces al documento o información relacionada. Además, el usuario dispone en la columna de la izquierda de diversas opciones para filtrar los resultados obtenidos, de la misma forma que Scopus. Se pueden modificar tanto los resultados por página como el método de ordenación. Por último, como se puede consultar en la figura 3.6, esta web también ofrece la opción de guardar una consulta, con la posibilidad de marcar la opción de notificar si se producen nuevos resultados.

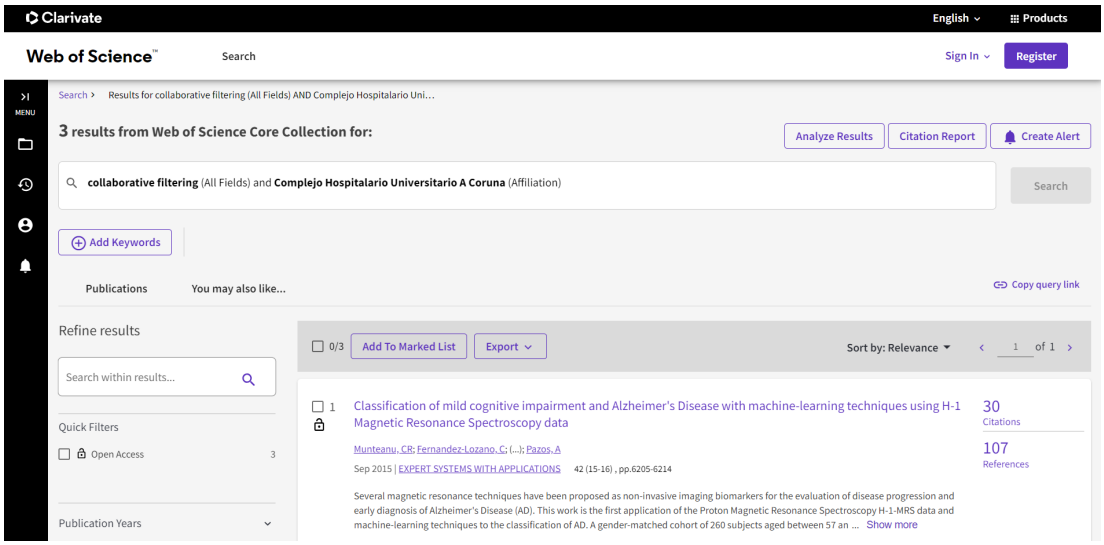


Figura 3.5: Resultados de búsqueda de documentos de Web of Science

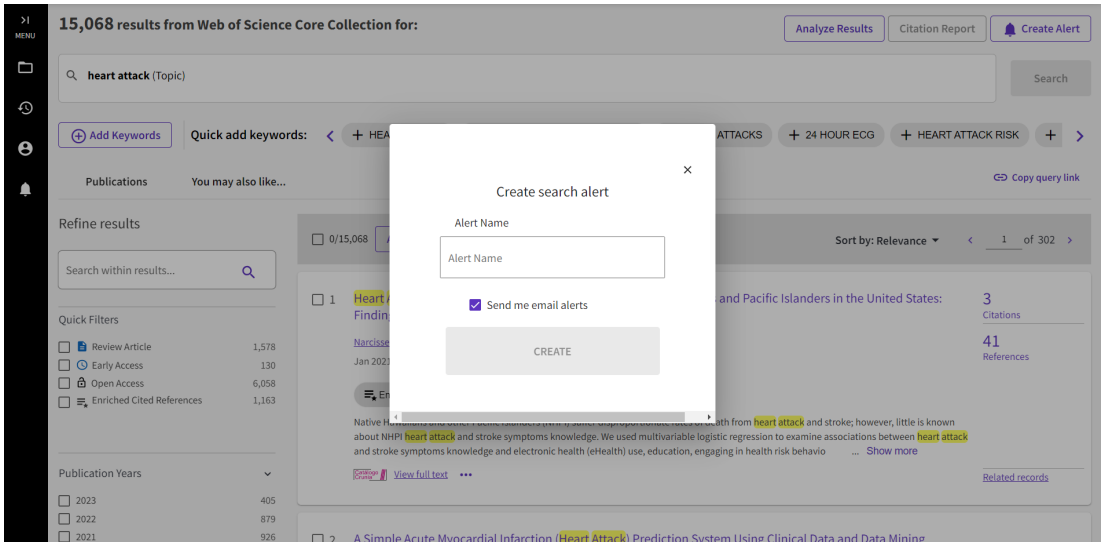


Figura 3.6: Guardar una consulta en Web of Science

3.2 Metabuscadores y herramientas relacionadas

En esta sección se verán otras herramientas similares al objetivo definido previamente para la aplicación a diseñar que no guardan relación directa con las bases de datos a usar.

1. Google Scholar (*Google académico* en español) [23] es la rama orientada a la documentación científica de Google, con el cual comparte, en gran medida, la interfaz. Cabe recalcar el menú de búsqueda avanzada, el cual difiere de los vistos previamente. En la figura 3.7 podemos ver cómo consta de una serie de entradas de texto acompañadas de unas breves frases las cuales les proporcionan contexto. El usuario puede cubrir los recuadros que considere necesario y luego realizar la búsqueda pulsando sobre el botón azul de la lupa en la esquina superior derecha de la sección.

2. Otra referencia interesante viene dada por el artículo de investigación escrito por Neil R. Smalheiser y su equipo [24]. En ella se expone un metabuscador que traduce y envía directamente las consultas a 5 bases de datos: PubMed, EMBASE, CINAHL, PsycINFO y Cochrane. Ofrece información relevante de cara al desarrollo del proyecto como estructura, posibilidades o problemas que se hayan podido encontrar.

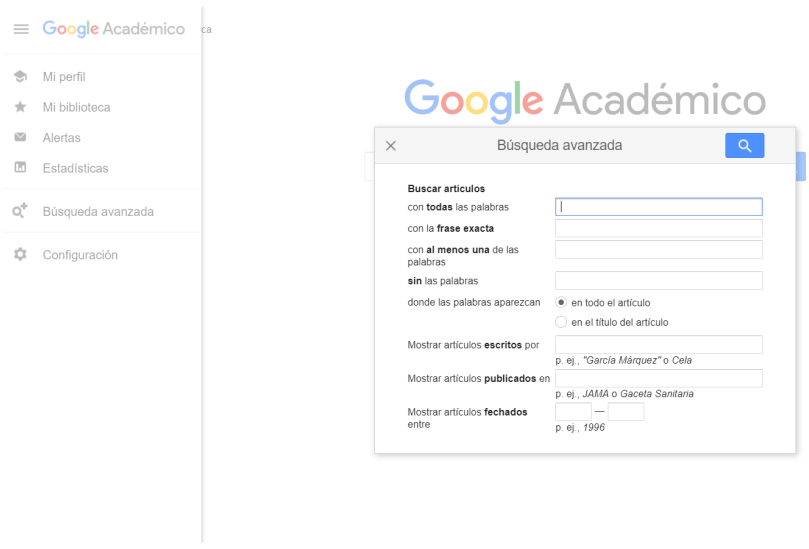


Figura 3.7: Búsqueda avanzada de Google Scholar.

Capítulo 4

Metodología y planificación de desarrollo

Durante este capítulo se tratará todo lo relativo a la metodología empleada, así como la planificación y estimación de costes del proyecto.

4.1 Especificación de requisitos

Un requisito, en el ámbito del desarrollo software, describe aquello que el sistema debe hacer, es decir, las funcionalidades, características y restricciones que la aplicación debe cumplir [25]. Estos son esenciales para determinar qué es lo que el cliente desea obtener, así como para la planificación del desarrollo.

Los requisitos globales del sistema pueden dividirse en dos grupos: requisitos funcionales y no funcionales.

4.1.1 Requisitos funcionales

Los requisitos funcionales son aquellos que describen las características y funcionalidades que el programa debe tener para así satisfacer las necesidades del cliente o, en última instancia, del usuario. En el caso de este proyecto, se definen los siguientes:

- Buscador de documentos. Permite al usuario, a partir de una serie de valores (bien palabras clave, identificadores u otros valores propios de un documento científico) realizar una consulta y obtener una lista de resultados. El objetivo es mostrar una información similar a los resultados disponibles en las bases de datos de Scopus y Web of Science.
- Guardar consultas. Permite al usuario almacenar en el sistema las consultas que considere de interés, permitiéndole acceder a ellas más adelante.

- Sistema de notificaciones. El sistema notifica a los usuarios sobre aquellas consultas guardadas si se detectan nuevos resultados no consultados.
- Sistema de autenticación de usuarios. Permite a los usuarios de la aplicación registrar sus datos para emplear las diferentes funcionalidades previamente mencionadas.

4.1.2 Requisitos no funcionales

Los requisitos no funcionales son aquellos que describen cualidades no relacionadas directamente con las funciones del sistema. Buscan definir cómo debe ser el resultado en lugar de lo que debe hacer. Al contrario que los requisitos funcionales, estos deben estar siempre presentes en el sistema, por lo que se tienen en cuenta en todo el proyecto. En este proyecto se han definido los siguientes:

- Uso sencillo. Para garantizar un funcionamiento satisfactorio, el sistema debe resultar intuitivo para el usuario. Se debe tener en cuenta que esta aplicación está orientada a académicos de diversos ámbitos, no solo de la informática.
- Diseño adaptable. Es importante adaptar la aplicación para que pueda ser empleada en diferentes dispositivos de diferentes resoluciones.
- Seguridad. Es necesario cumplir con unos parámetros de seguridad mínimos para aplicaciones web.

4.2 Metodología escogida

La metodología escogida para el desarrollo de cualquier proyecto debe escogerse teniendo en cuenta tanto su estructura como los tiempos definidos para la entrega del mismo. En este caso concreto, dichos tiempos vienen marcados por la entrega del Trabajo de Fin de Grado. Además, la estructura es fácilmente divisible en productos tangibles y funcionales que se pueden proporcionar a los clientes, en este caso los directores del proyecto. Es por ello que se interpreta como óptimo un desarrollo ágil y rápido, proporcionando resultados tangibles y operativos con el fin de obtener comentarios por parte de las partes implicadas y aplicando los cambios que se consideren oportunos.

Tras consultar distintas opciones que cumpliesen con las características descritas, se seleccionó el desarrollo iterativo e incremental como metodología a seguir.

4.2.1 Definición

El desarrollo iterativo e incremental consiste en a partir de los distintos requisitos de un producto, tratar de agruparlos y desarrollarlos como pequeños proyectos. Estos se conocen

como iteraciones y su finalidad es proporcionar una versión funcional al cliente en períodos cortos de tiempo, la cual permite obtener el punto de vista del cliente y modificarla según sus necesidades u opiniones. De esta forma, el resultado va evolucionando y desarrollándose hasta alcanzar lo que se definió en un principio.

Cada iteración se pueden subdividir en 4 fases: análisis, diseño, implementación y pruebas, descritas brevemente a continuación:

- Análisis. En esta fase se determina qué es lo que se va a hacer durante la iteración, es decir, las tareas necesarias para proporcionar el producto deseado.
- Diseño. En esta fase se define cómo se quiere presentar el resultado a nivel de cómo el usuario lo ve e interactúa con el mismo (interfaz), así como la estructura sobre la que se va a desarrollar el sistema.
- Implementación. Se corresponde con la parte de ”hacer” el proyecto. En ella se produce el código necesario para obtener el producto previamente definido
- Pruebas. En esta fase, como su nombre indica, se comprueba el correcto funcionamiento del sistema.

4.2.2 Aplicación al proyecto

Para el desarrollo, se decidió subdividir el proyecto en cuatro iteraciones, precedidas todas ellas de una fase previa donde se analizarían los servicios a emplear, es decir, las APIs ofrecidas por Elsevier y Clarivate. A continuación, se explica brevemente la distribución de la carga de trabajo:

1. En primer lugar, una fase de análisis de las APIs a emplear, que se enfocaría en obtener toda la información posible sobre las alternativas disponibles y recopilarla para su uso más adelante.
2. La primera iteración se planteó con el objetivo de ofrecer una interfaz mínima y funcional, buscando comentarios por parte de la dirección y enfocando el resto del desarrollo en base a dicha información. También es la fase en la que, dadas las tecnologías empleadas, se debía definir la estructura del proyecto, por lo que complicar la implementación podría haber sido contradictorio y llevar más tiempo del necesario. Finalmente, se decidió implementar únicamente el buscador simple empleando una de las APIs, a determinar según el análisis realizado de las mismas (que finalmente fue Scopus).
3. La segunda iteración se enfocó en desarrollar el buscador avanzado con la misma API que se había empleado previamente. Esta supondría una mayor carga en la parte de la

implementación frente a la reducción de este mismo aspecto en otros ámbitos como el análisis o el diseño.

4. La tercera iteración se enfocó en combinar los resultados de ambos buscadores. Se consideró esta parte de la implementación como la más complicada del proyecto, por lo que se le asignó un rango de tiempo superior.

5. La cuarta y última iteración se decidió enfocarla en añadir las dos funcionalidades restantes. Esta decisión fue tomada en parte porque era imposible implementarlas sin haber incluido todo lo correspondiente a las fases previas. A este hecho se le sumó que la implementación de las mismas por separado podría llegar a ser demasiado trivial dadas las facilidades ofrecidas por las diferentes herramientas empleadas, por lo que se decidió combinarlas en un solo incremento.

4.3 Planificación

Una vez aclarada la metodología a emplear y los requisitos del sistema, se realizó una planificación estimada de cómo debía desarrollarse el proyecto, siguiendo siempre lo definido previamente. El diagrama de la figura 4.1 ilustra el resultado.

Para la estimación se tuvo en cuenta una persona, el autor, para desenvolverse en los diferentes roles de analista, diseñador y desarrollador. Se consideró que se haría un trabajo de cuatro horas diarias, con un total de tiempo estimado de unas 300 horas.

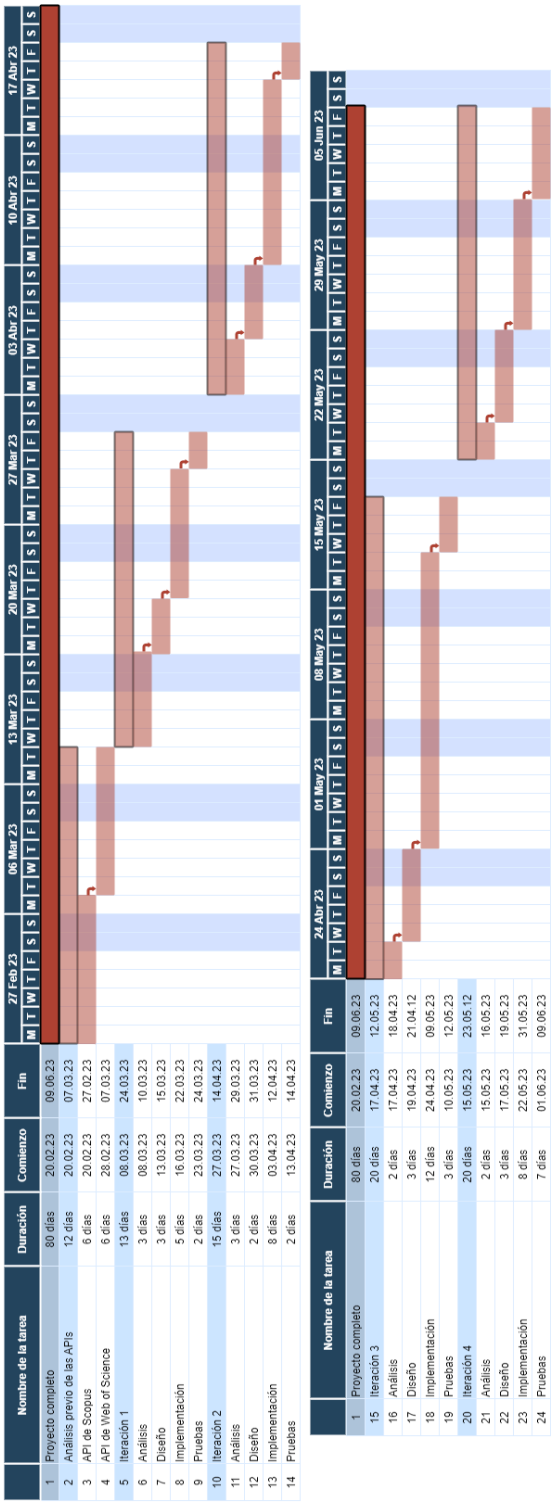
También se tuvo en cuenta otras dos personas más, los directores, que cumplieron con el rol de asesores del proyecto. Se estimó un trabajo semanal de una hora donde el autor plantearía diversas dudas sobre la realización de sus tareas. A mayores, se incluyeron las reuniones tras cada iteración, que se plantearon con una duración aproximada de dos horas cada una

4.4 Estimación de costes

Para un correcto planteamiento del proyecto, es necesario realizar una estimación aproximada de los costes asociados al correcto desarrollo del mismo, siguiendo la planificación realizada. Con el objetivo de contemplar este aspecto de forma completa, se decidió dividir la evaluación según los elementos que la componen.

4.4.1 Costes *hardware* y *software*

Respecto al *software* empleado durante el desarrollo del proyecto, todas las herramientas empleadas son de código abierto, por lo que, en este aspecto el coste es nulo. Tampoco se



	Nombre de la tarea	Duración	Comienzo	Fin
1	Proyecto completo	80 días	20.02.23	09.06.23
2	Análisis previo de las APIs	12 días	20.02.23	07.03.23
3	API de Scopus	6 días	20.02.23	27.02.23
4	API de Web of Science	6 días	28.02.23	07.03.23
5	Iteración 1	13 días	08.03.23	24.03.23
6	Análisis	3 días	08.03.23	10.03.23
7	Diseño	3 días	13.03.23	15.03.23
8	Implementación	5 días	16.03.23	22.03.23
9	Pruebas	2 días	23.03.23	24.03.23
10	Iteración 2	15 días	27.03.23	14.04.23
11	Análisis	3 días	27.03.23	29.03.23
12	Diseño	2 días	30.03.23	31.03.23
13	Implementación	8 días	03.04.23	12.04.23
14	Pruebas	2 días	13.04.23	14.04.23

	Nombre de la tarea	Duración	Comienzo	Fin
1	Proyecto completo	80 días	20.02.23	09.06.23
15	Iteración 3	20 días	17.04.23	12.05.23
16	Análisis	2 días	17.04.23	18.04.23
17	Diseño	3 días	19.04.23	21.04.12
18	Implementación	12 días	24.04.23	09.05.23
19	Pruebas	3 días	10.05.23	12.05.23
20	Iteración 4	20 días	15.05.23	23.05.12
21	Análisis	2 días	15.05.23	16.05.23
22	Diseño	3 días	17.05.23	19.05.23
23	Implementación	8 días	22.05.23	31.05.23
24	Pruebas	7 días	01.06.23	09.06.23

Figura 4.1: Diagrama de Gantt del proyecto

tendrá en cuenta el coste de las APIs ya que, dado que tanto la suscripción a Scopus y Web of Science como las claves empleadas durante el desarrollo tampoco tuvieron un gasto asociado.

Por la parte del *hardware*, fueron empleados diferentes elementos durante el desarrollo del proyecto. En primer lugar, el portátil con el que se realizó tanto el análisis como la programación de la aplicación cumple con las siguientes características:

- Marca y modelo: MSI GF63 Thin.
- Procesador: Intel Core i7-11800H (11ª Generación).
- Memoria RAM: 16 GB, 3200 MHz, DDR4.

El precio a día de la publicación de esta memoria ronda los 860,00€ (IVA incluido).

Para el cálculo de la amortización del mismo se empleará el *método lineal o de cuota fija según tablas*. Partiendo del precio previamente mencionado y aplicando un porcentaje del 25% sobre dicho valor por año, se obtiene un coste resultante, para cuatro meses de trabajo, de 71,67€.

También se emplearon otros elementos que se pueden englobar en esta clasificación, como una máquina virtual para poder realizar conexiones a los diferentes productos de Elsevier y Clarivate, además de ser el entorno de despliegue para que los directores del proyecto pudiesen probar el resultado de cada iteración. Dado que el servicio fue proporcionado por la UDC y no tiene un coste real asociado, se empleará como referencia los precios proporcionados por un proveedor externo, OVHcloud, los cuales para una instancia de las mismas características que la empleada (plan *Elite*) suponen 34,50€/mes (IVA incluido). Si tenemos en cuenta 4 meses de trabajo, se obtiene un total de 138,00€.

El total de coste *hardware* y *software* asciende a los 209,67€.

4.4.2 Costes de personal

El trabajo completo fue desarrollado por una única persona, que realizó los diferentes papeles de analista, programador y *tester*. A esto debe sumársele las horas realizadas por los directores del proyecto como asesores.

La tabla 4.1 muestra el total asociado a este ámbito en base a la planificación realizada. Los valores aplicados fueron calculados en base al BOE [26].

4.4.3 Coste total

Para el calculo total se sumarán los resultados calculados previamente. No se tendrán en cuenta otros costes, como los indirectos. El resultado final es de:

$$4486,00€ + 209,67€ = 4695,67€$$

Horas	Rol	Coste por hora	Subtotal
300	Programador Junior	12,50 €/h	3750,00 €
46	Asesor	16,00 €/h	736,00 €
Total:			4486,00 €

Tabla 4.1: Costes de personal

4.5 Seguimiento

A la hora de aplicar la planificación surgieron diferentes problemas que provocaron retrasos. Los resultados se muestran en la tabla 4.2

Fase	Horas estimadas	Horas realizadas	Desviación
Análisis APIs	48	48	0
Iteración 1	52	52	0
Iteración 2	60	76	16
Iteración 3	80	108	28
Iteración 4	60	60	0

Tabla 4.2: Seguimiento

Las desviaciones en la segunda y tercera iteración fueron causadas por problemas durante la implementación, los cuales se detallarán en el apartado correspondiente a dicha fase.

A los retrasos ya mencionados se le sumaron otros problemas personales al autor, lo que supuso pausar el proyecto y así aumentar, a mayores, un total de tres semanas la duración del proyecto.

Capítulo 5

Análisis de las APIs a emplear

En este capítulo se recoge toda la información relacionada con las APIs empleadas. Esta fue obtenida de la primera fase definida en la planificación dedicada al análisis de los servicios empleados, así como de diferentes errores y demás problemas encontrados durante la implementación del proyecto. Dado que uno de los objetivos es ver cuáles son las posibilidades ofrecidas por estos servicios, se considera relevante centralizar todos los conocimientos en un solo capítulo, aunque el transcurso de las iteraciones no parta completamente de la misma base.

5.1 APIs de Elsevier

Elsevier dispone a los usuarios la página web *Elsevier Developer Portal* [18], desde la cual, con una cuenta de usuario, puede crearse una clave que permite el acceso a las distintas APIs, aunque con ciertas restricciones. Durante esta sección, se detallará el acceso a la clave, las aplicaciones disponibles y las limitaciones sobre los servicios.

5.1.1 Acceso

En la página web de *Elsevier Developer Portal* se pone a disposición del usuario toda la información relacionada con las APIs así como el acceso a estas. Sin embargo, para poder usarlas deben cumplirse ciertos requisitos:

- Se debe disponer de una cuenta de usuario en *Elsevier Developer Portal*.
- El usuario debe disponer de una suscripción que permita el acceso a los recursos. Instituciones involucradas en la investigación pueden disponer de dicha suscripción, con lo que, en caso de ser miembro, solo sería necesario demostrar la pertenencia al organismo pertinente. Esto es posible durante el proceso de la creación de una cuenta en

Elsevier Developer Portal, donde se podrá seleccionar la institución a la que se pertenece y demostrar la pertenencia a la misma.

En el caso de este proyecto, la Universidad de A Coruña dispone de una suscripción que permite al autor emplear los servicios de acuerdo con las limitaciones establecidas en el servicio. En este caso, solo será necesario disponer de una cuenta de usuario en la institución e iniciar sesión con la misma durante la creación de la cuenta en *Elsevier Developer Portal*.

Una vez cumplidos los requisitos, el usuario puede obtener una clave en el apartado de *My API Key* compatible con los distintos productos en función de los acuerdos que haya aceptado al solicitarla.

Acuerdos de Elsevier

A la hora de solicitar una clave, el usuario debe aceptar como mínimo un acuerdo de uso sobre el servicio de las APIs. En ellos se exponen las distintas limitaciones impuestas. A continuación, se explicarán breve y resumidamente estos documentos

El primer acuerdo es el "*API Service Agreement*", común en este tipo de servicios *online* donde el usuario se compromete con una serie de limitaciones y condiciones impuestas por la empresa. Resumidamente, el usuario acepta las políticas impuestas por *Elsevier*, incluyendo la posibilidad de que si, de forma justificada, se determina que el uso de la API no está siendo correcto, se puede proceder a suspender la clave. Es obligatorio aceptarlo.

El segundo acuerdo es "*Elsevier Provisions for Text and Data Mining*". En él, se permite el acceso a varios servicios más limitados a cambio de comprometerse a emplearlos exclusivamente para investigación no comercial y/o de uso personal. Este acuerdo no es obligatorio aceptarlo, pero la información no estará disponible en caso de no hacerlo.

Los servicios (explicados en el siguiente apartado) incluidos con el segundo acuerdo son:

- Scopus Citation Overview
- Scopus Author Feedback
- ScienceDirect Full-Text Entitlement
- ScienceDirect Article Hosting Permissions
- ScienceDirect Holdings Report
- Embase Search and Retrieval
- Engineering Village Search and Retrieval
- Geofacets API

- Pharmapendium API
- SUSHI COP5 API

Finalmente se aceptó únicamente el primer acuerdo dado el uso que se hace de los servicios disponibles, explicado más adelante.

5.1.2 Productos

Elsevier proporciona acceso a numerosas APIs con distintas utilidades. A continuación, se listará cada producto acompañado de un breve resumen sobre lo que ofrece:

1. **Scopus**. Como se ha comentando en la sección 2.5, Scopus es la aplicación que ofrece Elsevier para la búsqueda de documentos de carácter científico, autores e instituciones. En el campo concreto de la búsqueda de documentos, esta aplicación dispone de varias APIs RESTful desde las cuales se pueden obtener datos genéricos, como título, autor(es) o citas, y otros datos técnicos, como el ISSN o el DOI. Además, el usuario puede acceder tanto al resumen como a datos relacionados con las citas del documento.

2. **ScienceDirect**. ScienceDirect es otro de los productos ofrecidos por Elsevier y que dispone de varias APIs RESTful. Permite la búsqueda de documentos científicos y devuelve datos similares a los que devuelve Scopus, con el añadido de incluir el documento al completo, tanto texto como cualquier otra información incluida en el mismo (tablas, imágenes, anexos, etc.).

3. **SciVal**. Otra aplicación de Elsevier con API RESTful disponible para el acceso a métricas de evaluación comparativa de rendimiento (*performance benchmarking*) tanto de investigadores como de instituciones.

4. **Engineering Village**. Esta API de arquitectura RESTful permite, de igual forma que la aplicación web homónima, consultar recursos de ingeniería de forma programática.

5. **Elsevier Embase API**. API RESTful que ofrece resúmenes, índices y otros metadatos proporcionados por la base biomédica de Elsevier: Embase.

6. **Geofacets API**. API RESTful que ofrece, de igual forma que la aplicación web homónima, recursos relacionados con la geología y ciencias de la Tierra.

7. **SUSHI COP5 API**. Se denomina Standard Usage Statistics Harvesting Initiative (SUSHI) a aquellos servicios que permiten obtener métricas de uso de recursos electrónicos siguiendo el estándar COP5.

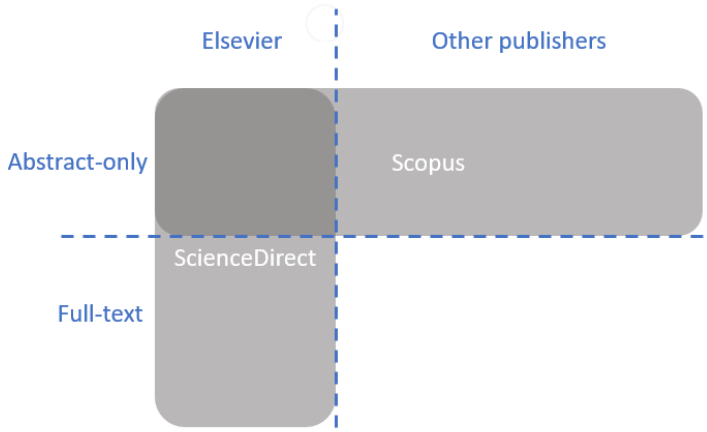


Figura 5.1: Comparativa gráfica del abarcamiento de los datos de ScienceDirect y Scopus

Cabe recalcar que Scopus y ScienceDirect difieren en los contenidos disponibles en su base de datos. En la figura 5.1, extraída de la sección de *FAQ* de Elsevier Developer Portal [18], se comparan y diferencian ambas bases de datos. Ambas aplicaciones comparten cierto subconjunto de artículos, el cual está constituido por publicaciones electrónicas de Elsevier (en concreto la colección de ScienceDirect), mientras que se expanden hacia objetivos distintos. En el caso de Scopus, se posibilita al usuario el acceso a documentos de otras editoriales mientras que, por otra parte, ScienceDirect permite acceso al documento al completo, posibilidad que Scopus no ofrece.

Las dos APIs de mayor interés, dada la casuística de la aplicación, son Scopus y ScienceDirect. Cabe recalcar que puede resultar de interés la API SUSHI COP5, pero por falta de tiempo se decidió descartarla.

A continuación se detallarán las APIs mencionadas de mayor interés, incluyendo vistas disponibles y aquellos campos que se consideren oportunos.

Scopus

Scopus dispone de varias APIs agrupadas por funcionalidades, entre las cuales se diferencian:

- Tres APIs dedicadas a la búsqueda.
    1. **Affiliation Search**. Búsqueda de instituciones.
    2. **Author Search**. Búsqueda de autores.
    3. **Scopus Search**. Búsqueda de documentos en Scopus.
- Tres APIs para obtener información detallada sobre un elemento concreto.

1. **Abstract Retrieval**. Permite obtener resúmenes de un documento concreto.
2. **Affiliation Retrieval**. Permite obtener el perfil de una institución concreta.
3. **Author Retrieval**. Permite obtener el perfil de un autor concreto.

- Cinco APIs para obtener metadatos y métricas relacionadas.

1. **Citations Count Metadata**. Devuelve el conteo, bajo el tipo de documento solicitado, del número de veces que se ha citado un documento especifico.
2. **Citations Overview**. Devuelve metadatos relacionados con las citas sobre un documento específico.
3. **PlumX Metrics**. Permite obtener métricas PlumX de documentos y otros elementos relacionados.
4. **Serial Title Metadata**. Devuelve metadatos sobre la revista, libro o cualquier otro medio de publicación de artículos.
5. **Subject Classifications**. Permite la búsqueda de información relacionada con la clasificación de materias definidas en Scopus.

- Una API para comentarios del autor.

1. **Author Feedback**. Permite obtener distintos datos relacionados con los autores de artículos.

ScienceDirect

ScienceDirect también dispone de numerosas APIs, las cuales se listan a continuación:

- Dos APIs dedicadas a la búsqueda.

1. **ScienceDirect Search V2**. Búsqueda en ScienceDirect.
2. **Article Metadata**. Búsqueda de autores.

- Cuatro APIs para obtener información detallada sobre un elemento concreto.

1. **Article Retrieval**. Permite obtener el documento al completo.
2. **Article Entitlement API**. Permite obtener los identificadores y derechos de los artículos a los que el usuario tiene acceso al texto completo.
3. **Article Hosting Permission API**. Representa el acceso (o no) a los documentos completos.
4. **Object Retrieval**. Permite obtener elementos por separado del documento al completo, como texto o imágenes.

- Tres APIs para obtener metadatos y métricas relacionadas.

    1. **Serial Title Metadata**. Permite buscar por el título de la fuente, en este caso de publicaciones seriadas (p.e. publicaciones en varias partes, revistas y diarios).
    2. **Nonserial Title Metadata**. Permite buscar por el título de la fuente, en este caso que no sean de publicaciones seriadas (p.e. libros).
    3. **Subject Classifications**. Permite la búsqueda de información relacionada con la clasificación de materias definidas en ScienceDirect.

- Una API para información de informes realizados hacia contenido de ScienceDirect.

    1. **Holdings Report API**. Muestra los informes realizados contra los contenidos de ScienceDirect.

5.1.3 Comparativa de productos

Volviendo al proyecto, el objetivo principal es buscar documentos, por lo que APIs como *Scopus Search*, *Article Metadata* o *ScienceDirect Search V2* son las más apropiadas para cumplir esta función.

Dado que *Scopus* proporciona más documentos, se considera que la opción óptima para el desarrollo de la aplicación es emplear su API frente a las otras dos posibilidades. Sin embargo, resulta de interés, no solo para este desarrollo, sino para el trabajo futuro a realizar en la aplicación, toda la información adicional que *ScienceDirect* pueda proporcionar.

Generalidades

Algo que se debe tener en cuenta antes de continuar con el análisis son las vistas disponibles, ya que algunos productos disponen de varias.

- Scopus Search ofrece dos vistas: la estándar (*Standard*), con parámetros básicos; y la completa (*Complete*), con muchos más metadatos.

- Article Metadata ofrece dos vistas con la misma denominación y menos campos que Scopus Search (detallado a continuación): la estándar (*Standard*) y la completa (*Complete*).

- ScienceDirect Search V2 ofrece una única vista, llamada estándar (*Standard*).

Valores devueltos

En la tabla 5.1 se muestra una lista de campos devueltos por respuesta disponibles, permitiendo comparar entre las tres APIs seleccionadas para su estudio. La información accesible

Campos	***Scopus Search***	***Article Metadata***	***SD Search***
Identificador	Scopus ID	DOI	DOI
DOI	Sí	Sí	Sí
PII	Sí	Sí	Sí
MEDLINE ID	Sí		
ORCID	Sí		
EID	Sí	Sí	
Título	Sí	Sí	Sí
Tipo fuente	Sí	Sí	
Tipo documento	Sí	Sí	
Veces citado	Sí		
Nombre fuente	Sí	Sí	Sí
ISBN/ISSN	Sí	Sí	
Volumen	Sí	Sí	Sí
Temática fuente	Sí		
Rango páginas	Sí	Sí	Sí
Fecha publicación	Sí	Sí	Sí
Autores*	Sí	Sí	Sí**
Autor principal	Sí	Sí	Sí
Open access	Sí	Sí	Sí
Institución*	Sí		
Resumen	Sí	Sí	
Palabras clave	Sí	Sí	
Número artículo	Sí	Sí	
Fundación*	Sí		
Texto completo		Enlace	Enlace

Tabla 5.1: Tabla de campos disponibles en las APIs de búsqueda *Scopus Search*, *Article Metadata* y *ScienceDirect Search V2*. Un asterisco (*) indica que los campos contienen más información anidada. Dos asteriscos(**) indican que en esa API el campo no tiene información anidada.

no se limita al documento en sí, sino que también incluye datos sobre la fuente que publica el artículo, pudiendo estas ser revistas científicas, libros o conferencias; instituciones o fundaciones que hayan contribuido o que, de alguna manera, estén relacionadas con la publicación; y metadatos como el número de veces que se ha citado el resultado. A modo de aclaración, el campo *Open Access* representa a un valor de igual nombre y permite confirmar si se puede obtener el documento al completo o el resumen para las APIs de ScienceDirect o Scopus, respectivamente. En caso afirmativo, se incluirá un enlace que referencia a la llamada de la API correspondiente.

Como se puede apreciar, la búsqueda de *Scopus* no solo es más completa en número de documentos, sino que también en información relacionada. Por la parte de *ScienceDirect*, el único campo exclusivo de sus APIs es el enlace al texto completo y otros derivados que no se incluyen por simplificación, como *Copyright* para *Article Metadata*. Si bien la decisión ya estaba tomada, la no disponibilidad de algunos campos, como el tipo de fuente que publica el artículo (Tipo fuente) o el número de veces que se ha citado una publicación (Veces citado) inclinan todavía más la balanza. Por otra parte, la API con menor cantidad de datos es la de *ScienceDirect Search V2*, limitando demasiado sus posibilidades como para emplearla como un método de búsqueda frente a la alternativa de *ScienceDirect*.

Limitaciones de peticiones

Nombre API	Cuota semanal	Peticiones por segundo
ScienceDirect Search v2	20 000	2
Article Metadata API	N/A	6
Scopus Search	20 000	9

Tabla 5.2: Limitaciones sobre las APIs.

La tabla 5.2 muestra de forma resumida las limitaciones establecidas al número de peticiones que se pueden hacer por cada API. Dado el público objetivo de esta aplicación, se considera que no se superará la cuota de 20 000 semanales, pero deben tenerse en cuenta de cara a futuras actualizaciones.

Ordenación

Por otro lado, *Scopus* y *ScienceDirect Search V2* permiten ordenación de resultados. En el primer caso, las posibilidades ofrecidas son: número de artículo, número de citas, fecha de publicación periódica, fecha de publicación del artículo, nombre de creador o primer autor,

número de páginas, rango de páginas (string con el formato *[<primera página>-<última pagina>]*), nombre de publicación periódica, año de publicación, relevancia o volumen. En el segundo caso son: fecha de publicación periódica y relevancia.

Paginación y resultados

Otro aspecto relevante es la paginación y las limitaciones asociadas a la misma. Cada API tiene diferentes limitaciones, pero todas tienen en común la posibilidad de indicar el primer resultado y el número de ellos que se quiere recibir por llamada. Estos valores están limitados por, según la documentación, el máximo por defecto del sistema.

- *Scopus* limita a 5000 el número de resultados disponibles empleando la paginación antes mencionada y 25 por respuesta (para la vista completa, que es la que se usará). Si se desean obtener todos, debe usarse la opción *cursor*, lo cual permitirá al usuario obtener, con cada llamada, un enlace a los siguientes documentos. De esta forma no se puede ir hacia atrás, debe empezarse siempre desde el principio.
- *ScienceDirect Search V2* permite poner como valor inicial cualquier número entre 0 y 6000 y para el número de resultados 10, 25, 50 y 100.
- *Article Metadata* impone 25 resultados por página y 6000 en total (por búsqueda).

Campos empleados en las consultas

Un último punto a analizar son los campos por los cuales se permite la búsqueda y filtrado de documentos. Para esta ocasión, la API *ScienceDirect Search V2* no se puede tener en cuenta ya que no permite emplear este tipo de filtrado en su búsqueda, por lo que expondremos los datos de las dos posibilidades restantes.

Article Metadata permite combinar hasta 15 campos distintos, mientras que *Scopus* permite 66 [27, 28]. Sin embargo, se debe tener en cuenta que algunos son combinaciones de otros. Por ejemplo, uno de ellos es *ALL*, el cual permite buscar por palabras clave en prácticamente cualquier ámbito de un documento, como título, autor, palabras clave, afiliación, fuente, etc. A su vez, cada valor de los mencionados previamente dispone de su propio campo para buscar exclusivamente sobre ello u otros que son subconjuntos de algunos de los más empleados o comunes. Aun así, la API de *ScienceDirect* sigue sin permitir filtrar por muchos campos.

La tabla 5.3 permite comparar las posibilidades ofrecidas por *Article Metadata* con las correspondientes de Scopus de forma más visual.

Campos disponibles para filtrado	
Scopus Search	Article Meatadata
key	keywords
srctype	content-type
author-name	authors
affil	affiliation
pubyear	pub-date
title	title
srctitle	srctitle
doi	doi
	eid
issn	issn
isbn	isbn
volumen	vol-issue
	available-online-date
	vor-available-online-date
openaccess	openaccess
...	

Tabla 5.3: Comparativa de campos disponibles de filtrado entre Scopus Search y Article Metadata.

5.1.4 Conclusión

En base a lo mencionado previamente, tanto características como cantidad de documentos disponibles, se decide emplear como método único de búsqueda para esta base de datos el *endpoint* de *Scopus*. Adicionalmente, se podrá usar como apoyo *ScienceDirect* para obtener los textos completos de los documentos, pero no como buscador.

5.2 APIs de Clarivate

Clarivate dispone de gran variedad de APIs en su *Clarivate Developer Portal*. Estas ofrecen diferentes servicios relacionados con los productos que ofrecen, como *EndNote*, *Incites* o *Web of Science*.

5.2.1 Acceso

El acceso a los productos de *Clarivate Developer Portal* es ligeramente distinto al de Elsevier. Como requisito general para poder emplear cualquiera de las APIs disponibles es necesario crear una cuenta en Clarivate y acceder con ella a *Clarivate Developer Portal*. También será necesario disponer de alguna suscripción para acceder a recursos como Web of Science (WoS).

Una vez autenticado, el usuario puede solicitar una clave en la API que considere. Sin embargo, es posible que dicha solicitud sea denegada porque el acceso puede ser de pago. En el caso del desarrollo de esta aplicación, se contactó con soporte para la solicitud de una clave de forma gratuita en los casos que no lo fuese.

5.2.2 Productos

Una vez se dispone de una cuenta en *Clarivate Developer Portal* se pueden consultar las diferentes APIs y sus funcionalidades. Dado que hay disponibles una gran cantidad de servicios, los agruparemos por aplicaciones para determinar cuáles son de mayor interés:

- **Web of Science**. Como ya se ha comentado previamente en la sección 2.5, Web of Science es el producto ofrecido para la consulta de documentos científicos por la empresa Clarivate. Dispone de gran variedad de APIs con diferentes funcionalidades.
- **InCites**. Esta herramienta, a partir de la documentación incluida en WoS, permite evaluar y comparar la investigación científica de los investigadores y/o instituciones.
- **EndNote**. Es una aplicación encargada de la gestión de citas y bibliografías a la hora de desarrollar un artículo o documento científico.
- **Converis**. Servicio que permite centralizar la gestión institucional, especialmente en el ámbito de la investigación. Dispone de una API para consultar la información del producto contratado de forma programática.
- **Derwent**. Aplicación que permite el acceso a más de 30 millones de patentes. Se dispone de una API para el acceso a dicha información.
- **CDDI SUSHI REST API**. Se denomina Standard Usage Statistics Harvesting Initiative (SUSHI) a aquellos servicios que permiten obtener métricas de uso de recursos electrónicos siguiendo el estándar COP5.
- Otras aplicaciones como Publons o Metabase, tienen APIs disponibles pero están obsoletas, por lo que no será necesario tenerlas en cuenta.

Para el caso concreto de esta aplicación, es de especial interés Web of Science, así que las investigaciones se centrarán en las APIs relacionadas con él.

Web of Science

Web of Science, como ya se ha explicado previamente en la sección 3.1, es un producto de Clarivate Analytics que permite el acceso a un conjunto de bases de datos de revistas científicas y metadatos relativos al contenido disponible. Dispone de varias APIs con distintas finalidades, listadas a continuación:

- *Web of Science API Lite* y *Web of Science API Expanded*. Si bien son dos APIs diferentes, proporcionan el mismo resultado con diferentes limitaciones. Permiten la búsqueda de documentos científicos en varias bases de datos empleando consultas con diferentes campos, de igual forma que el producto principal permite. La versión *Lite* es gratuita pero devuelve resultados más limitados (p.e. sin número de citas o sin resumen) que la versión *Expanded*, la cual es de pago.

- *Web of Science SUSHI API*. Ofrece un servicio similar a las dos anteriores con la ventaja de que los datos obtenidos se adaptan al protocolo definido por el *Code of Practice 5* (COP5) establecido por el estándar COUNTER.

- *Web of Science Reviewer Locator API*. Permite al usuario localizar e incluso contactar con los revisores de los documentos ofrecidos por la plataforma. Suele emplearse para integrar el servicio de *Web of Science Reviewer Locator* con otras plataformas.

- *Web of Science Journals API*. Proporciona métricas y metadatos para todas las revistas científicas contenidas en la *Web of Science Core Collection*. Se incluyen métricas ya conocidas, como el Journal Impact Factor, y otras nuevas.

- *Web of Science Starter API*. Permite acceso en tiempo real a diferentes variables relacionadas con los documentos alojados en el servicio, como el recuento de veces que un documento ha sido citado.

- *Web of Science Researcher API*. Habilita la búsqueda relacionada con autores, así como sus publicaciones.

- *ESTI service apis*. Permite la obtención de datos relacionados con la sesión ESTI. No hay mucha más información al respecto.

Para el desarrollo de esta aplicación nos centraremos en *Web of Science API Expanded* y *Web of Science API Lite*.

5.2.3 Comparativa de productos

Como se acaba de explicar, *Web of Science API Lite* y *Web of Science API Expanded* son muy similares en muchos aspectos por una sencilla razón: ofrecen el mismo servicio con distintas restricciones. Durante este capítulo se explicarán en profundidad sus características y las diferencias entre ellas para, finalmente, escoger la más adecuada de cara al desarrollo de la aplicación.

Generalidades

Una de las diferencias es el número de *endpoints* y, por lo tanto, de posibilidades que se ofrecen. La versión *Lite* ofrece los siguientes tres: uno para realizar búsquedas de forma programática, otro para, a partir de un *query identifier*, realizar una búsqueda y otro para obtener documentos a partir de un identificador concreto.

La versión *Expanded* ofrece un total de ocho *endpoints*, que incluye los 3 que proporciona la versión *Lite* con el añadido de un método más, el POST, para la operación de realizar búsquedas de forma programática. Los otros 5 permiten obtener documentos relacionados a partir de un identificador de WoS, las publicaciones que citan el documento consultado, las publicaciones citadas por el documento consultado, datos sobre las referencias citadas y publicaciones que citan el documento consultado y una lista exclusivamente compuesta por los identificadores de los resultados a partir de un *query identifier*.

Para el desarrollo de la aplicación es de especial relevancia los *endpoints* dedicados a la búsqueda de documentos de ambas APIs, así que se continuará el análisis enfocado exclusivamente en ellos.

Por otra parte, ambas APIs permiten seleccionar la base de datos en la que se quiere realizar la búsqueda. Las opciones disponibles son las siguientes:

- Web of Science Core Collection
- Arabic Citation Index (sólo en versión *Expanded*)
- Biological Abstracts
- BIOSIS Citation Index
- BIOSIS Previews
- CABI: CAB Abstracts® and Global Health® (sólo en versión *Expanded*)
- Chinese Science Citation Index SM (sólo en versión *Expanded*)
- Current Contents Connect

- Data Citation Index
- Derwent
- FSTA (sólo en versión *Expanded*)
- Inspec® (sólo en versión *Expanded*)
- Korean Journal Database (sólo en versión *Expanded*)
- Medline®
- Russian Science Citation Index
- SciELO Citation Index (sólo en versión *Expanded*)
- Zoological Records

A mayores, se proporciona un valor adicional para este campo en ambas APIs llamado *Web of Knowledge*. Este permite al usuario enviar la consulta por todas las bases de datos, siempre teniendo en cuenta la subscripción a la que está sujeto.

Valores devueltos

Otras de las diferencias clave son los valores devueltos. La versión *Lite* proporciona acceso a campos como: identificador de WoS (UT o *Unique Identifier*); autores (*Authors*); palabras clave del artículo determinadas por el autor (*Keywords* o *Author keywords*); tipo de documento (*Document type*); título (*Title*), el cual incluye una lista con todos los valores posibles, siendo la mayoría el título en varios idiomas; edición de la revista (*Issue*); número de páginas (*Pages*); fecha de publicación (*Publication date*); información relativa a la fuente en la que el documento fue publicado (*Source*), la cual incluye valores como el título, las páginas, fecha de publicación o el volumen entre otros; DOI; ISBN; o ISSN.

Web of Science API Expanded permite el acceso tanto a estos campos como a otros muchos más como número de citas y otros metadatos relacionados, idiomas en los que se ha publicado el artículo en cuestión o lista de contribuidores, siempre y cuando esa información esté disponible. Si bien esta versión es más completa, puede llegar a ser contraproducente dada la gran cantidad de información que devuelve y la cual no es toda completamente relevante. Sin embargo, la propia API dispone de un campo en el cual el usuario puede especificar qué información quiere recibir, permitiendo acortar la extensión de la respuesta. También dispone de un *flag* llamado *links* el cual, si está activado, incluye enlaces a las fuentes y las citas de cada documento devuelto como resultado.

Limitaciones de peticiones

En la figura 5.2 se muestran las tablas proporcionadas por Clarivate en el documento ”Términos de Producto/Servicio”. Como se puede ver, la versión *Lite* no tiene restricciones de documentos por año mientras que todos los planes de la versión *Expanded* sí. También destacar que las limitaciones aplicadas a la versión *Expanded* son claramente más restrictivas que las vistas previamente para Scopus y ScienceDirect.

Web of Science API Expanded			
Plan	**Requests per second**	**Web of Science documents per year**	**Maximum number of Web of Science documents returned by one request**
Basic	2	50,000	100
Intermediate	2	250,000	
Advanced	3	1,000,000	
Premium	5	3,000,000	

Web of Science API Lite		
Requests per second	**Web of Science documents per year**	**Maximum number of Web of Science documents returned by one request**
2	n/a	100

Figura 5.2: Tablas extraídas de los ”Términos de Producto/Servicio” de Web of Science (enlace)

Ordenación

Respecto a la ordenación de resultados, ambos servicios ofrecen las mismas posibilidades. Se permite ordenar ascendente o descendentemente a partir de: fecha de publicación, veces citado, primer autor, título de la fuente de publicación y título de la conferencia. Adicionalmente, se puede emplear: añadido recientemente (más recientes primero), relevancia.

Paginación y resultados

Debe tenerse en cuenta otro aspecto relevante como es la paginación. Si bien en *Scopus* existían ciertas limitaciones, *WoS* tampoco escapa a ellas en ninguna de sus posibilidades. Las dos APIs de *Clarivate* ofrecen la posibilidad de marcar el primer resultado así como el número de documentos por respuesta y estos están limitados a valores entre 1 y 100.000 y 0 y 100 respectivamente. Esto supone que, al contrario que con Scopus, si se obtienen más de 100.100 artículos de la consulta, no podrán ser observados por el usuario salvo que se aplique reordenación de los mismos.

Campos empleados en las consultas

Por último, los campos empleados en las consultas son los mismos para ambos servicios, aunque estos varían en función de la base de datos empleada.

5.2.4 Conclusión

A la hora de seleccionar la opción más adecuada para el proyecto, se optó por la *Web of Science API Expanded*. Los campos que ofrece la versión *Lite* limitaban demasiado la información a mostrar en los resultados, como, por ejemplo, el número de citas, que sí estaba disponible en *Scopus*. El resto de factores, como número de *endpoints* o bases de datos disponibles, no influyeron realmente en la decisión, ya que ambas opciones ofrecían todo lo necesario para el proyecto del buscador.

5.3 Aplicación al proyecto

De cara al proyecto, es necesario tener en cuenta diferentes aspectos en relación con el uso de las APIs:

- Valores en común entre servicios. Si se desea que las consultas sean aplicables a ambos sistemas deben conocerse los parámetros comunes al realizar consultas y ordenar resultados, así como toda información común devuelta. Para ello, se elaboraron tablas con todas las variables ofrecidas por cada opción. Están disponibles en el apéndice A para su consulta.

- Limitaciones. Como ya se ha visto previamente, existen limitaciones sobre el número de peticiones en cada servicio. Esto puede suponer un problema según el público objetivo. Teniendo en cuenta los servicios usados, Scopus Search y Web of Science Expanded, y las limitaciones en el número de peticiones realizables contra los mismos, 20000 por semana y 50000 al año respectivamente, podría suponer un problema si la cantidad de usuarios es muy alta, especialmente para el caso de WoS. Sin embargo, no se considera necesario trabajar con mayor margen por lo menos a corto plazo.

  Otro aspecto a tener en cuenta en las limitaciones es la disponibilidad de los servicios en Internet. Para el uso de los productos es necesario estar conectado a internet por medio de la red de la institución que dispone de una suscripción o disponer de una clave denominada *Institutional Key*, la cual no se pudo obtener. Dado que la VPN de la universidad permite el acceso a los recursos disponibles en ella pero no a la propia red, fue necesario establecer un túnel SSH contra un servidor alojado en la facultad de informática.

Una vez se tienen claros los puntos previamente mencionados, se puede proceder al desarrollo del proyecto.

Capítulo 6
Iteración 1

DADO que esta es la primera etapa de la implementación, durante esta iteración se tomarán muchas decisiones que afectarán al resto del proyecto. A lo largo de este capítulo se justificarán las distintas elecciones realizadas en conjunto con una explicación de cómo se han aplicado al proyecto.

6.1 Análisis

Durante esta iteración, como ya se comentó previamente, se desarrolla un buscador de documentos por palabras clave. Teniendo en cuenta que tanto usuario registrado como no registrado tendrán acceso completo a esta funcionalidad, la implementación no dispondrá de sistema de registro de usuarios, tan solo se centrará en desarrollar una interfaz web sencilla con la funcionalidad requerida. A continuación, se especificarán los actores del sistema y los casos de uso definidos e implementados durante este primer incremento.

6.1.1 Actores

Un actor es toda aquella figura externa al sistema pero que interactúa con él en uno o más casos de uso. No son excluyentes entre si, es decir, una persona puede cumplir con varios perfiles de actor simultáneamente o en distintos momentos a lo largo del tiempo.

Se pueden distinguir los siguientes grupos de actores:

- Actores principales: Todo aquel usuario que utiliza las funciones principales del sistema.
- Actores secundarios: Todo aquel involucrado en tareas administrativas o de mantenimiento del sistema.
- Elementos externos: Todo aquello que forma parte del ámbito de la aplicación pero que no se desarrolla con la misma.

- Otros sistemas: Sistemas externos al que se desarrolla pero que interactúan con él.

A continuación, se definen todos los actores involucrados en el proyecto:

- **Usuario no identificado**. El actor accede a la aplicación pero no realiza ninguna acción de inicio de sesión o registro en la aplicación. Se le permite únicamente realizar búsquedas en el sistema de forma ilimitada. También puede iniciar sesión o registrarse según sea necesario. Entra en la definición de actor principal.
- **Usuario identificado**. El actor accede a la aplicación tras iniciar sesión o registrarse. Se le permite realizar búsquedas en el sistema de forma ilimitada, así como guardar dichas búsquedas y ser notificado en caso de haber nuevos resultados. A mayores, se permite el cierre de sesión. Entra, de nuevo, en la definición de actor principal.
- **Scopus**. Es uno de los ya mencionados servicios con los que el usuario va a interactuar indirectamente al realizar sus consultas. Dicho de otra forma, es uno de los actores que provee información al sistema. Es externo al proyecto, por lo que se podría considerar adecuado añadirlo al grupo de otros sistemas.
- **Web of Science**. Es el otro servicio empleado para obtener los documentos a partir de las consultas que el usuario realice, es decir, provee información al sistema. De nuevo, es externo al proyecto, por lo que cumple con la definición de otros sistemas.

De los previamente mencionados solo se tendrán en cuenta dos actores para esta implementación: el Usuario no identificado y Scopus. Esto se debe a que, teniendo en cuenta que se va a permitir exclusivamente buscar por palabras clave, no es necesario incluir en esta primera versión el registro de usuarios, tal y como se mencionó en la parte respectiva a la planificación (4.2).

6.1.2 Casos de uso

Los casos de uso de la aplicación a implementar en esta primera iteración se exponen en las tablas 6.1, 6.2 y 6.3.

El segundo caso de uso difiere del finalmente implementado. En iteraciones posteriores se explicarán las decisiones detrás del cambio.

Otro matiz relevante es respecto a qué API se implementaría primero durante esta iteración. Se decidió en conjunto con los directores del proyecto que, dada la completitud y las diferentes herramientas proporcionadas para trabajar con el servicio, la mejor opción era *Scopus*. Es por ello que los casos de uso están definidos específicamente para esa base de datos.

En la tercera iteración, los casos de uso con el mismo funcionamiento pero distinta base de datos tendrán la misma numeración acompañada con una B en lugar de una A, permitiendo así distinguirlos.

CU-01A: Buscar documentos exclusivamente por palabras clave con Scopus.	
Descripción	Buscar documentos a partir de sólo unas palabras clave.
Actores	Usuario no identificado o usuario identificado & Scopus
Flujo de acciones	1. El usuario introduce una serie de palabras clave en el buscador. 2. El usuario pulsa el botón ”Buscar”. 3. El sistema recibe la petición del usuario y contacta con Scopus para obtener resultados en consecuencia. 4. El sistema, con la respuesta de la base de datos, devuelve una nueva vista con los resultados solicitados.
Flujo alternativo	Se produce algún error al recibir la respuesta de la API. Se le notifica al usuario.
Postcondiciones	El usuario puede consultar los 25 primeros documentos resultantes de su búsqueda.

Tabla 6.1: CU-01A: Buscar documentos exclusivamente por palabras clave con Scopus.

CU-02A: Obtener más resultados de Scopus	
Descripción	Mediante paginación obtener resultados previos o posteriores a los mostrados en la vista.
Actores	Usuario no identificado o usuario identificado & Scopus
Precondiciones	El usuario ha realizado una búsqueda y está en la página de resultados.
Flujo de acciones	1. El usuario pulsa uno de los botones al final de la página para ver los resultados siguientes, anteriores, primeros o últimos. 2. El sistema contacta con los servicios necesarios con el fin de obtener más resultados. 3. El sistema, con la respuestas de las bases de datos, devuelve una nueva vista con los resultados solicitados.
Flujo alternativo	Se produce algún error al recibir la respuesta de la API. Se le notifica al usuario.
Postcondiciones	El usuario puede consultar los resultados deseados.

Tabla 6.2: CU-02A: Obtener más resultados de Scopus

CU-03A: Ver detalles de un documento con Scopus	
Descripción	El usuario puede consultar detalles relacionados con uno de los documentos.
Actores	Usuario no identificado o usuario identificado & Scopus
Precondiciones	El usuario ha realizado una búsqueda y está en la página de resultados.
Flujo de acciones	1. El usuario selecciona el título de uno de los resultados disponibles, enviando así una petición contra el sistema para ver los detalles del documento. 2. El sistema obtiene dicha petición y la reenvía adecuadamente a la base de datos correspondiente. 3. El sistema redirige la información recibida de la base de datos al usuario en forma de vista.
Flujo alternativo	Se produce algún error al recibir la respuesta de la API. Se le notifica al usuario.
Postcondiciones	El usuario puede consultar la información deseada.

Tabla 6.3: CU-03A: Ver detalles de un documento con Scopus

6.2 Diseño

Una de las razones por las que se decidió comenzar con una iteración sencilla es porque, si bien la implementación será más breve, el resto de pasos requieren de mayor consideración. El diseño cuenta con varios de estos pasos. En este capítulo se explicarán muchas de las decisiones tomadas junto con su pertinente justificación.

6.2.1 *Framework* de desarrollo web

En el ámbito de desarrollo de aplicaciones web, existen multitud de herramientas que permiten tener resultados adaptados a diferentes necesidades. Aplicado al caso concreto de este proyecto, se necesitó una herramienta segura, que permita desarrollos ágiles y, a ser posible, que emplease Python como lenguaje de programación. Tras barajar diferentes posibilidades, se decidió emplear Django.

Además de ser código abierto, Django dispone de una estructura que permite realizar cambios continuos y modulares, favoreciendo el desarrollo ágil planteado para el proyecto. Por otro lado, permite crear aplicaciones altamente escalables y seguras. Si todo ello se le

suma el hecho de estar implementada en Python, se confirma que encaja con las necesidades mencionadas previamente.

6.2.2 Arquitectura del sistema

Como ya se ha comentado, Django es el *framework* seleccionado para la implementación del sistema. Esto define ciertas partes de la arquitectura como la estructura de la aplicación, algunos patrones de diseño empleados y otras técnicas que la herramienta usa.

Para explicar la arquitectura, es necesario explicar previamente el *Model-Template-View* (*MTV*) o Modelo-Plantilla-Vista, versión adaptada del patrón de diseño *Model-View-Controller* (*MVC*) o Modelo-Vista-Controlador la cual usa Django.

El nombre de *MVC* proviene de los tres principales capas del sistema que interactúan entre si [29]:

- Modelo (o *Model*). Es, básicamente, la capa encargada de interactuar con la base de datos.
- Vista (o *View*). La vista se encarga de definir cómo los datos se muestran al usuario. Dicho de otra forma, es la capa que se muestra al usuario y, por lo tanto, aquella con la que interactúa.
- Controlador (o *Controller*). Es la capa encargada de la lógica de negocio, además de conectar las dos capas anteriores.

MTV realiza ligeros cambios a la terminología previamente planteada. La Vista definida en MVC pasa a ser la Plantilla (o *Template*), ya que en Django se usan dichos documentos HTML para definir cómo se muestra la interfaz al usuario. El Controlador pasa a denominarse Vista, dado que el propio framework usa el término vista como aquella función que recibe como entrada la interacción del usuario (petición HTTP), interactúa con el Modelo si es necesario y finalmente devuelve una plantilla al usuario. Esta definición encaja con la dada previamente del controlador, ya que conecta Modelo y Vista o, en este caso, Modelo y Plantilla [30].

Otros conceptos clave para entender correctamente la arquitectura del sistema son los modelos empleados:

- Por una parte, se emplea la arquitectura cliente-servidor dada la estructura implícita de una página web. Esta es un modelo de aplicación distribuida que diferencia dos partes lógicas entre las cuales se reparte la carga computacional: la correspondiente al proveedor del servicio, el servidor; y la del usuario del mismo, el cliente.

  La estructura se basa en una relación donde el cliente realiza solicitudes contra el servidor, y este último responde compartiendo sus recursos. Dicho de otra forma, el cliente

cumple con el rol de consumidor de recursos y servidor de proveedor de los mismos [31].

En el caso de esta aplicación, esta estructura viene determinada por la relación entre navegador del usuario y servidor Django. El cliente debe establecer conexión en red de forma activa para obtener una respuesta la cual le permita el acceso a los recursos.

- Por otro lado, Django aplica la arquitectura *shared nothing*. Esta enuncia que los elementos que la constituyen son independientes entre sí, por lo que la parte del servidor puede, a su vez, subdividirse en otros dos elementos más: la base de datos y la aplicación que usa Django. Como resultado, el sistema se vuelve muy escalable, permitiendo ampliar o incluso cambiar cualquiera de los elementos sin grandes complicaciones.

El resultado final es un modelo cliente-servidor donde, a su vez, el servidor consta de dos partes: la aplicación web y la base de datos.

Para la base de datos a emplear, Django soporta oficialmente SQLite, MariaDB, MySQL, Oracle y PostgreSQL. Dada la experiencia académica del autor, las dos últimas son alternativas con las que ya ha trabajado y tiene cierto conocimiento sobre ellas. Finalmente, dado que es un *software* de código abierto, se decidió emplear *PostgreSQL*.

Se decidió separar base de datos y aplicación web en dos contenedores empleando la tecnología Docker, facilitando así su despliegue en diferentes entornos. Esto combinado con Visual Studio Code y su extensión Dev Containers [32] proporciona un entorno de desarrollo bastante simple, rápido de desplegar y altamente configurable, para el cual solo sería necesario abrir el gestor de contenedores y el editor.

Dado que el cliente, en este caso, es el navegador web, los esfuerzos en el desarrollo de este apartado deben centrarse en proporcionar una interfaz de acuerdo con los requisitos no funcionales mencionados en la sección 4.1.

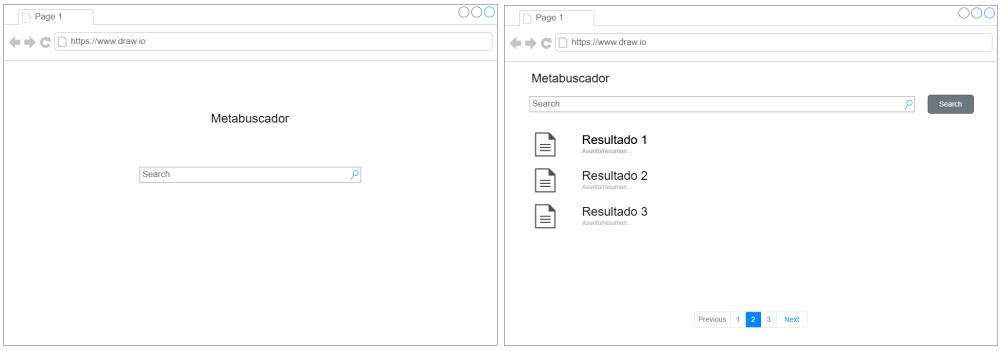
Por otra parte, para explicar la estructura interna de la aplicación web, es conveniente primero definir la nomenclatura empleada por Django. El framework define a la aplicación web completa como ”proyecto” y la subdivide en ”aplicaciones”, conjuntos modulares que incluyen una o más funcionalidades. Una de las grandes ventajas de este formato es que se puede reutilizar dichas ”aplicaciones” para varios ”proyectos”, ya que funcionan como librerías.

Para esta iteración se definió una única ”aplicación” llamada *searcher*. En ella se incluiría tanto el buscador simple de esta primera iteración como el buscador avanzado de la siguiente.

6.2.3 Diseño de interfaz

Para el diseño de la interfaz de esta iteración se tomó como referencia el buscador base y los resultados de Google Académico (o *Google Schoolar*), obviando sintaxis complejas o caracteres

especiales que este pueda utilizar. Los mockups disponibles en la figura 6.1 ilustran la idea inicial de la que se partió a la hora de realizar la implementación.



(a) Vista sin resultados. (b) Vista con resultados.

Figura 6.1: *Mockups* de la búsqueda simple.

6.2.4 Arquitectura interna

Por otra parte, para explicar la estructura interna de la aplicación web, es conveniente primero definir la nomenclatura empleada por Django. El framework define a la aplicación web completa como ”proyecto” y la subdivide en ”aplicaciones”, conjuntos modulares que incluyen una o más funcionalidades. Una de las grandes ventajas de este formato es que se puede reutilizar dichas ”aplicaciones” para varios ”proyectos”, ya que funcionan como librerías.

Para esta iteración se definió una única ”aplicación” llamada *searcher*. En ella se incluiría tanto el buscador simple de esta primera iteración como el buscador avanzado de la siguiente.

6.3 Implementación

El trabajo realizado durante esta iteración se ha enfocado, principalmente, en configurar el proyecto y proporcionar un resultado mínimo pero funcional. Durante este apartado se tratarán ambos frentes.

6.3.1 Configuración del proyecto

Como se ha comentado previamente en la sección 6.2.2, la idea inicial de este proyecto era desplegarlo en un entorno *Dockerizado* para así facilitar el despliegue en otros equipos. La estructura resultante se constituye en dos contenedores: uno con el servidor web *Django* y demás herramientas necesarias para el correcto funcionamiento del mismo y otro con la base de datos *PostgreSQL*. Como las herramientas son de código abierto, es habitual encontrar plantillas o proyectos de otros usuarios en diferentes comunidades o plataformas, a lo cual se

le suma el hecho de que el autor ya ha trabajado en un entorno similar, por lo que dispone de ficheros adaptados por él mismo.

Previamente se definen tres carpetas en las cuales irán los diferentes archivos del proyecto:

- ***.devcontainer***. En esta carpeta se incluirán los archivos para la configuración de los contenedores.
- ***requirements***. En esta carpeta se definirán dos ficheros en los que se irán incluyendo tanto las librerías como otras herramientas de utilidad para el desarrollo de la aplicación. Durante el despliegue se leerán esos ficheros y se instalará todo lo necesario.
- ***src***. En esta carpeta se incluirá el código del proyecto.

Comenzando por la carpeta *.devcontainer*, hay tres elementos de interés: las imágenes de los contenedores, el fichero de *docker-compose* y, por último, el *devcontainer.json* que emplearemos con Visual Studio Code para un despliegue sencillo y rápido.

Se decidió emplear ”postgres:latest” para la imagen del contenedor de *PostgreSQL*, ya que esta está correctamente configurada y es ampliamente usada. Además, como veremos más adelante, los ficheros que la comunidad proporciona para este tipo de estructuras emplean dicha versión. Por otro lado, para Django definiremos un *Dockerfile* con una imagen propia. En el fragmento de código 6.1 se muestra el resultado, en el cual simplemente se añaden las librerías de Python definidas en el archivo *requirements.txt*.

```
1  FROM python:latest
2  ENV PYTHONUNBUFFERED 1
3  RUN mkdir /workspace
4  WORKDIR /workspace
5  RUN apt-get update
6  RUN pip install --upgrade pip
7  COPY /requirements/requirements.txt requirements.txt*
       /workspace/
8  RUN if [ -f "requirements.txt" ]; then pip install
       --no-cache-dir -r requirements.txt && rm
       requirements.txt; fi
9  RUN apt-get autoremove -y \
10     && apt-get clean -y \
11     && rm -rf /var/lib/apt/lists/*
```

Código 6.1: Código de *Dockerfile*

Continuando con el apartado de Docker, se emplea el fichero *docker-compose.yml* para definir ambos contenedores y sus variables. De nuevo dicho fichero está disponible en el fragmento de código 6.2. Como se puede comprobar, las configuraciones realizadas son principalmente sobre los puertos y volúmenes a emplear. Recalcar que también se definen el usuario y contraseña de la base de datos, por lo que es recomendable tener especial cuidado con estos datos sensibles a la hora de un despliegue en producción.

```yaml
version: '3'
services:
  app:
    build:
      context: ..
      dockerfile: .devcontainer/Dockerfile
    ports:
      - "8000:8000"
    volumes:
      - ~/.gitconfig:/root/.gitconfig
      - ..:/workspace
    command: sleep infinity
    links:
      - 'db'
  db:
    image: postgres:latest
    restart: always
    ports:
      - "5432:5432"
    volumes:
      - postgres-data:/var/lib/postgresql/data
    environment:
      POSTGRES_USER: postgres
      POSTGRES_DB: postgres
      POSTGRES_PASSWORD: postgres
volumes:
  postgres-data:
```

Código 6.2: Código de *docker-compose.yml*

Por último, en lo referente al *.devcontainer*, para el fichero de *devcontainer.json* se empleó una plantilla disponible en la página oficial de https://containers.dev/ [32], la cual se adaptó

siguiendo la configuración de proyectos de características similares. En ella se definen numerosas variables a emplear para que el editor permita un trabajo más fluido.

6.3.2 Aplicación *Searcher*

La primera aplicación que se creará en *Django* será el buscador definido previamente. El objetivo será añadir todos los casos de uso relacionados directamente con la búsqueda simple y avanzada a esta aplicación. Sin embargo, en esta iteración solo se implementará la simple.

Tras crear el proyecto de *Django*, el cual denominaremos *metasearcher* por la naturaleza del mismo, se añadirá la aplicación *searcher*. Al hacerlo, se añadirán automáticamente todos los ficheros necesarios para su correcto funcionamiento. Durante la implementación se definirán tres vistas y las *URLs* correspondientes para que el usuario pueda acceder a ellas. Se definió una dirección adicional para que, en caso de que se quiera acceder a la aplicación directamente, no sea necesario incluir *index*. No es una funcionalidad relevante, pero se usará en las diferentes aplicaciones que se definan más adelante.

Durante esta implementación se decidió añadir todo el código necesario dentro del fichero correspondiente a las vistas, incluyendo el uso del *SDK* de *Scopus*, tratando de separar las diferentes funcionalidades de las propias vistas. Esto dio lugar a algunas dependencias en el código o repeticiones innecesarias que durante la segunda iteración fue necesario solventar.

6.4 Pruebas

Para probar el correcto funcionamiento del sistema se hicieron consultas de forma manual y de diferentes longitudes de texto y se compararon con los resultados ofrecidos por Scopus. Por ejemplo, la consulta "*heart attack*" en el buscador de este proyecto debe devolver los mismos 25 primeros resultados que los devueltos por el buscador de la aplicación de Elsevier empleando el campo *TITLE-ABS-KEY*.

A continuación, se listan las consultas realizadas para la comprobación del correcto funcionamiento:

- TITLE-ABS-KEY("heart attack")
- TITLE-ABS-KEY("best sort algorithms")
- TITLE-ABS-KEY("A neuropathologic feature of brain aging")

Es relevante que los contenidos vayan entrecomillados a la hora de formular las consultas en la aplicación de Scopus, ya que así es cómo se construye la consulta dentro del sistema.

Capítulo 7
Iteración 2

A lo largo de este capítulo se recopilará el desarrollo de la segunda iteración y sus cuatro etapas.

7.1 Análisis

Esta iteración se centrará en el desarrollo del buscador avanzado, similar a los que Scopus y Web of Science (WoS) ofrecen por defecto, empleando únicamente la API de Scopus. A mayores, se hicieron algunos cambios y se añadieron pequeñas modificaciones sugeridas por los directores a la iteración previa.

7.1.1 Casos de uso

Los casos de uso de la aplicación a implementar en esta segunda iteración están definidos en las tablas 7.1, 7.2 y 7.3

Como se comentaba en la iteración previa, el caso de uso CU-02 fue reescrito. Esta decisión fue tomada en conjunto con los directores tras la reunión pertinente al final de la primera iteración por problemas a la hora de mostrar los resultados.

Como ya se ha comentado en el análisis previo (capítulo 5), Scopus dispone de dos métodos para obtener resultados: empleando paginación y mediante el cursor. Sus diferencias son, resumidamente, que la paginación permite avanzar o retroceder como el usuario desee pero limita el número de resultados a 5000 y el cursor solo permite avanzar de 25 en 25 pero no limita los artículos obtenibles.

Tras tratarlo con los directores, se decidió emplear el cursor, lo que significó eliminar la paginación e implementar un *scroll* infinito. Esto tiene algunas ventajas, como mostrar todos los resultados en una sola página, y también algunos inconvenientes, como que cargar demasiados resultados puede suponer un uso de recursos excesivo para los navegadores.

CU-02A: Obtener más resultados de Scopus	
Descripción	Mediante ***scroll infinito*** obtener más resultados relacionados con la consulta realizada
Actores	Usuario no identificado o usuario identificado & Scopus
Precondiciones	El usuario ha realizado una búsqueda y está en la página de resultados.
Flujo de acciones	1. El usuario pulsa el botón de ”obtener más resultados” del final de la página. 2. El sistema envía una consulta a Scopus con el fin de obtener más resultados. 3. El sistema, con la respuesta de la base de datos, devuelve dinámicamente más información a la página web.
Flujo alternativo	Se produce algún error al recibir la respuesta de la API, por lo que se le notifica al usuario y se le pide que lo intente más tarde o recargue la página.
Postcondiciones	El usuario puede consultar los resultados deseados.

Tabla 7.1: CU-02A: Obtener más resultados (actualizado).

CU-04: Ordenar resultados	
Descripción	El usuario puede alternar entre las distintas opciones de ordenación.
Actores	Usuario no identificado o usuario identificado
Precondiciones	El usuario ha realizado una búsqueda y está en la página de resultados.
Flujo de acciones	1. El usuario selecciona el desplegable correspondiente al orden en el que se muestran los resultados. Se muestran las posibilidades disponibles. 2. El usuario selecciona la opción que prefiera y, automáticamente, se envía la petición al sistema. 3. El sistema devuelve, según el orden deseado, los resultados.
Flujo alternativo	Se produce algún error al recibir la respuesta de la API. Se le notifica al usuario.
Postcondiciones	El usuario puede consultar los resultados deseados.

Tabla 7.2: CU-04: Ordenar resultados.

CU-05A: Buscar documentos por uno o varios campos con Scopus	
Descripción	Buscar documentos a partir de uno o más campos empleando palabras clave y/o identificadores.
Actores	Usuario no identificado o usuario identificado & Scopus o Web of Science
Flujo de acciones	1. El usuario, mediante una interfaz interactiva, selecciona el campo por el que desea buscar e introduce las palabras clave o el identificador que corresponda. Esta acción se repite tantas veces como campos desee emplear el usuario, enlazándolos con un operador booleano. 2. El usuario pulsa el botón ”Buscar”. 3. El sistema recibe la petición del usuario y contacta con Scopus para obtener resultados en consecuencia. 4. El sistema, con la respuesta de la base de datos, devuelve una nueva vista con los resultados solicitados.
Flujo alternativo	Se produce algún error al recibir la respuesta de la API. Se le notifica al usuario.
Postcondiciones	El usuario puede consultar los 25 primeros documentos resultantes de su búsqueda.

Tabla 7.3: CU-05A: Buscar documentos por uno o varios campos con Scopus.

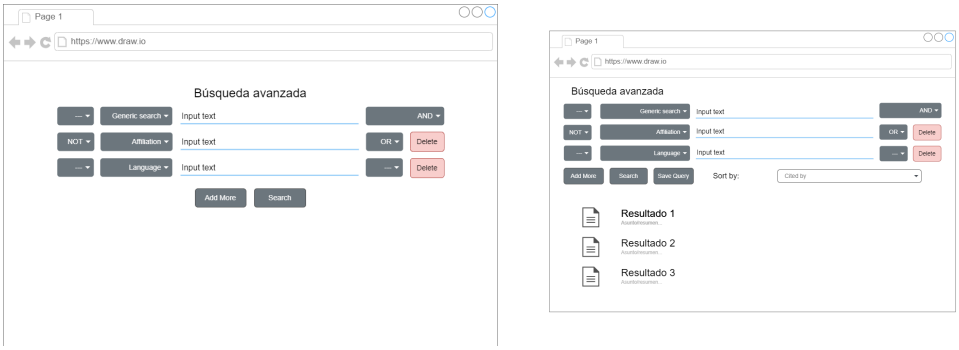
7.2 Diseño

El diseño en esta iteración se centró principalmente en ofrecer un buscador avanzado claro, intuitivo y completo tratando de ocultar, en la medida de lo posible, las diferentes limitaciones que las APIs imponen. Otros aspectos, como la arquitectura del sistema, permanecen invariables durante esta etapa.

Toda la implementación a realizar se hizo dentro de la aplicación definida previamente: *searcher*.

7.2.1 Diseño de interfaz

Para la interfaz de esta parte del proyecto, se tomaron los buscadores que tanto Scopus como Web of Science ofrecen como modelos a seguir. La estructura es muy similar: dos entradas, la primera un desplegable con los distintos campos disponibles y otra de texto donde el usuario introduce los términos por los que quiere realizar la consulta; y otro desplegable adicional por cada fila a mayores de la primera para seleccionar el nexo booleano que el usuario desee y así conectar los diferentes campos empleados en la consulta. A mayores, se dispone de botones con diferentes funcionalidades: uno para añadir una fila al final de las ya existentes; y otro por cada una de dichas filas que permita eliminarlas. Las figuras 7.1 muestran los conceptos previos a la implementación de la aplicación.



(a) Vista sin resultados. (b) Vista con resultados.

Figura 7.1: *Mockups* de la búsqueda avanzada.

7.3 Implementación

Partiendo de la base ya desarrollada, la aplicación *searcher* previamente creada, el objetivo de esta iteración sería añadir las vistas y funcionalidades pertinentes. Esto hizo que se descubriesen ciertas dependencias con el código previamente definido que no habían sido modularizadas correctamente, a lo cual se le sumó que el código perdió cierta claridad. Se decidió mover todas las funciones encargadas de contactar con la base de datos y transformar las respuestas recibidas en datos con los que la aplicación pudiese trabajar a un fichero llamado *utils.py*, ubicado en el mismo directorio de la aplicación.

7.3.1 Aplicación *Searcher*

El siguiente paso fue añadir las nuevas funcionalidades a la aplicación *searcher*. Se decidió comenzar por añadir primero la funcionalidad de búsqueda avanzada. Para ello, se ideó crear dos formularios: uno con un campo y palabras clave y otro con lo mismo incluyendo una

entrada a mayores para el enlace booleano. Por cada campo adicional que el usuario desee, se añade otro formulario. Para ello, fue necesario crear un pequeño *script* que permitiese añadir o eliminar campos al formulario sin recargar completamente la página. Esto no resultó sencillo debido a los pocos conocimientos de Javascript del autor y la dificultad añadida de modificar un formulario de Django dinámicamente.

Django permite generar formularios y extraer sus datos de forma sencilla desde las vistas. Sin embargo, si lo que se busca es trabajar con varios formularios a la vez, debe recurrirse a un *formset*. De esta forma, a partir de un único tipo de formulario, se puede obtener una estructura más compleja con varias copias del mismo elemento con diferente contenido.

Una vez se dispone de un *formset* bien configurado, debe incluirse el código de JavaScript para añadir o eliminar campos a la consulta. Para ello será necesario acceder a una serie de variables que indican el número de copias del formulario que hay en la vista y modificarlas cuando sea oportuno. A mayores, se debe incluir un formulario vacío para poder realizar copias del mismo. Como resultado se obtiene una interfaz sencilla e intuitiva.

Por otro lado, para añadir la ordenación de resultados, es necesario incluir un desplegable con las diferentes opciones disponibles. Se decidió incluirlo exclusivamente en las vistas de resultados, ya que en las demás no tenía sentido. Además, se consideró hacer que la página se recargase automáticamente, sin la necesidad de pulsar ningún botón. Para ello se generó, de nuevo, un pequeño código en JavaScript encargado de actualizar la ventana al percibir un cambio en la vista.

7.3.2 Reorganización de ficheros

Otro de los problemas encontrados durante esta iteración, aunque no supuso directamente un retraso en la planificación, fue la mala estructura que se había empleado durante la primera iteración a la hora de definir las diferentes funciones encargadas de la comunicación con la base de datos, en este caso Scopus.

Dado que el SDK había facilitado, en cierta medida, el trabajo, se consideró que desde las vistas podía implementarse en la totalidad la aplicación. Realmente, puede hacerse, pero el resultado es caótico y poco legible. Para solucionar este problema, dentro de la aplicación *searcher* se definió un nuevo fichero llamado *utils*. En él se definieron aquellas funciones encargadas de, a partir de una consulta y una base de datos, obtener los resultados de forma que las vistas pudiesen trabajar con ellos y exponerlos en las vistas.

Esta nueva estructura no solo se mantiene hasta el resultado final actual, sino que también aligeró muchos de los problemas que se fueron encontrando en funcionalidades añadidas durante las siguientes iteraciones como, por ejemplo, la traducción de consultas entre bases de datos.

7.4 Pruebas

De igual forma que en la iteración 1, para comprobar el correcto funcionamiento del sistema se realizaron consultas manualmente, en este caso empleando diferentes campos, y comparando los resultados con los ofrecidos por Scopus en su aplicación web. Dado que en este incremento se incluyó la posibilidad de cambiar el orden de los resultados, por cada consulta debía comprobarse también el correcto funcionamiento de esta funcionalidad alternando entre las diferentes posibilidades disponibles.

A continuación, se listan las consultas realizadas para la comprobación del correcto funcionamiento:

- TITLE(”collaborative filtering”) and AUTHOR-NAME(”cacheda”)
- AUTHOR-NAME(”cacheda”) or AUTHOR-NAME(”fernández”)
- AFFIL(”UDC”) and ISSN(”2296858X”)
- DOI(”10.1186/s40478-023-01638-2”)

Es relevante que los contenidos vayan entrecomillados a la hora de formular las consultas en la aplicación de Scopus, ya que así es cómo se construye la consulta dentro del sistema.

Capítulo 8

Iteración 3

En esta iteración estaba planeado incluir Web of Science (WoS) y combinar todos los resultados de ambas bases de datos pero, por problemas con las APIs, dicha funcionalidad tuvo que redirigirse a un escenario específico. A continuación, se explica el proceso seguido.

8.1 Análisis

Durante el análisis del trabajo a realizar se detectaron una serie de problemas con las APIs. En esta sección se describen los impedimentos que ocasionaron, así como las decisiones que se tomaron. Finalmente, se concluirá este apartado con las partes restantes del análisis.

8.1.1 Problema con las APIs: resultados incompletos

Como se acaba de comentar, el objetivo de este proyecto era combinar resultados de diferentes APIs para así proporcionar una interfaz en la que el usuario pudiese consultar todos los documentos desde una sola vista. Para ello, se debía generar un algoritmo capaz de combinar los datos de las dos aplicaciones y generar unos propios a partir de los valores disponibles. Sin emargo, surgieron una serie de problemas que ocasionaron hacer una adaptación de la idea original para cumplir con el objetivo marcado.

El principal inconveniente a la hora de comenzar a mezclar los datos es la incompletitud e incongruencia en gran parte de los resultados, hecho que en la documentación no se menciona. Por poner un ejemplo, datos como el DOI (u otros identificadores) pueden no estar incluidos, elementos necesarios para determinar la existencia de un mismo elemento en los dos servicios. A mayores, otros elementos como autor o título, que se podrían contemplar como alternativas para identificar duplicados, pueden no coincidir, bien porque cada servicio los almacena de una forma distinta o bien porque se pueden almacenar ligeras variaciones. Esto suponía un problema a la hora de mezclar los datos.

Otro problema, relacionado en parte con lo ya mencionado, son los métodos de ordenación y la información relativa a los mismos. Ambas APIs ofrecen una serie de posibilidades para organizar los resultados obtenidos, como relevancia o fecha de subida al sistema. Sin embargo, los valores por los cuales se realizan dichas clasificaciones no siempre se mostraban, lo cual puede llegar a ser entendible en casos como la relevancia, pero no en otros como la fecha de subida. Relacionado con esto, al hacer la ordenación de valores numéricos como pueden ser el rango de páginas, en el caso de Scopus la ordenación se realiza por la cadena en lugar de por el valor real, lo cual da lugar a que, por ejemplo, [8-9] sea mayor que [1011-2011].

Por lo visto, este tipo de inconvenientes no son exclusivos de los servicios empleados. En el capítulo 3 se menciona una aplicación que, empleando otras bases de datos, genera un metabuscador similar al que se trata de desarrollar y, casualmente, se encuentran con el mismo problema que se acaba de mencionar. Los implicados publicaron un artículo a mayores con todos los problemas encontrados durante el desarrollo del mismo [33].

Teniendo en cuenta todo lo mencionado anteriormente, se decidió ofrecer las siguientes posibilidades:

- Permitir al usuario obtener los resultados de cada base de datos por separado. Se le otorga la posibilidad de alternar de forma sencilla entre bases de datos, pudiendo obtener la totalidad de resultados en cada una de las posibilidades.
- Ofrecer la posibilidad a mayores de mezclar los resultados de ambas bases de datos considerando exclusivamente aquellos que incluyen el DOI entre sus datos. Dado el coste computacional de la operación, es recomendable no excederse con la cantidad de documentos empleados.

8.1.2 Casos de uso

Los casos de uso definidos para esta iteración vienen descritos en las tablas 8.1, 8.2, 8.3, 8.4, 8.5 y 8.6.

8.2 Diseño

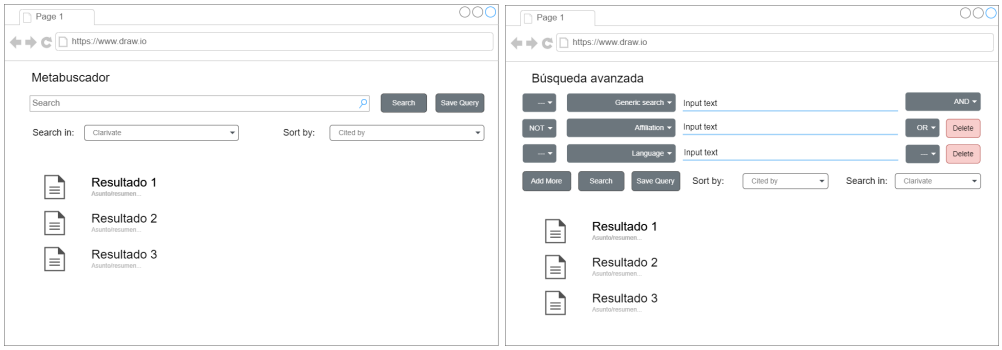
El siguiente paso de la iteración es el diseño. A continuación se describe brevemente el proceso seguido.

CU-01B: Buscar documentos exclusivamente por palabras clave con WoS.	
Descripción	Buscar documentos a partir de sólo unas palabras clave.
Actores	Usuario no identificado o usuario identificado & Web of Science
Flujo de acciones	1. El usuario introduce una serie de palabras clave en el buscador. 2. El usuario pulsa el botón ”Buscar”. 3. El sistema recibe la petición del usuario y contacta con Web of Science para obtener resultados en consecuencia. 4. El sistema, con la respuesta de la base de datos, devuelve una nueva vista con los resultados solicitados.
Flujo alternativo	Se produce algún error al recibir la respuesta de la API. Se le notifica al usuario.
Postcondiciones	El usuario puede consultar los 25 primeros documentos resultantes de su búsqueda.

Tabla 8.1: CU-01B: Buscar documentos exclusivamente por palabras clave con Web of Science.

8.2.1 Interfaz

Los cambios sobre la interfaz previamente definida en las iteraciones anteriores son mínimos. Simplemente se añade un desplegable a las vistas de resultados, a la misma altura que la ordenación, con las opciones disponibles. Las figuras 8.1a y 8.1b ilustran el concepto inicial.



(a) Vista del buscador simple. (b) Vista del buscador avanzado.

Figura 8.1: *Mockups* de la búsqueda avanzada.

CU-02B: Obtener más resultados de Web of Scence	
Descripción	Mediante ***scroll infinito*** obtener más resultados relacionados con la consulta realizada
Actores	Usuario no identificado o usuario identificado & Web of Science
Precondiciones	El usuario ha realizado una búsqueda y está en la página de resultados.
Flujo de acciones	1. El usuario pulsa el botón de ”obtener más resultados” del final de la página. 2. El sistema envía una consulta a Web of Science con el fin de obtener más resultados. 3. El sistema, con la respuesta de la base de datos, devuelve dinámicamente más información a la página web.
Flujo alternativo	Se produce algún error al recibir la respuesta de la API, por lo que se le notifica al usuario y se le pide que lo intente más tarde o recargue la página.
Postcondiciones	El usuario puede consultar los resultados deseados.

Tabla 8.2: CU-02B: Obtener más resultados de Web of Science.

CU-03B: Ver detalles de un documento con Web of Science	
Descripción	El usuario puede consultar detalles relacionados con uno de los documentos.
Actores	Usuario no identificado o usuario identificado & Web of Science
Precondiciones	El usuario ha realizado una búsqueda y está en la página de resultados.
Flujo de acciones	1. El usuario selecciona el título de uno de los resultados disponibles, enviando así una petición contra el sistema para ver los detalles del documento. 2. El sistema obtiene dicha petición y la reenvía adecuadamente a la base de datos correspondiente. 3. El sistema redirige la información recibida de la base de datos al usuario en forma de vista.
Flujo alternativo	Se produce algún error al recibir la respuesta de la API. Se le notifica al usuario.
Postcondiciones	El usuario puede consultar la información deseada.

Tabla 8.3: CU-03B: Ver detalles de un documento con Web of Science.

CU-05B: Buscar documentos por uno o varios campos con WoS	
Descripción	Buscar documentos a partir de uno o más campos empleando palabras clave y/o identificadores.
Actores	Usuario no identificado o usuario identificado & Web of Science
Flujo de acciones	1. El usuario, mediante una interfaz interactiva, selecciona el campo por el que desea buscar e introduce las palabras clave o el identificador que corresponda. Esta acción se repite tantas veces como campos desee emplear el usuario, enlazándolos con un operador booleano. 2. El usuario pulsa el botón ”Buscar”. 3. El sistema recibe la petición del usuario y contacta con Web of Science para obtener resultados en consecuencia. 4. El sistema, con la respuesta de la base de datos, devuelve una nueva vista con los resultados solicitados.
Flujo alternativo	Se produce algún error al recibir la respuesta de la API. Se le notifica al usuario.
Postcondiciones	El usuario puede consultar los 25 primeros documentos resultantes de su búsqueda.

Tabla 8.4: CU-05B: Buscar documentos por uno o varios campos con Web of Science.

CU-06: Permitir alternar entre bases de datos	
Descripción	El usuario puede cambiar la base de datos cuyos resultados está observando tras realizar una consulta.
Actores	Usuario no identificado o usuario identificado & Scopus & Web of Science
Precondiciones	El usuario ha realizado una búsqueda y está en la página de resultados.
Flujo de acciones	1. El usuario selecciona en un desplegable la base de datos que quiere consultar. El cliente envía la petición al percibir el cambio. 2. El sistema obtiene dicha petición, traduce la consulta para la base (o bases) de datos y la envía. 3. Una vez recibida la respuesta de la base (o bases) de datos, el sistema devuelve al cliente la nueva vista.
Flujo alternativo	Se produce algún error al recibir la respuesta de la API. Se le notifica al usuario.
Postcondiciones	El usuario puede consultar la información deseada si existe.

Tabla 8.5: CU-06: Permitir alternar entre bases de datos.

CU-07: Buscar combinando los resultados de ambas bases de datos	
Descripción	El usuario realiza una consulta que se envía a ambos sistemas y se devuelven los más relevantes.
Actores	Usuario no identificado o usuario identificado
Flujo de acciones	1. El usuario realiza una consulta con ”mezclar” como la base de datos seleccionada. 2. El sistema obtiene dicha petición, traduce la consulta para sendas bases de datos y las envía con un máximo establecido de resultados por servicio. 3. Una vez recibida la respuesta de la base de datos, el sistema mezcla exclusivamente los resultados que tengan DOI y los ordena por el parámetro que el usuario ha indicado. Tras ello, los envía en forma de vista al cliente.
Flujo alternativo	Se produce algún error al recibir la respuesta de alguna de las API. Se le notifica al usuario.
Postcondiciones	El usuario puede consultar la información deseada si existe.

Tabla 8.6: CU-07: Buscar combinando los resultados de ambas bases de datos.

8.3 Implementación

Para el desarrollo de esta iteración se siguió la estructura que se había definido en iteraciones previas, es decir, se añadieron las funciones encargadas de conectar y recopilar la información relativa a la base de datos al archivo *utils.py*, creado en la iteración 2, y se actualizaron las vistas para poder alternar entre base de datos siguiendo el análisis y diseño previamente definido.

8.3.1 Aplicación *Searcher*

El desarrollo de esta iteración se puede subdividir en dos partes, donde la primera sería la correspondiente a obtener y mostrar los resultados de Web of Science y la segunda la encargada de generar el algoritmo para obtener los resultados, mezclar los datos y pasarlos a la vista. Durante esta sección se seguirá dicha estructura.

Conexión con Web of Science

El primer paso fue establecer conexión con los servidores de Clarivate para poder realizar las consultas. Para ello, en este caso, fue necesario emplear la librería *requests*.

Una vez definidas las funciones necesarias, se debían obtener resultados homogéneos, no solo para ofrecer vistas cuya estructura fuese, como mínimo, similar, sino también para permitir en la segunda parte de esta iteración la combinación de resultados. Para ello se generó un *DataFrame* con la misma estructura que la empleada por el SDK de scopus.

El último paso fue el correspondiente a mostrar los resultados. Dado que se siguió la misma estructura que con Scopus, no fue necesario definir nada a mayores. Simplemente se pasó la información con la misma estructura y la plantilla de Django se completaría automáticamente.

Mezclado de datos

Para esta segunda parte de la iteración fue necesario generar una función que, a partir de dos listas de resultados, ofreciese todos aquellos documentos que tuviesen DOI ordenados y que mostrase información de ambos servicios, como, por ejemplo, el número de citas. Dado que son dos servicios con diferentes contenidos, los valores obtenidos pueden diferir. Para cumplir con dichos objetivos se empleó como apoyo la librería *Pandas*.

Pandas dispone de diversas funcionalidades para trabajar con datos de forma sencilla y, sobre todo, rápida. Para cumplir con dichos estándares emplea diferentes módulos implementados en C. Esta gran ventaja permite una mayor velocidad al ser código traducido directamente al lenguaje máquina. De esta forma, no es necesario emplear intérpretes que puedan dar lugar a un funcionamiento más lento.

Otro punto necesario para implementar esta funcionalidad fue limitar el número de resultados por base de datos a 100. Esta decisión se tomó para evitar largas esperas y, sobretodo, para no consumir demasiadas peticiones por consulta, ya que las disponibles pueden llegar a agotarse realmente rápido con mayores volúmenes de información.

Los pasos restantes son sencillos. El algoritmo debe eliminar todos aquellos resultados sin DOI, combinar aquellos que dispongan del mismo y devolverlos en forma de *DataFrame*.

8.4 Pruebas

Para comprobar el correcto funcionamiento de todos los casos de uso de esta iteración, fue necesario hacer diferentes pruebas para diferentes escenarios.

Toda la parte respectiva a añadir Web of Science supuso realizar las mismas consultas descritas hasta ahora en las dos iteraciones previas, solo que traducidas para dicha aplicación. Durante estas comprobaciones se descubrió que, inexplicablemente, ciertas consultas perfectamente construidas devuelven fallos. En concreto, devuelven el código 500 *Internal Server Error*. Se consideró que es problema del servicio.

A continuación, se listan las consultas realizadas para la comprobación del correcto funcionamiento traducidas:

- TS(”heart attack”)
- TS(”best sort algorithms”)
- TS(”A neuropathologic feature of brain aging”)
- TI(”collaborative filtering”) and AU(”cacheda”)
- AU(”cacheda”) or AU(”fernández”)
- OO(”UDC”) or OG(”UDC”) and ISSN(”2296858X”)
- DO(”10.1186/s40478-023-01638-2”)

Para probar el correcto funcionamiento de la mezcla de datos, se fue modificando el método de ordenación para una consulta y se fue comprobando que los datos estaban en la posición adecuada.

Capítulo 9
Iteración 4

Última iteración del proyecto en la que se recogerá todo el proceso seguido durante las cuatro etapas que lo constituyen.

9.1 Análisis

Para esta iteración se implementaron las funcionalidades restantes de la aplicación: la gestión de usuarios y permitir guardar consultas.

9.1.1 Actores

Como ya se comentó previamente, los dos actores definidos en la iteración 1 son los mismos a lo largo de todo el proyecto. Sin embargo, es en este incremento donde los casos de uso dejan de ser aplicables a los dos usuarios (registrado y no registrado). Solo aquellos usuarios registrados pueden acceder a la funcionalidad de guardar consultas, ya que de otra forma sería complicado almacenar la información que cada uno de ellos desea.

9.1.2 Casos de uso

Los casos de uso de la aplicación a implementar en esta cuarta iteración se exponen en las tablas 9.1, 9.2, 9.3, 9.4, 9.5, 9.6 y 9.7.

Esta iteración tiene mayor número de casos de uso pero, como ya veremos en el apartado correspondiente a la implementación de los mismos, requieren mucho menor esfuerzo.

9.2 Diseño

Durante esta sección se hará un repaso sobre los cambios realizados sobre el diseño de la aplicación.

CU-08: Registrar usuario en el sistema	
Descripción	Crear una cuenta de usuario en el sistema.
Actores	Usuario no identificado.
Precondiciones	El usuario está en el formulario de creación de cuenta.
Flujo de acciones	1. El usuario introduce su nombre y apellidos, e-mail, nombre de usuario y contraseña. Debe repetir este último campo para confirmar que es correcta. Si el formulario detecta algún error antes de ser avisado se lo notifica al usuario. 2. El usuario envía el formulario. 3. El sistema almacena los datos recibidos. El usuario pasa a estar autenticado.
Flujo alternativo	Se produce algún error al recibir el formulario del usuario, por lo que debe repetir el flujo de acciones desde el principio.
Postcondiciones	El usuario tiene un usuario y contraseña funcionales y tiene su sesión abierta.

Tabla 9.1: CU-08: Registrar usuario en el sistema.

CU-09: Iniciar sesión en el sistema	
Descripción	Iniciar sesión con una cuenta de usuario en el sistema.
Actores	Usuario no identificado.
Precondiciones	El usuario está en el formulario de inicio de sesión y tiene unos credenciales en el sistema.
Flujo de acciones	1. El usuario introduce sus credenciales en el sistema, es decir, nombre de usuario y contraseña. 2. El usuario envía el formulario. 3. El sistema comprueba que los datos recibidos concuerdan con los almacenados en la base de datos. En caso afirmativo, el usuario es autenticado.
Flujo alternativo	Los credenciales no coinciden con los del sistema, por lo que el usuario debe repetir el flujo de acciones desde el principio.
Postcondiciones	El usuario tiene su sesión abierta.

Tabla 9.2: CU-09: Iniciar sesión en el sistema.

CU-10: Cerrar sesión en el sistema	
Descripción	Cerrar una sesión que un usuario haya abierto en el sistema.
Actores	Usuario identificado.
Precondiciones	El botón de cerrar sesión es accesible por el usuario.
Flujo de acciones	1. El usuario pulsa el botón de cerrar sesión. 2. El sistema recibe la petición y cierra la sesión del usuario
Postcondiciones	El usuario no tiene una sesión abierta.

Tabla 9.3: CU-10: Cerrar sesión en el sistema.

CU-11: Guardar consulta	
Descripción	El usuario almacena la consulta que desea.
Actores	Usuario identificado.
Precondiciones	El usuario ha hecho una consulta al sistema y puede ver los resultados, así como el botón para guardar.
Flujo de acciones	1. El usuario pulsa el botón de guardar consulta. 2. El sistema recibe la petición y almacena la información necesaria para dicha asociación.
Postcondiciones	El usuario tiene una consulta guardada asociada a su cuenta.

Tabla 9.4: CU-11: Guardar consulta

CU-12: Eliminar consulta guardada	
Descripción	El usuario elimina una consulta guardada en el sistema.
Actores	Usuario identificado.
Precondiciones	El usuario ha guardado previamente una consulta en el sistema. Se encuentra en la vista correspondiente a la lista de consultas guardadas.
Flujo de acciones	1. El usuario pulsa el botón de eliminar una consulta. 2. El sistema recibe la petición y elimina la consulta para el usuario en cuestión.
Postcondiciones	El usuario ya no tiene la consulta asociada a su cuenta.

Tabla 9.5: CU-12: Eliminar consulta guardada

CU-13: Notificar usuario de nuevos resultados vía *e-mail*	
Descripción	Se le notifica al usuario mediante correo electrónico de que hay nuevos resultados disponibles en una de sus consultas.
Precondiciones	El usuario ha guardado previamente una consulta en el sistema.
Flujo de acciones	1. Periódicamente, el sistema comprueba si hay nuevos resultados para las consultas almacenadas de un usuario. 2. En el caso de que haya nuevos resultados para una o más consultas, se notifica a quien convenga vía correo electrónico.
Flujo alternativo	No hay nuevos resultados asociados a ninguna de las consultas de un usuario, por lo que el flujo de acciones termina en el primer paso.

Tabla 9.6: CU-13: Notificar usuario de nuevos resultados vía *e-mail*.

CU-14: Ver consultas guardadas	
Descripción	El usuario puede ver todas las consultas que haya guardado en el sistema.
Actores	Usuario identificado.
Precondiciones	El usuario ha guardado previamente una o más consultas en el sistema.
Flujo de acciones	1. El usuario envía una petición de acceder a la lista de consultas almacenadas en el sistema. 2. El sistema recibe la petición y, a partir de la sesión del usuario, obtiene todas las consultas guardadas.
Postcondiciones	El usuario puede ver la lista de consultas guardadas por él mismo.

Tabla 9.7: CU-14: Ver consultas guardadas.

9.2.1 Arquitectura interna

Para implementar los casos de uso de esta iteración se decidió añadir dos aplicaciones con objetivos distintos:

- ***members***. Esta se corresponde a la aplicación encargada de la autenticación de usuarios, apoyada por las herramientas proporcionadas por Django. Incluye los casos de uso CU-08, CU-09 y CU-10
- ***releases_notifier***. Esta aplicación se encarga de todo lo relacionado con guardar consultas y notificar los usuarios. Incluye los casos de uso CU-11, CU-12, CU-13 y CU-14.

A mayores, será necesario automatizar de alguna forma el envío de correos, activando algún proceso de forma periódica que compruebe los resultados consultados por cada usuario y notifique a aquellos que no estén al día. Se buscó alguna opción compatible con Django y se encontraron dos opciones: Celery y *django-crontab*.

Celery es una cola de tareas enfocada en el procesamiento en tiempo real que además permite la programación de tareas, cualidad que se busca para el envío de correos electrónicos. Está desarrollado en Python y es código abierto. En este caso, sería necesario crear un nuevo contenedor que aloje esta herramienta y conectarla con Django por medio de algún *broker*, como RabbitMQ.

django-crontab es una librería desarrollada por usuarios de Django para conectar el *framework* de páginas web con la herramienta que muchos sistemas Linux incluyen, Cron, el cual permite programar tareas periódicamente. Para su correcto funcionamiento, ambas herramientas deben ejecutarse en el mismo contenedor.

Tras contemplar las posibilidades ofrecidas y, en vista de la falta de tiempo de cara a la entrega, se decidió implementar la librería *django-crontab*. Además de requerir menos modificaciones sobre el proyecto y, por lo tanto, ser más sencillo, era una implementación que cumplía con lo necesario para el correcto funcionamiento del sistema. Celery es más sofisticado y podría ser útil para un sistema que programa numerosas tareas. En este caso, solo sería una, el envío de correos electrónicos a los usuarios que no han consultado los nuevos resultados, por lo que quizás es una solución excesiva para la magnitud del problema.

9.2.2 Arquitectura final

Dado que esta es la última iteración, se decidió incluir un esquema de la arquitectura final del sistema, disponible en la figura 9.1

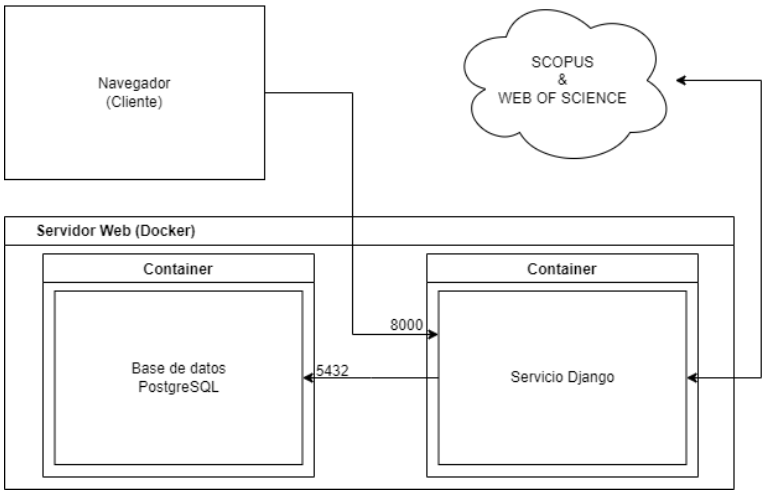


Figura 9.1: Diagrama de la arquitectura del sistema.

9.2.3 Modelado de datos

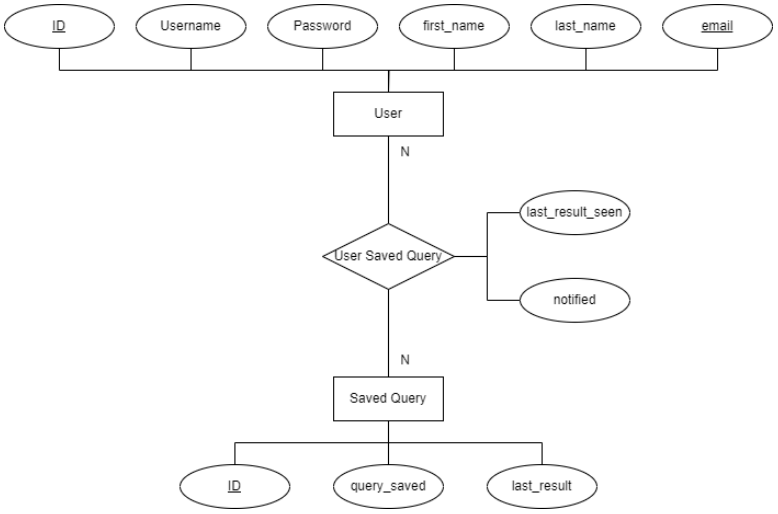
Dado que en esta iteración sí se va a usar una base de datos y a almacenar información es importante definir el diagrama Entidad-Relación y el modelo Relacional resultante.

En la figura 9.2a se muestra el modelo Entidad-Relación (ER). En él se definen dos entidades: las consultas y los usuarios. También se define una relación entre ellas, las consultas guardadas por cada usuario.

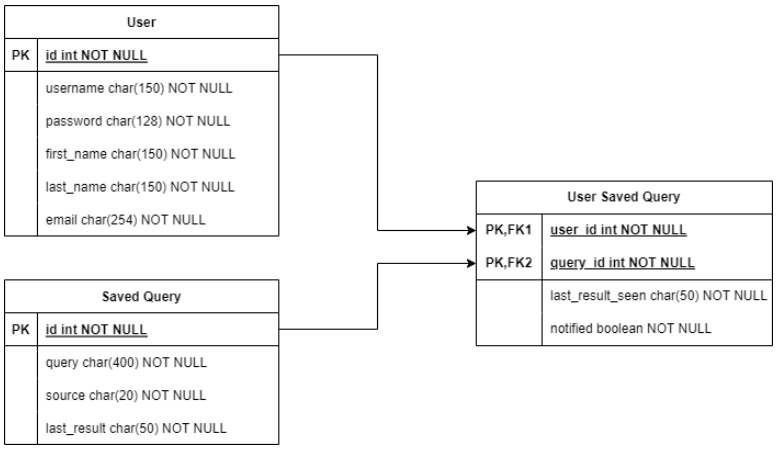
Los atributos de la entidad *User* vienen marcados por el paquete *auth* incluido en Django. La otra entidad, *Saved Query*, representa la consulta (*query*) junto con la base de datos contra la que se hace (*source*) y el último resultado obtenido por el sistema (*last_result*). Por ultimo, la relación *User Saved Query* tiene dos atributos: *last_result_seen*, que permite comprobar si el usuario está al tanto del último resultado almacenado en el sistema sin la necesidad de hacer una consulta; y *notified*, empleado para evitar el *spam* de correos electrónicos y enviar un solo mensaje hasta que se vuelvan a comprobar los resultados.

La figura 9.2b muestra el modelo relacional ideal obtenido del modelo ER. La estructura es similar a lo descrito en las secciones previas: una tabla por cada entidad y otra por la relación *N a N*, con los campos determinados por los distintos atributos descritos en el modelo ER.

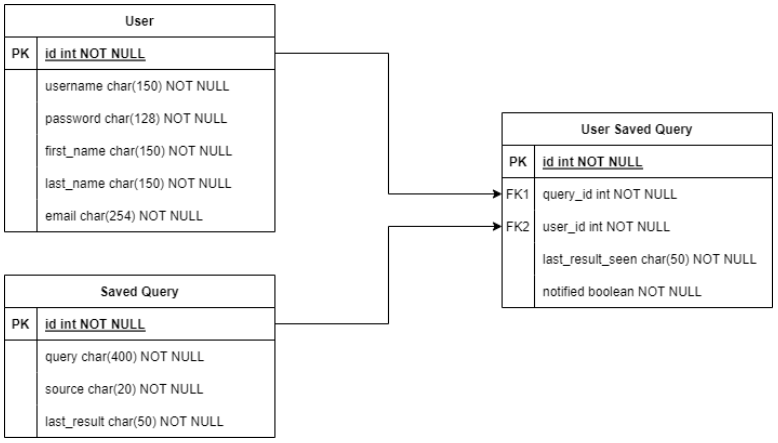
Sin embargo, el modelo es ”ideal” porque, por limitaciones de Django, no es posible definir claves compuestas en las tablas [34]. Si bien esto se podría hacer de una forma más manual, podría generar inconvenientes a la hora de que el *framework* trabajase con la base de datos. En la figura 9.2c se muestra el diagrama relacional finalmente empleado, el cual incluye un identificador adicional para cada consulta guardada por un usuario y que funciona como clave primaria de la tabla. El resultado es ligeramente menos óptimo, ya que se almacena un atributo a mayores de poca utilidad que ocupará espacio en la base de datos.



(a) Diagrama Entidad-Relación



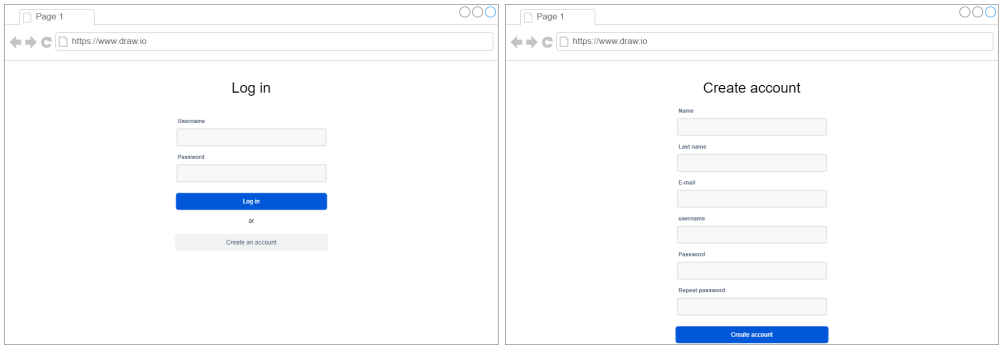
(b) Diagrama relacional obtenido a partir del Diagrama ER



(c) Diagrama Relacional obtenido a partir de Django

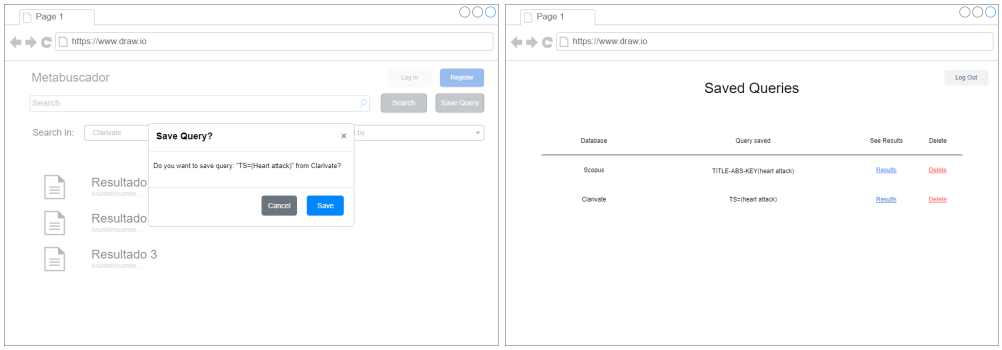
Figura 9.2: Diagramas del modelado de datos

9.2.4 Diseño de interfaz



(a) Vista de inicio de sesión. (b) Vista de creación de cuenta

Figura 9.3: *Mockups* de la aplicación de gestión de usuarios.



(a) Mensaje de confirmación de acción. (b) Vista de todas las consultas guardadas.

Figura 9.4: *Mockups* de la aplicación de guardar consultas.

Para el diseño de la interfaz de la aplicación dedicada a la gestión de usuarios, se diseñaron vistas genéricas con el objetivo de que resultasen familiares e intuitivas para el usuario. Las figuras 9.3a e 9.3b ilustran la idea de la que se partió.

Por otra parte, para el diseño de la interfaz de guardar consultas se tomó cierta inspiración de los ejemplos ya vistos en el capítulo 3. Se decidió separar la vista con las consultas guardadas para no complicar demasiado otras vistas. También se consideró añadir una ventana que permita confirmar la acción de guardar la consulta, ya que podría darse el caso de que el usuario interactuase de forma involuntaria con el botón. La idea inicial está disponible en la figura 9.4.

Como detalle adicional, es necesario permitir el acceso a la aplicación de gestión de usuarios desde las diferentes vistas de búsqueda de la aplicación. Por ello, se añadieron a todas ellas dos botones en la esquina superior izquierda para el inicio de sesión o creación de usuario que, una vez autenticado, pasarían a ser uno solo para el cierre de la sesión. En la figuras

9.4a y 9.4b se pueden ver ejemplos para ambos casos.

9.3 Implementación

El desarrollo de esta iteración fue mucho más fluido que las anteriores por varios motivos. Uno fue la experiencia del autor con el desarrollo de menús de usuario en Django, del cual ya se conocían las herramientas disponibles y se tenía cierta experiencia con ellas. También influyó que la implementación de la base de datos no supuso mayor problema que el comentando previamente en el apartado de Diseño: la restricción de no poder emplear claves compuestas. El *framework* facilitó mucho la gestión relativa a las tablas así como la modificación de la información contenida en ellas.

9.3.1 Configuración del proyecto

Para el correcto funcionamiento del servicio Cron en un contenedor Docker es necesario instalar e iniciar manualmente la aplicación. Para ello, se hizo una pequeña modificación en la imagen de la aplicación web: se incluyó la instalación del ya mencionado *software* de planificación de tareas.

Además, para un correcto funcionamiento del sistema, se debe iniciar el proceso de Cron en segundo plano y después poner en marcha Django.

9.3.2 Aplicación *members*

La aplicación *members* consta de un total de tres *URLs*: una para la vista de inicio de sesión, otra para la del registro de usuarios y una para cerrar sesión automáticamente, sin vista.

Se consideró que no era necesario crear una vista para el cierre de sesión, ya que con mostrar un mensaje que indicase que la acción se había realizado correctamente o, en caso contrario, que se había producido un error, es suficiente.

Para la implementación de los mensajes de confirmación o error entre vistas, se empleó la librería de *messages* incluida en Django. Simplemente debía indicarse al procesar la petición del usuario el texto a mostrar junto a la vista a la que sería redirigido. Por otra parte, lo ideal para un correcto funcionamiento y proporcionar una interfaz amigable, era redirigir al usuario a la vista en la que se encontraba siempre y cuando esto fuese posible. Para ello, se definió una variable en la *URL* para obtener la página desde la que se hacía la petición.

Respecto a los modelos de los usuarios, no fue necesario definirlos ya que *Django* los incluye por defecto. Sí fue necesario definir los formularios empleados en las vistas que, si bien la herramienta los proporciona ya hechos, no incluían todo lo deseado. Se añadieron campos adicionales, como nombre, apellidos y correo electrónico y se modificó el CSS usando las librerías de *django_crispy_forms* y *crispy_bootstrap5*.

9.3.3 Aplicación *releases_notifier*

El primer paso en el desarrollo de esta aplicación fue la definición de modelos. Como se vio en la sección 9.2, debían definirse tres clases: *UserQuery*, *QuerySaved* y *UserSavedQuery*. El código del fichero *models.py* está disponible en el *listing* 9.1, el cual encaja con el modelo definido previamente.

```
1  from django.db import models
2  from django.contrib.auth.models import User
3  class QuerySavedModel(models.Model):
4      query = models.CharField(max_length=400)
5      source = models.CharField(max_length=20)
6      last_result = models.CharField(max_length=50)
7      updated_on = models.DateTimeField(auto_now=True)
8      users = models.ManyToManyField(User,
       through="UserQueryModel")
9  class UserQueryModel(models.Model):
10     query = models.ForeignKey(QuerySavedModel,
       on_delete=models.CASCADE)
11     user = models.ForeignKey(User,
       on_delete=models.CASCADE)
12     last_result_seen = models.CharField(max_length=50)
13     notified = models.BooleanField(default=False)
```

Código 9.1: Código del fichero *models.py* de la aplicación *releases_notifier*

El siguiente objetivo fue establecer las *URLs* así como las vistas pertinentes. Se definieron cuatro, cinco contando la redirección a *index*. Cada dirección se corresponde con un caso de uso de los definidos previamente. Ninguna de las vistas de esta aplicación debe ser accedida por un usuario no registrado, así que, en el caso de que alguno lo intentase, será redireccionado a la vista de *login* de la aplicación *members*.

9.4 Pruebas

Para comprobar el correcto funcionamiento del sistema, se realizaron diversas pruebas. La primera de ellas fue la comprobación del correcto funcionamiento de la aplicación de gestión de usuarios. Para ello, se fueron creando usuarios en el sistema y se probó a realizar las consultas definidas en las iteraciones previas y a guardar algunas de ellas en cada uno.

El siguiente paso fue comprobar el correcto funcionamiento del envío de correos electrónicos. Para ello, se modificó manualmente la información de las consultas guardadas, forzando

así al sistema a comprobar que los resultados no estaban actualizados y, por lo tanto, a enviar la notificación a los correos de los usuarios.

Por último, para probar el sistema en un entorno de producción, se levantó el servicio en una máquina virtual de la UDC y los directores añadieron consultas y recibieron sus respectivos correos electrónicos cuando era necesario.

Capítulo 10

Conclusiones y trabajo futuro

En este capítulo se recapitulará el trabajo realizado y se concluirá el escrito con una serie de mejoras que se consideran de interés de cara al futuro.

10.1 Conclusiones

Tras un largo desarrollo, el sistema cumple con los objetivos indicados en el anteproyecto, los cuales fueron mencionados también en la introducción.

Respecto a otros aspectos, no relacionados con los objetivos funcionales, sino con aquellas características deseables con respecto al producto que se ofrece, se considera que se ha realizado una aproximación sencilla que se adapta al público objetivo de este proyecto. La interfaz es lo más simple posible e incluye toda la información disponible que se consideró necesaria, con un diseño adaptable para permitir su uso en dispositivos de distintas formas y tamaños.

La realización de este trabajo permitió al autor profundizar en varios aspectos relativos al desarrollo de una aplicación como el uso de Django, sobre el cual se ampliaron notablemente los conocimientos; aprendizaje sobre lenguajes con los que no se había trabajado como JavaScript o CSS, y otros con los que se ya había trabajado como Python o HTML; y aplicación de conocimientos y experiencia ganada en el ámbito de gestión y desarrollo de proyectos de este tipo. También recalcar el trabajo realizado con el análisis de las APIs con las que se trabajó.

10.2 Trabajo a futuro

Las principales mejoras que se podrían implementar a futuro en el proyecto son:

- Localizar el proyecto a diferentes idiomas. Se considera que un buen comienzo sería añadir los idiomas español, gallego y portugués.
- Mezclar datos no solo exclusivamente por DOI. Sería ideal emplear un algoritmo que permitiese detectar similitudes en nombre, autor y otros atributos que se consideren.

- Permitir emplear grandes masas de datos a la hora de mezclar datos. La limitación de 100 resultados en algunos casos queda escasa por sí sola.
- Añadir más métodos de notificación para que el usuario pueda ser avisado por el medio que más le convenga. Permitir conectar la aplicación con otras aplicaciones de mensajería como Telegram podría ser un buen paso en esa dirección.
- Conectividad con el servicio de autenticación de la UDC, permitiendo así enviar los avisos directamente al correo del usuario en la institución. Esto permitiría mayor flexibilidad a la hora de usar la aplicación a los miembros de la universidad.
- Mejoras en la interfaz, como la posibilidad de ocultar la consulta con un desplegable. También sería conveniente implementar tecnologías como AJAX en las vistas de resultados, evitando recargar la página completa.

Apéndices

Apéndice A

Tablas relación entre APIs

A continuación se muestran las distintas tablas de relación entre los diferentes parámetros de las APIs.

ORDENACIÓN:	WOS	Scopus
Load Date	Y	Y
Author	Y	Y
Conference Title	Y	
Cited Year	Y	
Local Times Cited	Y	
pagecount		Y
PubYear	Y	Y
Relevance	Y	Y
Source	Y	
Times Cited	Y	Y
Volume	Y	Y
Article Number		Y
pageFirst (or pages)	Y	Y
pagerange		Y
PubName		Y

Figura A.1: Campos de ordenación.

Tabla relación APIS	APIs (Aplicación, vista)			
Valores esperados	**Scopus Search (Scopus, complete)**	**Article Metadata (SD, complete)**	**WoS lite**	**WoS expanded (+ completa)**
Info Artículos				
Título	dc:title	dc:title	Title.Title	static_data.sumary.titles.title(type=item)
ID empleado	dc:identifier	dc:identifier	UT	UID
DOI	prism:doi	dc:identifier//prism:doi	Other.Identifier.Doi	dynamic_data.cluster_related.identifiers.identifier(type=doi)
EID	eid	eid		dynamic_data.cluster_related.identifiers.identifier(type=eid)
PubMed ID	pubmed-id			dynamic_data.cluster_related.identifiers.identifier(type=pubmed)
Autor/es	dc:creator//author *(all authors)*	dc:creator//authors *(all authors)*	Author.[Authors//BookAuthors//BookGroupAuthors] (listas)	static_data.summary.names.name(role=author)
authname	(author.)authname	author		display_name
url	(author.)author-url			
Institución	(author.)afid			
Institución	affiliation.			fullrecord_metadata.addresses.address_name.address_spec.organizations
nombre	affilname			organization(pref=Y)
id	afid			
ciudad	affiliation-city			
pais	affiliation-country			
Descripción (Abstract)	dc:description	dc:description//prism:teaser		static_data.fullrecord_metadata.abstracts.abstract.abstract_text (lista)
Tipo de documento	subtype//subtypeDescription	pubType	Doctype.Doctype (lista)	static_data.summary.doctypes.doctype//static_data.fullrecord_metadata.normalized_doctypes.doctype (lista)
Fecha publicación				static_data.summary.pub_info.sortdate//dates.date_created
Veces citado	citedby-count			dynamic_data.citation_related.tc_list.silo_tc.local_count
Palabras clave	authkeywords	authkeywords	Keyword.Keywords (lista)	static_data.fullrecord_metadata.keywords.keyword (lista)
Número del artículo	article-number		Other.Identifier.article_no (lista)	
Fundación	fund-arc//fund-no//fund-sponsor			static_data.fullrecord_metadata.fund_ack[grants.grant (lista) //fund_text]
Info Fuente				
Título	prism:publicationName	prism:publicationName	Source.SourceTitle (lista)	sumary.titles.title(type=source)
ISSN/ISBN	prism:issn/prism:isbn	prism:issn/prism:isbn	Other.Identifier.[Eissn//Issn]/Other.Identifier.Isbn (listas)	dynamic_data.cluster_related.identifiers.identifier(type=issn//eissn/isbn//eisbn)
PII	pii	pii		
Categoría (revista, libro...)	prism:aggregationType	prism:aggregationType		static_data.summary.pub_info.pub_type
Editorial				static_data.summary.publishers.publisher.names.name (lista)
Edición		prism:edition		
Volumen	prism:volume	prism:volume	Source.Volume (lista)	static_data.summary.pub_info.vol
Issue	prism:issueIdentifier	prism:issueIdentifier//prism:issueName	Source.[Issue//SpecialIssue//BookSeriesTitle] (listas)	static_data.summary.pub_info.[issue//special_issue]
Páginas	prism:pageRange	prism:startingPage&prism:endingPage	Source.Pages (lista)	static_data.summary.pub_info.page (contiene info. anidada)
Fecha (publicación fuente)	prism:coverDate//prism:coverDisplayDate	prism:coverDate//prism:coverDisplayDate	Source.Published.[BiblioDate//BiblioYear]	static_data.summary.pub_info.[coverdate//pubyear]
Paginación				
paginación	link.[self//first//prev//next//last]	link.[self//first//prev//next//last]		
cursor	cursor.[current//next]			
Otros				
Enlace a recurso en API	link.(ref=self)			
Enlace a página web	link.(ref=scopus)			links.record (flag *links* activada)
Copyright		prism:copyright		
Disclaimer			Other.ResearcherID.Disclaimer (lista)	r_id_disclaimer
Fecha contenido disponible		available-online-date//vor-available-online-date		
openaccess flag	openaccess(int)//openaccessFlag(bool)	openaccesss//openaccessArticle		
		openArchiveArticle//opeaccessUserLicense		
Base de datos				static_data.item.coll_id
Idiomas disponibles				static_data.fullrecord_metadata.languages.language (lista)

Figura A.2: Relación de valores devueltos por las APIs.

SCOPUS	WOK	WOS
ALL		ALL (All fields)
ABS**		AB (Abstract)
AF-ID**		
AFFIL [AFFILCITY, AFFILCOUNTRY, AFFILORG]**	OG (Organization-enhanced)	OG (Organization-enhanced)
	OO (Organization)	OO (Organization)
	AD (Address) // SA (Street Address)	AD (Address) // SA (Street Address)
	PS (Province/State)	
AFFILCITY	CI (City)	
AFFILCOUNTRY	CU (Country/Region)	CU (Country/Region)
AFFILORG		
	SG (Suborganization)	SG (Suborganization)
ARTNUM**		
	UT (Accession Number)	UT (Accession Number)
AU-ID**	AI (Author Identifiers)	AI (Author Identifiers)
AUTHOR-NAME [AUTHFIRST, AUTHLASTNAME, AUTHSUFFIX, AUTHNAME]	AU (Author)	AU (Author)
AUTH [AUTHFIRST, AUTHLASTNAME]**		
AUTHFIRST		
AUTHLASTNAME		
AUTHSUFFIX		
AUTHNAME		
AUTHCOLLAB**	GP (Group Author)	GP (Group Author)
AUTHKEY		AK (Author kewords) // KP (Keyword Plus)
CHEM [CASREGNUMBER, CHEMNAME]		
CHEMNAME**		
CASREGNUMBER		
CODEN**		
CONF [CONFNAME, CONFSPONSORS, CONFLOC]**	CF (Conference)	CF (Conference)
CONFLOC		
CONFSPONSORS		
CONFLOC		
DOCTYPE		
PUBSTAGE		
DOI**	DO (DOI)	DO (DOI)
EDITOR [EDFIRST, EDLASTNAME]**	ED (Editor)	ED (Editor)
EDFIRST		
EDLASTNAME		
EISSN		
EXACTSRCTITLE		
FIRSTAUTH		
FUND-SPONSOR	FO (Funding Agency)	FO (Funding Agency)
FUND-ACR		
FUND-NO	FG (Grant Number)	FG (Grant Number)
		FD (Funding Details)*
INDEXTERMS		
ISBN**	IS (ISSN/ISBN)	IS (ISSN/ISBN)
ISSN**	IS (ISSN/ISBN)	IS (ISSN/ISBN)
ISSNP		
ISSUE**		
KEY [AUTHKEY, INDEXTERMS, TRADENAME, CHEMNAME]**		
LANGUAGE**	*** esta opción está disponible como parametro	
MANUFACTURER**		
OPENACCESS		
PAGES [PAGEFIRST, PAGELAST]**		
PAGEFIRST		
PAGELAST		
PMID	PMID (PubMed ID)	PMID (PubMed ID)
PUBLISHER**		PUBL (Publisher)
PUBYEAR**	PY (Year Published)	PY (Year Published) // FPY (Final Publication Year)
		DOP (Date of Publication)
REF [REFAUTH, REFTITLE, REFSRCTITLE, REFPUBYEAR, REFPAGE]**		
REFAUTH		
REFTITLE		
REFSRCTITLE		
REFPUBYEAR		
REFPAGE		
REFARTNUM		
SEQBANK**		
SEQNUMBER**		
SRCTITLE**	SO (Publication Name)	
SRCTYPE		
	SU (Research Area)	SU (Research Area)
SUBJAREA	WC (Web of Science Category)	WC (Web of Science Category)
TITLE**	TI (Title)	TI (Title)
TITLE-ABS-KEY	TS (Topic)	TS (Topic)
TITLE-ABS-KEY-AUTH		
TRADENAME		
VOLUME**		
WEBSITE		
	ZP (Zip / Postal Code)	
** Los apartados marcados son incluidos en ALL		* Los apartados marcados NO están incluidos en ALL

Figura A.3: Campos de filtrado.

Bibliografía

[1] "Python wiki, beginners guide," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://wiki.python.org/moin/BeginnersGuide/Overview

[2] Mozilla Foundation, "Html: Lenguaje de etiquetas de hipertexto," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/HTML

[3] ——, "Javascript," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/JavaScript

[4] ——, "Css," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://developer.mozilla.org/es/docs/Web/CSS

[5] Django Software Foundation, "Django web page: overview," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://www.djangoproject.com/start/overview/

[6] Miguel Araujo, Daniel Feldroy and contributors, "Django crispy forms documentation," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://django-crispy-forms.readthedocs.io/en/latest/

[7] "Django crontab documentation," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://pypi.org/project/django-crontab/

[8] J. Spurlock, *Bootstrap: responsive web development.* "O'Reilly Media, Inc.", 2013, ch. 1.

[9] "Pandas web page," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://pandas.pydata.org/about/

[10] "Psycopg 2," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://pypi.org/project/psycopg2/

[11] K. Reitz, "requests," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://pypi.org/project/requests/

[12] "Elsapy github," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://github.com/ElsevierDev/elsapy

[13] "smtplib - smtp protocol client," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://docs.python.org/3/library/smtplib.html

[14] "Visual studio code web page," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://code.visualstudio.com/

[15] "Docker web page," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://www.docker.com/

[16] "Postgresql web page," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://www.postgresql.org/about/

[17] "Elsevier, página web oficial," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://www.elsevier.com/es-es/about

[18] E. B. V., "Elsevier developer portal," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://dev.elsevier.com/

[19] "Clarivate web page," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://clarivate.com/about-us/

[20] "Clarivate developer portal," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://developer.clarivate.com/

[21] Elsevier, "Scopus fact sheet," 2021, consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://www.elsevier.com/__data/assets/pdf_file/0017/114533/Scopus-fact-sheet-2022_WEB.pdf

[22] "Web of science core collection web page," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://clarivate.com/products/scientific-and-academic-research/research-discovery-and-workflow-solutions/webofscience-platform/web-of-science-core-collection/

[23] "Google academico," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://scholar.google.es/

[24] N. R. Smalheiser, "Design and implementation of metta, a metasearch engine for biomedical literature retrieval intended for systematic reviewers," *Health Inf Sci Syst.*, 2014.

[25] "Ieee standard for testability and diagnosability characteristics and metrics," *IEEE Std 1522-2004*, pp. 1–35, 2005.

[26] "Boletín oficial del estado, disposición 17238, número 117 de 2023," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://www.boe.es/boe/dias/2023/07/26/pdfs/BOE-A-2023-17238.pdf

[27] "Search tips for scopus search api," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://dev.elsevier.com/sc_search_tips.html

[28] "Search tips for article metadata api," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://dev.elsevier.com/sd_article_meta_tips.html

[29] J. Bucanek, "Model-view-controller pattern," *Learn Objective-C for Java Developers*, pp. 353–402, 2009.

[30] "Django faq," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://docs.djangoproject.com/en/4.2/faq/general/

[31] O. Lizama, G. Kindley, J. J. Morales, and A. Gonzales, "Redes de computadores arquitectura cliente-servidor," *Universidad Tecnica Federico Santa Maria*, pp. 1–8, 2016.

[32] "Dev containers," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://containers.dev/

[33] Y. Jiang, C. Lin, W. Meng, C. Yu, A. M. Cohen, and N. R. Smalheiser, "Rule-based deduplication of article records from bibliographic databases," *Database*, vol. 2014, 2014.

[34] "Extra fields on many-to-many relationships django documentation," consultado el 11 de septiembre de 2023. [En línea]. Disponible en: https://docs.djangoproject.com/en/3.2/topics/db/models/#extra-fields-on-many-to-many-relationships