Universidad Zaragoza

Trabajo Fin de Grado

Herramienta de población automática de ontologías con fuentes públicas externas

Autor

Luis García Garcés

Directoras

Paula Peña Larena

Raquel Trillo Lado

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2021

Escuela de
Ingeniería y Arquitectura
Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Luis García Garcés,

con nº de DNI 77215691E en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado Ingeniería Informática (Título del Trabajo) Herramienta de población automática de ontologías con fuentes públicos externas,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 31 de agosto de 2021

Fdo: Luis

AGRADECIMIENTOS

Quisiera comenzar este apartado dando gracias a mis tutores. Gracias a Raquel Trillo, Paula Peña y Rafael del Hoyo por guiarme durante el desarrollo de este proyecto, resolver mis dudas, aguantar las decenas de correos que enviaba cada semana y en especial por dedicar su tiempo de vacaciones en ayudarme a completar este trabajo y corregirlo. A su vez también me gustaría agradecer al Instituto Tecnológico de Aragón la oportunidad haber hecho prácticas en sus instalaciones durante el verano. Espero sinceramente haber estado a la altura y que se sientan satisfechos de haber trabajado conmigo y el fruto de ese trabajo. También me gustaría dedicárselo a mis padres porque siempre han creído en mi, me han ayudado en todo lo posible, apoyado siempre y me han dado toda su paciencia. Con este trabajo se cierra una etapa de mi vida muy importante, durante estos últimos años me he reído, he disfrutado, he aprendido, me he quejado, he sido presa de la ansiedad y me ha dolido la espalda una barbaridad, porque considero que el ritmo de trabajo de algunos cursos no es muy sano... Me gustaría darle las gracias a todas las personas que han estado a mi lado durante esta etapa, una vez más gracias a mis padres por estar siempre ahí. Gracias a mis compañeros de carrera por acompañarme durante esta etapa y haber hecho piña para afrontarla, echaré de menos los días en los que nos juntábamos todos a comer y como nos ayudábamos en épocas de exámenes. Gracias a mis amigos por ayudarme a desconectar y sacar siempre un hueco para escaparnos. Gracias a mi familia por estar siempre ahí y quererme tal y como soy. En definitiva muchas gracias a las personas que han estado ahí durante estos últimos años.

RESUMEN

Actualmente hay multitud de fuentes de datos libres y heterogéneas de las que se pueden obtener datos para un propósito específico. Integrar estos datos en general puede resultar costosos ya que cada fuente sigue su propio esquema. Es por tanto necesario llevar a cabo un proceso de integración sistemático que cree un esquema que aporte una visión global enriquecida y consistente de los diferentes componentes. Es aquí donde las ontologías juegan un papel clave debido a su esquema y a la capacidad de establecer nuevas relaciones. Una ontología es una definición formal que representa un conocimiento, mediante un esquema cuyos conceptos, propiedades y relaciones constituyen la especificación formal de un área de conocimiento. En el presente trabajo se va desarrollar una herramienta cuyo propósito es extraer conjuntos de datos de fuentes públicas y poblar automáticamente una ontología. Para su desarrollo se ha utilizado el *framework* de Python Flask integrado con una base de datos orientada a grafos, en este caso Neo4j. Aunque la extracción de los conjuntos de datos depende exclusivamente de la fuente de datos, la extracción se lleva cabo mediante el uso de la API que exponen las fuentes. Tras el proceso de extracción se ha utilizado la librería **rdflib** que permite crear grafos RDF.

Los grafos RDF se importan en Neo4j mediante el uso del *plugin* **Neosemantic**, que permite trabajar con tripletas RDF. Estas tripletas se crearán acorde a los conceptos y propiedades propios del modelo de información IDS.

Este proyecto se enmarca en el trabajo realizado en el área de Biga data y sistemas cognitivos del Instituto Tecnológico de Aragón, desde la cual se se estableció como requisito del trabajo el uso de la ontología desarrollada por IDSA[2] y que la herramienta permitiese la visualización de los conjuntos de datos integrados y la interacción del usuario con estos a través de una interfaz web. Para alcanzar el objetivo se ha utilizado **Jquery** y **Bootstrap** para desarrollar una interfaz rápida y simple. Finalmente se ha utilizado procesamiento del lenguaje natural para enriquecer las búsquedas que los usuarios realicen a **Neo4j** mediante el uso de la librería TXTAI para Python.

[2]https://internationaldataspaces.org/

ABSTRACT

Currently, there are a multitude of free and heterogeneous data sources from which data can be obtained for a specific purpose. Generally, the integration of this data can be costly since each source follows its own scheme. Therefore, it is needed to perform a systematic integration process in order to provide a global schema that provides more information. This is where ontologies play a key role due to their schema and the ability to establish new relationships. An ontology is a formal definition that represents knowledge, through a scheme whose concepts, properties and relationships constitute the formal specification of an area of knowledge. In the present work, a tool whose main purpose is the extraction of data sets from public sources and automatically populate an ontology is presented. Python Flask framework integrated with a graph-oriented database, in this case Neo4j, have been the tools selected to development this project. Although the extraction of the datasets depends exclusively on the public data source, the extraction is carried out using the API that the sources expose. **Rdflib** have been used right after the extraction. It is a Python library which allows user to create RDF graphs and export them. Because of the used of the **Neosemantic** plugin, which allows working with RDF triples, RDF graphs are imported into Neo4j. These triples will be created according to the concepts and properties of the IDS information model. The ontology developed by IDSA has been imposed by the coordinators of this project belonging to the Technological Institute of Aragon, due to the fact that this project is framed within the lines of the Institute. The tool has to allow the visualization of the data sets and user interaction through a web interface. For this, **Jquery** and **Bootstrap** have been used to develop a fast and simple interface. Finally, natural language processing has been used to enrich the searches made to **Neo4j** through the use of the TXTAI library for Python.

Índice

1. Introducción **11**
1.1. Contexto y Motivación 11
1.2. Objetivos 11
1.3. Estructura de la memoria 12

2. Visión general tecnológica **13**
2.1. Tecnologías de la Web semántica 13
2.1.1. RDF 13
2.1.2. OWL 14
2.2. Bases de datos 14
2.2.1. Bases de datos SQL 15
2.2.2. Base de datos NOSQL 15
2.2.3. Bases de datos orientadas a grafos 15
2.2.4. Modelo de base de datos elegida 16
2.2.5. Base de datos elegida 16
2.3. Procesamiento de lenguaje natural e indexación 17
2.3.1. Procesamiento de lenguaje natural 17
2.3.2. Indexación 17
2.3.3. Librería TXTAI 17
2.3.4. Justificación uso de TXTAI 18
2.4. Concepto de Ontología 18
2.4.1. Componentes 18
2.4.2. Ventajas 19
2.4.3. Dificultades 19
2.4.4. Selección del modelo de información IDS 20

3. Herramienta de población automática de ontologías con fuentes públicas externas **21**
3.1. Análisis 21
3.1.1. Requisitos del proyecto 21

3.1.2. Casos de uso . . . 22
3.2. Arquitectura . . . 24
3.2.1. Diagrama alto nivel . . . 24
3.2.2. Modelo entidad-relación . . . 25
3.3. Mapa de navegación . . . 26

4. Prototipo del sistema y Validación **29**
4.1. Backend . . . 29
4.1.1. Extracción de conjuntos de datos . . . 29
4.1.2. Población automática de la ontología . . . 31
4.1.3. Índices de búsqueda . . . 33
4.1.4. Búsqueda local de conjuntos de datos . . . 34
4.1.5. Conjuntos de datos recurrentes . . . 35
4.2. Frontend . . . 35

5. Conclusiones y Líneas de Trabajo Futuro **37**
5.1. Resultados . . . 37
5.1.1. Búsqueda en repositorios online . . . 37
5.1.2. Búsqueda por palabra clave . . . 41
5.2. Dificultades y problemas encontrados . . . 41
5.3. Metodología de desarrollo . . . 42
5.4. Conclusiones . . . 43
5.5. Planificación del proyecto . . . 43
5.6. Líneas de Trabajo Futuro . . . 43

Anexos **52**

A. Anexo 1 **55**
A.1. Ejemplo conjunto de datos en repositorio online . . . 55
A.2. Ejemplo tráfico red en repositorio online . . . 56
A.3. Estado del arte tecnologías bases de datos . . . 57
A.3.1. TerminusDB . . . 57
A.3.2. ArangoDB . . . 57
A.3.3. Neo4j . . . 58
A.4. Conceptos procesamiento lenguaje natural . . . 58
A.4.1. Word Embeddings . . . 58
A.4.2. Word2Vec . . . 58
A.4.3. GloVe . . . 59

A.4.4. Transformers . . . 59
A.5. Tecnologías desarrollo frontend . . . 60
A.5.1. Bootstrap . . . 60
A.5.2. JQuery . . . 61
A.5.3. Google Fonts . . . 61
A.5.4. Fontawesome . . . 61
A.6. Vocabulario utilizado . . . 61
A.6.1. Web Ontology Language . . . 61
A.6.2. Resource Description Framework . . . 61
A.6.3. Dublin Core . . . 62
A.6.4. Modelo de Información IDS . . . 62

B. Anexo 2 **65**
B.1. Explicación requisitos funcionales . . . 65
B.2. Explicación requisitos no funcionales . . . 66
B.3. Modelo de datos orientado a grafos . . . 67
B.4. Prototipado interfaces . . . 67
B.4.1. Pantalla de búsqueda avanzada . . . 67
B.4.2. Pantalla principal . . . 68
B.4.3. Pantalla de visualización de un conjunto de datos . . . 68
B.4.4. Pantalla de información . . . 69

C. Anexo 3 **71**
C.1. Diagrama de paquetes Frontend . . . 71
C.2. Interfaces . . . 71
C.2.1. Página inicial . . . 71
C.2.2. Búsqueda Avanzada . . . 73
C.2.3. Resultado búsqueda por publicador . . . 73
C.2.4. Resultado búsqueda por palabra clave . . . 74
C.2.5. Conjuntos de datos recurrentes . . . 75
C.2.6. Página información del uso de la herramienta . . . 76
C.2.7. Resultado de la búsqueda . . . 79
C.3. Diagramas de clase Backend . . . 82

D. Anexo 4 **85**
D.1. Repositorios públicos . . . 85
D.1.1. data.europa.eu . . . 85
D.1.2. data.gov.au . . . 85

D.2. Conjunto de datos para validación 85
D.3. Conjunto de datos validación con clave 86
D.4. Palabras clave ejemplo búsqueda 87
D.5. Resultado búsqueda avanzada 88
D.6. Resultado búsqueda por clave 89
D.7. Control de esfuerzo 90

E. Anexo 5 **95**
E.1. Extracción de información mediante *web crawling* y *web scraping* . . . 95
E.1.1. Web crawling 95
E.1.2. Web scraping 95
E.1.3. Tecnología crawler analizada 96
E.1.4. Inconvenientes uso de crawler 96
E.1.5. Solución alternativa 97
E.2. Estado del arte tecnologías *web scraping* y *crawling* 98
E.2.1. Apache Nutch 98
E.2.2. Stormcrawler 99
E.2.3. *crawler*4j 99
E.2.4. Scrapy 100
E.2.5. Beautiful Soup 100
E.2.6. Requisitos del proyecto para crawlers 100

F. Anexo 6 **103**
F.1. Glosario siglas y abreviaturas 103

Lista de Figuras **107**

Lista de Tablas **109**

Capítulo 1

Introducción

Existen multitud de fuentes de datos libres y heterogéneas de las que se pueden obtener datos para un propósito específico, pero en general resulta costoso integrarlos ya que previamente se requiere localizarlos, limpiarlos y homogeneizarlos. El problema a resolver en este proyecto consiste en la estructuración de la información de repositorios públicos de datos heterogéneos de forma automática gracias al uso de ontologías. Una ontología es un mecanismo de representación en forma de esquema conceptual con sus propios conceptos y vocabulario que permiten la estructuración y representación de un conocimiento (ver Sección 2.4 para más detalles). Durante el desarrollo de este proyecto se se analiza cómo es posible la integración de forma sistemática de los conjuntos de datos pertenecientes a repositorios diferentes gracias al uso de ontologías.

1.1. Contexto y Motivación

Este trabajo de final de grado se enmarca dentro de las líneas del Instituto Tecnológico de Aragón, también conocido como **ITAINNOVA**[3]. Se trata de un proyecto propuesto por el departamento de *Big Data y Sistemas Cognitivos*, departamento en el actual he estado haciendo prácticas desde junio de 2021 hasta el día de hoy (31/08/2021). Código disponible en `https://github.com/luisgg98/TFG-ITA-EINA`.

1.2. Objetivos

El objetivo del proyecto es el desarrollo de una herramienta que permita la extracción de información de fuentes públicas para poblar automáticamente una ontología. Para ello, además de capturar datos y adaptarlos creando las relaciones semánticas necesarias, se deberá mantener la trazabilidad de las operaciones realizadas

[3]https://www.itainnova.es/

sobre estos y su origen, con el objetivo de fomentar y garantizar una mayor calidad de los datos. Adicionalmente, se requiere que la herramienta permita la visualización de los resultados poblados en la ontología. En resumen, la herramienta ha de poder extraer los datos de los repositorios, estandarizar su información acorde con la ontología, poblar la ontología y visualizar su contenido. Los requisitos que han de cumplir se detallan en la Subsección 3.1.1.

1.3. Estructura de la memoria

En el Capítulo 2 se explican los conceptos y las tecnologías necesarios para la correcta comprensión y el correcto desarrollo de este proyecto.

En el Capítulo 3 se analizan los requisitos que ha de cumplir el proyecto, se muestran los diagramas acordes al diseño de la herramienta y los primeros prototipos de su interfaz.

El Capítulo 4 describe cómo se ha llevado a cabo la implementación de las tecnologías explicadas anteriormente para cumplir los requisitos y objetivos del proyecto propuestos.

El Capítulo 5 resume las conclusiones a las que se ha llegado tras haber completado este trabajo y también contiene los resultados.

Con el fin de facilitar la lectura de esta memoria la información complementaria de cada capítulo ha sido incluida en cincos anexos, que son accesibles desde el texto principal. Al final de esta memoria se encuentra un glosario en el que se explican los términos y siglas utilizadas en Apéndice F.

Capítulo 2

Visión general tecnológica

En este capítulo se llevará a cabo una explicación de los conceptos y tecnologías necesarios para el correcto entendimiento de este proyecto.

2.1. Tecnologías de la Web semántica

Antes de comenzar es necesario aclarar el significados de dos tecnologías, RDF y OWL, propios de la web semántica. La Web semántica supone un cambio de filosofía y uso de la Web. Se trata de una Web en la que los agentes software pueden interactuar de forma automática, integrar y reusar información basándose en la idea de agregar metadatos semánticos y ontológicos a la Web [1].

2.1.1. RDF

RDF[4], cuyas siglas significan **Resource Description Framework** establece un modelo estándar que sirve para proporcionar información descriptiva de los recursos disponibles en la Web, facilitando el intercambio de datos en la Web y la descripción de las relaciones entre los diferentes recursos disponibles. Un recurso puede ser un concepto del que obtener información [2].

El componente principal de RDF son las tripletas formadas por Sujeto, Predicado y Objeto (ver en Figura 2.1). En las tripletas (Sujeto, Objeto, Predicado) el sujeto y el objeto representan recursos y el predicado una propiedad.

Las tripletas se pueden representar de múltiples formas y sintaxis. En la representación de las tripletas el sujeto puede ser una URI (Identificador de Recursos Uniforme) o un nodo en blanco, en cambio, el objeto también puede ser un literal. El predicado en cambio es una URI. Tanto los recursos como las propiedades se identifican con URIs. Las URIs son identificadores únicos [3].

[4]http://www.w3.org/1999/02/22-rdf-syntax-ns

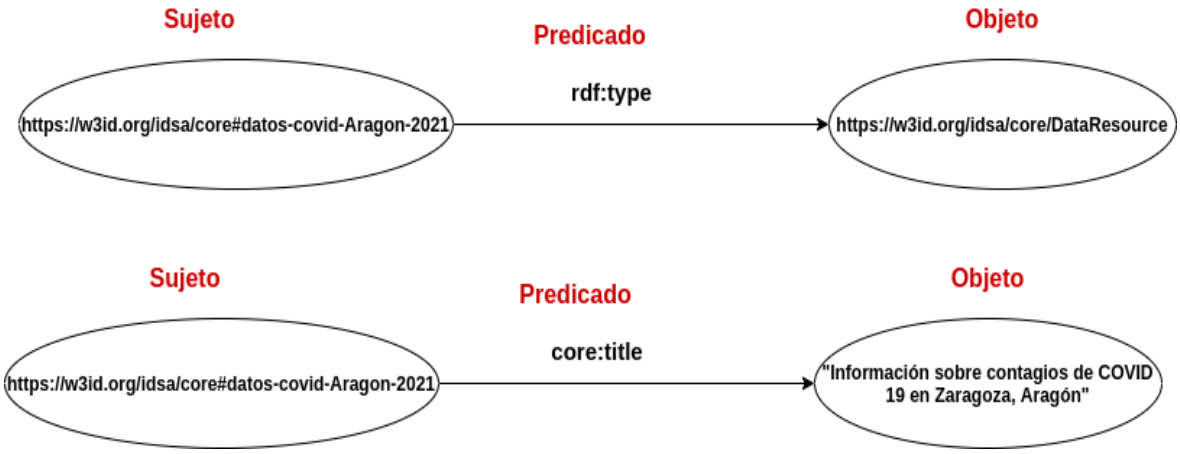

Figura 2.1: Ejemplos de tripletas RDF

2.1.2. OWL

OWL[5] es el acrónimo del inglés **Web Ontology Language**. Es un lenguaje declarativo usado en la web para describir ontologías. Este lenguaje de marcado permite publicar y compartir datos a través de la descripción y definición de ontologías. OWL se basa en RDFS, que se refiere a **RDF Schema**. Este último es un lenguaje basado en RDF cuya finalidad es la de definir vocabularios para RDF. Simplificando su definición podría decirse que se trata de un lenguaje para definir metadatos para RDF. Los metadatos se conocen como "datos acerca de los datos" y sirven para suministrar información sobre los datos producidos [4].

Sin embargo, OWL agrega más vocabulario para describir propiedades y clases: entre otros, relaciones entre clases, cardinalidad, igualdad, tipificación más rica de propiedades, características de propiedades y clases enumeradas [5].

2.2. Bases de datos

En esta sección inicialmente se hará una breve introducción a las bases de datos SQL y NOSQL. Dentro de las NOSQL se va a detallar qué son las bases de datos orientadas a grafos. Posteriormente se justificará porqué se ha decidido utilizar en este proyecto una base de datos orientada a grafos. Finalmente se explicará qué otras bases de datos se han decidido utilizar en el proyecto.

[5]http://www.w3.org/2002/07/owl

2.2.1. Bases de datos SQL

Las bases de datos SQL son aquellas que utilizan el lenguaje SQL para definir su esquema y para las operaciones de actualización, borrado y creación. Las bases de datos SQL siguen el modelo relacional que permite la representación de datos relacionados en tablas [6]. Los datos se almacenan como filas de tablas que han sido previamente creadas y definidas. Estas tablas están formadas por columnas que se corresponden a los atributos que queremos guardar de la información que estemos almacenando. Todos los datos que se guardan han de tener los mismos atributos dentro la tabla. Las datos de las diferentes tablas se relacionan entre sí gracias al uso de claves, uso de clave primarias para identificar a la propia columna y el uso de claves ajenas para identificar con qué otras instancias de la base de datos se relaciona. Los sistemas gestores de bases de datos que sigue el modelo relacional se caracterizan garantizar la coherencia y la disponibilidad de los datos a expensas de la tolerancia a las particiones [7].

2.2.2. Base de datos NOSQL

Las bases de datos **NOSQL** son todas aquellas que presentan un modelo distinto al modelo relacional. Los sistemas gestores de bases de datos NOSQL no tienen un esquema rígido predefinido, puede haber redundancias en los datos ya sea para ofrecer tolerancia a fallos o para evitar el uso de **operaciones JOIN**. Estas operaciones son aquellas que unen el contenido de varias tablas y generan un vista a partir de su unión. Las transacciones de bases de datos NOSQL tienden a cumplir las propiedades **BASE** (Basically Available, Soft State, Eventual and Consistency) [8]. No todas las bases de datos NOSQL cumplen los mismos puntos del **teorema CAP** (Consistency, Availability and Partition Tolerance) para ser escalables y distribuidas. Aunque, generalmente priorizan más la tolerancia a particiones y la disponibilidad que la consistencia [9].

Muchos de los SGBD (Sistema Gestor de Bases de Datos) NOSQL son proyectos de código abierto. Dentro de las bases de datos NOSQL hay diferentes tipos: Bases de datos documentales, bases de datos clave/valor, bases de datos multivalor, bases de datos de Arrays, etc. (véase más en [10]) En este proyecto nos centraremos en las bases de datos orientadas a grafos.

2.2.3. Bases de datos orientadas a grafos

Una base de datos orientada a grafos es una base de datos NOSQL cuya finalidad es la de representar un conocimiento mediante la creación y manipulación de grafos. Estos grafos contienen nodos y propiedades utilizados para almacenar datos y representar las

relaciones entre estos de forma diferente a las bases de datos relacionales [11].

2.2.4. Modelo de base de datos elegida

Finalmente, se ha decidido el uso de una base de datos orientada a grafos para este proyecto, en lugar del uso de una base de datos relacional. En las bases de datos orientadas a grafos se evita el uso de JOINs debido a que se almacenan las relaciones entre distintas instancias de los datos, por lo tanto a la hora de realizar una consulta es mucho menos costoso. Los grafos permiten un estudio más flexible de los datos y aplicar algoritmos de grafos para detectar patrones, relaciones entre los datos implícitas o factores influyentes. En definitiva permite un análisis más eficiente y rápido que en una base de datos relacional tradicional, ya que el propio sistema gestor facilita la aplicación de algoritmo y evita el uso de JOINs.

Debido al almacenamiento explícito de las relaciones entre grafos y que cuentan con operaciones para navegar rápidamente entre estos, permiten una mejor representación del conocimiento que en una base de datos relacional [12]. El objetivo de este proyecto consiste en la población automática de una ontología. Las ontologías definen un vocabulario y cómo se relacionan los elementos que la componen. Al fin y al cabo las ontologías permiten las representación de un conocimiento. Los SGBD que más se adaptan a la idea de representar un conocimiento son los orientados a grafos. Se ha llevado a cabo un análisis de bases de datos orientadas a grafos disponible en Sección A.3.

2.2.5. Base de datos elegida

Finalmente, de los tres sistemas gestores orientados a grafos candidatos, explicados en la Sección A.3, se ha elegido Neo4j. Otros ejemplos son, **ArangoDB** y **TerminusDB**, ambos son dos proyectos que llevan mucho menos tiempo activos en comparación con Neo4j y esto refleja que Neo4j cuenta con una comunidad más numerosa de usuarios y que cuenta con el soporte de una empresa. Pueden encontrarse numerosos tutoriales y recursos adicionales, como libros gratuitos (por ejemplo véase en [13] y [14]), para aprender a manejar Neo4j. A esto se le suma su documentación, ya que es abundante y clara tanto para el uso de la base de datos, como para el aprendizaje de CYPHER, su lenguaje de consulta, como para su *plugin* **Neosemantic**. Cierto es que nativamente no soporta RDF y que plantea un enfoque distinto, porque se trata de una base de datos orientada a grafos de propiedades y no de tripletas [15], pero con el *plugin* **Neosemantic** se solventa. Además, cuenta con una aplicación de escritorio que permite visualizar, administrar la base de datos e instalar *plugins*.

2.3. Procesamiento de lenguaje natural e indexación

Una vez haya sido poblada la ontología han de poder realizarse consultas enriquecidas semánticamente a la base de datos. Para cumplir este objetivo es necesario el uso de una librería que permita la búsqueda semántica y creación de un índice sobre el que realizar estas consultas. Antes de describir la tecnología elegida y explicar porqué ha sido seleccionada, se describirán brevemente los conceptos procesamiento de lenguaje natural, el cual será referido como PLN, e indexación. En esta sección se hará mención a dos técnicas de aprendizaje en procesamiento del lenguaje natural *word embedding* y *transfomers*, las cuales se encuentran explicados en la Sección A.4.

2.3.1. Procesamiento de lenguaje natural

Es un campo de la inteligencia artificial y lingüística que estudia las interacciones entre los ordenadores y el lenguaje humano, conocido como lenguaje natural. El procesamiento del lenguaje natural se ocupa de la aplicación de técnicas computacionales para la comunicación entre personas y máquinas por medio del lenguaje natural.

2.3.2. Indexación

El proceso indexación denota la formación de un índice en el que los documentos se recogen y clasifican, generalmente utilizando palabras clave y se disponen para realizar búsquedas posteriormente.

2.3.3. Librería TXTAI

TXTAI[6] es una librería para Python que a través de herramientas de aprendizaje automático procesa las consultas realizadas en lenguaje natural, las transforma y permite la creación de índices sobre los que realizar consultas. Permite el uso de modelos de PLN basados en *word embeddings* y *Transformers*, los cuales serán utilizados en este proyecto y explicado su uso en el Capítulo 4. Es capaz de generar *embeddings* para modelos *transformers* preentrenados gracias a que utiliza el *framework* Sentence *Transformers*[7].

Además **TXTAI** permite la clasificación de textos, detección de emociones y traducción de estos. Esto es posible ya que utiliza los modelos *transformers*

[6]https://github.com/neuml/txtai
[7]https://github.com/UKPLab/sentence-transformers

preentrenados proporcionados por **Hugging Face**[8] gracias al uso de su API(Interfaz de Programación de Aplicaciones). Esta librería se trata de un proyecto de código abierto desarrollado por la empresa NeuML[9] disponible para uso bajo los términos de la licencia Apache License 2.0.

2.3.4. Justificación uso de TXTAI

Finalmente, esta librería se ha elegido por sus cualidades descritas anteriormente. Permite búsqueda semántica, debido a que procesa el lenguaje natural de las consultas y multidioma, ya que, aunque sólo procesa las consultas realizadas en inglés permite traducir textos al inglés. De esta forma puede solventarse este escollo lingüístico y trabajar con conjuntos de datos que no estén exclusivamente en inglés. Además permite la creación y actualización de índices de forma rápida y sencilla. Se trata de una tecnología de rápida configuración, muy fácil y sencilla de utilizar. Se ha elegido esta tecnología por encima de otras herramientas que también permiten crear índices y realizar búsquedas como **ElasticSearch** o **Solr** por las cualidades mencionadas y porque fue planteada por los directores de este proyecto.

2.4. Concepto de Ontología

En informática, una ontología es un mecanismo de representación en forma de esquema conceptual, cuya finalidad es la de constituir la especificación formal de los conceptos en un área de conocimiento (véase en *What is an Ontology?* de Gruber 1992 [16]). Un esquema conceptual proporciona una descripción lógica de los datos compartidos, lo que permite que los programas y las bases de datos interoperen.

Cada ontología presenta su vocabulario que describe varios aspectos del dominio, área del conocimiento, que está modelando. Junto a este vocabulario las ontologías también describen las relaciones entre sus componentes, de esta forma proporcionan una especificación explícita que refuerza su significado. Gracias al uso de ontologías se consigue una representación formal que mejora el intercambio de información entre uno o varios elementos, ya que define un vocabulario común para todos los elementos que al mismo tiempo es independiente de su naturaleza y comportamiento [17].

2.4.1. Componentes

En este apartado se indicarán brevemente los componentes que forman las ontologías [18].

[8]https://huggingface.co/models
[9]https://neuml.com/

- **Instancias**. Denominadas también como individuos u objetivos, representan elementos que pertenecen a un concepto dentro del dominio de acuerdo a la estructura de la ontología.

- **Conceptos**. Son los principales elementos que componen la ontología. Dentro de los conceptos se engloba además a las clases y entidades. Representan el tipo de elementos a modelar.

- **Roles**. Los roles comprenden las relaciones, propiedades y atributos. En el caso de las relaciones estas representan tipos de asociaciones entre conceptos del modelo, o entre conceptos. En el caso de los atributos y propiedades representan aspectos, propiedades o características de los conceptos.

- **Funciones**. Las funciones en una ontología son complejas estructuras cuya finalidad es la calcular e obtener información de otros elementos dentro de la ontología.

- **Axiomas**. Estos elementos son restricciones, reglas o lógica correspondiente a las definiciones de los conceptos que deben ser cumplidos en las relaciones entre los distintos elementos de la ontología. Las reglas y restricciones definen la estructura de la ontología ya que validan las nuevas entradas a la ontología.

2.4.2. Ventajas

Las ontologías son empleadas para especificar y estandarizar vocabulario para el intercambio de datos entre sistemas. Las ontologías proporcionan servicios para responder consultas, publicar bases de conocimiento reutilizables y ofrecer servicios para facilitar la interoperabilidad entre múltiples sistemas y bases de datos heterogéneos. Además, una cualidad clave de las ontologías consiste en que son capaces de especificar un modelo de datos que va más allá de la especificación del esquema de una base de datos. En las ontologías escritas en OWL su esquema juega un papel muy importante, ya que permite el enriquecimiento y la extensión de sus consultas [19].

2.4.3. Dificultades

Las herramientas que trabajan con ontología no suelen rechazar actualizaciones que provocan inconsistencia, simplemente muestran una señal de error [20]. También unas de las desventajas que presentan las ontologías es la dificultad para adaptar las bases de datos tradicionales a su esquema. Las ontologías permiten describir complejas expresiones o relaciones entre conceptos y las bases de datos relacionales tienden a

la simplificación, basándose en el almacenamiento de filas en tablas y necesitando de complejas operaciones para calcular las relaciones entre los datos. Otra dificultad añadida, es que el uso y la adaptación del esquema de una base de datos a una ontología es un proceso manual y costoso.

2.4.4. Selección del modelo de información IDS

Para este proyecto se ha elegido como ontología el modelo de información de IDSA (International Data Spaces Association)[10]. Esta es una asociación cuyo propósito es el intercambio de datos de forma segura y fiable.

El modelo de información IDS (International Data Spaces)[11] se basa en una ontología que cumple los estándares de RDFS/OWL, que define los conceptos fundamentales que permiten describir a los actores, sus interacciones, los recursos que intercambian y las restricciones en un espacio de datos. Nace de la necesidad de crear un estándar para el intercambio de datos empresariales entre socios de forma segura. Es decir, se trata de un modelo para el intercambio de información digital en el mundo empresarial (para más información consulte [21]).

Aunque este proyecto puede ser extensible a otras ontologías finalmente se ha elegido la ontología desarrollada por la IDSA. La razón de la elección de esta ontología ha sido por decisión de los directores de este proyecto.

[10]https://internationaldataspaces.org/
[11]https://w3id.org/idsa/core

Capítulo 3

Herramienta de población automática de ontologías con fuentes públicas externas

En este capítulo se detallarán los requisitos de la herramienta. A su vez se explicará cómo se va a utilizar, mediante el uso de un diagrama de casos de uso, su arquitectura, cómo se va a desplegar y los bocetos de la primera aproximación de la interfaz web.

3.1. Análisis

En esta sección se analizarán los requisitos que ha de cumplir la herramienta para correcto cumplimiento de su propósito, el poblado automático de una ontología, y se explicarán los posibles casos de uso para los usuarios.

3.1.1. Requisitos del proyecto

Requisitos funcionales

Los requisitos funcionales son aquellos que establecen los comportamientos del sistema y son independientes de la implementación del sistema [22]. En la Sección B.1 se incluyen los requisitos mostrados en la Tabla 3.1 con más detalle.

Código	Título
RF-1	Búsqueda de conjuntos de datos.
RF-2	Búsqueda por palabra clave.
RF-3	Filtrar los conjuntos de datos por publicador, país y categoría.
RF-4	Búsqueda por categoría, publicador o país.
RF-5	Descarga del contenido de las distribuciones.
RF-6	Gestionar apartado de conjuntos de datos recurrentes o favoritos .
RF-7	Los conjuntos de datos han de adaptarse al esquema de la ontología.
RF-8	Permite la heterogeneidad de los repositorios de datos abiertos.
RF-9	Establece relaciones entre los conjuntos de datos que tengan propiedades en común.
RF-10	Rápida incorporación de repositorios de datos.
RF-11	Búsqueda semántica.
RF-12	Guardar las relaciones en la base de datos.

Tabla 3.1: Requisitos funcionales

Requisitos no funcionales

Los requisitos no funcionales describen las facilidades que debe de proporcionar el sistema en cuanto a la implementación [22]. En la Sección B.2 se encuentran los requisitos mostrados en la Tabla 3.2 detallados.

Código	Título
RNF-1	Despliegue con Docker.
RNF-2	Uso de una librería de búsqueda semántica basada en embedded.
RNF-3	Ontologia IDS.
RNF-4	Uso de GitHub.

Tabla 3.2: Requisitos no funcionales

3.1.2. Casos de uso

En este apartado se presentarán los diferentes casos de uso de esta herramienta. Un caso de uso especifica el comportamiento o parte del sistema sin llegar a explicar cómo implementa dicho comportamiento. Los casos de uso describen una situación de uso del sistema interactuando con actores. Se utiliza la técnica de diagrama y descripción de casos de uso para capturar información de cómo un sistema trabaja o de cómo se desea que trabaje [23].

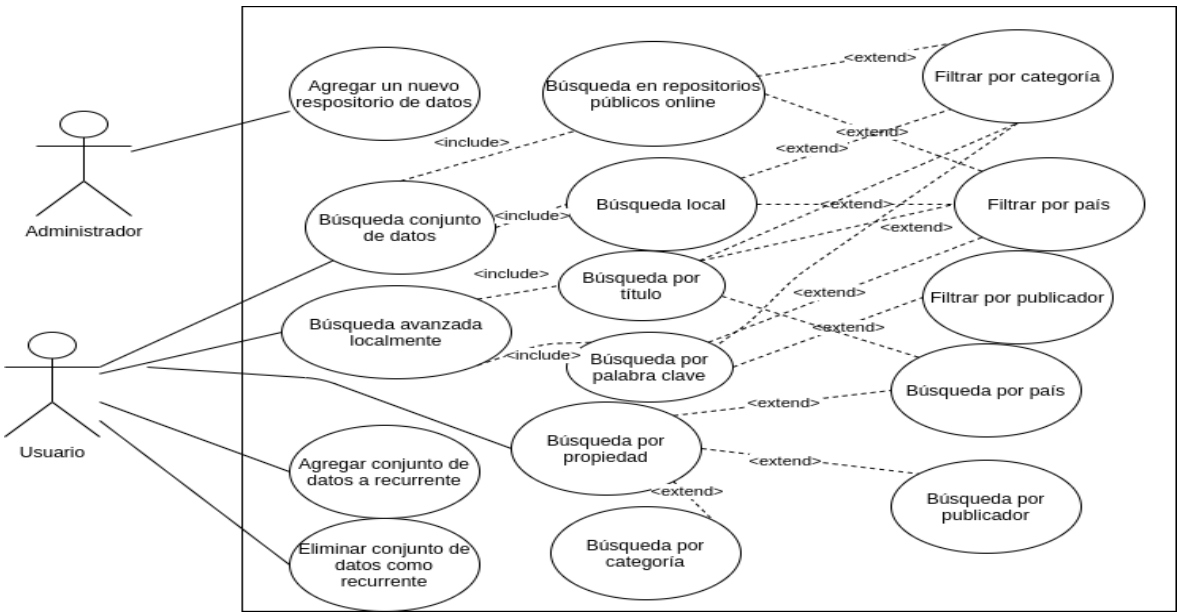

Figura 3.1: Diagrama de casos de uso

Podemos diferenciar dos tipos de actores. Un tipo de actor denominado administrador y un usuario final. El administrador puede incorporar nuevos repositorios de datos programando un *script* que descargue los conjuntos de datos de estos repositorios y utilice la API de la librerías para poblar la ontología y estandarizar estos conjuntos de datos.

El usuario puede realizar búsquedas de conjuntos datos. La búsqueda del usuario se realizará en la base de datos local y en los repositorios de datos *online*. Existe la posibilidad de que usuario quiera limitarse a buscar conjuntos de datos que pertenezcan a una categoría, a un país o a ambas.

El usuario puede realizar búsquedas en la base de datos local a través de un sistema de búsqueda avanzada. Estas búsquedas pueden realizarse consultando a la base de datos por palabras clave, devolviendo los conjuntos de datos que tengan una palabra clave semejante, o por el título del conjunto de datos, devolviendo conjuntos de datos con un título lo más parecido posible.

Proporcionados una categoría, un país o un publicador por el usuario, el sistema ha de devolver todos los conjuntos de datos que cumplan con una de estas características. El usuario puede agregar conjuntos de datos al conjunto de datos recurrentes para acceder a estos conjuntos de datos de forma más rápida. De la misma forma el usuario puede eliminar conjuntos de datos de los conjuntos de datos recurrentes.

3.2. Arquitectura

3.2.1. Diagrama alto nivel

Debido a que se perseguía desarrollar un prototipo funcional demostrador de la viabilidad, se ha optado por una arquitectura basada en el patrón modelo-vista-controlador y la implementación de una aplicación monolítica conectada a una base de datos. Como se detallará más adelante en el Capítulo 4 la aplicación se desarrollará en **Flask**[12]. Los detalles se implementación se explicarán más adelante, pero como ya se ha mencionado que es requisito el uso de la librería TXTAI se ha añadido al esquema. Para el procesamiento del lenguaje natural, **TXTAI** descarga los modelos que utiliza del repositorio de modelos **HuggingFace**. La aplicación se encuentra conectada a los repositorios de datos públicos de los que descarga los conjuntos de datos. Se conecta a la base de datos a través del uso de un protocolo binario denominado BOLT. La base de datos se trata de **Neo4j** que utiliza su *plugin* **Neosemantic** (explicado en la Subsección A.3.3). Los detalles sobre el desarrollo y cómo se ha llegado a esta arquitectura se encuentra en el Capítulo 4.

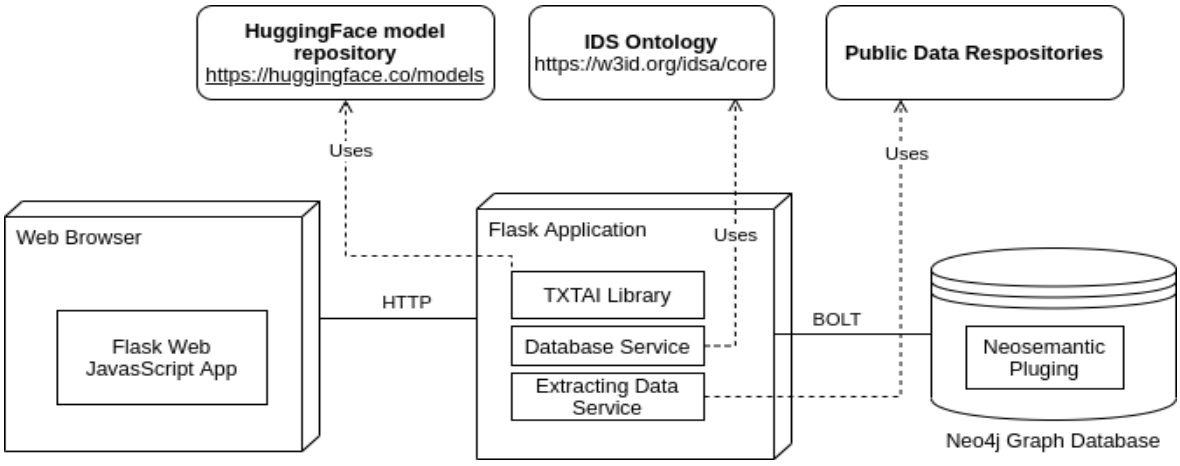

Figura 3.2: Diagrama alto nivel

Diagrama de despliegue

Como se ha mencionado anteriormente, uno de los requisitos no funcionales relacionados con el despliegue de la herramienta es que se utilice **Docker**. Se utilizarán dos contenedores, uno que albergue la aplicación **Flask** y un segundo que contenga la base de datos. El contenedor que contiene la aplicación Flask necesitará importar la imagen de Python 3.8 e instalar las librerías utilizadas. Una vez se ha desplegado quedará disponible en el puerto 5000. Con respecto al contenedor que alberga la base de datos, será necesario para su correcto despliegue la instalación del *plugin*

[12]https://flask.palletsprojects.com/en/2.0.x/

Neosemantic, la importación de la imagen de Neo4j y la declaración de volúmenes para no perder el contenido almacenado. Una vez desplegada la base de datos estará disponible a través del uso del protocolo binario BOLT en el puerto 7687 al cuál se conectará la aplicación Flask.

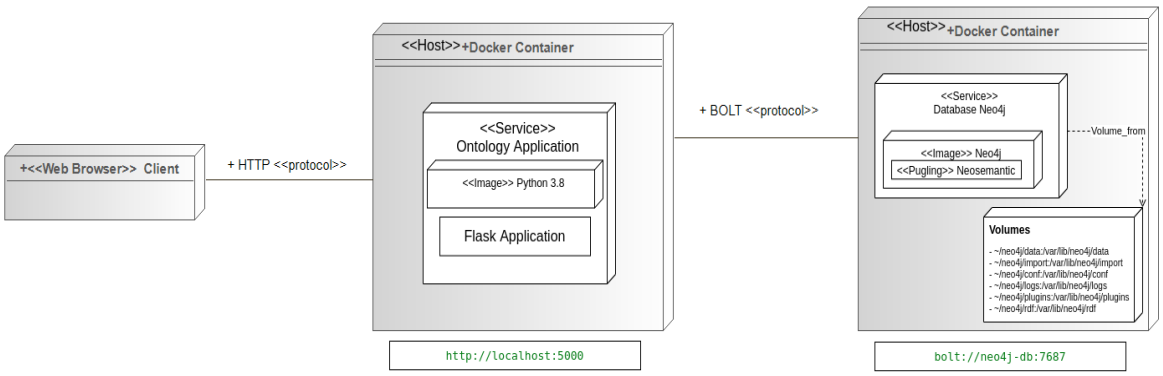

Figura 3.3: Diagrama de despliegue

3.2.2. Modelo entidad-relación

En esta sección se utilizará un modelo entidad-relación para mayor comprensión de los datos almacenados. El modelo entidad-relación es un modelo conceptual para facilitar la comprensión del dominio y los componentes que se tienen en cuenta, pero no hace referencia a la implementación de algoritmos ni al almacenamiento [24].

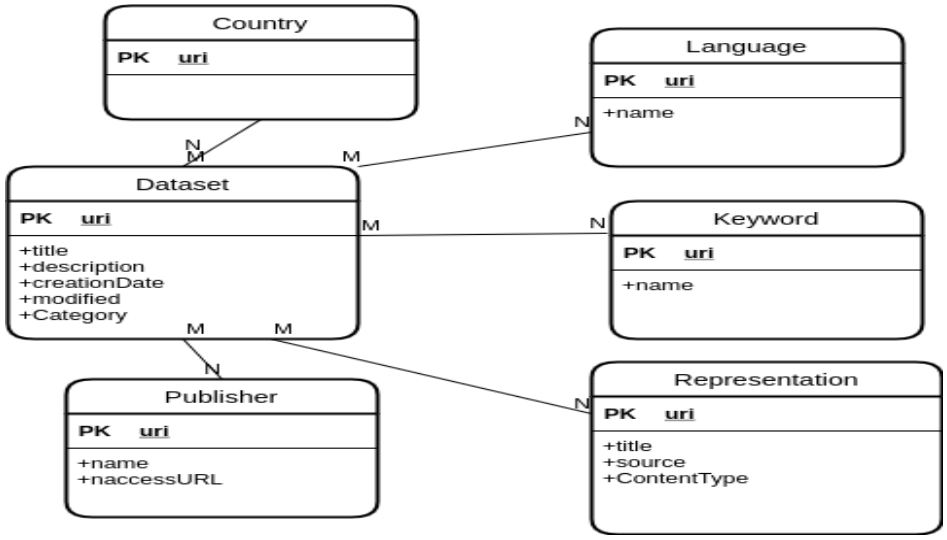

Figura 3.4: Diagrama entidad-relación

– **Dataset**. Representa a los conjuntos de datos y sus atributos almacenados. Los conjuntos de datos contarán con un título, una descripción, una fecha de creación, una fecha de modificación en caso de que su repositorio de datos original lo

permita y una categoría asociada. Cada conjunto de datos esta publicado por un publicador.

- **Publisher**. El publicador de un Dataset se encuentra representado en la entidad, que puede relacionarse con más de un Dataset.

- **Representation**. Utilizada para guardar el contenido de una distribución de un conjunto de datos. Cada distribución cuenta con un título, un enlace a su contenido descargable e información sobre el formato de este último. Un conjunto de datos puede tener más de una sola distribución.

- **Keyword**. Representa las palabras clave del conjunto de datos, es muy importante dado que permite establecer relaciones implícitas entre diferentes datasets.

- **Language**. Representan el idioma en el que está disponible el conjunto de datos.

- **Country**. Representa el país al que pertenece, generalmente es el mismo que el de su publicador.

Como ejemplo auxiliar se incluye un diagrama mostrando cómo se relacionan los componentes de la ontología tras haberla importando (disponible en la Figura B.1), aunque este proceso se encuentra explicado y detallado en el Capítulo 4.

3.3. Mapa de navegación

En esta sección se incluye el mapa de navegación con los bocetos de la interfaz de la herramienta. Como puede observarse es posible acceder a la visualización del contenido de un conjunto de datos a través de todas pantallas salvo de la página informativa. Esto es debido a que una vez se haya completado la búsqueda se mostrarán los conjuntos de datos resultados y clicando sobre estos resultados es posible acceder a la pantalla que muestra su información completa. Para visualizar mejor el mapa de navegación los bocetos de cada página junto a su explicación se encuentran especificados en la Sección B.4.

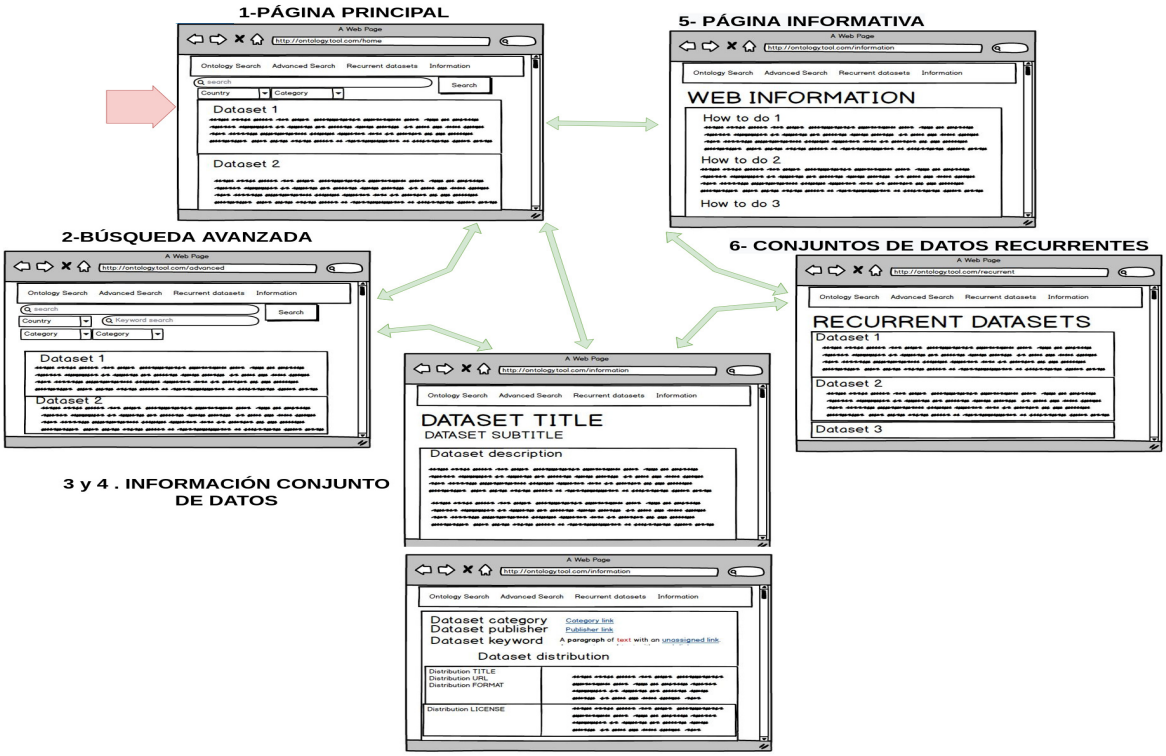

Figura 3.5: Mapa de navegación

Capítulo 4

Prototipo del sistema y Validación

Tras haber llevado a cabo el análisis del contexto y requisitos (ver en Capítulo 3), en este capítulo se explicará cómo se ha implementado la aplicación y qué tecnologías se han utilizado para cumplir los requisitos propuestos y alcanzar el objetivo de este proyecto.

4.1. Backend

En esta sección se detallará cómo se ha implementado la lógica de la aplicación. Se ha elegido **Flask** como tecnología para desarrollar esta herramienta. Flask es un *framework* para desarrollar aplicaciones web en Python. Seleccionado debido a su simplicidad y a que emplea el mismo lenguaje de programación que la librería de **Python TXTAI** mencionada en el Capítulo 2.

Además, como base de datos se ha utilizado Neo4j, mencionada también en el Capítulo 2. **Py2neo**[13] se trata de una librería para Python que permite establecer una conexión a la base de datos Neo4j y ejecutar sentencias en CYPHER, lenguaje de consultas en **Neo4j**. Los diagramas de clases se encuentran disponibles en la Sección C.3.

4.1.1. Extracción de conjuntos de datos

Para la extracción de los datos de las fuentes de datos de interés se han analizado diferentes alternativas:

1. La localización y extracción de los datos de forma automática mediante técnicas de recuperación de información basadas en *web crawling* (ver en Subsección E.1.1).

[13]https://py2neo.org/2021.1/

2. La localización de fuentes de datos por parte de los administradores y el uso de técnicas de *web scraping* para la extracción de datos (ver en Subsección E.1.2).

3. La localización de fuentes de datos de interés por parte de los administradores de sistemas y el uso de APIs de dichas fuentes para la extracción de forma semiautomática empleando un patrón adaptador.

Finalmente, tras la implementación de diversas pruebas se optó por la tercera opción, las razones por las que se tomó esta decisión se encuentra detalladas en la Subsección E.1.5. A su vez en Sección E.2 se encuentra el estudio de las tecnologías de *crawling* y *scraping* analizadas para este proyecto.

Existen portales de datos abiertos que exponen APIs públicas, como por ejemplo **datos.gob.es**[14], o portales cuya API puede ser accedida mediante el uso de las herramientas de desarrollador del navegador. Por lo tanto la forma de acceder a los conjuntos de datos se ha implementado mediante peticiones *get* utilizando la librería **requests**[15].

Con el objetivo de añadir repositorios de datos lo más rápido posible se ha creado una clase interfaz **AccessScript**, de la que heredarán el resto. De esta forma es posible cumplir los requisitos RF-8 y RF-10. Las clases que hereden de esta interfaz han de sobreescribir los métodos de su clase padre de tal forma que cumplan los siguientes pasos:

1. Crear una *url* a la que solicitar los conjuntos de datos. Dado el país del repositorio y su la categoría de los conjuntos de datos ha de crear una *url* a la que solicitar los datos necesarios.

2. Solicitar a la *url* creada los conjuntos de datos. Este paso ha de repetirse hasta que no haya más conjuntos de datos disponibles.

3. Analizar el resultado obtenido, crear una instancia de la clase Dataset de la ontología IDS para ser utilizada por la clase CreateRDF que se encargará de poblar la base de datos. Este paso deberá repetirse para cada uno de los conjuntos de datos obtenidos.

4. Crear una tupla con el identificador único del conjunto de datos, su título más su descripción en inglés y las palabras clave de este conjunto de datos.

5. Invocar la función encargada de actualizar los índices de búsqueda proporcionándole como parámetros la lista de tuplas generada en el paso anterior.

[14]https://datos.gob.es/es/apidata
[15]https://docs.python-requests.org/en/master/

Sus clases hijas los implementará en función de lo que se necesite para extraer la información de cada repositorio de datos. Lo que han de tener en común todas las clases que hereden de AccessScript para agilizar la incorporación de nuevos repositorios es la devolución de una lista de tuplas y la transformación del conjunto de datos obtenido como respuesta en una instancia de la clase Dataset. Las tuplas que han de devolver deben tener el siguiente formato:

(ID, TÍTULO + DESCRIPCIÓN, PALABRAS CLAVE)

Figura 4.1: Formato tupla

4.1.2. Población automática de la ontología

En este apartado se va a detallar cómo funciona en proceso de poblado de la ontología una vez se ha descargado un conjunto de datos de su fuente de datos pública y cómo se cumplen los requisitos RF-7, RF-9 y RF-12 (Tabla 3.1). En este apartado se va a utilizar el *plugin* Neosemantic de **Neo4j**. Una de las funcionalidades por las que se ha elegido este *plugin* para este proyecto es su capacidad para importar y exportar tripletas RDF en múltiples formatos, aunque en este proyecto simplemente se ha utilizado el formato RDF/XML. Además, y en este caso es fundamental para completar el objetivo del proyecto, permite la importación y exportación de ontologías.

Al importar una tripleta RDF, este *plugin* realiza un mapeo para que se adapte al formato de grafo en Neo4j. Los sujetos de las tripletas se mapean como nodos, las etiquetas de los nodos se obtienen a partir de rdf:type. Cada nodo ha de tener una URI única. Para ello al inicializar la base datos hay que establecer que todo recurso de la base de datos ha de tener una URI identificadora única. Esto se realiza mediante la ejecución del siguiente comando en CYPHER.

```
CREATE CONSTRAINT n10s_unique_uri ON (r:Resource)
ASSERT r.uri IS UNIQUE;
```

Los predicados de las tripletas son mapeados a propiedades de nodo si el objeto de la tripleta es un dato literal. En cambio los predicados de las tripletas son mapeados a relaciones si el objeto de la tripleta es un recurso, que en este caso acabaría siendo otro nodo[25].

Es necesario establecer la configuración antes de comenzar a utilizar con la base de datos. Para este proyecto se ha utilizado la siguiente configuración:

- **KeepLangTag:True** Este atributo establece que al importar las tripletas mantenga las etiquetas que indican el idioma.

– **handleMultival: ARRAY** Este atributo indica que si un atributo tiene varios valores ha de tratarse como una lista, de esta forma si un atributo tiene varios valores, estos no se sobreescriben y se conservan ambos.

Esta configuración se aplica ejecutando los siguientes comandos de CYPHER al iniciar la herramienta:

```
CALL n10s.graphconfig.init();
CALL n10s.graphconfig.set({keepLangTag: true,
handleMultival: "ARRAY"});
```

Una vez se ha descargado el contenido de un conjunto de datos y se ha guardado como instancia de la clase **Dataset** se utilizan los métodos de la clase **CreateRDF**. Esta clase utiliza la librería **rdflib**[16], se trata de una librería en Python que permite trabajar con RDF. La API de **rdflib** permite la lectura y la creación de fichero RDF. En este proyecto se ha utilizado para crear un grafo RDF por cada conjunto de datos.

Para poblar la ontología y estandarizar los conjuntos de datos acorde al vocabulario de esta ontología es necesario construir el grafo RDF con tripletas que utilicen este vocabulario. Para ello **rdflib** utiliza la clase **Namespace**, dado una URI nos permite crear un grafo utilizando el vocabulario de dicha URI. Usando la función **bind** es posible asociar un prefijo a cada URI simplificando así la tarea de creación del grafo. En este proyecto es necesario el uso de los espacios de nombres OWL, RDF, CORE[17] y DCTERMS[18]. CORE se corresponde con el vocabulario del modelo de información IDS y DCTERMS engloba los términos de los metadatos mantenidos por la asociación Dublin Core Metadata Initiative. Los anteriores espacios de nombres mencionados definen, qué vocabulario se utiliza de cada uno de ellos y cómo queda mapeada la información al importar el grafo se encuentran explicados en la Sección A.6.

Una vez se ha creado el grafo RDF con el uso de **rdflib**, se exporta como *string* utilizando la función *serialize*, que permite especificar el formato siendo en este caso **pretty-xml**. Una vez exportado el grafo mediante el uso de Py2neo se ejecuta la sentencia **CALL n10s.rdf.import.inline**, que permite importar el grafo en la base de datos Neo4j, adaptarlo y guardar su contenido (ver ejemplo en la Figura 4.2).

[16]https://rdflib.readthedocs.io/en/stable/index.html
[17]https://w3id.org/idsa/core/
[18]http://purl.org/dc/terms/

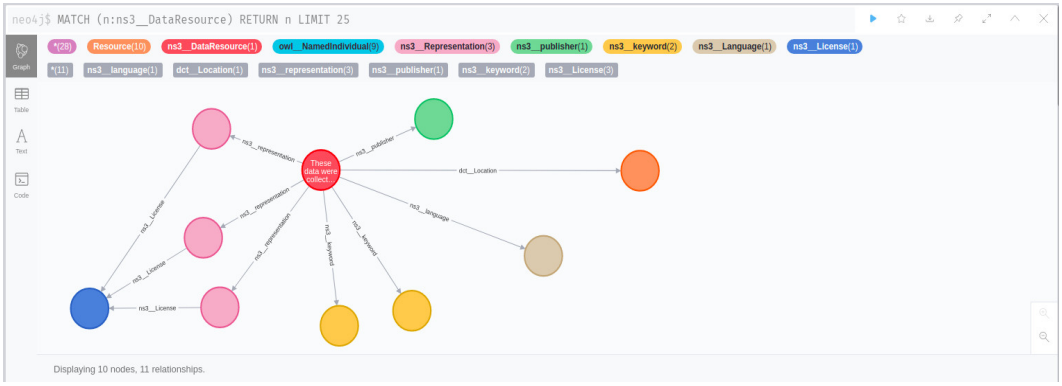

Figura 4.2: Ejemplo conjunto de datos en Neo4j

4.1.3. Índices de búsqueda

Una vez se ha completado el proceso de extracción de los conjuntos de datos y se ha poblado la ontología hay que construir un índice de búsqueda sobre el que realizar las consultas.

El índice se genera gracias al uso de la librería TXTAI (detallada en la Subsección 2.3.3). Para generar estos índices se utilizan tuplas, en este proyecto se utilizan dos índices de búsqueda, un índice para buscar conjuntos de datos por su contexto y un segundo para buscarlos por palabras clave. Las tuplas que se utilizan siguen la misma estructura: (URI de conjuntos, contenido para indexar).

Al iniciar la aplicación se crean dos instancias de la clase *Embeddings*, esta clase nos permite especificar el método de transformación de palabras en lenguaje natural a vectores para ser procesadas y el modelo preentranado que se va a utilizar. Es decir, si por ejemplo se quiere utilizar un modelo preentrenado *transformers*, hay que proporcionarle este modelo e indicar en el atributo método que se trata de un modelo que utiliza el método *transformers*.

Con el fin de poder buscar conjuntos de datos por palabras clave se ha utilizado una instancia con el método *word embedding* que utiliza el modelo GloVE (ver en la Sección A.4). De esta forma podemos generar un índice con las palabras clave y realizar consultas que nos devuelvan los conjuntos de datos con las palabras clave más próximas y que más relacionadas estén con la consulta. El modelo GloVE hay que descargarlo y se encuentra en la carpeta de contenido estático del proyecto.

Existe una segunda instancia con método *transformers* y el modelo **sentence-transformers/bert-base-nli-mean-tokens**[19] para buscar conjuntos de datos en función del contexto. El modelo *transformers* utilizado para este índice se descarga del repositorio de modelos *transformers* **Hugging Face**. Esta librería

[19]https://huggingface.co/sentence-transformers/bert-base-nli-mean-tokens

utiliza modelos preentrenados de este repositorio y permite su uso para traducción, clasificación o similitud. Gracias al uso de un índice generado por el método *transformers* es posible obtener conjuntos de datos que se asemejen a la consulta realizada en lenguaje natural.

Cada vez que se guardan nuevos conjuntos de datos en la base de datos se actualizan o se crean los índices de búsqueda. Si los índices no existían previamente se crearán y si ya existían estos índices se cargarán y serán actualizados. Al realizar búsqueda sobre índices y contar con herramientas de procesamiento del lenguaje natural que ayudan a enriquecer la consulta en la base de datos local se cumple el requisito RF-11.

4.1.4. Búsqueda local de conjuntos de datos

En esta sección se explica cómo se llevan a cabo las consultas en la base de datos local, cumpliendo los requisitos RF-1, RF-2, RF-3 y RF-4 (ver en la Tabla 3.1). Se producen consultas a la base de datos en dos situaciones, desde la página principal de la aplicación y desde la página de búsqueda avanzada. Las búsquedas en los índices se llevan a cabo utilizando la librería TXTAI y las consultas directas a la base de datos a través de la librería Py2neo.

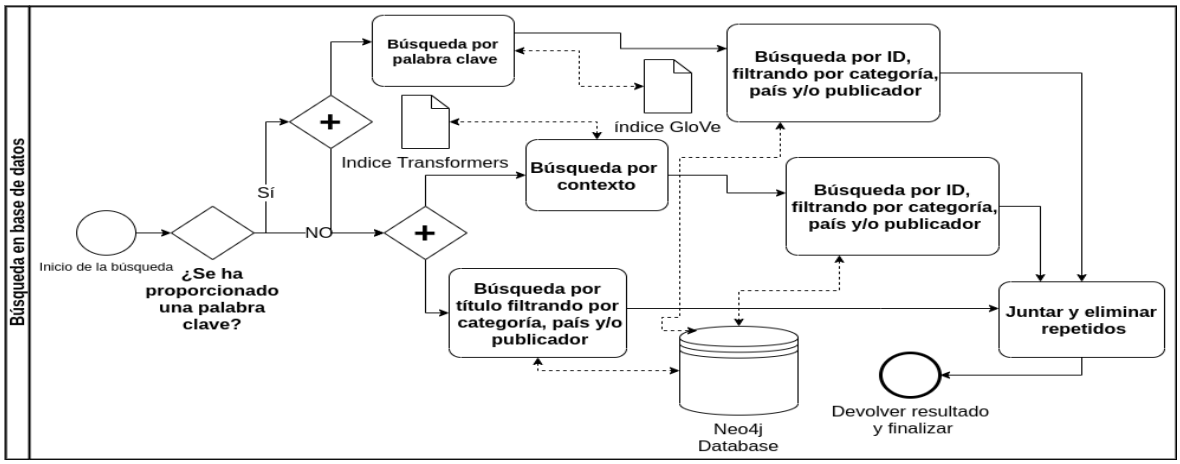

Figura 4.3: Búsqueda en base de datos

Cuando se realiza una consulta, primero se busca por título y a continuación se consulta el índice para obtener los conjuntos de datos en función del contexto, devolviendo ambos resultados. Cuando se búsqueda directamente en la base de datos por título se tienen en cuenta todas las propiedades que ha de cumplir el conjunto de datos para obtener el resultado. En otras palabras, si se aporta el país de origen, el publicador y/o la categoría, estas propiedades se añaden a la búsqueda y simplemente se devuelven los conjuntos de datos que las cumplan todas. En cambio cuando se consulta el índice, inicialmente no se tienen estas propiedades.

Al consultar el índice se obtiene una lista de URIs de conjuntos de datos. Posteriormente se buscan en la base de datos los conjuntos de datos que tengan estas URIs y que cumplan las propiedades seleccionadas. Una vez se han obtenido los resultados de ambas búsquedas se comprueba que no haya ninguno repetido y se devuelve el resultado. Si se ha proporcionado una palabra clave se paraleliza el proceso de búsqueda. Se lanzan dos hilos de tal forma que uno se encarga de la búsqueda en el índice de palabras clave y el segundo realiza la búsqueda por título y en el índice de contexto explicado anteriormente.

4.1.5. Conjuntos de datos recurrentes

Gracias al uso de una base de datos NOSQL es posible añadir propiedades a las los datos almacenados en Neo4j. Con tal de cumplir el requisito RF-6 cuando el usuario interactúa a través del *Frontend* de la herramienta se añade la propiedad favorito al conjunto de datos.

4.2. Frontend

En este apartado se explicará como se ha desarrollado la interfaz gráfica de esta herramienta. Al igual que para el *backend* de la herramienta se ha utilizado **Flask** para el Frontend. **Flask** es un *framework* de Python para desarrollo de aplicaciones web [26], que generalmente sigue el patrón Modelo Vista Controlador, el cual será denominado como MVC a partir de ahora. MVC es un patrón arquitectural para diseño de software [27]. En MVC se separan los datos, la lógica de negocios y la gestión de la interfaz.

Las interfaces de esta herramienta se han desarrollado utilizando: **Bootstrap**, **JQuery**, **Google Fonts** y **Fontawesome**. Con el fin de agilizar el proceso de desarrollo de estructura de la interfaz web se ha utilizado **Bootstrap**. **JQuery** se ha utilizado para ejecución de código asíncrono. Finalmente **Google Fonts**, librería de fuentes tipográficas y **Fontawesome**, paquete de iconos, se han utilizado para embellecer la interfaz. Todas estas tecnologías se encuentran explicadas en la Sección A.5.

En la Sección C.2 se han adjuntado imágenes de la última versión de las interfaces desarrolladas con las tecnologías descritas anteriormente.

Capítulo 5

Conclusiones y Líneas de Trabajo Futuro

En este capítulo se exponen las conclusiones a las que se ha llegado tras haber completado con éxito este proyecto y los resultados obtenidos. Además de las conclusiones también se expone la metodología de trabajo, la planificación del proyecto y los posibles trabajos futuros para la ampliación este este proyecto.

5.1. Resultados

En este apartado se mostrarán los resultados del uso de la aplicación. Para la validación del proceso de poblado automático de la ontología en este proyecto se han utilizado dos repositorios de datos públicos *online*. Estos repositorios son European Data Portal, principal portal de datos públicos en Europa y Data.gov.au, portal público del gobierno australiano. Estos repositorios se explican con detalle en la Sección D.1.

5.1.1. Búsqueda en repositorios online

Durante este apartado se va a mostrar un ejemplo de población de un conjunto de datos. Antes de poblar la base de datos, puede verificarse que se ha importado correctamente la ontología. Los nombres de sus componentes han quedado guardados en Neo4j como etiquetas de nodo (ver en la Figura 5.1) que se asignarán a los nodos de los juntos de datos una vez se haya poblado.

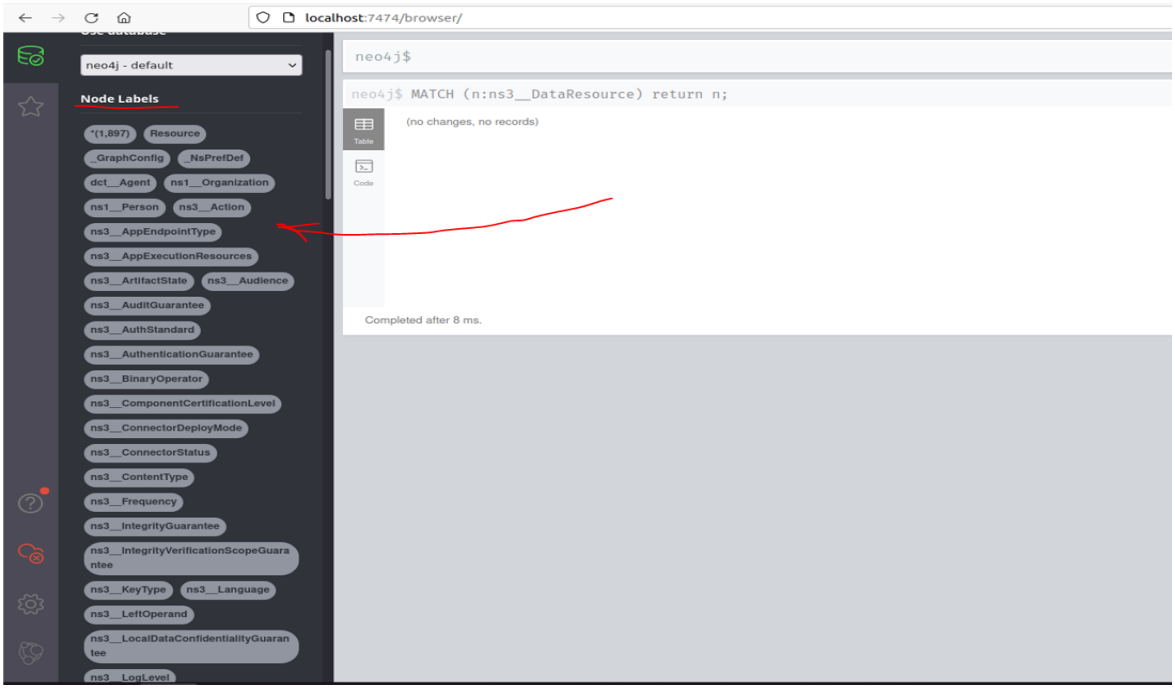

Figura 5.1: Base de datos sin poblar

En primer lugar, se poblarán todos los conjuntos de datos relacionados con coches eléctricos en Australia. Completar este proceso requiere utilizar el buscador disponible en la página principal y seleccionar Australia como país (ver en la Figura 5.2).

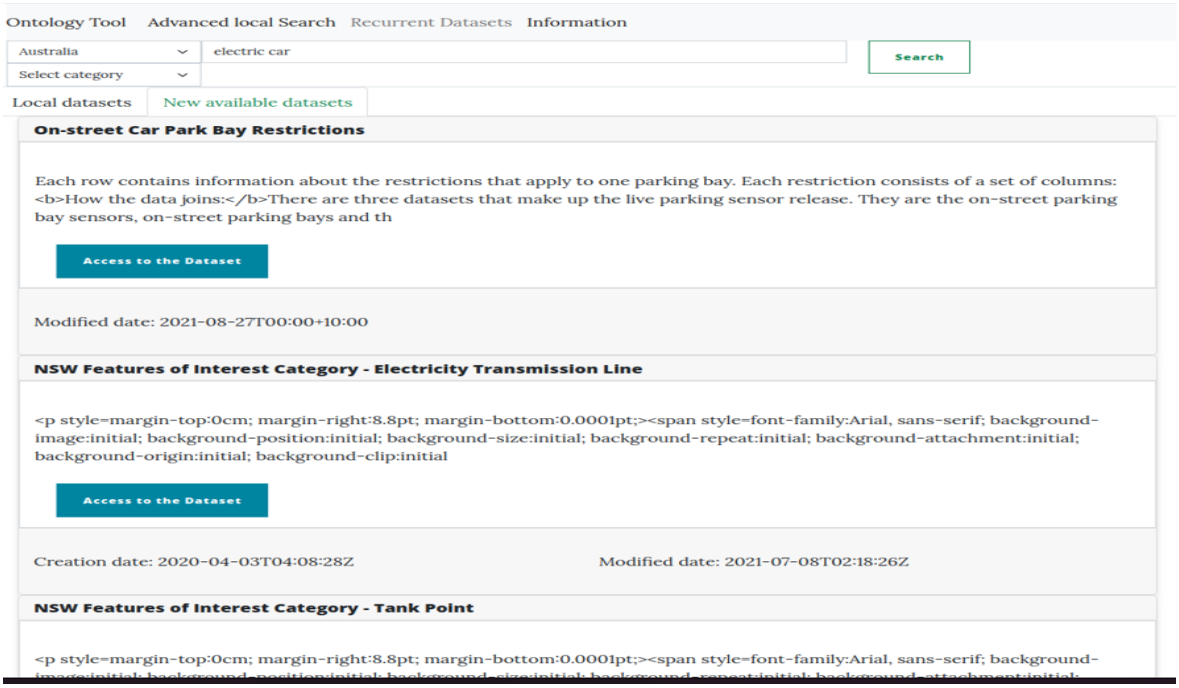

Figura 5.2: Búsqueda repositorio *online electric car*

Tras haber completado esta búsqueda la ontología se ha poblado con éxito y ahora puede comprobarse observando el contenido de la base de datos (ver en la Figura 5.3).

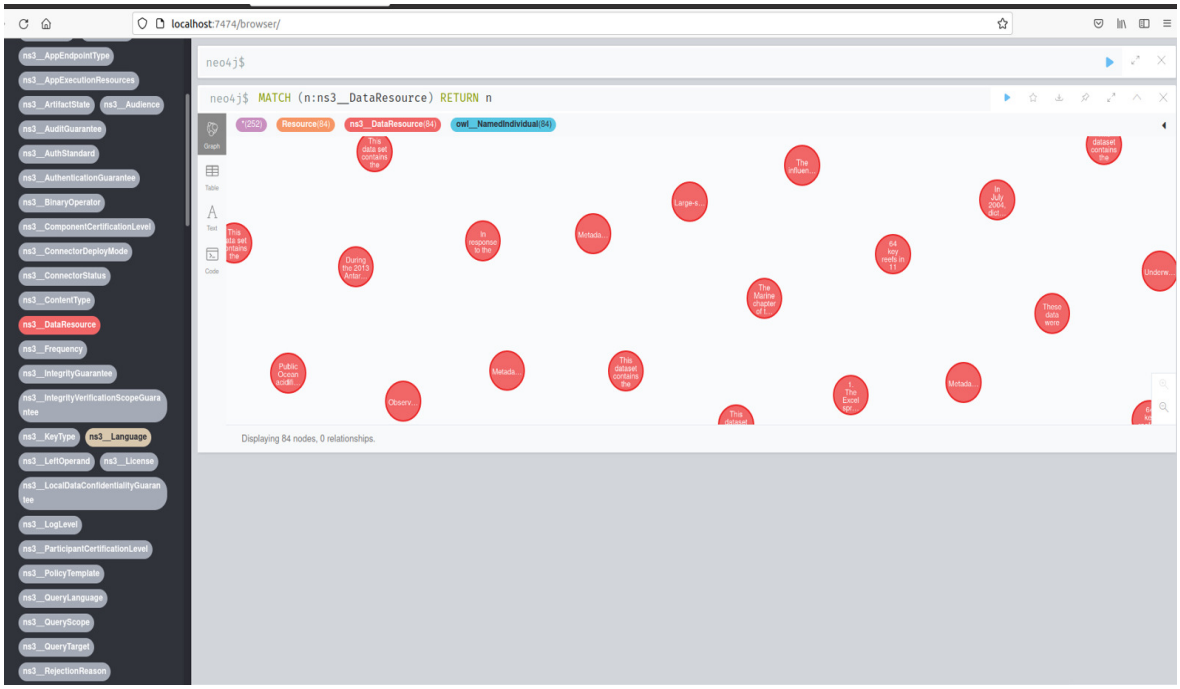

Figura 5.3: Base de datos poblada

Una vez poblada la base de datos y actualizados los índices de búsqueda, se buscará el conjunto de datos extraído *On-street Car Bay Restrictions*. Como puede comprobarse es un conjunto de datos disponible en la base de datos y que se ha guardado durante el proceso anterior (ver en la Figura 5.4).

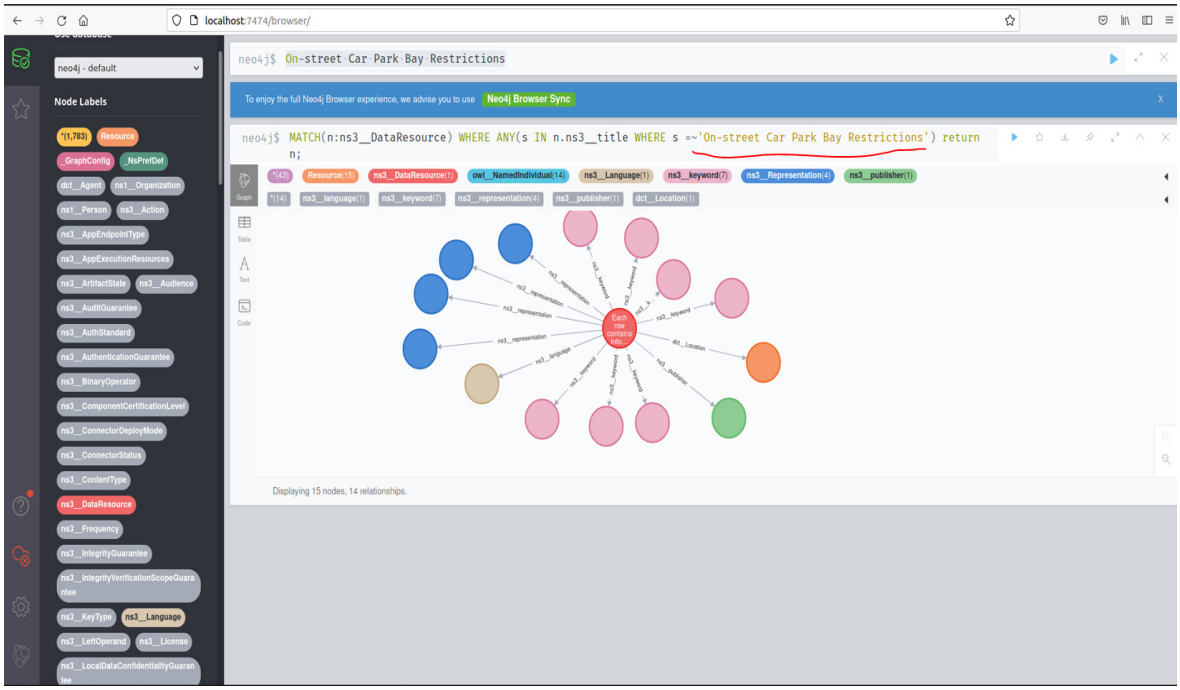

Figura 5.4: Conjunto de datos en base de datos

En la página de búsqueda avanzada se llevará a cabo la búsqueda de este conjunto

de datos mediante el uso del procesamiento del lenguaje natural. Obteniendo finalmente el resultado deseado (ver en la Figura 5.5).

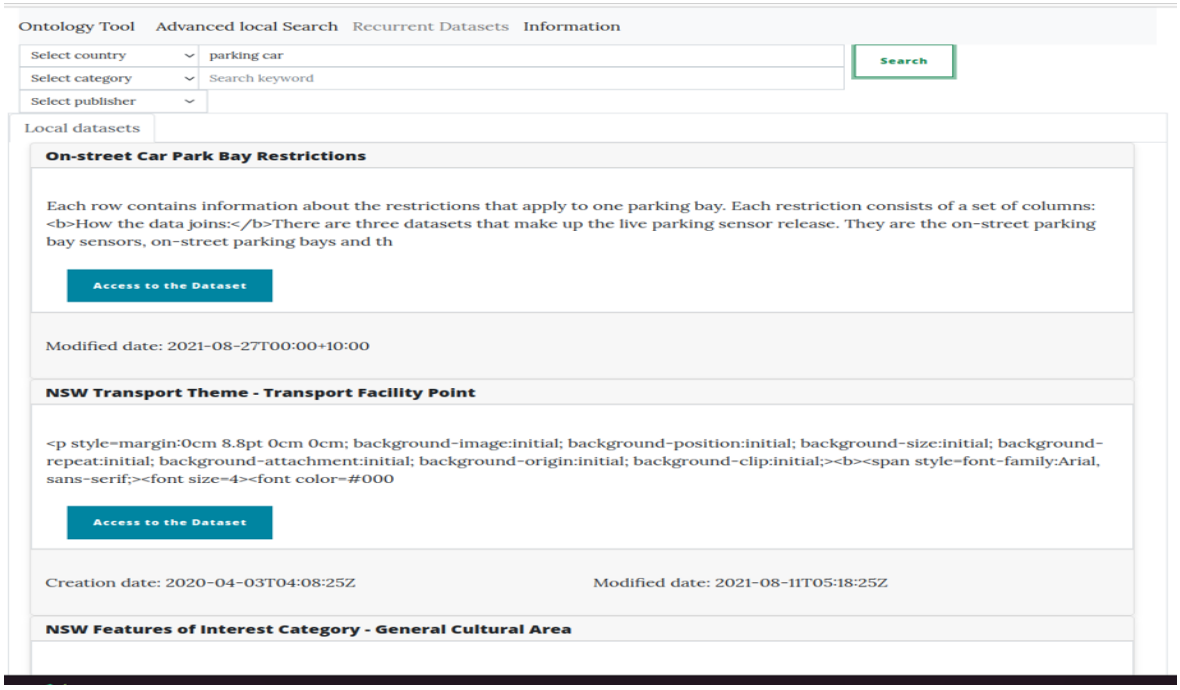

Figura 5.5: Búsqueda avanzada *parking*

A los conjuntos de datos extraídos anteriormente se van a sumar nuevos conjuntos de datos importados de European Data Portal (ver en la Sección D.2). Se comienza buscando en la página principal de la aplicación todo lo relacionado con coches eléctricos en España. Tras un tiempo de espera, en el que la herramienta está descargando los conjuntos de datos, transformándolos e importándolos en Neo4j, aparece el resultado de la búsqueda (ver en la Sección D.5).

Nos centraremos en el conjunto de datos "Puntos Carga Vehículos Eléctricos Illes Baleares". Si se hace clic sobre este podemos acceder a su contenido, el cual se encuentra en la Subsección C.2.7 de esta memoria. Una vez se encuentra poblado en la base de datos y estandarizado con el vocabulario de la ontología es posible buscarlo desde la página de búsqueda avanzada. Mediante el uso del procesamiento de lenguaje natural es capaz de procesar las consultas y devolver el conjunto de datos que se buscaba.

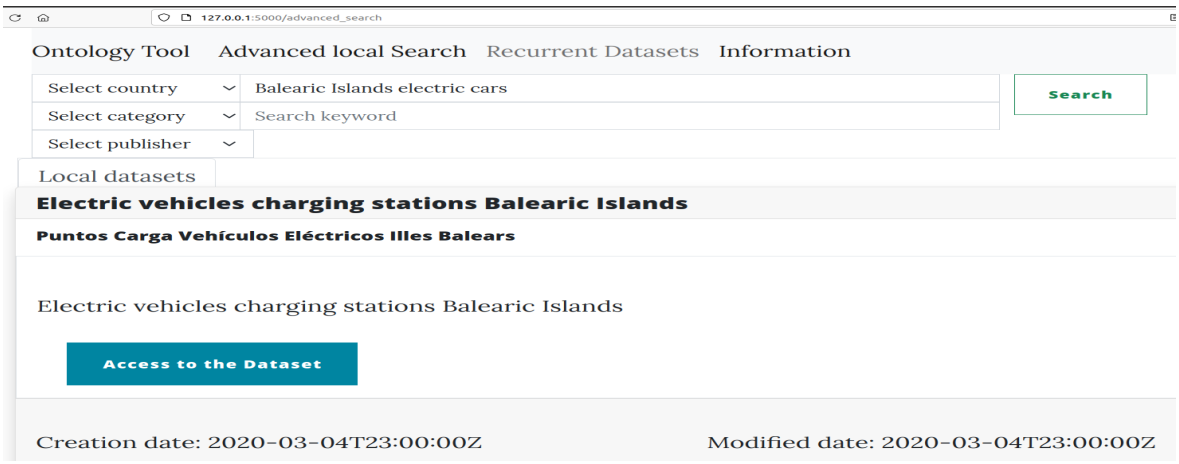

Figura 5.6: Búsqueda conjunto de datos por título interpretado

5.1.2. Búsqueda por palabra clave

A continuación, se va mostrar cómo es posible buscar conjuntos de datos (explicados en la Sección D.3) localmente utilizando una palabra clave. Las palabras claves que contiene cada conjunto de datos están disponibles en la Sección D.4. Como puede comprobarse se obtiene el resultado deseado (ver en la Sección D.6).

5.2. Dificultades y problemas encontrados

Durante el desarrollo de este proyecto se han encontrado varios problemas, producidos por fallos a la hora de entender conceptos.

1. El primer problema fue un planteamiento erróneo de la población de la ontología. En lugar de instanciar nodos con las etiquetas propias al vocabulario de la ontología se querían instanciar nodos que estuvieran relacionados con nodos con las etiquetas propias de la ontología. En otras palabras, en lugar de tener un nodo con la etiqueta DataResource, tener un nodo sin etiqueta con toda la información relacionado con un nodo vacío con la etiqueta DataResource.

2. El segundo problema fue la elección de los repositorios iniciales para validar el proyecto. Inicialmente se añadieron al proyecto los repositorios Open Data Aragón[20], datos.gob.es[21] y European Data Portal[22]. El problema es que al utilizar estos tres repositorios se estaban repitiendo conjuntos de datos. Ya que muchos de los conjuntos de datos que se publican en Open Data Aragón, se publican en datos.gob.es y estos a su vez en European Data Portal.

[20]https://opendata.aragon.es/
[21]https://datos.gob.es/
[22]https://data.europa.eu/

3. Neo4j no admite Multithread [28] por lo tanto se ha tenido que usar un semáforo antes de cada acceso para escribir en la base de datos. En un principio también se utilizaron semáforos para escribir en los ficheros RDF que posteriormente se importarían en Neo4j, porque se utilizaba la función **n10s.rdf.import.fetch** para importar ficheros. Estos últimos semáforos dejaron de utilizarse cuando pasó a utilizarse la función **n10s.rdf.import.inline**.

4. Debido a no haber cursado asignaturas en las que se tratara el tema del procesamiento de lenguaje natural y lo sencilla que es de utilizar la librería de TXTAI, se planteó un método erróneo de búsqueda por palabra clave. Se estaba utilizando un modelo Transformers para encontrar palabras de significado similar, en lugar de un modelo basado en *word embedding*. Tras profundizar en este tema lo suficiente, esto último se cambió.

5. Tanto data.gov.au como European Data Portal cambiaron su API durante el desarrollo de este proyecto, dando lugar a muchos fallos incomprendidos hasta que se descubrió que las lecturas de los JSON de repuesta a las peticiones eran erróneas.

5.3. Metodología de desarrollo

La metodología de desarrollo seguida para este proyecto ha sido una metodología ágil. El desarrollo del proyecto ha sido iterativo, de rápido prototipado e incremental. Se ha dividido en el cumplimiento progresivo de los requisitos. Cada semana se tenía una reunión con los coordinadores del proyecto, se evaluaba el trabajo realizado y se planteaba el trabajo para la próxima semana. Durante estas reuniones se acordaba qué requisitos se habían cumplido y cuáles debían de ser los siguientes a cumplir en el próximo prototipo que se mostraría en la siguiente reunión.

Antes de añadir un requisito era necesario buscar una tecnología que fuera capaz de cumplirlo, por lo tanto antes del desarrollo había una fase previa de búsqueda y análisis de tecnologías. Antes de cumplir cada requisito ha debido realizarse un proceso de búsqueda, análisis y finalmente implementación. Las reuniones se han realizado de forma presencial, en el edificio del **Instituto Tecnológico de Aragón** ubicado en el Campus Río Ebro, como *online* a través de **Google Meet**[23].

[23]https://meet.google.com/

5.4. Conclusiones

Con la realización del trabajo se ha aprendido a desarrollar un proyecto en un instituto de investigación y a trabajar fuera de la universidad. La necesidad del uso de ontologías para estandarizar y relacionar un conocimiento con estructura heterogénea. Aprovechándose así al máximo este conocimiento, ya que gracias al uso de las ontologías pueden establecerse nuevas relaciones entre datos inconexos. También se ha iniciado en el campo del procesamiento del lenguaje natural, área no vista anteriormente en la universidad, y se ha aprendido sus nociones básicas.

En cuanto a la dificultad de este trabajo, el mayor escollo ha sido la falta de conocimiento previo, ya que no se había visto antes nada sobre ontologías o procesamiento del lenguaje natural. Esto ha significado el estudio de estos campos y el posterior desarrollo del estado del arte, disponible en Capítulo 2. En cuanto a las herramientas manejadas, este trabajo ha servido para ampliar los conocimientos sobre el uso de Python y sus bibliotecas.

5.5. Planificación del proyecto

Este proyecto junto a su documentación ha llevado un total de 334 horas. Se ha recogido en una tabla el trabajo realizado cada día que se ha invertido en este proyecto. Debido al tamaño de la tabla se ha incluido en la Sección D.7 del Anexo.

5.6. Líneas de Trabajo Futuro

A continuación se detallan las posibles líneas de trabajo futuro para mejorar y expandir este proyecto:

- Aplicación del número de repositorios *online* a los que se accede en este proyecto. Actualmente hay dos repositorios integrados en este proyecto, pero gracias a su diseño es posible integrar muchos más. Como también sería posible añadir más idiomas, ya que actualmente sólo se guardan conjuntos de datos en español y en inglés.

- Paralelización de tareas y el paso a una arquitectura de microservicios. Actualmente toda la lógica de negocio se encuentra en una aplicación monolítica. Una mejora interesante que se podría hacer es la división de las tareas en servicios desplegados en diferentes máquinas.

- Profundizar más en los conceptos que ofrece la ontología y añadir más relaciones a

las actualmente utilizadas, como también el uso de más metadatos proporcionados por parte de los repositorios públicos.

– Mejora de su interfaz y el uso de un *framework* de diseño web. Actualmente se esta utilizando Flask tanto para la lógica de la aplicación como para la interfaz. Sería interesante el uso de Angular[24] o React[25] para mejorar la interfaz y separar la lógica de la aplicación de las interfaces.

[24]https://angular.io/
[25]https://es.reactjs.org/

References

[1] W3C. *Semantic Web*. 2015. URL: https://www.w3.org/standards/semanticweb/.

[2] Carlos Tellería Orriols Raquel Trillo Lado Ramón Hermoso Traba. *Tecnología semántica en la Web de los datos - Grado en Ingeniería en Informática e Ingeniería de Tecnologías y Servicios de Telecomunicación*. 2019. URL: https://moodle.unizar.es/add/pluginfile.php/2307340/mod_resource/content/2/2019_2020_04_LasTecnologiasDeWebDeDatosRDF_RDFSySPARQL.pdf.

[3] Digital Guide IONOS by 1 y 1. *URI: ¿qué es el identificador de recursos uniforme?* 2020. URL: https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/uri-identificador-de-recursos-uniformes/ (visitado 15-06-2021).

[4] Gobierno de Perú. *Qué son los Metadatos*. URL: https://www.geoidep.gob.pe/conoce-las-ides/metadatos/que-son-los-metadatos.

[5] W3C. *OWL Web Ontology Language*. 2004. URL: https://www.w3.org/TR/2004/REC-owl-features-20040210/#s1 (visitado 17-06-2021).

[6] Javier Lacasta Miguel. *Bases de datos. Recuperación de Información Modelo Relacional. Universidad de Zaragoza*. 2019. URL: https://moodle.unizar.es/add/pluginfile.php/1849642/mod_resource/content/2/03_modeloRelacional.pdf.

[7] IBM Cloud Education. *Teorema de CAP*. 2019. URL: https://www.ibm.com/es-es/cloud/learn/cap-theorem (visitado 20-05-2021).

[8] Sergio Ilarri Artigas. *Bases de datos 2. Sistemas Gestores de Bases de Datos: NoSQL. Universidad de Zaragoza*. 2020. URL: https://moodle.unizar.es/add/pluginfile.php/2518642/mod_resource/content/2/3-3-SGBGD-noSQL.pdf.

[9] Rubenfa. *NoSQL: clasificación de las bases de datos según el teorema CAP*. 2014. URL: https://www.genbeta.com/desarrollo/nosql-clasificacion-de-las-bases-de-datos-segun-el-teorema-cap.

[10] Grapheverywhere. *Bases de Datos NoSQL — Qué son, marcas, tipos y ventajas*. URL: https://www.grapheverywhere.com/bases-de-datos-nosql-marcas-tipos-ventajas/ (visitado 24-06-2021).

[11] Oracle España. *¿Qué es una base de datos orientada a grafos?* 2021. URL: https://www.oracle.com/es/big-data/what-is-graph-database/ (visitado 20-05-2021).

[12] Neo4j Inc. *Why Graph Databases?* 2021. URL: https://neo4j.com/why-graph-databases/ (visitado 28-07-2021).

[13] Amy E. Hodler Mark Needham. *Graph Algorithms: Practical Examples in Apache Spark and Neo4j*. 2019. URL: https://neo4j.com/lp/book-graph-algorithms/?utm_program=emea-prospecting&utm_source=google&utm_medium=cpc&utm_campaign=emea-search-sandbox&utm_adgroup=neo4j-sandbox&gclid=Cj0KCQjwytOEBhD5ARIsANnRjVh_wd3XiAqadZ5F5565-svnJZyCY9zwVeo-KIwMzYFLf40tMpbX84IaAtnDEALw_wcB (visitado 23-06-2021).

[14] Jim Webber Ian Robinson. *Graph Databases*. 2015. URL: https://neo4j.com/graph-databases-book/?utm_program=emea-prospecting&utm_source=google&utm_medium=cpc&utm_campaign=emea-search-sandbox&utm_adgroup=neo4j-sandbox&gclid=Cj0KCQjwytOEBhD5ARIsANnRjVjAC1ri7V2byoMG2Y6WnayTCtl6Ww57LwKhkdcmD1L_rCyyBUz-ENUaAnXaEALw_wcB (visitado 23-06-2021).

[15] Neo4j Jesús Barrasa Field Engineer. *RDF Triple Stores vs. Labeled Property Graphs: What's the Difference?* 2016. URL: https://neo4j.com/blog/rdf-triple-store-vs-labeled-property-graph-difference/ (visitado 23-06-2021).

[16] Thomas Gruber. *What is an Ontology?* 1992. URL: https://web.archive.org/web/20100716004426/http://www-ksl.stanford.edu/kst/what-is-an-ontology.html (visitado 20-06-2021).

[17] Thomas R. Gruber. *A Translation Approach to Portable Ontology Specifications*. 1993. URL: https://web.archive.org/web/20110715104718/http://tomgruber.org/writing/ontolingua-kaj-1993.pdf.

[18] Mireya Tovar-Vidal Cecilia Reyes-Pna. *Ontology: Components and Evaluation, a Review*. 2019. URL: https://www.rcs.cic.ipn.mx/2019_148_3/Ontology_%20Components%20and%20Evaluation_%20a%20Review.pdf (visitado 01-08-2021).

[19] Tom Gruber. *Encyclopedia of Database Systems*. 2007. URL: https://tomgruber.org/writing/ontology-definition-2007.htm (visitado 01-08-2021).

[20] Ian Horrocks. *What Are Ontologies Good For?* 2013. URL: https://www.cs.ox.ac.uk/people/ian.horrocks/Publications/download/2013/Horr13a.pdf (visitado 01-08-2021).

[21] *The International Data Spaces Information Model An Ontology for Sovereign Exchange of Digital Content*. 2020. URL: http://dbis.rwth-aachen.de/cms/publications/iswc-ids-infomodel (visitado 01-08-2021).

[22] Miguel Ángel Latre Javier Nogueras Iso José Merseguer. *Ingeniería del software. Fase de Requisitos. Universidad de Zaragoza.* 2019. URL: https://moodle.unizar.es/add/pluginfile.php/2248531/mod_resource/content/10/2a_faseRequisitos.pdf.

[23] Miguel Ángel Latre Javier Nogueras Iso José Merseguer. *Ingeniería del software. Diagrama de Casos de Uso. Universidad de Zaragoza.* 2019. URL: https://moodle.unizar.es/add/pluginfile.php/2248532/mod_resource/content/5/2b_DiagramasCasosDeUso.pdf.

[24] Javier Lacasta Miguel. *DISEÑO CONCEPTUAL DE BASES DE DATOS*. 2019. URL: https://moodle.unizar.es/add/pluginfile.php/1849638/mod_resource/content/5/02_modeloE-R_basico.pdf.

[25] Jesús Barrasa. *Importing RDF data into Neo4j*. 2016. URL: https://jbarrasa.com/2016/06/07/importing-rdf-data-into-neo4j/ (visitado 15-06-2021).

[26] José Domingo Muñoz. *¿Qué es Flask?* 2017. URL: https://openwebinars.net/blog/que-es-flask/ (visitado 12-08-2021).

[27] Carlos Tellería Orriols Raquel Trillo Lado Ramón Hermoso Traba. *Patrón MVC - Grado en Ingeniería en Informática e Ingeniería de Tecnologías y Servicios de Telecomunicación*. 2019. URL: https://moodle.unizar.es/add/pluginfile.php/2408605/mod_resource/content/1/PatronMVC.pdf.

[28] *Neo4j graph operations in a multithreaded environment*. 2013. URL: https://stackoverflow.com/questions/12701185/neo4j-graph-operations-in-a-multithreaded-environment (visitado 13-05-2021).

[29] Lewis John McGibbney. *Welcome to the Apache Nutch Wiki*. 2021. URL: https://cwiki.apache.org/confluence/display/NUTCH/Home#Home-GeneralInformation/ (visitado 25-02-2021).

[30] Apache Foundation. *Apache Nutch News*. 2021. URL: http://nutch.apache.org/ (visitado 25-02-2021).

[31] Sebastian Nagel. *Web Crawling with Apache Nutch*. 2014. URL: https://www.slideshare.net/sebastian_nagel/aceu2014-snagelwebcrawlingnutch/ (visitado 26-02-2021).

[32] Tony's Programming Stuff. *Apache Nutch 2.0 Tutorial (with Elasticsearch)*. 2019. URL: https://youtu.be/AvyBiGuBc64.

[33] The free encyclopedia Wikipedia. *Apache Nutch*. 2021. URL: https://en.wikipedia.org/wiki/Apache_Nutch (visitado 25-02-2021).

[34] DigitalPebble Ltd. *A collection of resources for building low-latency, scalable web crawlers on Apache Storm*. 2021. URL: http://stormcrawler.net/ (visitado 28-02-2021).

[35] Wikipedia. *StormCrawler*. 2020. URL: https://en.wikipedia.org/wiki/StormCrawler (visitado 28-02-2021).

[36] Apache Software Foundation. *Why use Apache Storm?* 2019. URL: http://storm.apache.org/ (visitado 28-02-2021).

[37] Julien Nioche. *StormCrawler 1.16 + Elasticsearch 7.5.0*. 2020. URL: https://youtu.be/8kpJLPdhvLw (visitado 28-02-2021).

[38] DigitalPebble Ltd. *StormCrawler 2.0*. 2020. URL: https://github.com/DigitalPebble/storm-crawler/releases/tag/2.0 (visitado 28-02-2021).

[39] Yasser Ganjisaffar. *Crawler4j Issues*. 2020. URL: https://github.com/yasserg/crawler4j/issues (visitado 28-02-2021).

[40] Yasser Ganjisaffar. *Open Source Web Crawler for Java*. 2020. URL: https://github.com/yasserg/crawler4j#quickstart (visitado 28-02-2021).

[41] ScrapeHero. *Best Web Crawling Tools and Frameworks in 2020*. 2021. URL: https://www.scrapehero.com/best-web-crawling-tools-and-frameworks/ (visitado 28-02-2021).

[42] Amy DeGregorio. *A Guide to Crawler4j*. 2019. URL: https://www.baeldung.com/crawler4j (visitado 03-03-2021).

[43] Yasser Ganjisaffar. "Open Source Web Crawlers". En: *UCI Donald Bren School of Information and Computer Sciences* (2018). URL: https://www.ics.uci.edu/~djp3/classes/2009_01_02_INF141/Lectures/Discussion02.pdf.

[44] Scrapy developers. *Architecture overview*. 2021. URL: https://docs.scrapy.org/en/latest/topics/architecture.html (visitado 05-03-2021).

[45] Attreya Bhatt. *Python Scrapy Tutorial*. 2019. URL: https://youtu.be/ve_0h4Y8nuI (visitado 05-03-2021).

[46] Leonard Richardson. *Beautiful Soup Documentation*. 2020. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (visitado 06-03-2021).

[47] Uni Python. *El modelo Embeddings (Incrustaciones) de Palabras*. 2021. URL: https://unipython.com/el-modelo-embeddings-incrustaciones-de-palabras/ (visitado 06-07-2021).

[48] David Mezzetti. *Part 7: Apply labels with zero-shot classification*. 2021. URL: https://github.com/neuml/txtai/blob/master/examples/07_Apply_labels_with_zero_shot_classification.ipynb (visitado 10-07-2021).

[49] Rabeh Ayari. *Word Embedding Techniques Demystified*. 2020. URL: https://towardsdatascience.com/nlp-embedding-techniques-51b7e6ec9f92 (visitado 12-07-2021).

[50] Alejandro Vaca. *Transformers en Procesamiento del Lenguaje Natural*. 2021. URL: https://www.iic.uam.es/innovacion/transformers-en-procesamiento-del-lenguaje-natural (visitado 13-07-2021).

[51] Carlos Santana. *La Siguiente Gran Revolución: NLP (Procesamiento del Lenguaje Natural)*. 2020. URL: https://youtu.be/cTQiN9dewIg (visitado 20-07-2021).

[52] Francesca Bitto. *Graph Databases TerminusDB vs Neo4j*. 2021. URL: https://blog.terminusdb.com/graph-databases-terminusdb-vs-neo4j (visitado 22-06-2021).

[53] Wikipedia. *TerminusDB*. 2021. URL: https://en.wikipedia.org/wiki/TerminusDB (visitado 22-06-2021).

[54] Paradigma. *¡Hola! ¿Conoces ArangoDB?* 2018. URL: https://www.paradigmadigital.com/dev/hola-conoces-arangodb/ (visitado 23-06-2021).

[55] Community driven projects for the ArangoDB database. *ArangoDB Community*. 2021. URL: https://github.com/ArangoDB-Community/ (visitado 23-06-2021).

[56] Neo4j Inc. *Neosemantics Neo4j RDF and Semantics toolkit*. 2021. URL: https://neo4j.com/labs/neosemantics/ (visitado 23-06-2021).

[57] Neo4j Inc. *Getting Started with Neo4j*. 2021. URL: https://neo4j.com/developer/get-started/ (visitado 23-06-2021).

[58] Neo4j Inc. *Using Neo4j from Python*. 2021. URL: https://neo4j.com/developer/python/ (visitado 23-06-2021).

[59] Jorge Gracia del Río. *Introducción al PLN. Tema 6 – Semántica distribucional. Departamento de Informática e Ingeniería de Sistemas Universidad de Zaragoza.* 2021. URL: https://moodle.unizar.es/add/pluginfile.php/3217212/mod_resource/content/1/6_IntroPLN_202021_SemanticaDistribucional.pdf.

[60] Jorge Gracia del Río. *Introducción al PLN. Tema 9 – Transformers. Departamento de Informática e Ingeniería de Sistemas Universidad de Zaragoza.* 2021. URL: https://moodle.unizar.es/add/pluginfile.php/3234961/mod_resource/content/1/9_IntroPLN_202021_Transformers.pdf.

[61] Raquel Trillo Lado. *Semantic Techniques for Improving Keyword-based Searching*. 2012. URL: https://www.researchgate.net/publication/228073112_Semantic_Techniques_for_Improving_Keyword-based_Searching (visitado 15-06-2021).

[62] *International Data Spaces Information Model*. 2021. URL: https://international-data-spaces-association.github.io/InformationModel/docs/index.html.

[63] Duncan Brown. *Introducing Bolt, Neo4j's Upcoming Binary Protocol – Part 1*. 2015. URL: https://dzone.com/articles/introducing-bolt-neo4js-upcoming-binary-protocol-p (visitado 07-08-2021).

[64] Ashish Vaswani Noam Shazeer Niki Parmar Jakob Uszkoreit Llion Jones Aidan N Gomez Łukasz Kaiser Illia Polosukhin. *Attention Is All You Need*. 2017. URL: https://papers.nips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[65] *FullSemanticsNamedIndividuals*. 2008. URL: https://www.w3.org/2007/OWL/wiki/FullSemanticsNamedIndividuals (visitado 17-06-2021).

[66] World Wide Web Consortium. *RDF Schema 1.1*. 2014. URL: https://www.w3.org/TR/2014/REC-rdf-schema-20140225/#ch_type (visitado 05-07-2021).

[67] Tomas Mikolov Kai Chen Greg Corrado Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*. URL: https://arxiv.org/pdf/1301.3781.pdf.

[68] W3 Schools. *jQuery Tutorial*. 2021. URL: https://www.w3schools.com/jquERy/default.asp (visitado 20-06-2021).

[69] Francisco Javier Fabra Caro. *Práctica 1: Diseño y desarrollo de interfaces*. 2021. URL: https://moodle.unizar.es/add/pluginfile.php/3174773/mod_resource/content/2/P1-Bootstrap.pdf.

[70] Gobierno de las Islas Baleares. *Puntos Carga Vehículos Eléctricos Illes Balears*. 2020. URL: https://data.europa.eu/data/datasets/https-catalegdades-caib-cat-api-views-qa96-dprj?locale=es.

[71] *Sea ice bio optical measurements*. 2017. URL: https://data.gov.au/dataset/ds-aodn-88ae8b38-1363-407e-8860-75e4b9ecab10/details?q=sea%20ice%20bio-optical.

[72] *Marine environmental data layers for Southern Ocean species distribution modelling*. 2018. URL: https://data.gov.au/dataset/ds-aodn-000239cf-b0e2-440b-be3e-1f0bea645d85/details?q=marine%20environmental%20data%20layer.

[73] SolrTutorial 2020. *¿Qué es solr?* 2020. URL: https://solrtutorial.es/que-es-solr.html (visitado 23-02-2021).

[74] *The Transformer - model architecture*. URL: https://miro.medium.com/max/1838/1*BHzGVskWGS_3jEcYYi6miQ.png.

[75] Rafal Kuć. *Solr vs. Elasticsearch: Performance Differences More. How to Decide Which One is Best for You*. 2021. URL: https://sematext.com/blog/solr-vs-elasticsearch-differences/ (visitado 23-07-2021).

[76] Developers Mozilla. *JavaScript*. 2021. URL: https://developer.mozilla.org/es/docs/Web/JavaScript.

[77] The free encyclopedia Wikipedia. *API*. 2021. URL: https://en.wikipedia.org/wiki/API.

[78] Mike Chapple. *Abandoning ACID in Favor of BASE in Database Engineering*. 2020. URL: https://www.lifewire.com/abandoning-acid-in-favor-of-base-1019674.

[79] The free encyclopedia Wikipedia. *Bolt Network Protocol*. 2021. URL: https://en.wikipedia.org/wiki/Bolt_(network_protocol).

[80] The free encyclopedia Wikipedia. *NoSQL*. 2021. URL: https://es.wikipedia.org/wiki/NoSQL.

[81] The free encyclopedia Wikipedia. *SQL*. 2021. URL: https://es.wikipedia.org/wiki/SQL.

[82] The free encyclopedia Wikipedia. *Sistema de gestión de bases de datos*. 2021. URL: https://es.wikipedia.org/wiki/Sistema_de_gesti%C3%B3n_de_bases_de_datos.

[83] Neo4j Inc. *Cypher Query Language*. 2021. URL: https://neo4j.com/developer/cypher/.

[84] The free encyclopedia Wikipedia. *Procesamiento de lenguajes naturales*. 2021. URL: https://es.wikipedia.org/wiki/Procesamiento_de_lenguajes_naturales.

[85] The free encyclopedia Wikipedia. *Identificador de recursos uniforme*. 2021. URL: https://es.wikipedia.org/wiki/Identificador_de_recursos_uniforme.

[86] The free encyclopedia Wikipedia. *JavaScript Object Notation*. 2021. URL: https://es.wikipedia.org/wiki/JSON.

[87] The free encyclopedia Wikipedia. *Docker*. 2021. URL: https://es.wikipedia.org/wiki/Docker_(software).

[88] The free encyclopedia Wikipedia. *Apache Hadoop*. 2021. URL: https://es.wikipedia.org/wiki/Apache_Hadoop.

[89] Mozilla e individual contributors. *HTML: Lenguaje de etiquetas de hipertexto*. 2021. URL: https://developer.mozilla.org/es/docs/Web/HTML.

[90] Mozilla e individual contributors. *HTTP*. 2021. URL: https://developer.mozilla.org/es/docs/Web/HTTP.

[91] Mozilla e individual contributors. *JavaScript*. 2021. URL: https://developer.mozilla.org/es/docs/Web/JavaScript.

[92] Ivan de Souza. *XML ¿qué es y para qué sirve este lenguaje de marcado?* 2019. URL: https://rockcontent.com/es/blog/que-es-xml/.

Anexos

Anexos A

Anexo 1

A.1. Ejemplo conjunto de datos en repositorio online

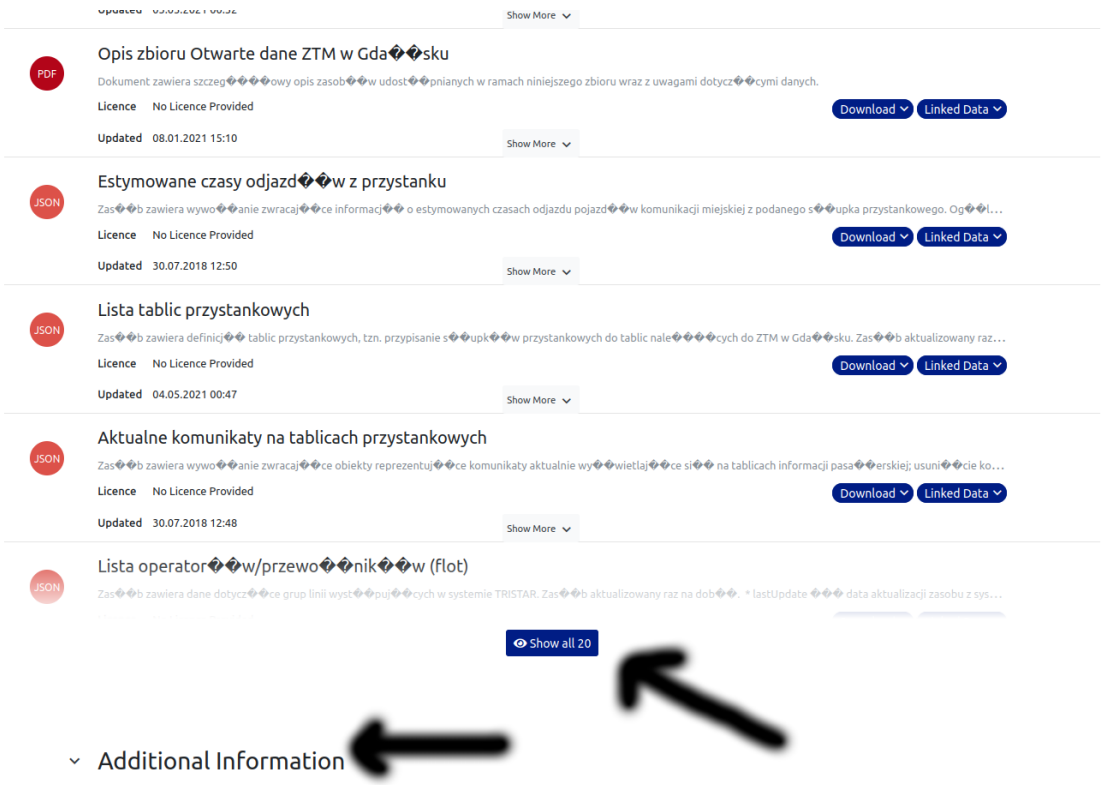

Figura A.1: Captura de pantalla Data Europa

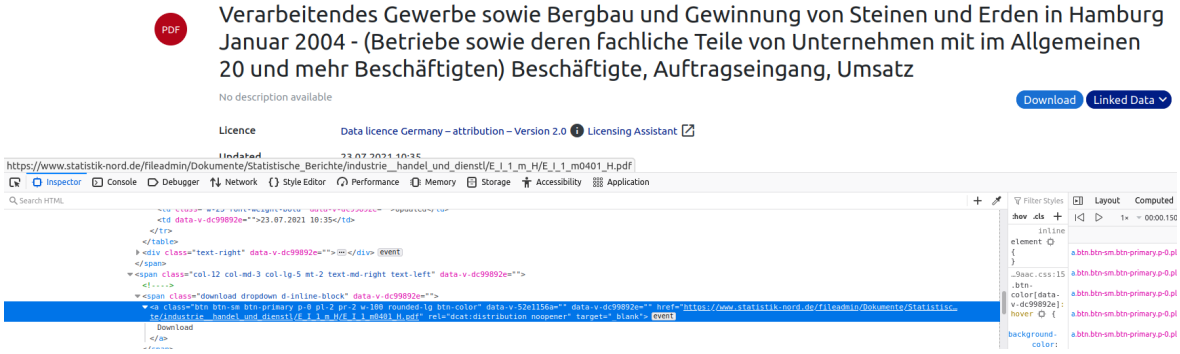

Figura A.2: Ejemplo conjunto de datos en Data Europa parte 1

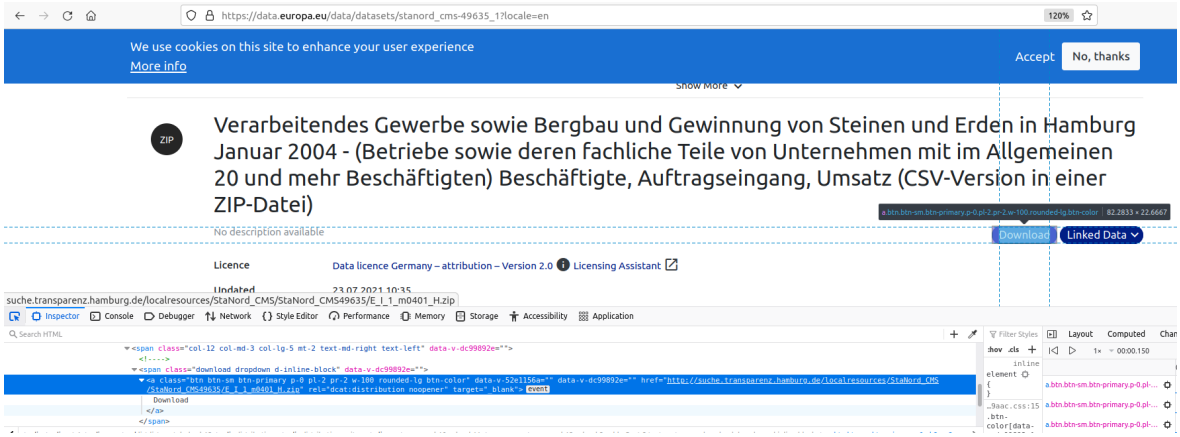

Figura A.3: Ejemplo conjunto de datos en Data Europa parte 2

A.2. Ejemplo tráfico red en repositorio online

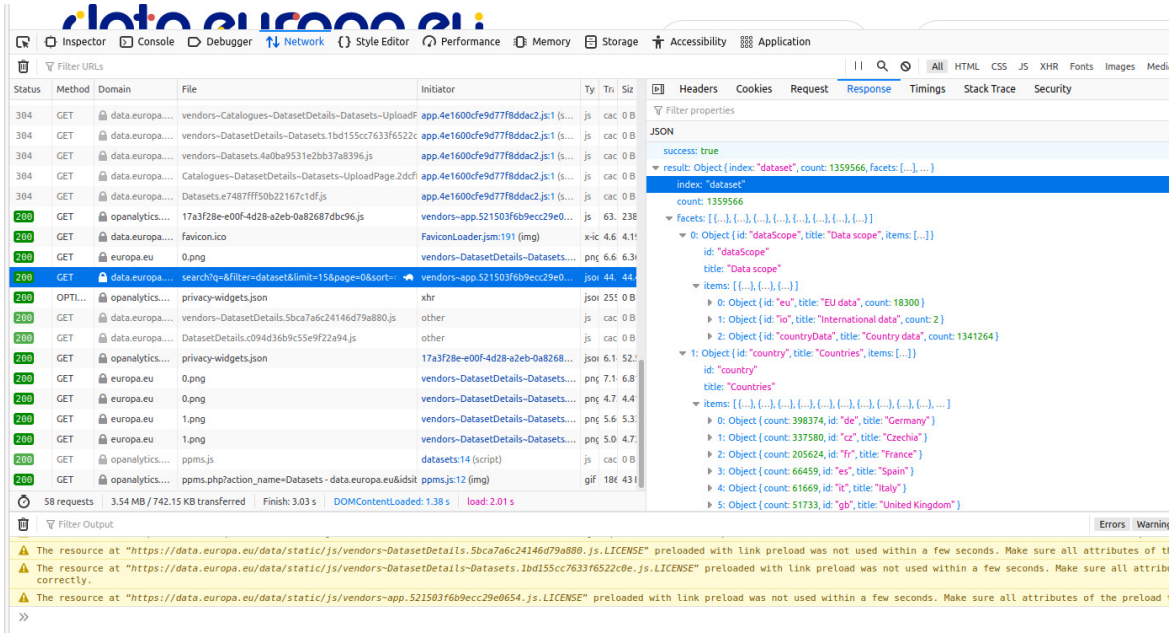

Figura A.4: Tráfico red Data Europa

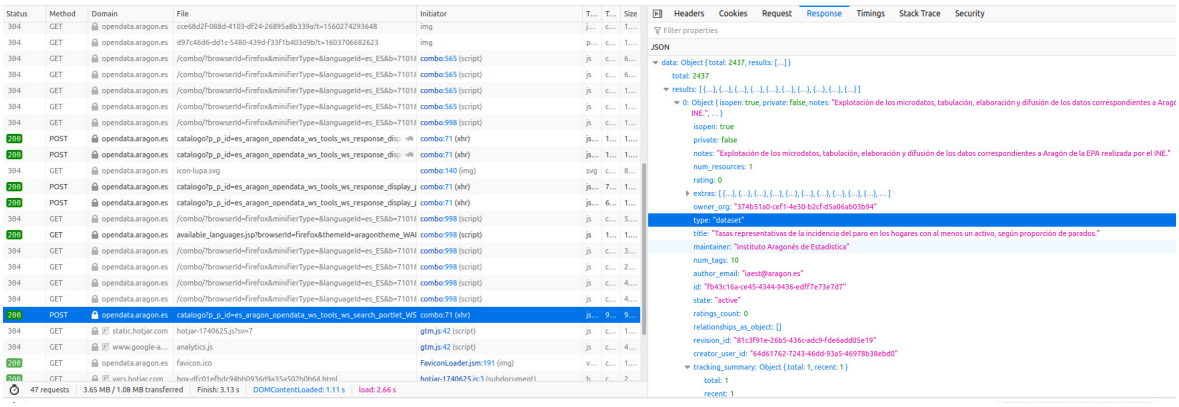

Figura A.5: Tráfico red Open Data Aragón

A.3. Estado del arte tecnologías bases de datos

A continuación se llevará a cabo una breve introducción a las bases de datos orientadas a grafos de código abierto y la explicación de la elección para este proyecto.

A.3.1. TerminusDB

TerminusDB[26] es una base de datos noSQL, orientada a grafos, que utiliza su propio lenguaje de consulta **WOQL** (Web Object Query Language) que a su vez permite escribir consultas directamente en JavaScript, Python o JSON [52]. A su vez, también cuenta con un driver en Python. Es un proyecto de código abierto, cuya licencia es **Apache License 2.0**, diseñado para construir grafos de forma colaborativa con un control de versiones similar a Git. Permite guardar documentos y ficheros RDF. **TerminusDB** utiliza el **lenguaje OWL** con modificaciones para el diseño de su esquema [53].

A.3.2. ArangoDB

ArangoDB[27] se trata de una base de datos orientada a grafos. Cuenta con varios drivers en Python desarrollados por su comunidad [55]. Permite flexibilidad en el modelo de datos, los datos pueden almacenarse como pares de clave-valor, documentos o grafos, por lo que no es exclusivamente un base de datos orientada a grafos. Cuenta con su propio lenguaje para las consultas llamado **AQL** (ArangoDB Query Language). **ArangoDB** es un proyecto de código abierto bajo licencia **Apache License 2.0** que cuenta con interfaz web y soporta *multithread*. Al tratarse de una base de datos noSQL

[26] https://terminusdb.com/
[27] https://www.arangodb.com/

prioriza la consistencia de la información frente a la disponibilidad y la tolerancia a particiones [54].

A.3.3. Neo4j

Neo4j[28] es un sistema gestor de bases de datos orientado a grafo desarrollado por la empresa Neo4j, Inc[29] . Es un proyecto de código abierto con licencia GPLv3. Soporta transacciones que cumplen las propiedades ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad). Neo4j está desarrollado en Java, tiene su propio lenguaje de consulta denominado CYPHER. Pueden ejecutarse consultas en CYPHER desde otros programas a través del uso del protocolo HTTP o mediante el protocolo binario *bolt* [63]. Hay disponibles varios drivers en Python explicados en la documentación oficial [58]. Neo4j tiene un comunidad muy amplia y buena documentación [57]. Nativamente no soporta RDF, pero cuenta con un *plugin* llamado **Neosemantics**[30] desarrollado por el equipo de Neo4j que permite importar ficheros RDF y trabajar con los vocabularios asociados a RDF [56].

A.4. Conceptos procesamiento lenguaje natural

A.4.1. Word Embeddings

Word embedding es el nombre de un conjunto de lenguajes de modelado y técnicas de aprendizaje en procesamiento del lenguaje natural. Las *word embeddings*, o en español palabras incrustadas, se tratan de representaciones de vectoriales de palabras. El texto plano ha de ser transformado para ser utilizado y procesado en una red neuronal, es necesaria una representación numérica. Esta representación numérica queda plasmada en un vector que representa a cada palabra del vocabulario perteneciente al texto de lenguaje natural que se va a procesar. Las características semánticas o diferentes significados de cada palabra se extraen de forma explícita ya que a palabras similares les corresponden vectores similares, es decir, cercanos en el espacio [59]. Los dos modelos de *word embeddings* tratados en este proyecto son Word2Vec y GloVe, que serán explicados a continuación.

A.4.2. Word2Vec

Word2vec [67] se trata de un modelo predictivo desarrollado por Tomas Mikolov en Google en 2013. Utiliza *word embeddings* no contextuales, ya que el contexto se extrae

[28]https://github.com/neo4j/neo4j
[29]https://neo4j.com/
[30]https://neo4j.com/labs/neosemantics/

de forma explícita en función de qué palabras se encuentran cerca.

Uno de los inconvenientes del uso de *word embeddings* con *one-hot-encoding* y representar las palabras con vectores es el hecho de que depende del tamaño del vocabulario los vectores y la matriz de ocurrencia que finalmente se genera pueden ser de dimensiones demasiado grandes.

Word2vec genera un espacio vectorial multidimensional de 300 dimensiones solventando el problema de tener vectores de dimensiones excesivamente grandes, ya que estos son computacionalmente ineficientes [59].

En Word2Vec se entrena una red neuronal con dos modelos de aprendizaje diferentes *CBOW model*, *Continous Bag of Words model* o Modelo de bolsa de palabras continuada, y *skip-gram model* o modelo de programa continuo de salto.

EL modelo de CBOW se utiliza para predecir palabras en base a un contexto, por el contrario el modelo *skip-gram* se utiliza para predecir el contexto en base a un palabra proporcionada en [49].

A.4.3. GloVe

GloVe[31] se trata de un proyecto de código abierto desarrollado en 2014 por la Universidad de Stanford. Es un algoritmo de representación de palabras basado en Word2Vec. GloVe combina las estadísticas globales con el aprendizaje local basado en el contexto en Word2Vec [47]. GloVe construye una matriz explícita co-ocurrencia de palabras usando estadísticas a través de todo el corpus del texto.

A.4.4. Transformers

En el año 2017 trabajadores de Google en colaboración con la universidad de Toronto publican un *paper* en el que proponen la arquitectura de *Transformer* [64]. Este modelo presentaba como principal innovación el uso de capas de atención (ver en la Figura A.6).

[31] https://nlp.stanford.edu/projects/glove/

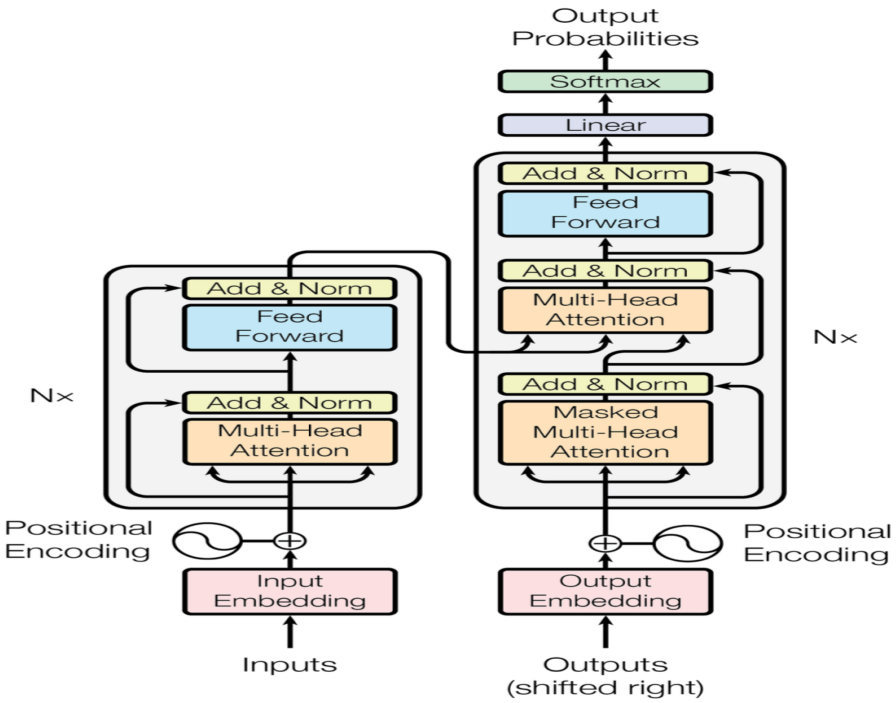

Figure 1: The Transformer - model architecture.

Figura A.6: Modelo arquitectural Transformer[74]

Estas capas de atención codifican cada palabra de una frase en función del resto de la secuencia, de esta forma es posible codificar el contexto en un vector que representa a dicha palabra. El modelo *Transformers* introduce el contexto en la representación matemática. Utiliza *embeddings* contextuales, ya que cada palabra cuenta con una representación diferente para cada uno de sus significados [50].

El uso de modelos basados en *Transformers* ha producido una revolución en el campo del procesamiento del lenguaje natural ya que es más preciso a la hora de encontrar el contexto y predecir la próxima palabra [51].

Es un modelo que permite ahorra tiempo de entrenamiento. Esto es posible porque un *Transformers* entrenado como modelo del lenguaje, es posible adaptarlo para que realice otras tareas añadiéndole capas a la arquitectura [60].

A.5. Tecnologías desarrollo frontend

A.5.1. Bootstrap

Bootstrap[32] es una biblioteca de código abierto que proporciona herramientas para desarrollo *web responsive* con HTML, CSS y JavaScript. Proporciona clases CSS y componentes predefinidos [69].

[32]https://getbootstrap.com/

A.5.2. JQuery

JQuery[33] es un biblioteca de JavaScript que permite la ejecución de peticiones asíncronas [68]. En este proyecto se ha utilizado JQuery para actualizar la interfaz sin tener que cambiar de página y mostrar los resultados de las búsquedas de conjuntos de datos tanto en los repositorios *online* como en la base de datos local.

A.5.3. Google Fonts

Google Fonts[34] es un directorio interactivo que proporciona fuentes tipográficas. Para este proyecto se han importado Lora y Open Sans.

A.5.4. Fontawesome

Fontawesome[35] se trata de un paquete que proporciona una gran variedad de iconos para desarrollo web.En este proyecto se ha utilizado para añadir una señal de error cuando no se encuentra conjuntos de datos.

A.6. Vocabulario utilizado

En esta sección se explicarán el vocabulario que se ha utilizado para construir los grafos RDF que se importarán a la base de datos gracias al uso del *plugin* Neosemantic para Neo4j.

A.6.1. Web Ontology Language

OWL (Web Ontology Language), se encuentra explicado en Subsección 2.1.2. De todo el vocabulario disponible perteneciente a OWL simplemente se ha utilizado la etiqueta **owl:NamedIndividual**. Esta etiqueta se utiliza para declarar cada una de las instancias de las clases de la ontología [65].

A.6.2. Resource Description Framework

RDF, explicado en Subsección 2.1.1, ha sido necesario el uso del vocabulario asociado a RDF para la construcción de las tripletas. La etiqueta **rdf:about** permite asociar URIs para ser utilizadas como sujeto en las tripletas RDF. En este caso es utilizada para darle una URI a cada instancia de la ontología que después será utilizada como identificador único de cada una de estas instancias en la base de datos.

[33] https://jquery.com
[34] https://fonts.google.com/
[35] https://fontawesome.com/

rdf:type es una etiqueta utilizada para especificar que un recurso pertenece una determinada clase. Este caso se ha utilizado para diferenciar a qué clase pertenece la instancia a poblar en la ontología. *rdf:resource* es una etiqueta utilizada para establecer la URI objeto de las tripletas RDF [66]. Véase este cómo utilizar estas dos últimas etiquetas en Listing A.1. En este ejemplo se declara que la instancia owl:NamedIndividual con URI `https://w3id.org/idsa/core#ISOPODS` se trata de una clase cuya URI es `https://w3id.org/idsa/core/keyword`.

Listing A.1: Ejemplo uso rdf:about

```
<owl:NamedIndividual rdf:about="https://w3id.org/idsa/core#ISOPODS">
    <core:name>isopods</core:name>
    <rdf:type rdf:resource="https://w3id.org/idsa/core/keyword"/>
</owl:NamedIndividual>
```

A.6.3. Dublin Core

Dublin Core es un modelo de metadatos desarrollado por la organización Dublin Core Metadata Initiative[36]. De estos metadatos simplemente se ha utilizado la etiqueta *dct:Location* para indicar a qué país pertenece cada conjunto de datos.

A.6.4. Modelo de Información IDS

Se trata del vocabulario perteneciente al modelo de información propuesto por la International Data Space Assocation explicado en Subsección 2.4.4. Se han utilizado tanto clases como propiedades. Las clases utilizadas de este vocabulario son: *DataResource*, *License*, *Representation*, *Language* y *Resource*.

- **DataResource** es la clase que identifica a los conjuntos de datos.
- **Representation** identifica a las distintas distribuciones de los conjuntos de datos, ya que estas distribuciones son diferentes representaciones del conjunto de datos en distintos formatos.
- **License** es la clase que identifica a las licencias, en este proyecto cada licencia esta asociada a una distribución.
- **Language** es la clase que identifica a los idiomas. En este proyecto se usa para identificar si un conjunto de datos esta en inglés, en español o en ambos idiomas.
- **Resource**, se trata de la clase que comprende al resto de generales recursos que forman parte de la ontología, aquellos a los que no son instancia de una clase en concreto.

[36]https://dublincore.org/

A continuación se indicarán las propiedades que se han utilizado. En un primer momento se mencionarán aquellas que **Neosemantic** ha interpretado como relaciones entre grafos y posteriormente aquellas que ha interpretado como propiedades de los grafos. Aquellas que han quedado guardadas como relaciones son: *keyword*, *representation*, *language* y *publisher*.

- **Keyword**. La relación *keyword* indica que un instancia de clase *Resource* es una palabra clave perteneciente a un conjunto de datos.
- **Publisher**. La relación *publisher* establece quién ha sido el publicador de un conjunto de datos.
- **Representation**. Esta relación vincula a un conjunto de datos con sus diferentes distribuciones
- **Language**. La relación *language* establece en qué idioma o idiomas esta disponible un conjunto de datos

Las relaciones *keyword* y *publisher* anteriormente mencionadas relacionan una instancia de la clase *DataResource* y una , o varias en el caso de *keyword*, instancias de la clase Resource porque esta ontología no cuenta con clases *keyword* o *Publisher* que representen específicamente una palabra clave o al publicador del conjunto de datos. Aquellas que son propiedades de los nodos son: *name*, *title*, *source*, *accessURL*, *description*, *modified*, *creationDate* y *contentType*.

- **Title**. Utilizado para los títulos de las licencias, conjuntos de datos y sus distribuciones.
- **Name**. Indica el nombre el publicador y de las palabras clave.
- **AccessURL**. Utilizado para indicar cuál es la web del publicador.
- **Source**. Etiqueta las *urls* asociadas a las distintas distribuciones, de estas *urls* es de donde se puede obtener el contenido de los conjuntos de datos.
- **ContentType**. Indica el formato de la distribución.
- **Description**. Utilizado para etiquetas las descripciones de los conjuntos de datos.
- **Modified**. Utilizado para guardar la fecha de última actualización de un conjunto de datos.
- **CreationDate**. Utilizada para guardar la fecha de la creación de un conjunto de datos.

Anexos B

Anexo 2

B.1. Explicación requisitos funcionales

A continuación se describen estos requisitos detalladamente.

- **RF-1**. Dada una consulta por parte del usuario la herramienta ha de devolver los conjuntos de datos que más se asemejen.
- **RF-2**. Dada una palabra clave ha de devolver los conjuntos de datos que contengan esa palabra clave o semejantes.
- **RF-3**. A las búsquedas de los requisitos **RF-1** y **RF-2** han de poder añadirse un filtro por publicador, país y categoría. De esta forma se descartan los conjuntos de datos que no pertenezcan a la categoría, país o publicador.
- **RF-4**. Dada una categoría, publicador o país han de obtenerse todos los conjuntos de datos almacenados que cuenten con una de estas propiedades.
- **RF-5**. Los enlaces del contenido de las distribuciones han de estar disponibles para poder acceder a ellos y descargar su contenido.
- **RF-6**. Los conjuntos de datos a los que se acceda con más frecuencia han de poder ser marcados para acceder más rápidamente a ellos. Han de poder desmarcarse y marcarse los conjuntos de datos en cualquier momento.
- **RF-7**. Cuando se descarguen los conjuntos de datos el contenido de sus metadatos ha de adaptarse al vocabulario propio de la ontología. De esta forma se ha de contar con un proceso de estandarización de los conjuntos de datos al poblar la ontología en esta herramienta.
- **RF-8**. La herramienta no puede estar restringida por uno o varios repositorios de datos, ha de poder ser extensible a más repositorios.

– **RF-9**. Aquellos conjuntos de datos que compartan propiedades en común han de estar relacionados en la base de datos.

– **RF-10**. Incorporar un nuevo repositorio público de datos no ha de alterar el resto del funcionamiento de la aplicación y ha de ser una tarea rápida y sencilla.

– **RF-11**. Las consultas han de estar enriquecidas mediante el uso de procesamiento de lenguaje natural.

– **RF-12**. La base de datos ha de poder guardar las relacione y ha de contar con operaciones para poder navegar entre ellas.

B.2. Explicación requisitos no funcionales

– **RNF-1**. Ha de utilizarse **Docker**[37] para desplegar la herramienta. Es necesario el uso de **Docker** para este proyecto debido a su portabilidad, compatibilidad y rápido despliegue.

– **RNF-2**. Uso de una librería de búsqueda semántica basada en embedded para el procesado de las consultas y creación de los índices para las búsquedas.

– **RNF-3**. La ontología ha ser poblada en este proyecto ha de ser la ontología de **International Data Spaces Association**. Ontología impuesta por los directores de este proyecto.

– **RNF-4**. El código de este proyecto ha de almacenarse en un repositorio de GitHub para su revisión durante su desarrollo.

[37]https://www.docker.com/

B.3. Modelo de datos orientado a grafos

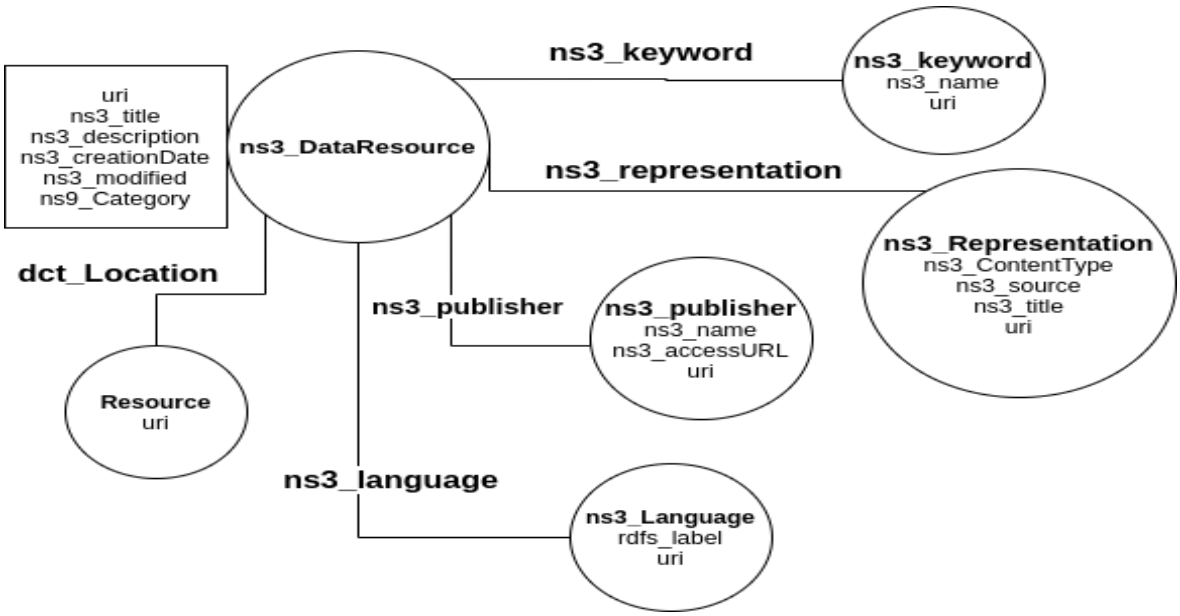

Figura B.1: Diagrama modelo de datos

B.4. Prototipado interfaces

En esta sección se incluirán imágenes de los bocetos iniciales de la interfaz de la herramienta. Los bocetos que se han incluido se corresponden a las pantallas de búsqueda avanzada, principal, conjunto de datos recurrentes, información y visualización de un conjunto de datos.

B.4.1. Pantalla de búsqueda avanzada

En esta pantalla podrá consultarse a la base de datos local filtrando por país, categoría y publicador. Las consultas podrán hacerse por título, palabra clave o ambas. Se mostrarán los resultados debajo, de cada resultado se mostrará su título y descripción.

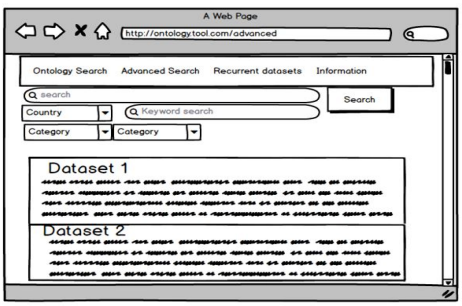

Figura B.2: Pantalla búsqueda avanzada

B.4.2. Pantalla principal

En la página principal se encontrará la barra de búsqueda que permite descargar los conjuntos de datos de los repositorios *online* públicos. Se podrá seleccionar un país y una categoría. De los resultados se mostrará su título y descripción.

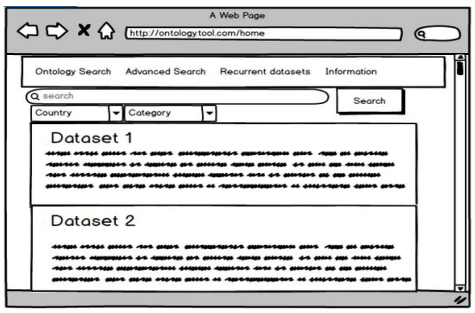

Figura B.3: Pantalla principal

B.4.3. Pantalla de visualización de un conjunto de datos

En esta pantalla se visualizará el contenido de un conjunto de datos. Se mostrará su título, en caso de tener un segundo título se mostrará como subtítulo, su descripción, su publicador, sus palabras clave y sus distribuciones. De sus distribuciones se mostrará la información asociada a la distribución, título, enlace y formado, junto a la licencia de cada distribución.

Figura B.4: Pantalla conjunto de datos parte 2

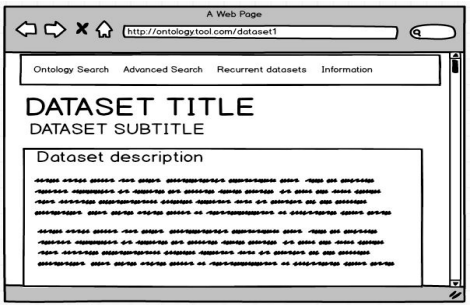

Figura B.5: Pantalla conjunto de datos parte 1

B.4.4. Pantalla de información

Esta última pantalla contiene ha de contener una breve guía en la que se explique al usuario cómo interactuar con la herramienta.

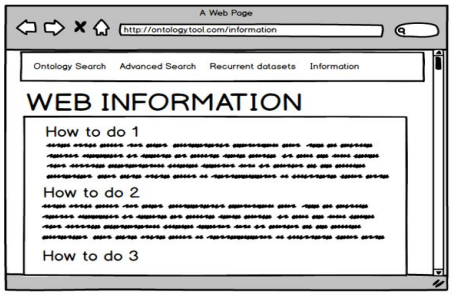

Figura B.6: Pantalla de información sobre la herramienta

Anexos C

Anexo 3

C.1. Diagrama de paquetes Frontend

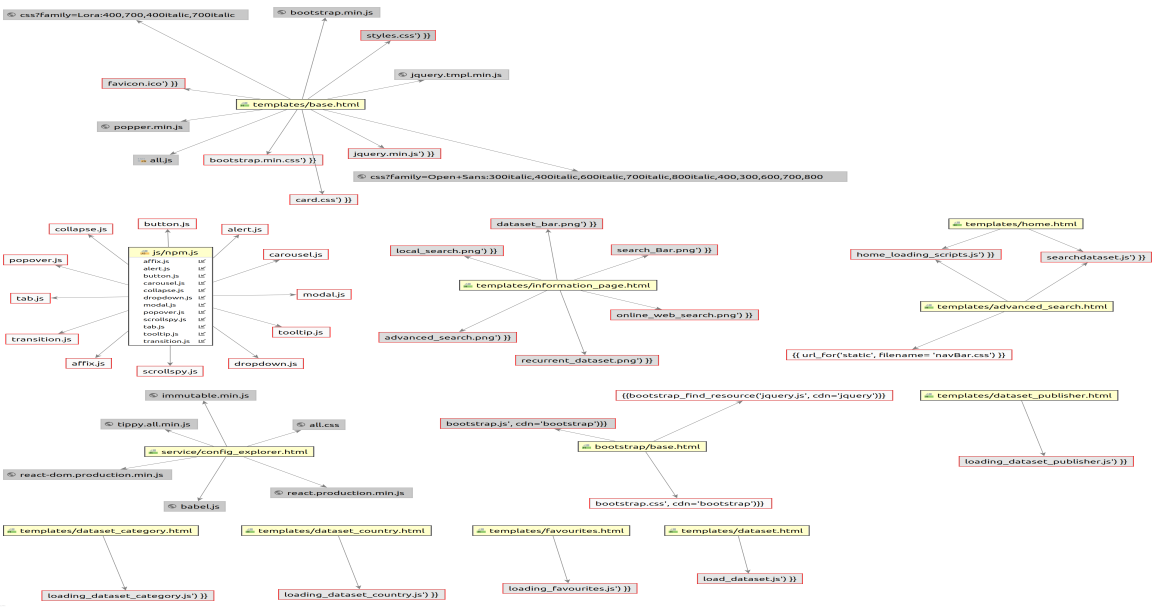

Figura C.1: Diagrama de paquetes Frontend

C.2. Interfaces

A continuación se muestra el resultado final de la interfaz de la aplicación.

C.2.1. Página inicial

Esta es la interfaz de la página principal de la herramienta. Desde este página es desde donde se realiza la extracción de nuevos conjuntos de datos pertenecientes a los repositorios *online* conectados a esta herramienta. A la búsqueda pueden añadirse los filtros de búsqueda por país y por categoría. Se muestran los resultados, tanto de una búsqueda local como de una búsqueda en los repositorios.

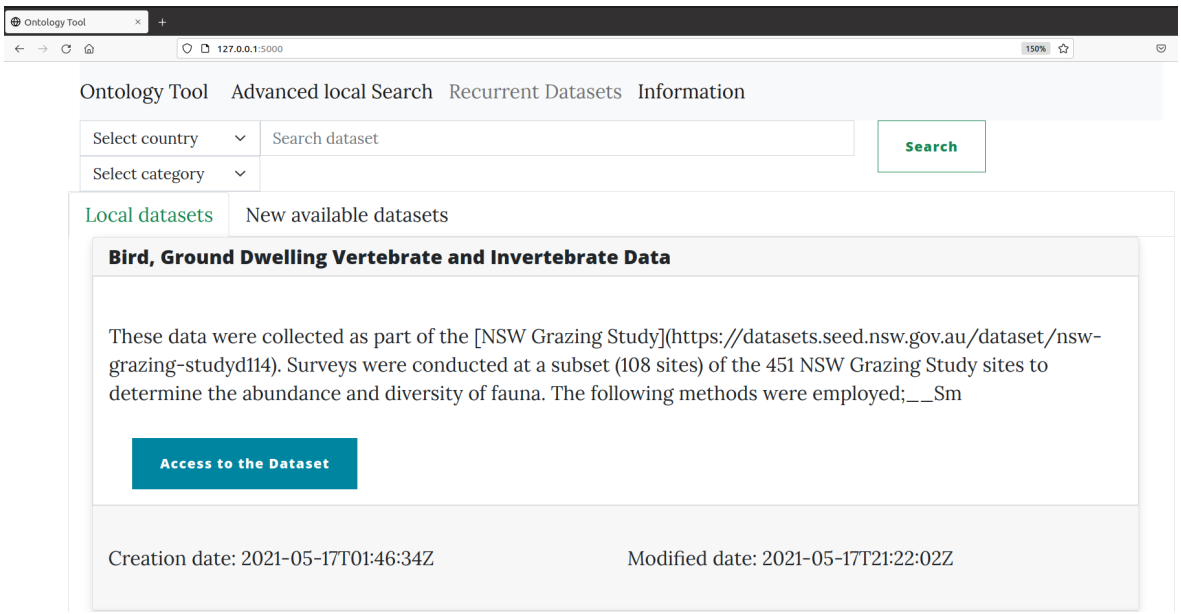

Figura C.2: Pantalla inicial

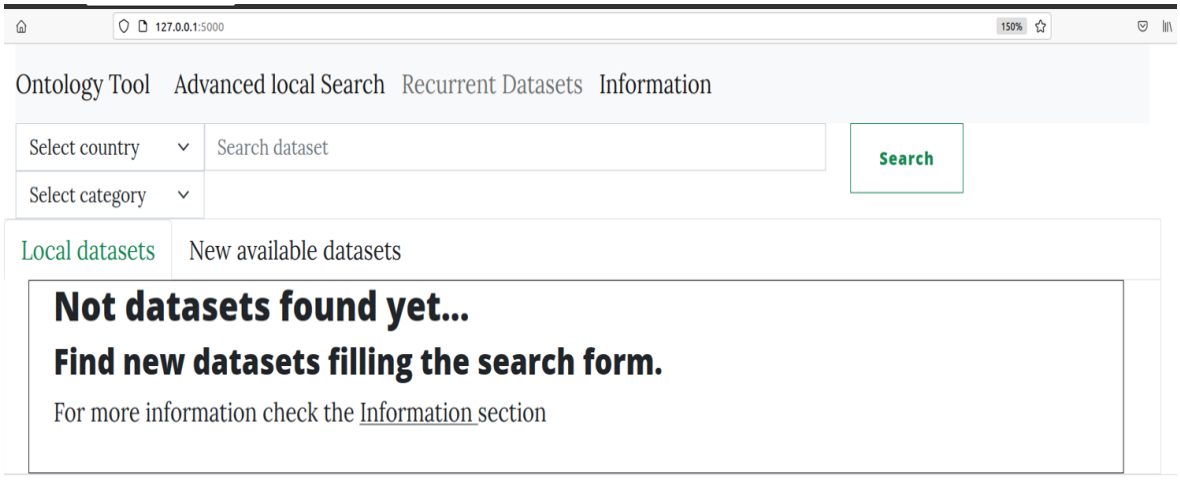

Figura C.3: Página inicial antes de buscar

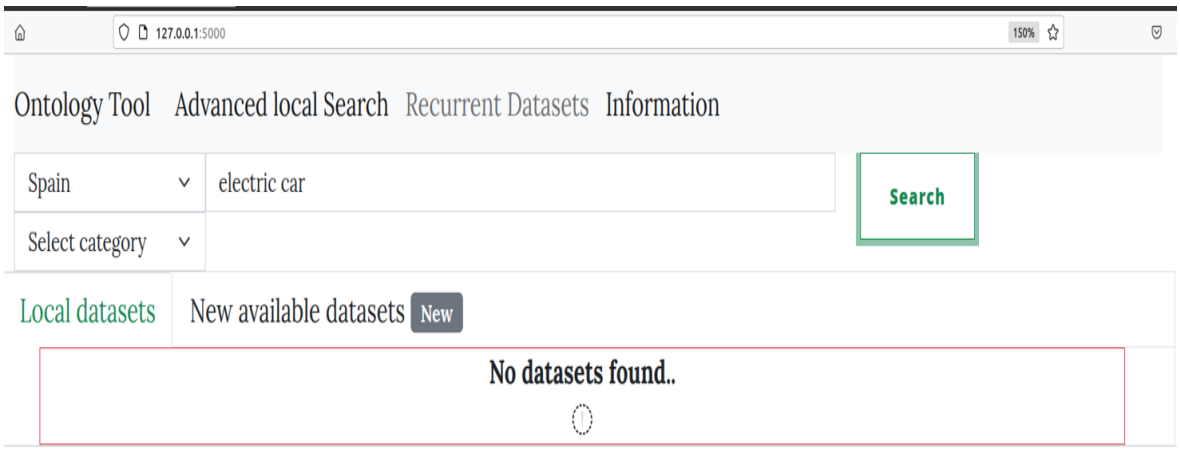

Figura C.4: Página inicial cuando falla la búsqueda local

C.2.2. Búsqueda Avanzada

Interfaz de la búsqueda avanzada. En esta pantalla se realiza búsquedas de conjuntos de datos en la base de datos local. Pueden buscarse los conjuntos de datos por título y por palabra de clase. Las búsquedas pueden filtrarse por país, categoría y publicador.

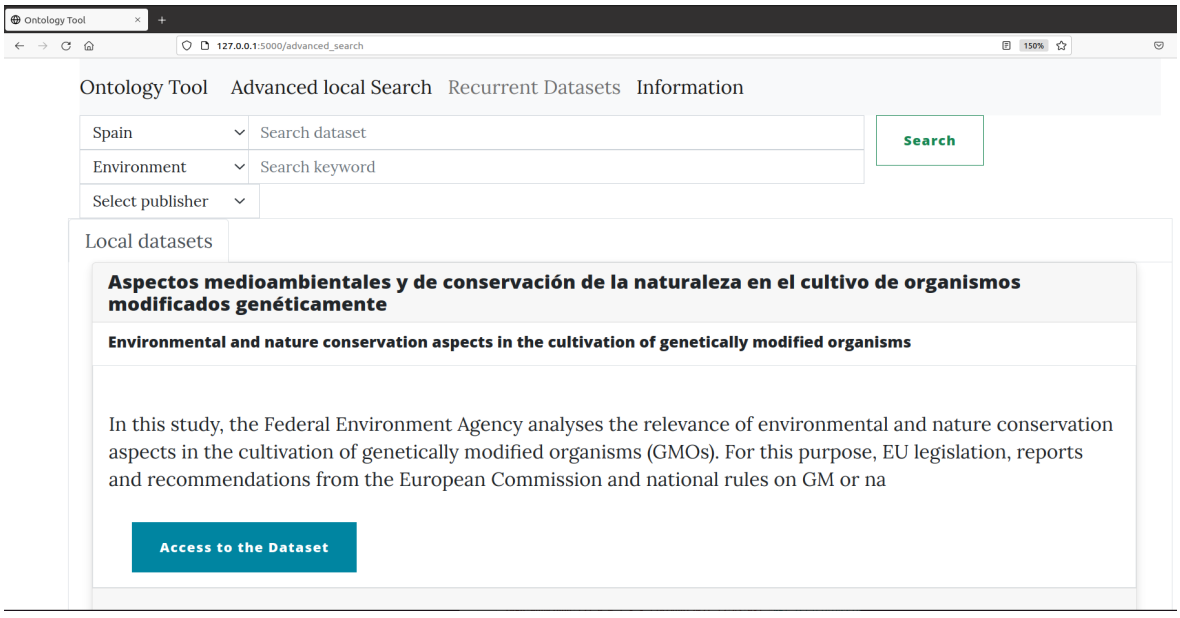

Figura C.5: Búsqueda Avanzada

C.2.3. Resultado búsqueda por publicador

Interfaz que muestra los conjuntos de datos cuyo publicador es el mismo.

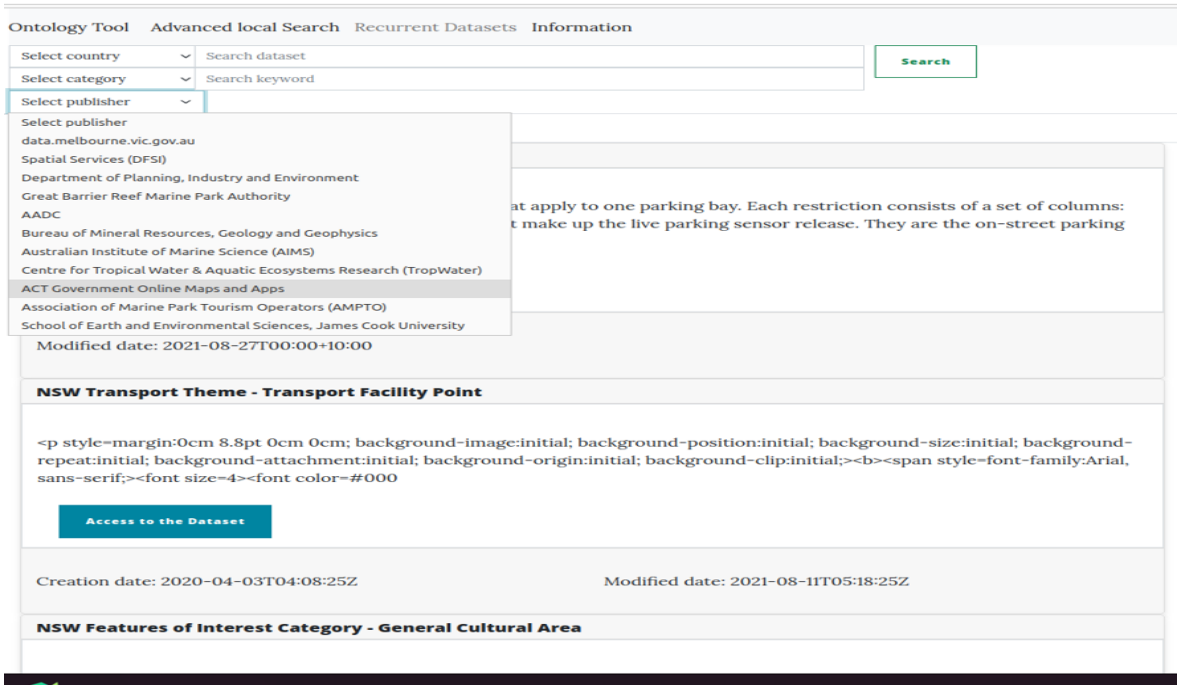

Figura C.6: Elección de publicador

Figura C.7: Conjunto de datos que comparten publicador

C.2.4. Resultado búsqueda por palabra clave

Interfaz que muestra los conjuntos de datos que tienen la palabra clave *ocean chemistry*.

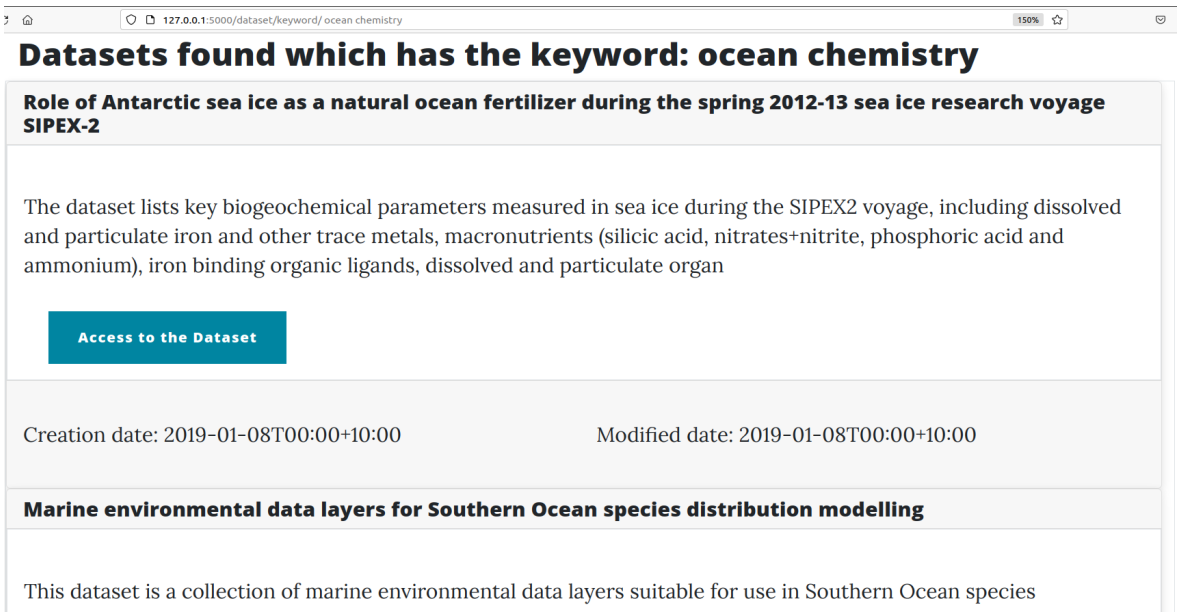

Figura C.8: Resultado palabra clave *ocean chemistry*

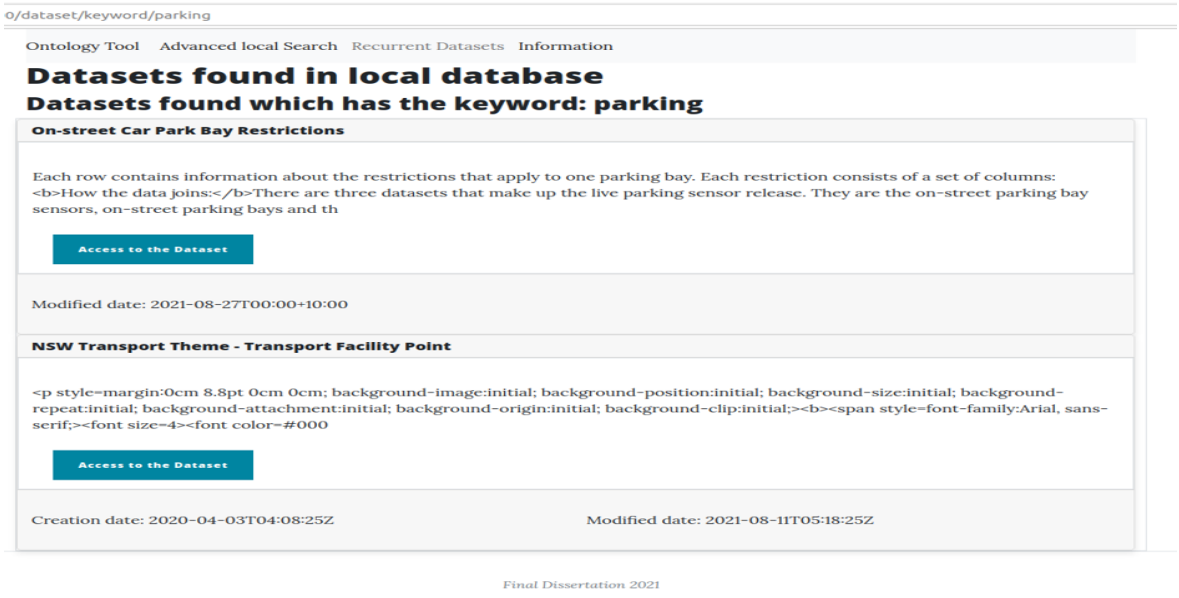

Figura C.9: Resultado palabra clave **parking**

C.2.5. Conjuntos de datos recurrentes

Interfaz en la que se muestran los conjuntos de datos que han sido marcados como conjuntos de datos recurrentes.

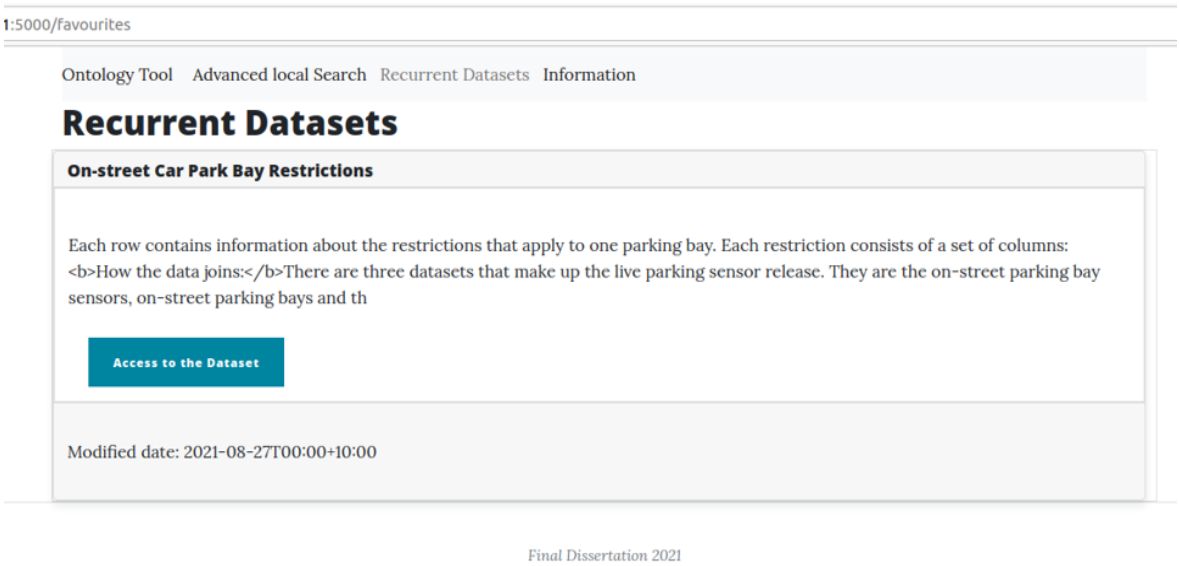

Figura C.10: Conjuntos de datos recurrentes

C.2.6. Página información del uso de la herramienta

En esa interfaz se explica cómo utilizar la herramienta.

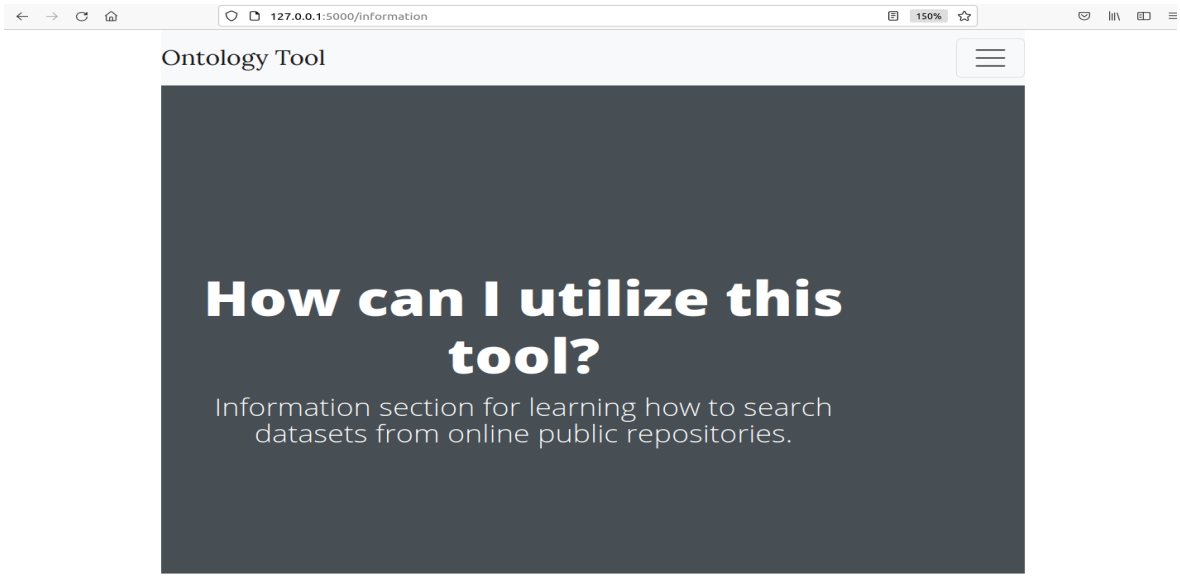

Figura C.11: Título de la página tutorial

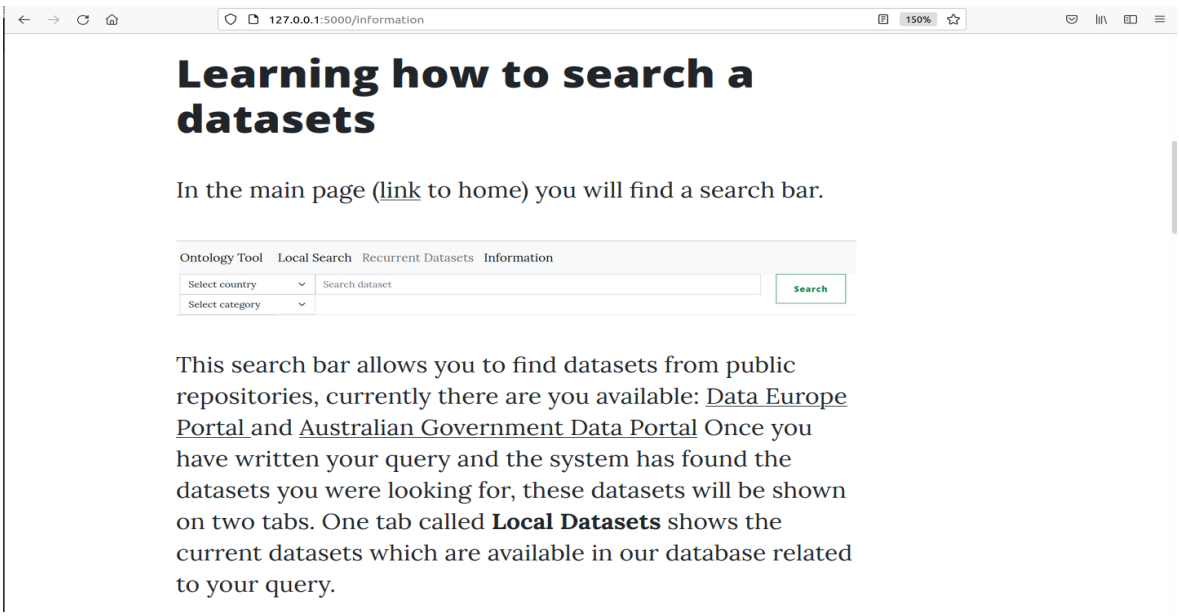

Figura C.12: Explicación página principal, búsqueda local

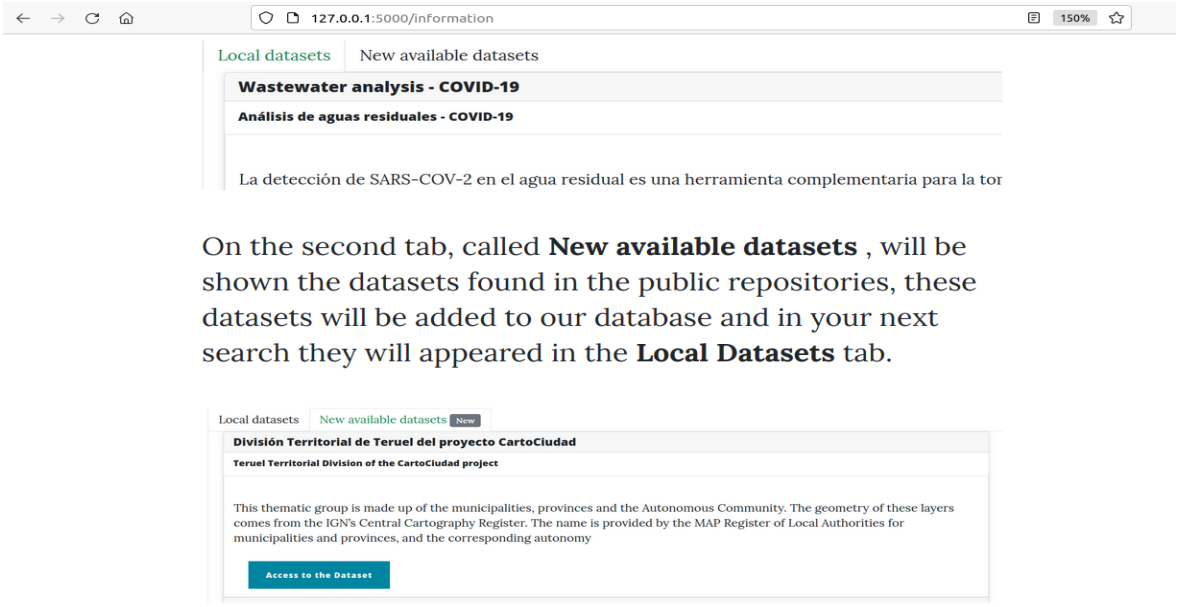

Figura C.13: Explicación página principal, búsqueda repositorios online

Figura C.14: Explicación búsqueda avanzada

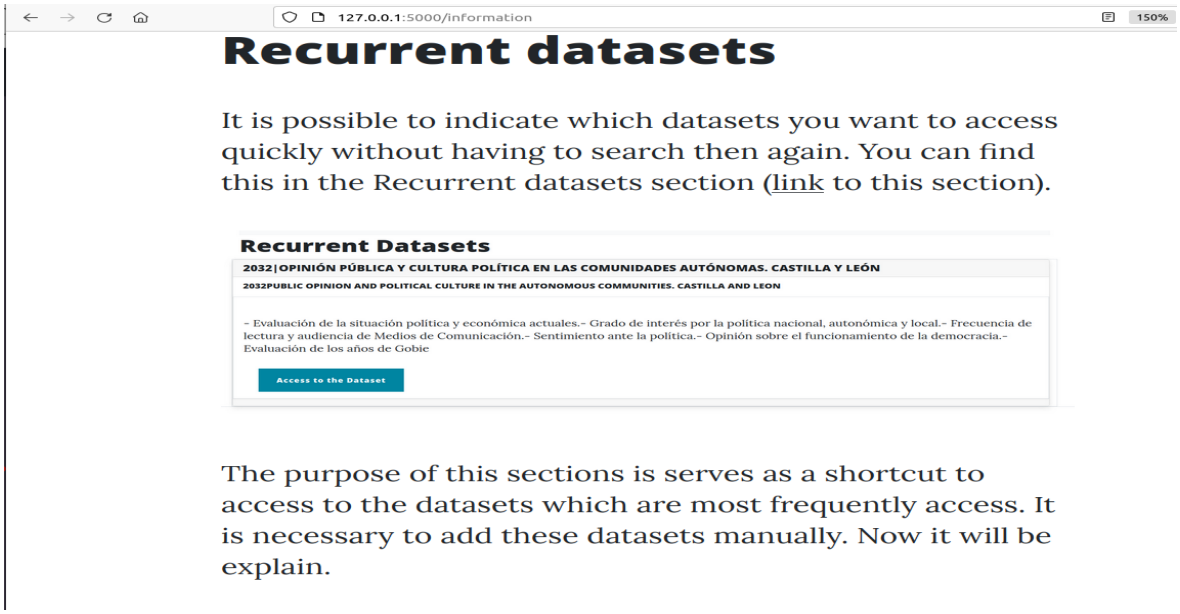

Figura C.15: Explicación datos recurrentes, parte 1

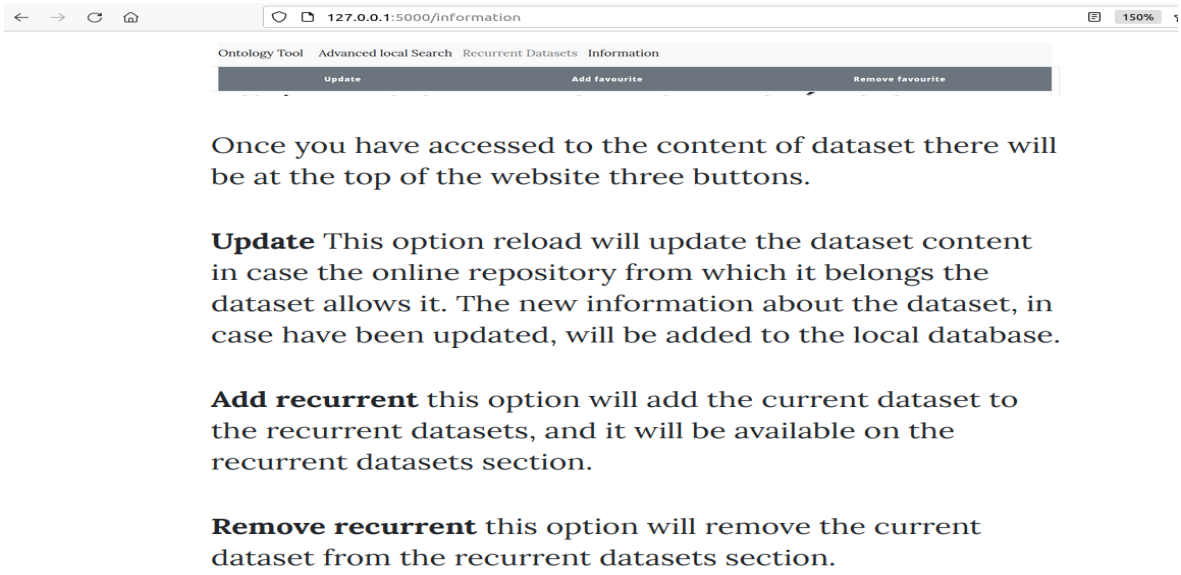

Figura C.16: Explicación datos recurrentes, parte 2

C.2.7. Resultado de la búsqueda

En este apartado se muestra la interfaz que contiene la información de un conjunto de datos.

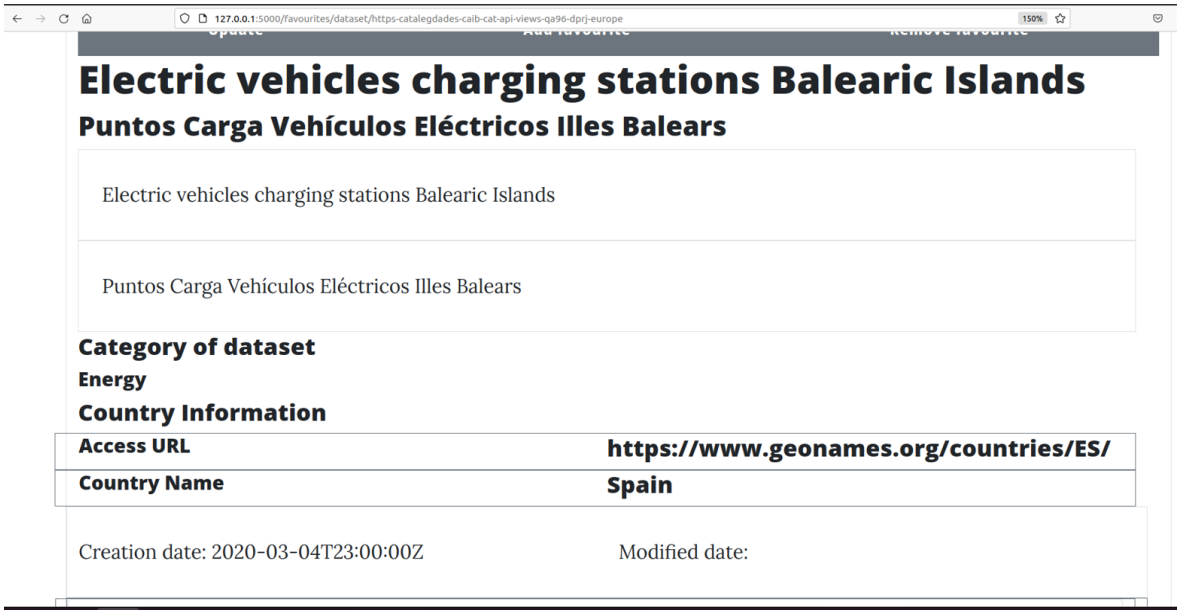

Figura C.17: Conjunto de datos resultado parte 1

Figura C.18: Conjunto de datos resultado parte 2

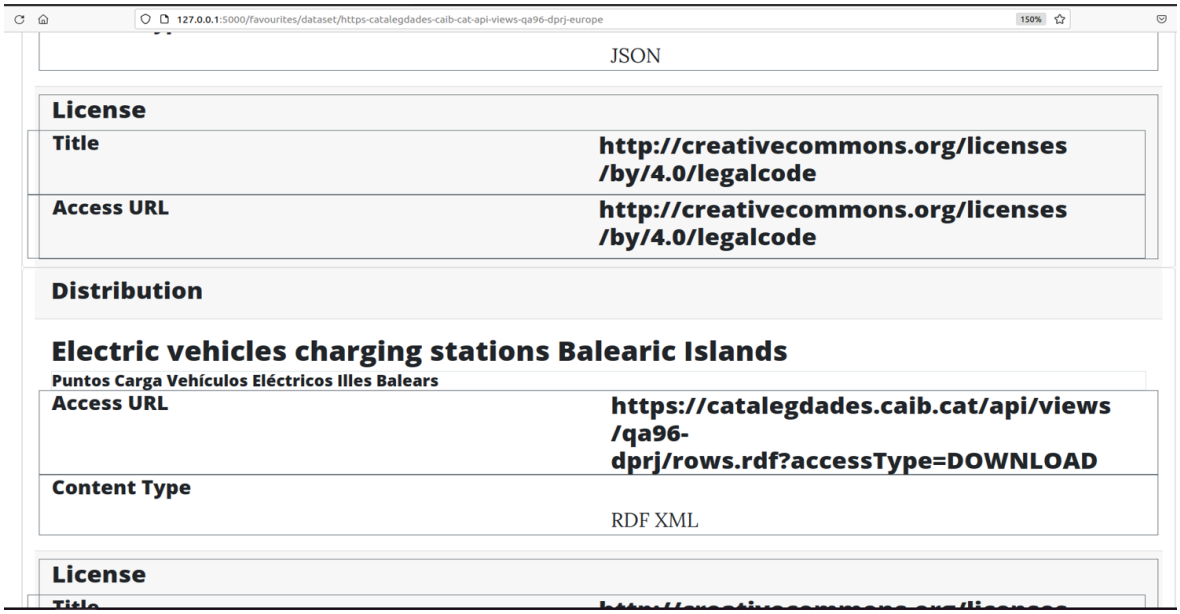

Figura C.19: Conjunto de datos resultado parte 3

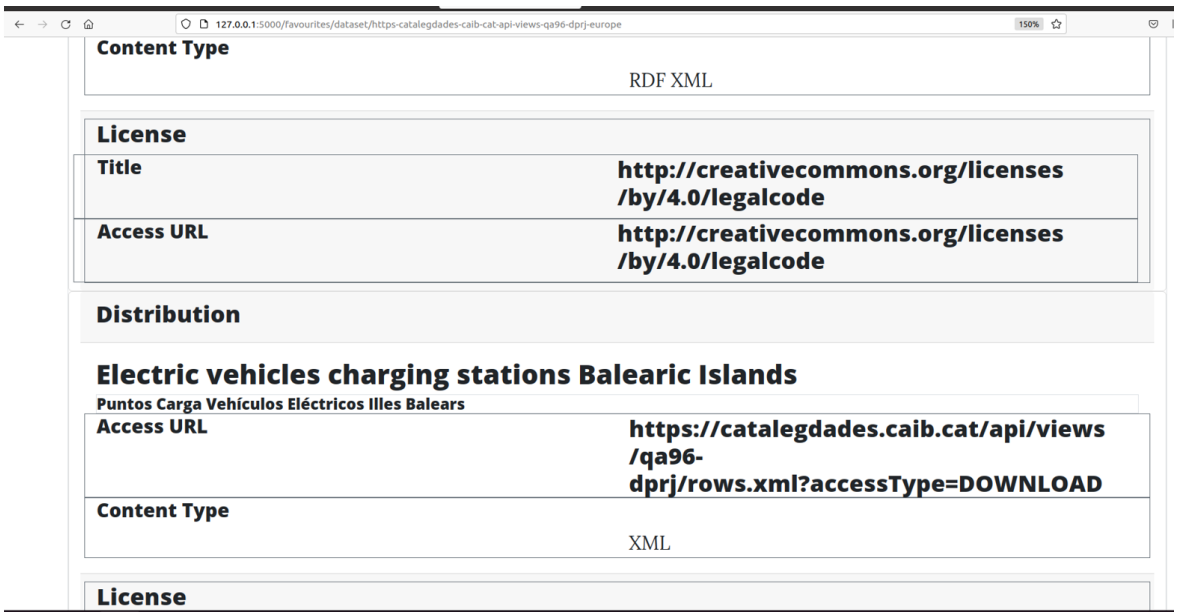

Figura C.20: Conjunto de datos resultado parte 4

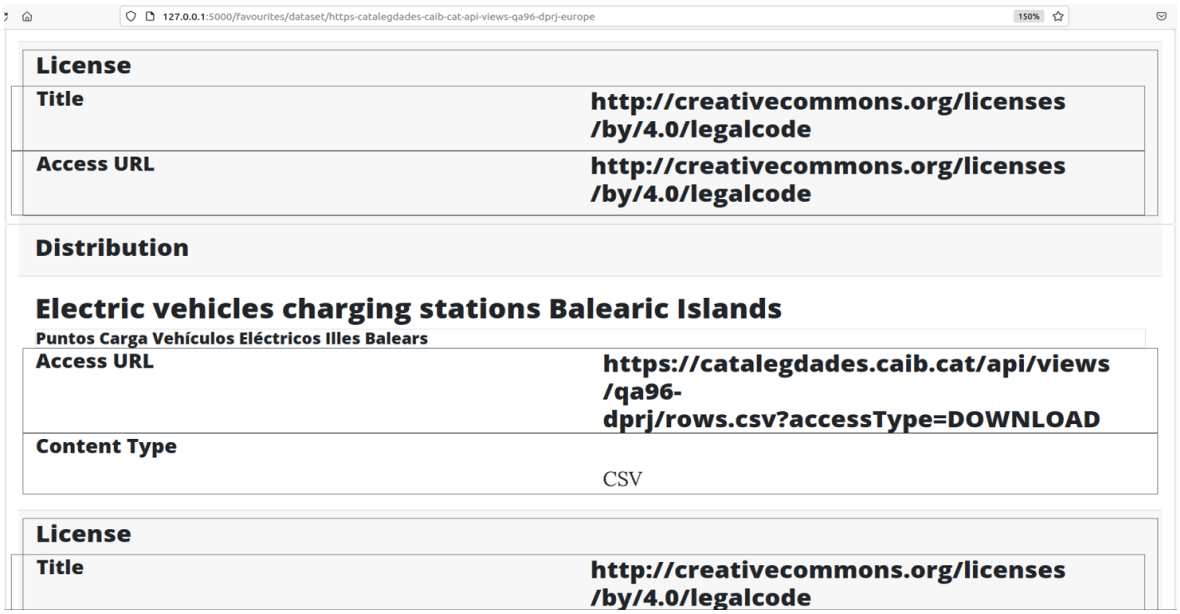

Figura C.21: Conjunto de datos resultado parte 5

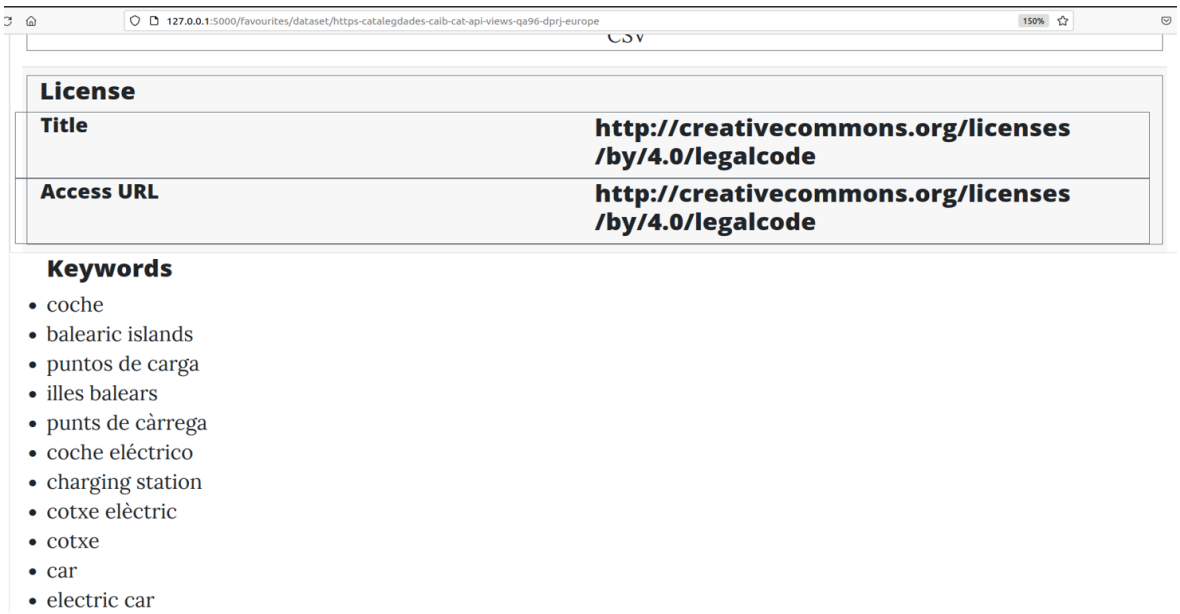

Figura C.22: Conjunto de datos resultado parte 6

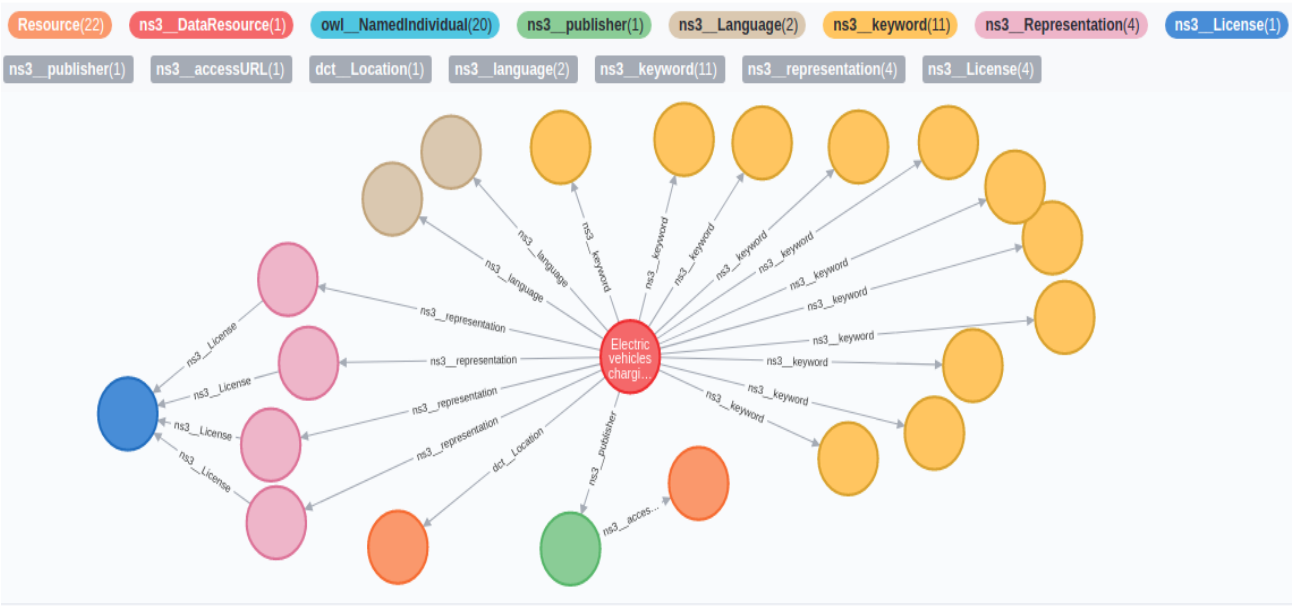

Figura C.23: Conjunto de datos en base local Neo4j

C.3. Diagramas de clase Backend

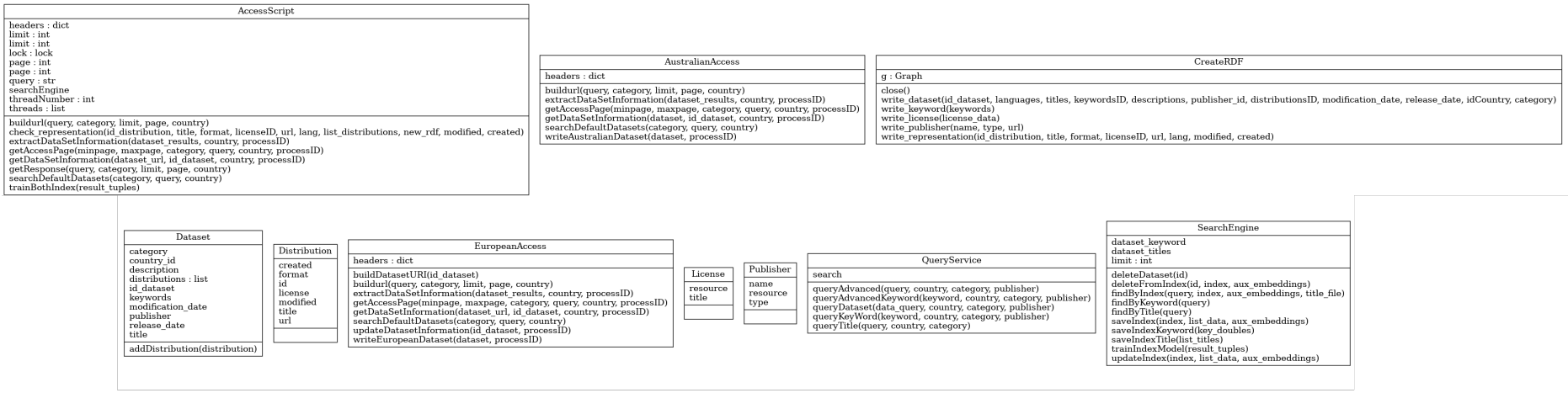

Figura C.24: Diagrama de clases

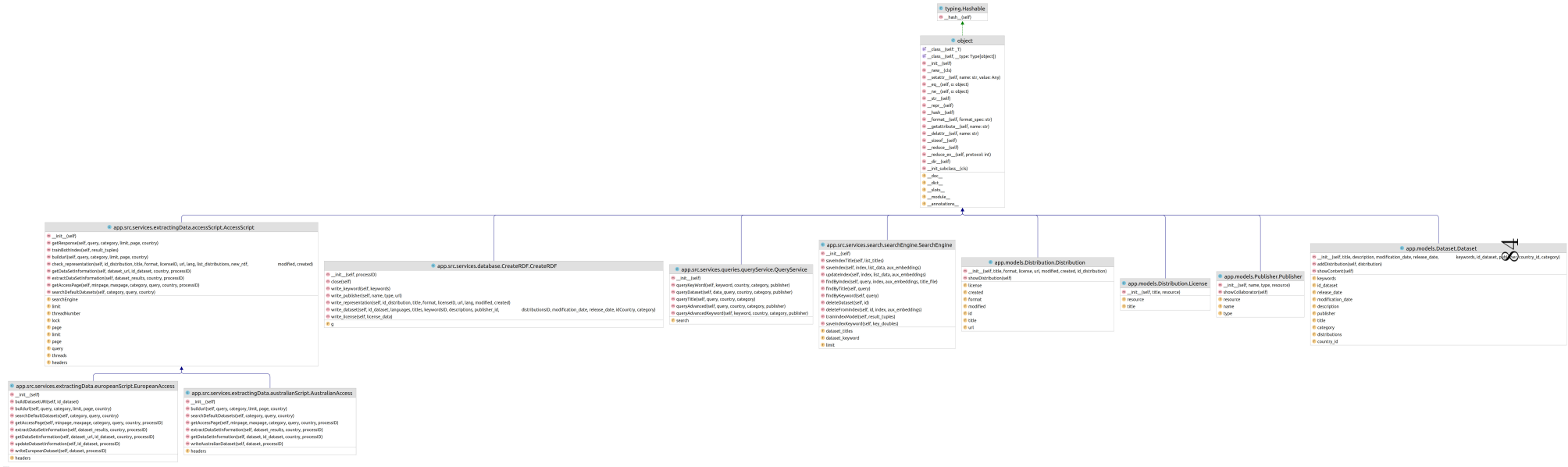

Figura C.25: Diagrama de paquetes y clases más detallado

Anexos D

Anexo 4

D.1. Repositorios públicos

D.1.1. data.europa.eu

Data.europa.eu[38] se trata de el portal de datos público donde las instituciones europeas publican sus conjuntos de datos públicos. Es uno de los principales portales de datos de Europa y esta es una de las razones por las que se ha seleccionado. Su información esta disponible en múltiples idiomas, en este proyecto se han guardado los conjuntos de datos que estuvieran en español y en inglés. En el caso de haber conjuntos de datos exclusivamente en español ha sido necesario traducirlos al inglés para poder utilizar la biblioteca de procesamiento de lenguaje natural TXTAI, ya que esta no procesa el español.

D.1.2. data.gov.au

Data.gov.au[39] se trata del portal público de conjuntos de datos del gobierno australiano. Se ha elegido este repositorio porque proporciona conjuntos de datos en inglés y por rápida incorporación a la aplicación debido a su similitud con el portal europeo de datos. El único inconveniente de este portal es que se quiere el uso de la biblioteca TXTAI para analizar el contenido del conjunto de datos y poder clasificarlo en una categoría, ya que no tienen ninguna categoría asociada.

D.2. Conjunto de datos para validación

El junto de datos utilizado en la Subsección 5.1.1 se encuentra disponible en el siguiente enlace [70], aporta información sobre los puntos de carga de vehículos eléctricos en Islas las Baleares.

[38]https://data.europa.eu/es
[39]https://data.gov.au/

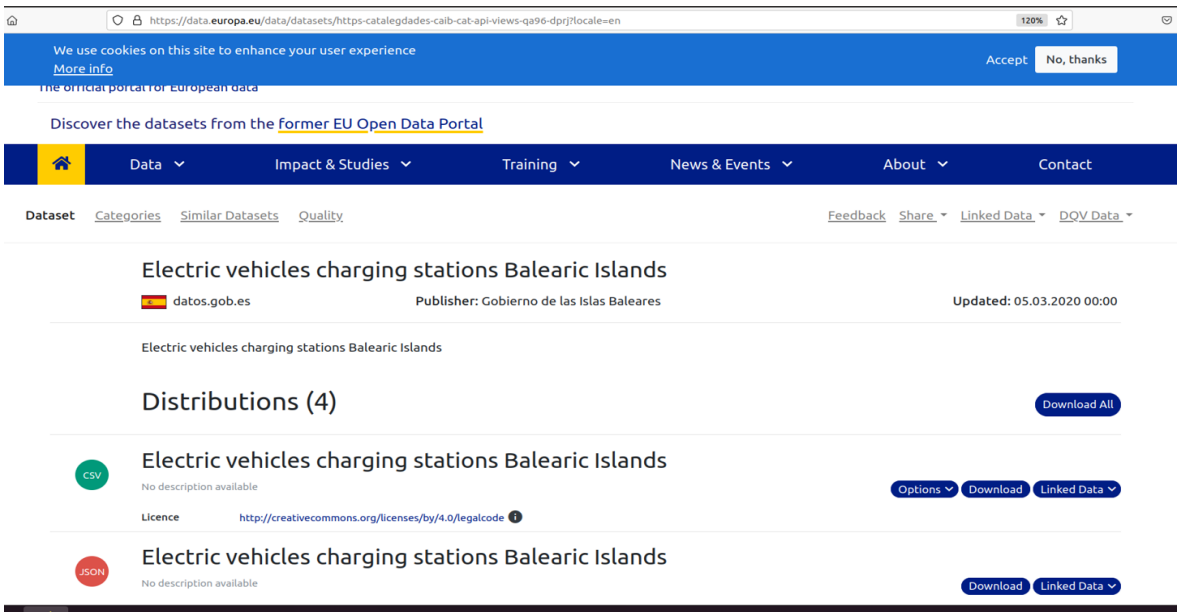

Figura D.1: Conjunto de datos elegido como ejemplo

D.3. Conjunto de datos validación con clave

Los datos de prueba para la validación de la búsqueda por clave (Subsección 5.1.2) se han utilizado dos conjuntos de datos disponibles en data.gov.au de temática relacionada con el océano. Estos conjuntos de datos se encuentran disponibles en [71] y en [72].

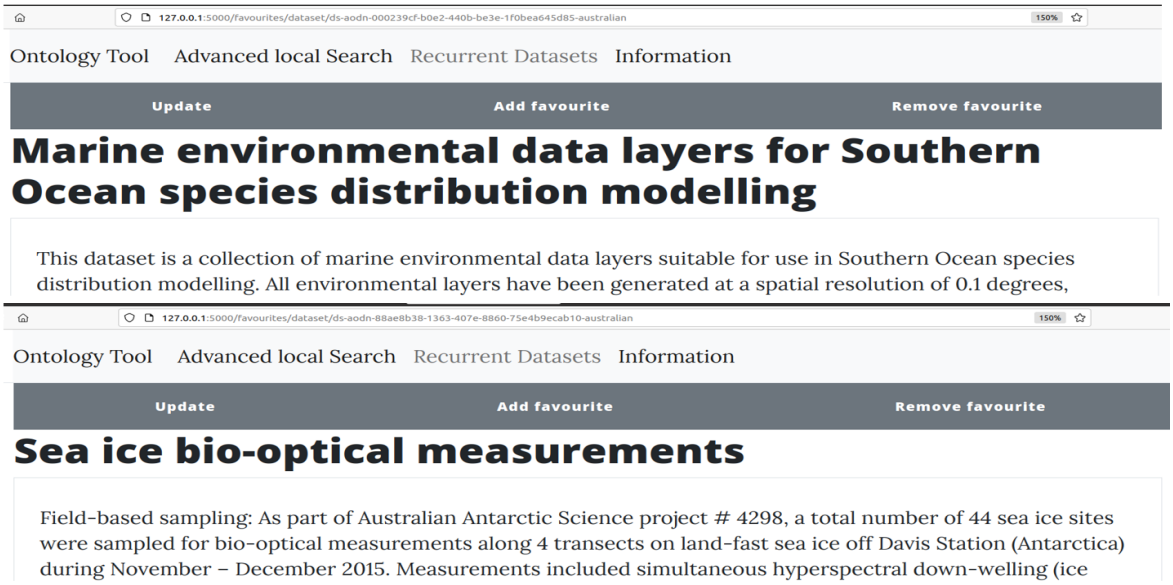

Figura D.2: Conjuntos de datos a buscar

D.4. Palabras clave ejemplo búsqueda

Palabras clave conjunto de datos
chlorophyll
polar
amd
ocean chemistry
models
earth science
environmental layers
pigments
oceans
ceos
density
water temperature
ocean temperature
continent
southern ocean
geographic region
salinity
ocean
marine environment monitoring
computer , computer
au
antarctica, antarctica
carbon

Tabla D.1: Palabras clave Marine Environmental Data Layer

Palabras clave conjunto de datos
polar
hplc
ceos
crc
ace
biological classification
high-performance liquid chromatograph
spectroradiometers
au
cryosphere
sea ice , sea ice
microalgae
ice cores
fast ice
geographic region
earth science
amd
oceans
ice algae
field investigation
bio-optics
plants

Tabla D.2: Palabras clave Sea ice bio-optical measurements

D.5. Resultado búsqueda avanzada

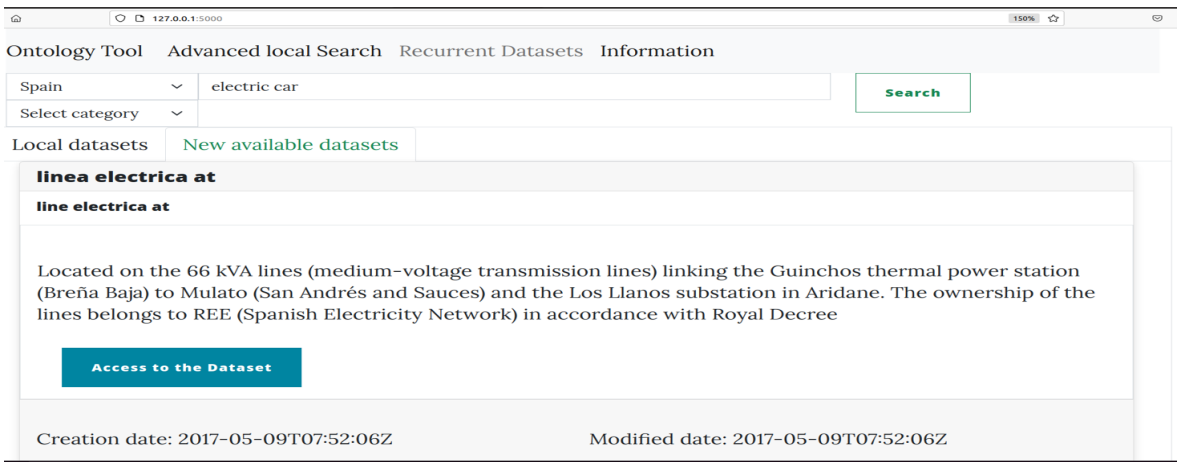

Figura D.3: Resultados búsqueda *electric car* en España, parte 1

Figura D.4: Resultados búsqueda *electric car* en España, parte 2

D.6. Resultado búsqueda por clave

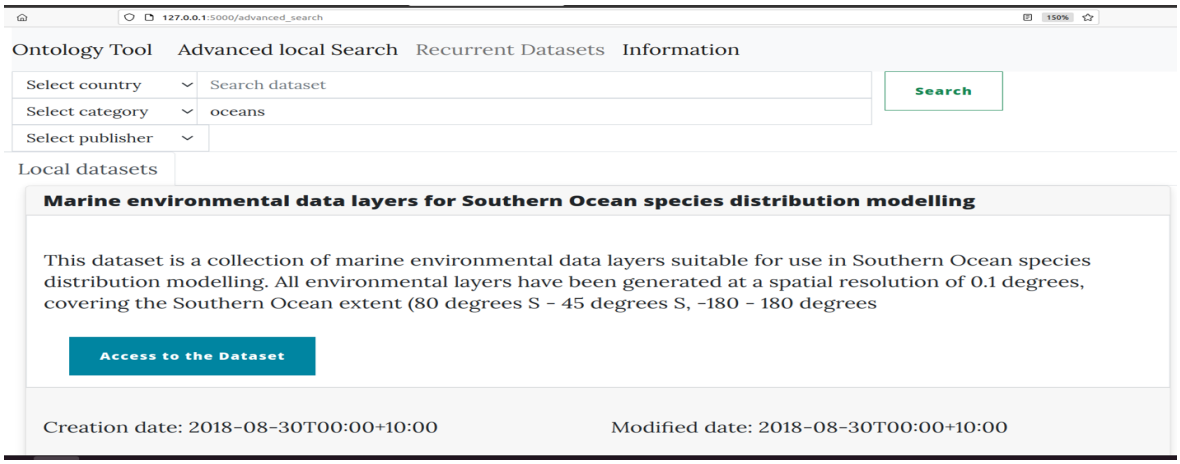

Figura D.5: Búsqueda por palabra clave, parte 1

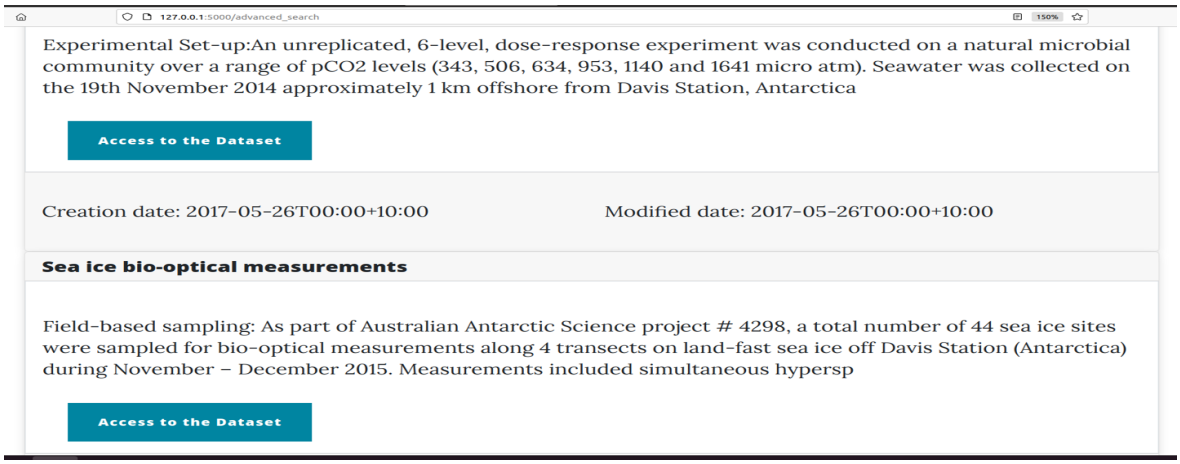

Figura D.6: Búsqueda por palabra clave, parte 2

D.7. Control de esfuerzo

Fecha	Descripción	Horas
08/02/2021	Búsqueda y aprendizaje de información sobre OWL	4
10/02/2021	Completar documentación para beca en el ITA	1
11/02/2021	Estudio apuntes recuperación información	4
17/02/2021	Estudiar tesis proporcionada por tutora (véase en [])	2
18/02/2021	Ver tutoriales sobre OWL	1
19/02/2021	Reunión con coordinadores	1
20/02/2021	Búsqueda y aprendizaje de información sobre DCAT	4
25/02/2021	Búsqueda tecnologías crawling	5
26/02/2021	Reunión con coordinadores	1
06/03/2021	Organizar documentación y comparar crawlers	6
11/03/2021	Leer documentación de Scrapy	4
12/03/2021	Reunión con coordinadores	1
14/03/2021	Configurar crawler de Scrapy	4
18/03/2021	Crear repositorio, instalar plugins scrapy: Scrapy-Selenium, Scrapy-Splash	4
19/03/2021	Reunión con coordinadores	1
22/03/2021	Crear scripts en Python para acceder a Aragon Opden Data	4
23/03/2021	Encontrar la forma de representar JavaScript en el sitio web	4
04/04/2021	Instalación de Splash para hacer clic en los botones que aparecen en el sitio web del portal europeo de datos abiertos	4
04/05/2021	Intentar reducir el tiempo de Splash clicando botones	4

09/05/2021	Reunión con coordinadores	1
14/04/2021	Termniar los Scrips para acceder a Data Europa y Aragon Open Data	4
19/04/2021	Crear scripts para acceder a datos.gob.es	3
20/04/2021	Limpiar código, corregir errores y fusionar clases de conjuntos de datos	4
21/04/2021	Reescribir código	3
22/04/2021	Crear y actualizar documentación en Overleaf	4
29/04/2021	Escribir documentación en Overleaf	4
30/04/2021	Reunión con coordinadores	1
04/05/2021	Comparar bases de datos orientadas a grafos	5
05/05/2021	Leer documentación e instalar Neo4j	4
06/05/2021	Leer documentación sobre Neosemantic y Neo4j	3
18/05/2021	Instalar y configurar Neo4j	3
20/05/2021	Leer documentación sobre la ontología	2
28/05/2021	Reunión con coordinadores	1
31/05/2021	Poblar la base de datos utilizando fichero RDF	6
01/06/2021	Reunión con coordinadores	2
02/06/2021	Script para poblar datos de Aragon Open data	3
03/06/2021	Script para poblar datos de datos.gob.es	3
04/06/2021	Arreglar error script European Portal	3
05/06/2021	Tutoriales Flask y Django	2
06/06/2021	Arreglar bugs en la configuración Flask	4
07/06/2021	Buscar bibliotecas para RDF, Neo4j y Flask	4
08/06/2021	Avanzar con la implementación de RDF y Neo4j	4

09/06/2021	Importar de forma automática los datasets, solucionar problemas	4
10/06/2021	Avanzar con la estandarización e importación de datasets	7
11/06/2021	Reunión con coordinadores	2
12/06/2021	Actualizar proyecto a Django y solucionar errores con las keyword	4
13/06/2021	Deshacer y volver al uso de Flask	4
17/06/2021	Testear búsquedas con TXTAI	4
18/06/2021	Actualizar requisitos, leer documentación TXTAI	4
20/06/2021	Intentar solucionar errores de la búsqueda semántica	5
21/06/2021	Tests para búsqueda semánticas, usar bibliotecas para traducción	5
22/06/2021	Agregar Flask con Jquery, ajax y bootstrap	3
23/06/2021	Avanzar con el desarrollo de la interfaz	4
25/06/2021	Reunión con coordinadores	2
26/06/2021	Avanzar con el desarrollo de la interfaz	4
28/06/2021	Diseñar interfaz con bootstrap, prepara mockups	3
29/06/2021	Comenzar a integrar backend con la interfaz	4
30/06/2021	Avanzar con la integración de backend y frontend	4
06/07/2021	Reunión tutores e implementar búsqueda semántica	4
07/07/2021	Implementar búsqueda semántica	4
08/07/2021	Integrar interfaz con búsqueda semántica	4
09/07/2021	Duplicar búsqueda, en local y en repositorios	4
17/07/2021	Mejorar interfaz	3
18/07/2021	Terminar primera versión funcional de la herramienta	5

19/07/2021	Desplegar en docker, solucionando sus errores	11
20/07/2021	Mejorar interfaz y solucionar fallos	5
21/07/2021	Arreglar fallos y terminar segunda versión	4
22/07/2021	Estudiar búsqueda semántica	4
23/07/2021	Estudiar procesado de lenguaje Natural, word embedding	4
24/07/2021	Estudiar y entender Transformers	6
25/07/2021	Traducir memoria de inglés a español	4
26/07/2021	Capítulo 2, estado del ARTE, crawlers	4
27/07/2021	Capítulo 2, estado del ARTE,bases de datos	5
28/07/2021	Capítulo 2, estado del ARTE, TXTAI	3
31/07/2021	Memoria Ontología IDS	2
01/08/2021	Apartado Información en la herramienta, desarrollo web	4
05/08/2021	Modelo de datos, capítulo 4 de la memoria	4
06/08/2021	Capítulo 4 de la memoria, diagramas	5
07/08/2021	Terminar capítulo 4 de la memoria	3
09/08/2021	Documentar código y modificar script	3
12/08/2021	Comenzar apartado 5 de la memoria	4
13/08/2021	Avanzar apartado 5 de la memoria	3
14/08/2021	Tomar capturas de pantalla para la memoria	3
15/08/2021	Pruebas de despliegue con Docker para revisar cambios	4
18/08/2021	Apartado de Introducción y casi todas las conclusiones	4
23/08/2021	Correcciones de la memoria	2
24/08/2021	Terminar correcciones de la memoria	2
25/08/2021	Resumen y agradecimientos	2

26/08/2021	Solucionar error detectado con datos de data.gov.au	4
27/08/2021	Adaptar memoria tras haber solucionado el error	4
28/08/2021	Ajustar imágenes y solucionar error de formato	8
29/08/2021	Arreglar fallo despliegue en Docker	1

Tabla D.3: Control de esfuerzo

Anexos E

Anexo 5

E.1. Extracción de información mediante *web crawling* y *web scraping*

Inicialmente se analizó la posibilidad de desarrollar un *crawler*, con el propósito de extraer los datos de los repositorios, y con ello diversas tecnologías relacionadas, aunque finalmente fue descartado.

En esta sección se explicará brevemente qué son los procesos de *crawling* y *scraping*. A continuación se concretará qué tecnología decidió utilizarse antes de declinar la idea de utilizar un *crawler* y se justificarán los motivos de esto. Finalmente se detallará cómo se resuelve el problema de la extracción de datos sin el uso de *crawler*. En Sección E.2 se adjunta un estudio de tecnologías de *web crawling* y *web scraping*.

E.1.1. Web crawling

Web crawling es un proceso por el cual un programa, generalmente conocido como araña o *crawler*, inspecciona y navega a través de páginas web. Cuando un *crawler* accede a una página web explora su contenido, lo descargará y sigue los enlaces que esta contiene para continuar con el proceso.

E.1.2. Web scraping

Web scraping es un proceso de análisis de datos estructurados extraídos de una página web para transformarlos en datos no estructurados. Generalmente la tarea de *scraping* es llevada a cabo por un programa que se encarga de analizar y parsear código HTML del que extraer información y la estructura para procesarla o almacenarla en una base de datos.

E.1.3. Tecnología crawler analizada

Las características y requisitos que ha de cumplir la tecnología elegida para este proyecto se encuentran detallados en la Subsección E.2.6. La tecnología elegida ha sido Scrapy junto con el *plugin* **Splash-Scrapy**[40] ya que para acceder a los conjuntos de datos proporcionados por los repositorios de datos públicos es necesario cargar y renderizar JavaScript, lenguaje de programación interpretado muy popular para desarrollo web [76].

Splash-Scrapy es un *plugin* desarrollado por la comunidad que permite acoplar a **Scrapy** el servicio **Splash** que permite procesar el contenido JavaScript de la Web. Aunque a pesar de configurar y desplegar un *crawler* funcional capaz de descargar el HTML de los repositorios de datos públicos tuvo que descartarse la idea del uso de un *crawler*. El motivo por el cual no se ha utilizado ninguna tecnología *crawler* se explica a continuación.

E.1.4. Inconvenientes uso de crawler

Un primer inconveniente identificado fue que eran necesarios 5 segundos para cargar la página principal, debido a que tenía que renderizar su código JavaScript. A esta espera debía de añadirse el tiempo de carga de cada página dedicada a cada conjunto de datos. Asimismo, había que añadirle el tiempo que se tarda en emular el comportamiento de un usuario para pulsar los botones o el simplemente emular hacer clic para cargar la página del conjunto de datos.

El segundo inconveniente encontrado era la cantidad de botones que son necesarios para acceder al contenido de los conjuntos de datos. Las distintas representaciones de los conjuntos de datos y la información del publicador cuentan cada una con un botón que desvela el contenido. En la siguiente imagen se toma como ejemplo el European Data Portal[41] donde observamos la cantidad de botones que hay para acceder a las diferentes distribuciones de un conjunto de datos (ver ejemplo en la Figura A.1).

Otro inconveniente asociado a los botones era la falta de identificadores únicos que diferenciaran a cada uno de los botones. No hay forma de diferenciar a los botones que pertenecen a una misma clase (ver en ejemplo en la Figura A.2). A continuación, se muestra a modo de ejemplo un conjunto de datos alemán con dos botones pertenecientes a distribuciones diferentes de un mismo conjunto de datos.

[40] https://github.com/scrapy-plugins/scrapy-splash
[41] https://data.europa.eu/es

Estos dos botones pertenecen a la misma clase, pero carecen de un identificador único que los diferencie, de tal forma que no se controla a qué botón se esta haciendo clic exactamente. Además, cada conjunto de datos tiene un número indefinido de distribuciones con dos botones cada una.

Debido a todos estos inconvenientes, la cantidad de tiempo que es necesario para extraer un sólo conjunto de datos incrementó excesivamente, haciendo muy lento el proceso de extracción ya que el objetivo del proyecto requiere la extracción de cientos e incluso miles de conjuntos de datos. No sólo el tiempo de renderizado y de pulsado de los botones era un problema, sino que también el hecho de no poder identificar individualmente cada uno de los botones. Debido a todos estos inconvenientes fue necesario pensar en otra forma de acceder y extraer los conjuntos de datos.

E.1.5. Solución alternativa

Utilizando las herramientas de desarrollador del navegador **Mozilla Firefox** es posible observar qué peticiones está llevando a cabo el portal web. En el apartado de Red es posible visualizar el tráfico que genera esta plataforma y localizar la petición que esta extrayendo la información que necesitamos (ver ejemplo en la Figura A.4).

Puede observarse que en una petición la aplicación web solicita los conjuntos de datos junto a su información para incrustarla en la página de resultado. La respuesta a la petición devuelve en formato **JSON** los conjuntos de datos encontrados. Esta es una petición que puede realizarse desde un dominio que no sea el de European Data Portal ya que no está protegida, en la cabecera de la petición no hay ningún atributo que restrinja su respuesta. Esto es algo común a los repositorios de datos públicos, como puede observarse esto mismo sucede en Aragón Open Data[42] (ver ejemplo en Figura A.5).

La descarga de estos ficheros JSON toma mucho menos tiempo que renderizar la página del conjunto de datos y no acarrea ninguno de los problemas explicados anteriormente. Debido a esto finalmente se ha optado por solicitar los conjuntos de datos a través de peticiones HTTP (Protocolo de Transferencia de Hipertexto). No es una solución universal, en el caso de que el portal no disponga de una API, pública o no, de la que extraer la información habrá que utilizar tecnologías *crawler* para obtener los datos.

[42]https://opendata.aragon.es/

E.2. Estado del arte tecnologías *web scraping* y *crawling*

El objetivo de este apartado es realizar una breve introducción a estas tecnologías y compararlas para ver cuál es la que me mejor se adapta al proyecto. Esta sección toma como referencia la comparación de tecnologías de *crawling* realizada en el portal web **Scrape Hero** (véase en [41]).

E.2.1. Apache Nutch

Apache Nutch[43], se trata un *crawler* web desarrollado por **Apache Software Foundation**. Este es un proyecto de código abierto con licencia **Apache License 2.0**. No es una biblioteca ni un *framework* para desarrollo de *crawler* sino un programa en si mismo quer permite acceder y parsear miles de URLs. Actualmente pueden encontrarse dos versiones de **Apache Nutch**, estas son **Nutch 1.x** y **Nutch 2.x**. **Nutch 1.x** cuenta actualmente con soporte técnico ya que se trata de un proyecto activo y su última actualización fue lanzada el 24 de junio de 2021. Por el contrario **Nutch 2.x** es un proyecto inactivo, al tratarse de un proyecto inactivo [29] esta versión será obviada y a partir de ahora nos referiremos a **Nutch 1.x** simplemente como **Nutch**.

Nutch se encuentra completamente desarrollado en Java y presenta una arquitectura modular que permite a los usuarios crear *plugins* y *addons* para conectar este *crawler* con otras plataformas o proyectos [31]. Puede escalar dinámicamente debido a que depende de la estructura de datos de **Hadoop** y hace uso del *framework* distribuido de **Hadoop**. Este *crawler* es comúnmente utilizado para construir índices de búsqueda y puede implementarse sobre estos índices que genera si se combina con proyectos abiertos como **ElastiscSearch** [32] o **Apache Solr**.

Pros	Contras
Arquitectura modular, muy personalizable	No aporta web scraping
Escalable dinámicamente con Hadoop	No es ideal para latencias bajas
Búsqueda utilizando ElasticSearch	Díficil de configurar
Procesa y parsea miles de URLs	Tiene una gran curva de aprendizaje
Comunidad activa	Herramienta muy pesada
	Operaciones pueden llevar mucho tiempo

Tabla E.1: Cualidades de Apache Nutch

[43] http://nutch.apache.org/

E.2.2. Stormcrawler

Stormcrawler[44] ofrece una colección de herramienta y recursos de código abierto para realizar *web crawling* en **Apache Storm** [45]. **Apache Storm** es un sistema de computación distribuido de código abierto desarrollado por la **Fundación Apache**. Permite a sus usuarios crear web *crawlers* para latencias bajas en sistemas distribuidos. Su licencia de código abierto es **Apache License 2.0** y esta principalmente escrito en Java [34].

Stormcrawler puede conectarse a otros proyectos de la **Fundación Apache**, de esta forma por ejemplo puede añadirsele un motor de búsqueda si se conecta con **ElasticSearch** [37] o **Apache Solr**, anteriormente mencionados. Este *crawler* esta recomendado para proyectos a gran escala. Actualmente se encuentra activo y con soporte siendo publicada su última versión el 20 de julio de 2020 [38].

Pros	Contras
Apropiado para proyecto a gran escala	Gran curva de aprendizaje
Búsqueda implementada con ElasticSearch	No hay mucha documentación
Bueno para latencias bajas	Difícil de configurar
Procesa y parsea miles de URLs	
Fácil de ampliar	

Tabla E.2: Cualidades de Stormcrawler

E.2.3. *crawler*4j

***crawler*4j**[46] se trata de un *crawler* de código abierto para Java. ***crawler*4j** proporciona una API para configurar y crear *crawlers* para Java [42]. Permite realizar *web scraping* aunque han de añadirse bibliotecas específicas de *scraping* como **Jsoup**. Permite *multi-threading* para escalar, aunque sólo trabaja sobre una sola máquina. Se trata de un proyecto activo aunque su última actualización fue publicada en 2018 y su última *issue* cerrada en **GitHub** fue en octubre de 2020 [39].

Pros	Contras
Fácil de configurar	No hay mucha documentación
Fácil de complementar y extender	Comunidad inactiva
Apropiado para grandes y pequeños projecto	No web scraping nativo
Procesa y parsea miles de URLs	Última actualización en 2018

[44]http://storm.apache.org/
[45]https://storm.apache.org/
[46]https://github.com/yasserg/*crawler*4j

E.2.4. Scrapy

Scrapy[47] es un *framework* que permite realizar *web scraping* y *web crawling* en Python. Proporciona una API que permite crear rápida y eficientemente *crawlers* sencillos para extraer y procesar datos de páginas web. No escala dinámicamente y cuenta con una compleja arquitectura [44], aunque es ideal para pequeños proyectos ya que su configuración es simple y rápida. Cuenta con mucha documentación oficial y vídeos explicativos en **Youtube** debido a que tiene un amplia comunidad activa [45].

Scrapy cuenta con muchos *plugins* creados por su comunidad que permiten conectar **Scrapy** con otros servicios como **ElasticSearch** o **Splash** [48], el cual se trata ser un servicio para renderizar Javascript ya que nativamente Scrapy no puede procesarlo.

Pros	Cons
Fácil de configurar y utilizar	No procesa JavaScript nativamente
Amplia comunidad de desarrolladores	No esta pensado para sistemas distribuidos
Documentación cuantiosa y detallada	No puede escalar dinámicamente
Comunidad activa	
Muchos plugins desarrollados por su comunidad	

E.2.5. Beautiful Soup

Beautiful Soup[49] es una biblioteca para *web scraping* en Python. Se utilizada para extraer datos de ficheros HTML o XML. Depende del uso de otros tecnologías para obtener los fichero HTML por lo tanto no puede considerarse una tecnología de *crawling*, ya que exclusivamente esta pensada para hacer *scraping* (véase en [46]).

Pros	Contras
Buena biblioteca para scraping	No es una tecnología para crawling
Comunidad activa	

E.2.6. Requisitos del proyecto para crawlers

El contenido explicado en este apartado se simplifica visualizado en Tabla **??**. El principal objetivo del *crawler* en este proyecto es el de acceder y extraer conjuntos de datos de portales públicos de datos abiertos. Debido a esto tecnologías apropiadas

[47] https://docs.scrapy.org/en/latest/
[48] https://splash.readthedocs.io/en/stable/
[49] https://www.crummy.com/software/BeautifulSoup/

para proyecto a gran escala que pueda procesar miles de URLs e indexarlas no serán necesarias.

No va a desplegarse en un sistema distribuido, va a funcionar en una sola máquina por lo tanto no es un requisito que pueda escalar dinámicamente o que funcione sobre sistemas distribuidos. Se necesita una tecnología que permite una descarga y un procesado rápido de los conjuntos de datos extraídos. Además dadas las características de los portales públicos de datos es necesario que la tecnología permita renderizar JavaScript y *multi-threading* para dividir la carga de trabajo y agilizar el proceso de extracción y procesado de datos.

Requisitos
Renderizar JavaScript
Compatible con web semántica
Procesar y extraer conjuntos de datos
Multi-thread
Rápida descarga de ficheros
Web scraping

Anexos F

Anexo 6

F.1. Glosario siglas y abreviaturas

1. **API**. Application Programming Interface o en español Interfaz de Programación de Aplicaciones. Conjunto de subrutinas, funciones y métodos que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción [77].

2. **BASE**. Basically Available, Soft State and Eventual Consistency o en español Disponibilidad Básica, Estado Débil y Consistencia Eventual [78].

   - **Basically Available**. Las bases de datos priorizarán la disponibilidad de sus datos aunque no sean consistentes.
   - **Soft State** La base de datos no se encargará de gestionar la consistencia de los datos.
   - **Eventual Consistency** No atómicamente, pero llegará el momento en el que los datos almacenados llegarán a un estado de consistencia.

3. **BOLT**. Este es un protocolo de red orientado a la conexión que se utiliza para la comunicación cliente-servidor en aplicaciones de bases de datos [79].

4. **Teorema CAP**. Consistency, Availability and Partition Tolerance o en español Consistencia, Disponibilidad y Tolerancia a Particiones. También conocida como **Conjetura de Brewer**, afirma que en sistemas distribuidos es imposible garantizar que se cumplan las propiedades de disponibilidad, consistencia y tolerancia a particiones de forma simultánea. Un sistema no puede asegurar más de dos de estas tres características simultáneamente [9].

   - **Consistency** Cualquier lectura recibe como respuesta la escritura más reciente o un error.

- **Availability** Las peticiones reciben respuestas no errónea, es decir hay datos disponibles aunque no se garantiza sus consistencia.
- **Partition Tolerance**. El sistema sigue en funcionamiento aunque varios de sus nodos hayan sido desconectados o hayan sufrido percances.

5. **RDF**. Resource Description Framework o en español Marco de Descripción de Recursos. Modelo estándar que sirve para proporcionar información descriptiva de los recursos disponibles en la Web, facilitando el intercambio de datos en la Web y la descripción de las relaciones entre los diferentes recursos disponibles (ver más en Subsección 2.1.1).

6. **OWL**. Web Ontology Language. Es un lenguaje declarativo usado en la web para describir ontologías (ver más en Subsección 2.1.2).

7. **RDFS**. Se refiere a **RDF Schema**. Este último es un lenguaje basado en RDF cuya finalidad es la de definir vocabularios para RDF.

8. **NOSQL**. Not Only SQL o No SQL. Se refiere a una amplia clase de sistemas de gestión de bases de datos que difieren del modelo de sistema de gestión de bases de datos relacionales [80].

9. **SQL**. Structured Query Language o en español Lenguaje de Consulta Estructurada. Es un lenguaje de dominio específico utilizado en programación, diseñado para administrar, y recuperar información de sistemas de gestión de bases de datos relacionales [81].

10. **SGBD**. Sistema Gestor de Bases de Datos. Software que permite el almacenamiento, modificación y extracción de los datos almacenados en una base de datos [82].

11. **CYPHER**. Cypher es el lenguaje de consulta de grafos de Neo4j que permite a los usuarios almacenar y retirar datos de la base de datos orientada a grafos [83].

12. **IDSA** International Data Spaces Association o en español Asociación Internacional de Espacios de Datos. Asociación cuyo propósito es el intercambio de datos de forma segura y fiable (ver en Subsección 2.4.4).

13. **PLN**. Procesamiento del Lenguaje Natural. Campo de las ciencias de la computación, de la inteligencia artificial y de la lingüística que estudia las interacciones entre las computadoras y el lenguaje humano [84].

14. **URI** Identificador de Recursos Uniforme. Cadena de caracteres que identifica los recursos de una red de forma unívoca [85].

15. **JSON** JavaScript Object Notation o en español Notación de Objeto de JavaScript. Formato de texto sencillo para el intercambio de datos [86].

16. **Docker**[50] Proyecto de código abierto que automatiza el despliegue de aplicaciones dentro de contenedores de software, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones en múltiples sistemas operativos [87].

17. **Hadoop**[51]. Entorno de trabajo para software, bajo licencia libre, para programar aplicaciones distribuidas que manejen grandes volúmenes de datos [88].

18. **Apache Lucene**[52]. Motor de búsqueda de alto rendimiento y escalable, y tiene multitud de funcionalidades orientadas a la búsqueda.

19. **Apache Solr**[53]. Motor de búsqueda de código abierto escritos en Java y basado en la librería de Java Lucene desarrollado por la Fundación Apache Software.

20. **Elasticsearch**[54]. Motor de búsqueda de código abierto escritos en Java y basado en la librería de Java Lucene desarrollado por Shay Banon.

21. **HTML**. HyperText Markup Language o en español Lenguaje de Marcas de Hipertexto. Lenguaje de marcado que define el significado y la estructura del contenido web [89].

22. **HTTP**. Hypertext Transfer Protocol o en español Protocolo de Transferencia de Hipertexto. Protocolo de la capa de aplicación para la transmisión de documentos hipermedia, como HTML [90].

23. **Python**. Python[55] es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código.

24. **JavaScript**. Lenguaje de programación ligero, interpretado o de secuencias de comandos que te permite implementar funciones complejas en páginas web [91].

[50] https://www.docker.com/
[51] https://hadoop.apache.org/
[52] https://lucene.apache.org/core/
[53] https://solr.apache.org/
[54] https://www.elastic.co/es/
[55] https://www.python.org/

25. **Flask**. Flask es un framework minimalista escrito en Python que permite crear aplicaciones web rápidamente y con un mínimo número de líneas de código.

26. **XML**. Extensible Markup Language o en español Lenguaje de Marcado Extensible. Es un lenguaje de marcado que define un conjunto de reglas para la codificación de documentos [92].

Lista de Figuras

2.1. Ejemplos de tripletas RDF 14

3.1. Diagrama de casos de uso 23
3.2. Diagrama alto nivel 24
3.3. Diagrama de despliegue 25
3.4. Diagrama entidad-relación 25
3.5. Mapa de navegación 27

4.1. Formato tupla 31
4.2. Ejemplo conjunto de datos en Neo4j 33
4.3. Búsqueda en base de datos 34

5.1. Base de datos sin poblar 38
5.2. Búsqueda repositorio *online electric car* 38
5.3. Base de datos poblada 39
5.4. Conjunto de datos en base de datos 39
5.5. Búsqueda avanzada *parking* 40
5.6. Búsqueda conjunto de datos por título interpretado 41

A.1. Captura de pantalla Data Europa 55
A.2. Ejemplo conjunto de datos en Data Europa parte 1 56
A.3. Ejemplo conjunto de datos en Data Europa parte 2 56
A.4. Tráfico red Data Europa 56
A.5. Tráfico red Open Data Aragón 57
A.6. Modelo arquitectural Transformer[74] 60

B.1. Diagrama modelo de datos 67
B.2. Pantalla búsqueda avanzada 67
B.3. Pantalla principal 68
B.4. Pantalla conjunto de datos parte 2 68
B.5. Pantalla conjunto de datos parte 1 68

B.6. Pantalla de información sobre la herramienta 69

C.1. Diagrama de paquetes Frontend 71
C.2. Pantalla inicial 72
C.3. Página inicial antes de buscar 72
C.4. Página inicial cuando falla la búsqueda local 73
C.5. Búsqueda Avanzada 73
C.6. Elección de publicador 74
C.7. Conjunto de datos que comparten publicador 74
C.8. Resultado palabra clave *ocean chemistry* 75
C.9. Resultado palabra clave **parking** 75
C.10.Conjuntos de datos recurrentes 76
C.11.Título de la página tutorial 76
C.12.Explicación página principal, búsqueda local 77
C.13.Explicación página principal, búsqueda repositorios online 77
C.14.Explicación búsqueda avanzada 78
C.15.Explicación datos recurrentes, parte 1 78
C.16.Explicación datos recurrentes, parte 2 79
C.17.Conjunto de datos resultado parte 1 79
C.18.Conjunto de datos resultado parte 2 80
C.19.Conjunto de datos resultado parte 3 80
C.20.Conjunto de datos resultado parte 4 81
C.21.Conjunto de datos resultado parte 5 81
C.22.Conjunto de datos resultado parte 6 82
C.23.Conjunto de datos en base local Neo4j 82
C.24.Diagrama de clases 83
C.25.Diagrama de paquetes y clases más detallado 84

D.1. Conjunto de datos elegido como ejemplo 86
D.2. Conjuntos de datos a buscar 86
D.3. Resultados búsqueda *electric car* en España, parte 1 88
D.4. Resultados búsqueda *electric car* en España, parte 2 89
D.5. Búsqueda por palabra clave, parte 1 89
D.6. Búsqueda por palabra clave, parte 2 89

Lista de Tablas

3.1. Requisitos funcionales 22
3.2. Requisitos no funcionales 22

D.1. Palabras clave Marine Environmental Data Layer 87
D.2. Palabras clave Sea ice bio-optical measurements 88
D.3. Control de esfuerzo 94

E.1. Cualidades de Apache Nutch 98
E.2. Cualidades de Stormcrawler 99