Facultade de Informática
UNIVERSIDADE DA CORUÑA

TRABALLO FIN DE GRAO
GRAO EN ENXEÑARÍA INFORMÁTICA
MENCIÓNS EN COMPUTACIÓN E ENXEÑARÍA DO SOFTWARE

Sistema para la construcción automatizada y explotación de taxonomías

Estudante: Juan Luis Filgueiras Rilo
Dirección: Álvaro Barreiro García
Javier Parapar López

A Coruña, novembro de 2020.

A mamá, a papá y a todos los que habéis confiado en mí para llegar hasta aquí

Agradecimientos

En primer lugar, a Javier y a Álvaro por esta propuesta tan interesante y desconocida para mí, además del trato y el ánimo constantes a pesar de las complicadas circunstancias actuales. Es un honor haber tenido las puertas abiertas desde el primer momento y haber podido profundizar en esta área, con un proyecto que ha supuesto un reto personal y un gran aporte didáctico.

A mi familia, por su apoyo incondicional y su interés continuo, que indudablemente me han ayudado a estar donde estoy. En especial, a mis padres, a mis tíos y a mi primo, que siempre han sabido valorar mi trabajo incluso cuando yo mismo no lo hice.

A Juli, a Pablo y a Ishwi. A Mario. A Adri. A Edu, Andrés y Antonio, algunos de los compañeros incondicionales que he tenido a lo largo del grado y que, de forma impecable, han hecho que el paso por aquí haya sido un auténtico placer.

A Lía, a Diego, a Math, a Jacobo y Álex y a toda mi compañía de teatro. Más que amigos, son el sustento de mi motivación actual y espero poder seguir contando con ellos de la misma forma.

A Sergio, por su paciencia infinita y por ayudarme en todo lo ajeno a este proyecto, que ha hecho que valga la pena cada segundo de dedicación. Literalmente, has hecho que me sienta en casa, compañero.

Por último, a todos aquellos docentes, desde mi formación más elemental hasta la actual, que se han tomado la molestia en aconsejarme, alentarme, intercambiar perspectivas y hacerme ver que siempre hay una forma de aprender, mejorar y avanzar como persona y como profesional. En especial, a Ramón Vilalta y a Miguel Rubio.

Resumen

Las taxonomías son estructuras de información que ofrecen valor jerárquico en diversas aplicaciones basadas en conocimiento como la comprensión de consultas, la respuesta de preguntas en lenguaje natural y la recomendación personalizada. Sin embargo, a día de hoy, la mayoría de métodos automatizados de construcción de taxonomías son limitados respecto a la poca expresividad que poseen las relaciones entre nodos, lo que provoca que su elaboración se vea supeditada al esfuerzo y coste intensivos, derivados de la construcción manual y la necesidad de conocimiento experto. Este proceso proporciona resultados de suma importancia, pero poco adaptables a los cambios y, raramente completos. De esta situación se deriva la incipiente necesidad de métodos de elaboración de taxonomías automatizados y de calidad, que permitan relaciones de mayor expresividad y ofrezcan más flexibilidad ante cambios.

HiExpan es un «framework» de construcción de ontologías que deduce la relación semántica entre nodos de la misma clase a partir de una taxonomía semilla, la cual se irá expandiendo hasta conseguir un resultado convergente, de forma no-supervisada. El objetivo de este proyecto es adaptar, mejorar y ofrecer una forma de plataforma que permita la construcción y explotación de taxonomías de manera sencilla e interactiva, para poder comprobar la utilidad de los resultados obtenidos.

Para poder lograr la consecución de los anteriores objetivos se ha adoptado una metodología ágil con ciclos iterativos e incrementales, para adaptarse a las posibles necesidades emergentes durante el desarrollo. De esta forma, se han alcanzado los objetivos en forma de incrementos funcionales cada vez más expresivos, con el apoyo de herramientas para el aseguramiento de la calidad, obteniendo un producto fiable, completo y de interés multidisciplinar, con una diversa aplicabilidad en el ámbito de la Recuperación de la Información, e incluso fuera del mismo.

Abstract

Taxonomies are data structures that offer hierarchical value at diverse knowledge-based applications, such as query understanding, question answering and personalized recommendation. However, the vast majority of automatized methods are, nowadays, limited in means of poor expresivity between nodes, causing the process elaboration to be contigent on intensive effort and cost, derived from manual construction and the need of expert knowledge. This process offers results of great value, but poorly adaptative to change and, rarely complete. This situation derives the incipient need of automatized taxonomy construction methods, that allow more expresive relationships and offer more flexibility for changes.

HiExpan is a framework of taxonomy construction that derives the semantic relation between nodes of same semantic class starting from a seed taxonomy, which will be iteratively expanded until having a convergent result, in a non-supervised way. The goal of this proyect is to adapt, improve and offer a platform that allows the construction and exploitation of taxonomies in a easy and interactive way, so its easy to check the usefulness of obtained results.

In order to achieve the previous objectives, an agile methodology is adopted, with iterative and incremental cycles, to ensure adaptability to the possible emerging needs during development. This way, each goal is achieved in the form of a functional increment, adquiring more expresivity as it progresses through the iterations, with the help of development tools for the quality assurance, resulting in a reliable, complete product of multidisciplinar interest, with a diverse aplicability whether in the scope of Information Retrieval or elsewhere.

Palabras clave:

- Recuperación de la Información
- Construcción de Taxonomías
- Extracción de frases
- Expansión jerárquica de árboles
- Motor de búsqueda
- Sistemas de recomendación

Keywords:

- Information Retrieval
- Taxonomy Construction
- Phrase Mining
- Hierarchical Tree Expansion
- Search Engine
- Recommendation Systems

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Inevitablemente, a lo largo de toda la historia del ser humano, la clasificación ha sido una de las necesidades constantes para comprender mejor el mundo que yace a nuestro alrededor. Desde Aristóteles –padre de la biología– desarrollando la primera taxonomía conocida sobre la clasificación de animales (vigente hasta el siglo XVIII), hasta llegar a Charles Darwin, con las categorías precursoras de las ontologías de clasificación de seres vivos de hoy en día, el poder clasificador de estas estructuras es de importancia suma debido a su capacidad de expresar las características comunes entre elementos de un dominio común. A pesar de ser predominantes en el ámbito de la biología, estos árboles de clasificación han ido adquiriendo importancia conforme las ciencias de la computación solicitaban más modelos y estructuras para la organización de la creciente cantidad de datos, proclives a estar ordenados de alguna forma, habitualmente con relaciones de Hiperonimia.

Estas relaciones de hiperonimia son fáciles de identificar, pero el proceso de convertir una jerarquía a partir de relaciones semánticas entre sus miembros determinando aspectos comunes, en ocasiones, no es trivial. Cualquier taxonomía actual posee un volumen de términos desmesurado para su construcción, reorganización, estudio y salvaguardado en márgenes de tiempo aceptables. Asimismo, existe una dificultad añadida derivada de la necesidad de aplicar la ciencia de la taxonomía a todos los dominios que tienen una necesidad de clasificación, y es que, si bien las entidades poseen dominios diversos, las relaciones no son intuitivas en la gran mayoría de los casos. Si se requiere establecer una jerarquía territorial de todos los países del mundo, por ejemplo, será difícil determinar qué relación semántica subyace entre un país y una subdivisión territorial que, irremediablemente, es arbitraria.

Como añadido a la situación presentada, la necesidad de las taxonomías ha ido en aumento, no sólo para la comprensión del estado del arte de diversas ramas científicas, lingüísticas y socio-políticas, sino que cobran utilidad técnica cuando se combinan con la Recuperación de la Información, un ámbito de la investigación que convive muy estrechamente con las relaciones semánticas y el poder expresivo de una unidad de información. Desde la Teoría de

la Información de Shannon, la comunicación es un elemento clave para la obtención de una necesidad de información. Una de las finalidades de esta teoría es la disminución del ruido existente en la comunicación, inherente a los diversos elementos que la componen. Las ontologías aprovechan la expresividad de una entidad –siendo esta expresividad inversamente proporcional a la frecuencia de aparición– y les da un carácter más trascendental, permitiendo abordar problemas complejos relacionados con el lenguaje natural dentro del ámbito tecnológico.

1.1 Motivación

La aplicabilidad de las taxonomías, dentro de este contexto, es muy diversa. Desde el apoyo a la resolución de búsqueda por consultas a través de la comprensión de las mismas hasta el apoyo a sistemas de recomendación, son una parte fundamental del conocimiento de diversos sistemas en prácticamente cualquier dominio. Este hecho las convierte en una opción estratégica para la inversión en esfuerzo, tiempo y coste que supone su construcción, y es por ello que su demanda es alta. El principal problema asociado a ellas es que la labor de construcción es, en esencia, un proceso complejo realizado por expertos humanos y excesivamente extensivo en consumo de tiempo y esfuerzo, proporcionando un resultado poco adaptativo al cambio –debido a la rigidez que las caracteriza– y, muy a menudo, poco completo.

Ante esta problemática, la construcción automatizada de taxonomías se presenta como una potencial solución, dada la situación actual en la que se encuentra la investigación sobre el procesamiento de lenguajes naturales. A día de hoy, las soluciones automatizadas propuestas para la construcción de ontologías son muy básicas, proporcionando resultados sobre relaciones simples de Hiperonimia (relaciones «ES_UN») , cuya semántica se extrae fácilmente mediante sustitución. Por ejemplo, la clasificación de grupos de seres vivos simple se basa en este tipo de relación: El término «gato» posee una relación «ES_UN» con «mamífero», que, a su vez, posee una relación con «vertebrado» –sucesivamente hasta llegar al término «ser_vivo»–. Este tipo de relaciones se identifican con nitidez a través del procesamiento de lenguajes naturales y las restricciones posicionales de una palabra, que le aportan valor semántico. Sin embargo, esta aproximación no es realista para la mayoría de divisiones taxonómicas del mundo real. La relación semántica propuesta es muy inflexible para la forma diversa en la que se organizan los diversos contextos de la realidad y, por otro lado, la generación de ontologías de este modo es muy limitada para satisfacer necesidades específicas del usuario en ámbitos aplicativos con tareas muy específicas.

Esta situación motiva a invertir esfuerzo en la automatización de la construcción de taxonomías guiada por tarea, proporcionada por el usuario, para mejorar el paradigma actual de generación de estas estructuras jerárquicas, en beneficio de cualquier aplicación basada en

conocimiento.

1.2 Objetivos

El objetivo principal consiste en la elaboración de taxonomías a través de una señal del usuario –que representará la tarea sobre la que se guiará el proceso de construcción– y un corpus de entrada sobre el que se extraerán los términos a partir de los cuales se expandirá dicha señal hasta formar una taxonomía completa y ajustada a las necesidades del usuario.

Además, la relación entre los elementos del árbol resultante no estará sujeta a una definición estricta, sino que se obtendrá a partir de la estructura de la entrada del usuario. De esta forma, no sólo se consigue una solución diferente a las ya existentes para la construcción automática (permitiendo ampliar la variedad de dominios sobre los que se puede construir), sino que el ámbito de la aplicabilidad se ajusta a las necesidades del usuario, de forma que se reducen los costes de construcción derivados del conocimiento experto y la clasificación manual.

Como complemento a lo anterior, este proceso puede seguir siendo muy complejo para el usuario objetivo, ya que requerirá amplios conocimientos técnicos y de Recuperación de Información, que no están al alcance de todos los departamentos técnicos de una organización. Es por ello que otra de las motivaciones es permitir que la generación de taxonomías sea accesible, configurable y provea una forma de comprobar los resultados obtenidos y la razón de esos mismos resultados. Si bien el proceso de construcción puede llegar a ser muy complejo y realizarse a través de múltiples pasos para la obtención de toda la información de valor que permita elaborar de forma automática una taxonomía ajustada a las necesidades del ámbito aplicativo, a efectos prácticos no debería ser necesario conocer los detalles de implementación del proceso de construcción, ya que el verdadero objetivo es permitir la configuración a alto nivel del proceso. De esta forma, el usuario sólo requerirá conocer ciertos aspectos del ámbito lingüístico –inevitablemente necesarios para la elaboración de taxonomías– y podrá centrarse en ellos, sin importar el cómo esos aspectos se obtienen y dando más importancia a saber los aspectos que se quieren tener en cuenta para elaborar estas estructuras.

Para poder centrar el foco en lo anterior, es importante que la generación produzca resultados visibles de forma intuitiva (principalmente en su estructura de árbol), independientemente de la representación interna de los datos o de como éstos sean la entrada posterior de otros sistemas para el apoyo y complemento de su base de conocimiento. Relacionado con esto, no bastará con visualizar adecuadamente los resultados, es importante que sean accesibles –exportables por necesidad de uso– y se pueda realizar búsqueda sobre ellos, para poder corroborar que la ontología resultante es representativa de la colección de documentos que se ha proporcionado, suponiendo que esta colección es representativa de la tarea que el usuario

necesita complementar con una jerarquía. La mejor forma de comprobar que estos resultados son representativos, lo más óptimo es habilitar una búsqueda facetada a través de las categorías de clasificación de la ontología.

En cuanto al ámbito no funcional, un aspecto muy importante es la alta disponibilidad, ya que la construcción automática de taxonomías es un proceso muy laborioso que puede tardar mucho tiempo en dar sus frutos. Pese a que la eficiencia de construcción no es un punto clave –debido a que el proceso de construcción manual es mucho más dilatado en el tiempo «per se»–, es importante que las estructuras sean de fácil acceso y estén representadas de forma que su exploración interactiva es rápida para el usuario. Por otro lado, debe haber una gran tolerancia a fallos, para favorecer la gestión de «feedback» con el usuario e informarle sólo de errores que él mismo haya podido cometer, subsanando en la medida de lo posible los errores que puedan surgir durante el complejo proceso de construcción.

1.3 Estructura de la memoria

La estructura de la memoria será la siguiente:

1. **Introducción**: En este apartado se explica el marco contextual del proyecto, la necesidad de la que surge la motivación del mismo, y expresa los objetivos a alcanzar. También da a conocer la estructura de la memoria y el plan de trabajo.

2. **Conceptos previos**: Da a conocer aquellos conceptos específicos del dominio acotado previamente y el estado del arte de los mismos en cuanto a su uso y alcance del proyecto.

3. **Aspectos tecnológicos**: Explica, de forma detallada aquellas tecnologías empleadas para completar los objetivos del proyecto y su motivación y alcance de uso.

4. **Metodología y Gestión de proyecto**: Comenta la metodología escogida, las adaptaciones al contexto de este proyecto, la planificación, estimación de costes y gestión de riesgos asociada al mismo.

5. **Desarrollo**: En este apartado se comentará, paso a paso, la arquitectura escogida, el modelo de datos, y la implementación en base a la planificación.

6. **Conclusiones y trabajo futuro**: Detalla la evaluación del resultado final de forma global, así como las posibilidades que ofrece este proyecto para ampliar líneas de trabajo que aporten valor al mismo.

7. **Apéndices**: Esta sección está compuesta de las siguientes secciones adicionales:

- **Documentación sobre la planificación y los resultados del desarrollo**: Donde se adjuntará la documentación gráfica que verifica que se ha realizado la planificación y una demostración de la interfaz.
- **Glosario**: Donde se definen los términos de definición técnica empleados en el proyecto.
- **Bibliografía**: Donde se recogen los libros, artículos y documentos que sustentan al proyecto.

1.4 Plan de trabajo

De forma general, el proyecto ha ido sustentado sobre la siguiente relación de etapas, guiada por la metodología escogida:

- Estudio del área de Recuperación de la Información en profundidad, y documentación sobre el estado del arte en generación automática de taxonomías y su ámbito aplicativo.
- Estudio del concepto de búsqueda facetada, su objetivo, su alcance y su relación con las estructuras jerárquicas.
- Estudio del ámbito tecnológico disponible para la consecución de objetivos, así como para favorecer el diseño de una arquitectura de alta disponibilidad y tolerancia a fallos.
- Realización de un diseño arquitectural preliminar, que se irá complementando conforme vaya avanzando el desarrollo, y un modelo de datos acorde a la aplicación que dará soporte a la construcción de taxonomías de forma interactiva.
- En posteriores iteraciones, se conseguirán incrementos de software que permitirán alcanzar los objetivos a corto plazo que se establezcan.
- Elaboración de la memoria y documentación asociada.

Capítulo 2

Conceptos previos

Este capítulo contendrá todos aquellos conceptos introductorios necesarios para la comprensión del dominio de desarrollo del trabajo. Se empezará definiendo el área de trabajo (Recuperación de Información), y proseguirá con la formulación del problema, los distintos aspectos teóricos abordados y una visión detallada de la solución empleada para satisfacer la motivación de este proyecto.

2.1 Recuperación de la información

La Recuperación de la Información es la rama de las ciencias de la computación que busca satisfacer las necesidades de información de los usuarios en base a la información digitalizada. Esta necesidad se satisface a través de consultas a grandes colecciones de documentos, tratadas para su fácil comprensión.

Aunque no es la única forma, los motores de búsqueda son los ejemplos más representativos de Recuperación de la Información. El usuario puede formalizar una necesidad de información a partir de la cual el motor proporcionará los resultados más adecuados, presentándolos de forma accesible.

2.1.1 Búsqueda facetada

A partir de la anterior definición, la búsqueda facetada en este ámbito es una técnica para poder acceder a la información del mismo modo que con un motor de búsqueda, pero organizando la información de acuerdo a un sistema de clasificación de facetas. Este sistema ofrecerá una forma de navegación por filtros a través de una colección de resultados, los cuales podrán ser ampliados o reducidos ante la inclusión o exclusión de nuevas facetas.

Las facetas son elementos de información, que representan agrupaciones de interés desde el punto de vista del usuario. Un ejemplo sencillo puede ser contemplar la extracción del autor en una colección de documentos, de forma que uno de los sistemas de clasificación de

facetas puede ser en base al nombre del autor. Las facetas ofrecen la posibilidad de ampliar la búsqueda en torno a una temática, mejorando la experiencia de usuario a través de sugerencias que permitan refinar su búsqueda o realimentar futuras consultas ampliando el espacio de búsqueda.

En este contexto, las facetas serán los distintos nodos de la taxonomía resultante. Esto se debe a que una taxonomía es una estructura de clasificación por definición, lo que permite delimitar los documentos que han sido determinantes a la hora de influir en el resultado del proceso de construcción.

2.2 Formulación del problema

Se busca obtener una taxonomía resultado $\mathcal{T}$, a partir de un conjunto de documentos $\mathcal{D}$ y una «taxonomía semilla» $\mathcal{T}^0$, estructurada en forma de árbol y con pocos elementos, que sirva de guía en la tarea y sea base para la obtención de relaciones entre entidades y candidatos para la ontología. Cada nodo $e \in \mathcal{T}$ representa una entidad (o término) del corpus $\mathcal{D}$. Cada arista del árbol $\langle e_1 e_2 \rangle$ representa un par de entidades que satisfacen la relación específica de tarea, expresada a través de $\mathcal{T}^0$. La taxonomía resultado se podría expresar como $\mathcal{T} \stackrel{def}{=} (\mathcal{E}, \mathcal{R})$, donde $\mathcal{E}$ es el conjunto de nodos entidad y $\mathcal{R}$ el conjunto de aristas relación.

2.3 Conceptos relacionados

En esta sección se formularán tres conceptos que asientan las bases de la solución adoptada al planteamiento del problema.

2.3.1 Construcción de taxonomías

El proceso de construcción de taxonomías es, en esencia, dependiente del conocimiento experto, difícil de automatizar (debido a la abstracción que poseen las posibles relaciones entre términos) y costosa en su elaboración por ser, generalmente, manual. Tradicionalmente, se requiere información específica del dominio de aplicación del cual se busca realizar una ontología, lo que provoca sobrecostes derivados de la búsqueda de conocimiento experto a través del estudio del ámbito, en muchas ocasiones guiado por expertos (internos o ajenos a un contexto organizativo).

Además, la diversidad semántica que puede haber en las relaciones de las taxonomías presenta una limitación para los algoritmos de construcción de los que se dispone actualmente. La mayoría de aproximaciones para la resolución de este problema consiste en la extracción de términos a través de una relación de hiperónimo-hipónimo [1]. donde cada para «padre-hijo» se relaciona a través de una relación del tipo «ES_UN». Habitualmente, se extraen primero

las parejas hiperónimo-hipónimo y luego se organizan en una jerarquía, hasta conseguir el resultado en una estructura de árbol, típicamente.

Para extraer estas relaciones, existen tanto métodos basados en patrones como en métodos distributivos. La primera aproximación consiste en detectar patrones sintácticos (patrones «Hearst» [2]) entre términos. Posteriormente, se le han incorporado normas lingüísticas a este método para afinarlo, y, paralelamente, han surgido diseños de patrones generalizados (p. ej.: «star-pattern» [3]). Estos métodos se caracterizan por proporcionar una alta precisión pero un bajo «recall». Por otro lado, la aproximación distributiva se caracteriza por predecir si dos entidades tienen una relación entre sí respecto a la representación distributiva de estas. Inicialmente, se partía de estadísticas sobre los términos y se extraen candidatos a través de su similitud por parejas (p.ej.: similitud de coseno). Más adelante, se toman «word embeddings»[1] [4] pre-entrenados y un conjunto de datos de entrenamiento para formar modelos supervisados de clasificación y regresión. A pesar de todo, ninguna de las dos técnicas son aplicables al ámbito actual de la formulación del problema, ya que, aunque estas técnicas asienten las bases de la construcción automatizada de ontologías, se busca una elaboración de las mismas que sea más generalizada que la relación hiperónimo-hipónimo. Se trata de conseguir una taxonomía guiada por tarea[2], donde las relaciones extraídas están deducidas de la entrada del usuario y, por lo tanto, son específicas de la tarea proporcionada por este.

Para organizar los nodos obtenidos en una jerarquía, muchos métodos organizan todos los términos en un grafo con todas las relaciones de Hiperonimia (implicando la presencia de aristas que contienen ruido o una relación poco representativa) y después extraen un árbol jerárquico del mismo. Ninguno de estos dos métodos es aplicable al contexto actual, dado que en el «framework» propuesto se aprovecha la relación existente entre términos en la «semilla» proporcionada y la expande manteniendo esa relación, que no ha de ser necesariamente de Hiperonimia. En este caso, tanto la selección de candidatos de la ontología y la construcción de esta se realiza paralelamente en una única tarea conjunta, permitiendo la retroalimentación del sistema a la hora de consolidar la obtención de candidaturas a partir de la relación entre (cada vez más) entidades y también facilitando la construcción y reorganización de la taxonomía sobre la marcha a partir de la selección de candidatos realizada hasta el momento. De esta forma, el objetivo de reforzar la semántica representada a través de las relaciones entre unas pocas entidades en la entrada del usuario se cumple, buscando afianzar, iterativamente, no sólo la coherencia entre los pares de entidades –directamente relacionados– que expandirán la jerarquía inicial, sino que también la cohesión global de la estructura, revisada en cada iteración y optimizada al final del proceso de construcción. Es importante destacar que

[1] «Word Embedding»: Nombre de un conjunto de lenguajes de modelado y técnicas de aprendizaje en procesamiento del lenguaje natural en dónde las palabras o frases del lenguaje natural son representadas como vectores de números reales.

[2] En inglés: «task-guided»

no solo es necesario que las relaciones sean semánticamente similares a las proporcionadas en la «semilla», sino que es esencial que la estructura resultante tenga sentido y proporcione un sentido global a los distintos niveles de la jerarquía –de nada sirve saber que los «hijos» de la entidad país «USA» son los distintos estados si para la entidad «España» se tiene como hijos directos las distintas ciudades y no las comunidades autónomas y/o provincias que la componen–.

2.3.2 Expansión de «sets»

La expansión de «sets» [5] (estrechamente relacionada con la tarea de expansión de la taxonomía en el «framework» HiExpan) es una técnica consistente en obtener, a partir de un «set» pequeño de términos, un «set» completo de entidades que comparten la misma clase semántica. En el contexto de este desarrollo, no solo se busca la expansión de los «sets», sino que se requiere la organización de los mismos en una estructura jerárquica (típicamente, de árbol, como una ontología clásica). Una opción para conseguir esto consiste en la obtención de un ránking de documentos a partir del «set» que se busca expandir a través de un motor de búsqueda en línea y extraer los términos candidatos de los mejores resultados. Otra aproximación consiste en expandir dicho conjunto de entidades en base a un corpus, el cual se procesa para la extracción de candidatos con posibles semejanzas semánticas, de forma «offline». Esta técnica, por sí sola, no resuelve el problema planteado, pero aporta un punto de apoyo para la obtención de candidatos, los cuales se reorganizarán en estructura de jerarquía aplicando otros métodos.

2.3.3 Extracción de relaciones débilmente supervisada

Relacionado con los dos conceptos previos, la extracción de relaciones es esencial, ya que tiene por objetivo la obtención de relaciones semánticas a partir de un corpus, que se da generalmente entre dos o más términos (p. ej.: «París está en Francia», posee una relación [París, está en, Francia], donde «París» y «Francia» son dos entidades y «está en» la relación que les da valor semántico). El concepto «débilmente supervisado» consiste en una forma de etiquetado (en nuestro caso, se correspondería con el etiquetado de relaciones, es decir, las aristas del árbol jerárquico) en la cual, en un aprendizaje supervisado, los conjuntos de entrenamiento son etiquetados de forma imprecisa, limitada o con ruido, ocasionado por la falta de datos previamente etiquetados, falta de conocimiento en el dominio, o insuficiente tiempo para etiquetar apropiadamente un conjunto de entrenamiento, a cambio de reducir costes y agilizar un proceso en esencia muy pesado.

En este contexto, realizar un pre-etiquetado manual para cualquier taxonomía sería verdaderamente costoso en tiempo y recursos expertos, requiriendo de conocimientos de cada uno de los diversos dominios y la suficiente información como para disponer de un «dataset»

sobre el cual extraer las relaciones en las que se basará la ontología final. Una forma de afrontar la extracción consistiría en la identificación de patrones de forma iterativa, comenzando con patrones básicos que acabarían refinándose durante el proceso[3], llevando a la extracción de relaciones dentro del conjunto de documentos. La contrapartida de este método de extracción es la necesidad de ingentes cantidades de relaciones entre entidades para poder tener un clasificador efectivo, lo cual es un impedimento si se parte de un conjunto de «semillas» muy limitado.

2.4 Planteamiento de la solución: HiExpan

Ante el problema propuesto, y en vista de que los métodos planteados en la sección anterior no proporcionan una solución completa por sí solos, pero aportan valor en su resolución, la idea principal del «framework» HiExpan [6] consiste en aglutinar estos conceptos en un único procedimiento coordinado para la construcción de ontologías guiadas por tarea. Para este método, la idea principal consiste en visualizar cada «set» de nodos subyacentes a una entidad de la taxonomía como un conjunto coherente bajo la misma clase semántica2.1. A partir del corpus $\mathcal{D}$, primero se extraen los términos mediante una herramienta de minado de frases[4], y se filtran por categoría gramatical (para descartar aquellas categorías no relevantes como los verbos, preposiciones, etc.). La extracción de frases viene motivada por la ausencia de contexto semántico que permite determinar con certeza la categoría gramatical de aquellas palabras que puedan poseer ambigüedades. Por ejemplo: en el contexto de la oración «Interpreta al bueno de la serie», «bueno» juega un papel de sustantivo, aunque eso no es lo habitual para esta palabra, que suele encontrarse como adjetivo por norma general. Sin embargo, sin el contexto de la oración, es muy poco probable que el término se clasifique como adjetivo, sino como sustantivo, porque es más probable que así aparezca en muchos contextos. Es por ello que, para poder realizar un filtrado más preciso o que permita mayores matices según el tipo de frases u oraciones, es necesario que estas se extraigan de esta forma más sofisticada y se evalúe cada término dentro del contexto que le corresponda.

Debido a que el resultado de la extracción contiene mucho ruido, la obtención de los términos más relevantes se realiza a través de una técnica de expansión de «sets», de donde se extraen los mejores candidatos para cada iteración, evitando así tener que explorar todos los términos de uno en uno para buscar el mejor candidato en cada iteración. Este proceso se identificará como expansión en anchura. Sin embargo, existen algunos nodos dentro de la taxonomía que no poseen «subsets» durante el proceso y podrían poseerlos. Para solventar

[3] En inglés, este tipo de técnica que parte de algo sencillo y a partir de ello se forman elementos de mayor complejidad se denomina «bootstrapping»

[4] En inglés: «phrase-mining tool», consiste en una herramienta que, en vez de extraer palabras o grupos de palabras, extrae frases con significancia gramatical y, en ocasiones pero no necesariamente, semántica

este limitante, se realiza una expansión en profundidad del nodo, es decir, una extracción de relaciones (débilmente supervisada), de forma que a partir de ese nodo se pueda obtener un «conjunto semilla» de nodos hijo. Al final del método, se realiza un proceso de optimización que podrá reestructurar el árbol resultante de aplicar los dos métodos anteriores de forma iterativa e intercalada.

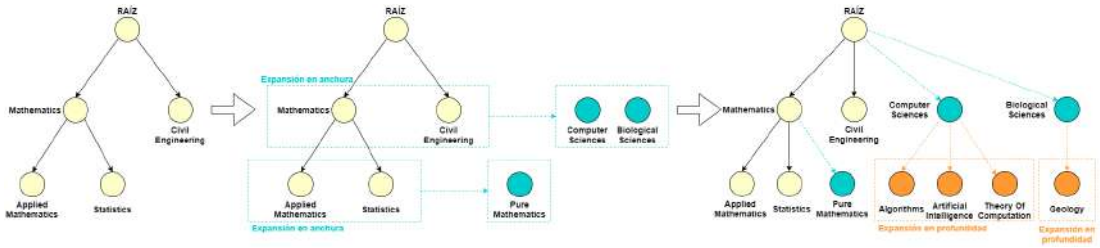

Figura 2.1: Ejemplo de expansión de taxonomía según el planteamiento de HiExpan

2.4.1 Extracción de términos clave

Para la extracción de términos clave, el «framework» utiliza AutoPhrase [7], un extractor de frases que proporciona un listado de términos en conjunto a la lista de frases donde se identifican los términos, para cada documento. Después de extraer las frases, se les aplica a los términos obtenidos un etiquetado de categorías gramaticales[5]. De todos los términos, se filtran solo aquellos que posean una etiqueta de sustantivo (p. ej.: Sustantivo Singular (NN), Sustantivo Plural (NNS), Nombre Propio (NNP)). Evidentemente, el resultado en este paso contiene ruido y será relativamente amplio y díficil de manejar, aunque es importante mantener un alto «recall» en etapas tempranas para filtrar los falsos positivos en los posteriores pasos.

2.4.2 Expansión jerárquica del árbol

La expansión jerárquica del árbol se basa en los conceptos definidos en la sección anterior. Para la expansión en anchura, se aprovechará la expansión de «sets» para encontrar candidatos que compartan la misma clase semántica. Para la expansión en profundidad, se utilizará la extracción de relaciones débilmente supervisada a partir de un pequeño conjunto de correspondencias entre términos para obtener pares de entidades que posean enlaces similares.

En el siguiente pseudocódigo se encuentra definido todo el proceso de expansión jerárquica, de forma que sirva de presentación al detalle de cada uno de los procedimientos internos:

[5] En inglés: «Part-of-Speech» (POS) tagger

Algorithm 1: Expansión jerárquica del árbol

Entrada: Una taxonomía guiada por tarea $\mathcal{T}$

Salida : Una taxonomía $\mathcal{T}^0$; una lista de candidatos V; número máximo de iteraciones max_iter.

$\mathcal{T} \leftarrow \mathcal{T}^0$;

for *iter = 1 to max_iter* **do**

  $q \leftarrow queue([\mathcal{T}.rootNode])$;

  while *q is not empty* **do**

    $e_t \leftarrow q.pop()$;

    if $e_t.children$ *is empty* **then**

      $S \leftarrow$ `ExpansiónProfundidad`(e_t);

      $e_t.children \leftarrow S$;

      $q.push(S)$;

    $C_{new} \leftarrow$ `ExpansiónAnchura`$(e_t.children)$;

    $e_t.children = e_t.children \oplus C_{new}$;

    $q.push(C_{new})$;

  `ResoluciónConflictos`(T);

Medidas de similitud

Para la resolución del problema de expansión se dispondrá del algoritmo SetExpan, propuesto en el «framework», con unas ampliaciones en cuanto a las características requeridas para la ampliación de los conjuntos.

Respecto a estas características que se extraerán de la colección de entrada, distinguimos tres tipos:

- **«skip-pattern».** Los «skip-patterns» son palabras en el contexto de un término e_i donde este es sustuituído por un marcador de posición. Así pues, la información de valor se halla en las palabras que rodean a la entidad que es sustituida, imponiendo restricciones posicionales. Un ejemplo de «skip-pattern», si se toma la entidad «Cat» en la frase «My Cat is Object-Oriented.», un posible skip-pattern sería «My _ is». Se extraen hasta seis patrones de diferente longitud para cada término e_i.

- **«term embedding».** Los «embeddings» son representaciones matemáticas (vectores de números reales) de los términos mediante técnicas de lenguajes de modelado y técnicas de aprendizaje en procesado de lenguaje natural. De esta forma se aumenta el rendimiento del análisis sintáctico y semántico al trabajar con estas representaciones.

- **«entity type».** Para obtener el tipo de cada entidad, se enlazan los términos con una

instancia (local u «online») de Probase, una base de conocimiento para la deducción de tipos a partir de términos o frases. De haber entidades que no fuesen enlazables, no se tiene en cuenta tipo alguno para ellas.

Estas tres características se utilizarán para conocer la similitud de dos entidades «hermanas» (pertenecientes al mismo «set») $\text{sim}_{\text{sib}}(e_1\ e_2)$. Para ello, primero se computan todos los pesos entre cada par de entidades y sus «skip-patterns». Por otro lado, se halla la robustez de la asociación entre un término y su tipo deducido a través de la obtención de sus «entity types», con la «confidence score» obtenida mediante Probase, es decir, cómo de «seguro» se está de que la entidad e tiene un tipo *ty*. Así pues, se computa, para una entidad y con los «skip-patterns» de ésta:

$$f_{\text{e,sk}} = \log(1 + X_{\text{e,sk}}) \left[\log |V| - \log(\sum_{e'} X_{\text{e',sk}}) \right], \tag{2.1}$$

donde $\text{X}_{\text{e,sk}}$ es la co-ocurrencia de la entidad y el «skip-pattern» (en crudo, es decir, el número de veces que se encuentra en el mismo documento el patrón y la entidad). A su vez, |V| es el número total de entidades candidatas extraídas. De igual forma, se calcula el peso de la robustez de un término respecto a un tipo asociado:

$$f_{\text{e,ty}} = \log(1 + X_{\text{e,ty}}) \left[\log |V| - \log(\sum_{e'} C_{\text{e',ty}}) \right], \tag{2.2}$$

donde $\text{C}_{\text{e',ty}}$ es el «confidence score» obtenido a través de Probase, que refleja la seguridad con la que se deduce que una entidad tiene un tipo determinado. A partir de estos pesos y, a la hora de calcular la similitud entre dos entidades dentro del mismo «set», primero se computa la similitud entre dos entidades ($e_1\ e_2$) a través de los «skip-patterns»:

$$sim^{sk}_{sib}(e_1, e_2|SK) = \frac{\sum_{sk \in SK} \min(f_{e_1,sk}, f_{e_2,sk})}{\sum_{sk \in SK} \max(f_{e_1,sk}, f_{e_2,sk})}, \tag{2.3}$$

donde SK es un conjunto de patrones «discriminativos». Es decir, por cada patrón de ese conjunto (más adelante se verá cómo se seleccionan los patrones más «discriminativos»), se guardan el peso mínimo de los dos en un acumulador y el peso máximo en otro acumulador, para luego dividir la suma de mínimos entre la suma de máximos. De esta forma, cuanto más pequeña sea la diferencia de los pesos, mayor será la similitud.

De forma similar, se calcula la similitud de tipos para dos entidades, pero en vez de seleccionar un subconjunto de discriminantes, como en el caso anterior, se toman todos los tipos de forma que:

$$sim^{tp}_{sib}(e_1, e_2) = \frac{\sum_{ty} \min(f_{e_1,ty}, f_{e_2,ty})}{\sum_{ty} \max(f_{e_1,ty}, f_{e_2,ty})}. \tag{2.4}$$

Por último, el tercer factor determinante es la similitud del coseno entre las dos entidades a traves de sus «embeddings»:

$$sim_{sib}^{emb}(e_1, e_2) = cos(e_1, e_2) = \frac{\vec{v_{e_1}} \cdot \vec{v_{e_2}}}{\|\vec{v_{e_1}}\|\|\vec{v_{e_2}}\|}, \tag{2.5}$$

siendo esta medida una representación de la «distancia» entre dos términos en cuanto a su semántica. Una semántica más similar proporcionará un coseno más alto.

Una vez obtenida la medida de las tres similitudes por separado, y sabiendo qué es lo que mide cada una, se formula la combinación de las tres de forma que, un par de entidades son afines si aparecen en contextos similares, tienen tipos similares y tienen «term embeddings» similares, extraídos mediante word2vec [8], un software para la extracción de representaciones matemáticas vectoriales de términos. Por tanto, cada una tiene un aporte directo y multiplicativo al cómputo final, dando lugar a la siguiente fórmula:

$$sim_{sib}(e_1, e_2|SK) = \sqrt{(1 + sim_{sib}^{sk}(e_1, e_2|SK)) \cdot sim_{sib}^{emb}(e_1, e_2) \cdot (1 + sim_{sib}tp(e_1, e_2))}. \tag{2.6}$$

2.4.3 Expansión en anchura

Una vez presentadas las medidas de similitud para cada par de entidades, y a partir de un «set» S de entidades del árbol de nodos y una lista de entidades candidatas V, la idea principal consiste en calcular la similitud promedio de todas las entidades candidatas respecto a las entidades de S, a través de las medidas propuestas en el apartado anterior. Sin embargo, teniendo en cuenta que la lista de candidatos es muy ruidosa en cuanto a que muchos de los candidatos son de mínima relevancia para S, y que el conjunto de «skip-patterns» extraídos totales es desorbitadamente alto, es necesario acotar el espacio de cómputo de alguna forma.

El primer paso a seguir es reducir el número de «skip-patterns» que se va a utilizar. Para esto, se computará el «score» acumulado de cada patrón respecto a su similitud con las entidades en el «set» S. Una forma de hacerlo es utilizar la forma mostrada en el apartado anterior donde se calcula el peso de una entidad respecto a un patrón ($f_{\text{e,sk}}$). Tras calcular el agregado por cada entidad de S (p.ej.: $score(sk) = \sum_{e \in S} f_{e,sk}$), de estos pesos para todos los patrones, se ordenan de forma descendente a modo de ránking y se escogen los 200 primeros «skip-patterns». A partir de estos 200 patrones, se realiza un muestreo sin reemplazo (es decir, los individuos seleccionados se descuentan de la población total en las subsiguientes elecciones) para generar 10 conjuntos SK_t de 120 patrones cada uno. Una vez el espacio de patrones está acotado, se toman las entidades candidatas de V para acotar el ruido que contiene respecto al «set» actual S. Ya que los subconjuntos SK_t están construidos sobre S, es coherente pensar que las entidades de V que tienen al menos una ocurrencia dentro de al menos uno de los

conjuntos de SK_t es una entidad apta para su consideración. De esta forma, no sólo se acotan los «skip-patterns», sino que también se acotan las entidades candidatas, ya que aquellas que no tienen patrones «altamente relacionados» no deberían ser consideradas.

Una vez reducidas las entidades y obtenidos aquellos SK que se mencionaban en el apartado anterior, se computará el «score» de una entidad considerada e de la siguiente forma:

$$score(e|S, SK_t) = \frac{1}{|S|} \sum_{e' \in S} sim_{sib}(e, e'|SK_t). \tag{2.7}$$

Profundizando un poco, se puede observar como el «score» de cada entidad candidata es mayor cuanto mayor sea la similaridad con todos las entidades que ya pertenecen a S. Esto proporciona, para cada conjunto SK_t, el ránking de las entidades candidatas ordenado de forma descendente por sus «scores» (sea L_t). La posición en el ránking de una entidad, por lo tanto, se define como r_t^i para la entidad e_i en L_t y si e_i no aparece en L_t, $r_t^i = \infty$. Así pues, se computa el «Mean Reciprocical Rank» (mrr) de cada entidad e_i y se añaden al «set» S de la siguiente forma:

$$mrr(e_i) = \frac{1}{10} \sum_{t=1}^{10} \frac{1}{r_t^i}, \qquad S = S \cup \{e_i | mrr(e_i) > \frac{1}{r}\}. \tag{2.8}$$

La decisión de adoptar por una fórmula como la del mrr, es decir, agregar las posiciones en los distintos ránkings para cada conjunto de «skip-patterns» relacionados, se debe a la concepción de que una entidad relevante para un «set» coherente aparecerá en muchos de estos conjuntos de patrones S_t con un ránking alto en L_t, y, por lo tanto, tendrá un buen ránking promedio. En este caso, r es un parámetro umbral sobre el cual se decidirá si una entidad pertenece o no al «set» a expandir, teniendo un valor constante $r = 5$.

2.4.4 Expansión en profundidad

En el caso de expansión de anchura, se parte de cada uno de los conjuntos coherentes de entidades que tienen una entidad «padre» común. Sin embargo, cabe la posibilidad de que aquellos nodos que se introducen dentro de un «set» como resultado del proceso de expansión se encuentren en un conjunto de entidades que tienen, a su vez, nodos «hijo» que forman «sets» coherentes. Esta situación deja a las entidades recién añadidas sin la posibilidad de formar un «set» de nodos hijo, ya que la entrada del proceso de expansión de profundidad requiere, necesariamente, al menos un descendiente.

Para solventar la problemática, se realiza una expansión en profundidad de aquellos nodos que no posean descendientes, siempre y cuando exista dicho nivel de profundidad en el resto del árbol de la taxonomía. Esta expansión tiene como objetivo obtener los primeros descen-

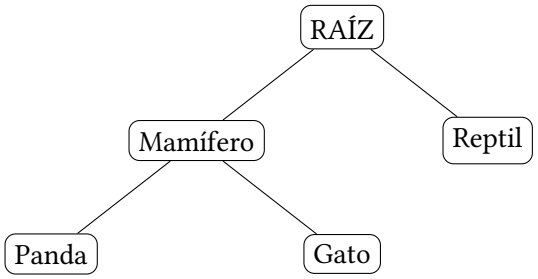

Figura 2.2: Ejemplo simple de taxonomía

dientes de una entidad, ponderando su relación semántica con los nodos con los que forma un «set» (sus «hermanos») y las relaciones de estos «hermanos» con sus respectivos nodos descendientes.

El algoritmo de expansión en profundidad utiliza los ya definidos «term embeddings», las representaciones matemáticas de los términos que refleja, de forma codificada, la semántica del mismo (denotado por $v(t)$, el vector asociado al término[6]). Al tratarse de vectores, la distancia entre vectores implica deducir la relación semántica entre ambos. Partiendo de esta base, y sobre el ejemplo de la Figura 2.2, se puede afirmar que la distancia entre el nodo *Mamífero* y sus hijos *[Panda, Gato]* es una distancia objetivo similar que debería alcanzar el nodo *Reptil* –al cual buscamos añadirle profundidad– y los futuros candidatos para ser descendiente de este nodo. Estos candidatos podrían ser *Caimán*, *Serpiente*, *Tortuga*...

Así pues, dado un nodo objetivo para la expansión e_o y un conjunto de aristas de referencia $E = \{\langle e_p, e_h \rangle\}$, donde e_p es el nodo «padre» de e_h (el nodo «hijo»), se calcula la afinidad de colocar un nodo candidato e_x como descendiente de e_p de la siguiente forma:

$$sim_{par}(\langle e_o, e_x \rangle) = cos\left(\vec{v_{e_o}} - \vec{v_{e_x}}, \frac{1}{|E|}\sum_{\langle e_p, e_c \rangle} \vec{v_{e_p}} - \vec{v_{e_c}}\right), \tag{2.9}$$

donde $cos(v(x), v(y)$ representa la similitud del coseno entre los vectores $v(x)$ y $v(y)$. Tras calcular la $sim_{par}(\langle e_o, e_x \rangle)$ de todos los candidatos, se seleccionan los n-candidatos con mayor «score» para formar el conjunto inicial que subyacerá al nodo e_o y será objetivo de la expansión en profundidad en subsiguientes iteraciones. El «embedding» de un término se extrae por aprendizaje a través de REPEL [9], un modelo para la extracción débilmente supervisada a través de aprendizaje por patrones (ya mencionado anteriormente). En este caso, el aprendizaje se realiza mediante las relaciones entre términos de la taxonomía «semilla» y permite obtener las representaciones matemáticas a través de sus dos módulos, un módulo basado en el aprendizaje por patrones, y otro módulo distribucional, sobre el cual se realiza un

[6] En ocasiones, y sólo para el contexto de este proyecto, se usará de forma indistinta el vocablo «término» y «entidad»

clasificador predictivo de las representaciones. Como ambos módulos se supervisan entre sí, el módulo de patrones proporciona patrones cada vez más fiables y el módulo distribucional ajusta el modelo de clasificación.

2.4.5 Resolución de conflictos

Como el algoritmo propuesto de expansión del árbol jerárquico es iterativo, en cuanto a que aplica la expansión en anchura y la expansión en profundidad de forma alterna y repite este proceso varias veces hasta conseguir el árbol expandido en su totalidad, es importante tener en cuenta que la supervisión del usuario es muy limitada, y consiste sólo en la entrada de la taxonomía «semilla» T^0. Esta señal de supervisión, como se ha expuesto, no solo es débil (por ejemplo, para la extracción de «term embeddings» o para el cómputo de la similitud entre nodos de un mismo «set»), sino que se va reforzando en cada iteración.

Esto lleva a la necesidad de asegurarse de que los nodos introducidos por cualquiera de estos métodos en las primeras iteraciones de la expansión tengan posiciones de calidad, es decir, que la posición dentro de la taxonomía sea coherente tanto con el «set» al que pertenece como con la altura a nivel jerárquico dentro de ésta. El impacto de introducir un nodo en una posición errónea en las primeras etapas de la expansión es mucho mayor que un error introducido en posteriores etapas, ya que su introducción de forma prematura induce a que la expansión de la taxonomía pueda tomar otro rumbo a nivel cohesivo respecto a su semántica global. Por lo tanto, si un mismo término aparece en múltiples posiciones de la taxonomía en construcción, se denomina «conflicto», y la prioridad es resolverlo, encontrando la mejor posición para esa entidad.

Dado un conjunto de nodos conflictivos C (las distintas posiciones de la misma entidad), se aplican tres reglas para determinar el mejor lugar para colocar el nodo. Primero, si uno de los nodos pertenece a la taxonomía «semilla» T^0, entonces el mejor candidato es ese y se da por finalizado el conflicto. De cualquier otra forma, para cada par de nodos en el conjunto C, se comprueba si uno de los nodos es el ascendente del otro, y de cada par se mantiene solo el ancestro, si lo hubiere. Una vez cribado, se calcula la «confianza»[7] de cada nodo $e \in C$ de la siguiente manera:

$$conf(e) = \frac{1}{|sib(e)|} \left(\sum_{e' \in sib(e)} sim_{sib}(e, e'|SK) \right) sim_{par}(\langle par(e), e \rangle), \tag{2.10}$$

donde $sib(e)$ es el conjunto de nodos del mismo «set» que e[8], y $par(e)$ se refiere al nodo ascendente al candidato. SK, a su vez, es el conjunto de «skip-patterns» seleccionado a partir

[7] En inglés: «confidence score», mide la certeza con la que se puede afirmar que un nodo pertenece a ese lugar
[8] Del inglés: sib proviene de «siblings», es decir, los «hermanos del mismo conjunto»

de la robustez acumulada de todas las entidades de $sib(e)$. La idea principal consiste en capturar la similitud conjunta del candidato respecto a sus «hermanos» y su «padre». Por lo tanto, aquel nodo que posea mayor similitud conjunta será el candidato seleccionado. No obstante, todavía falta por determinar qué es lo que ocurre si debajo de los nodos repetidos y descartados (que deben ser eliminados de la taxonomía) existe un subárbol de descendientes, o qué es lo que sucede con los nodos «hermanos» de este descarte que han sido añadidos «a posteriori» del mismo (ya que su presencia dentro del «set» ha podido contribuir de forma negativa a la expansión del conjunto). Para solventar esta situación, de cada nodo repetido que se va a eliminar, se eliminan también todos los nodos subyacentes a éste y todos los nodos añadidos a posterior del mismo (si verdaderamente una entidad borrada por este efecto pertenece a ese conjunto, en posteriores iteraciones será añadido a la taxonomía final). Finalmente, el nodo descartado será almacenado dentro de la «lista negra» de nodos hijo a descartar automáticamente en futuras iteraciones, para no reintroducir el conflicto.

2.4.6 Optimización global de la taxonomía resultado

Hasta ahora, la aplicación del algoritmo presentado ha valido para encontrar candidatos que posean coherencia con un conjunto de nodos, de forma local (computando similitudes entre nodos del mismo «set» y/o entre ascendente y descendente). Esto es útil para fomentar la coherencia dentro de un mismo «set» y para definir bien el subárbol candidato de una entidad. Sin embargo, un aspecto que ya se ha mencionado pero que no se ha teratado de forma explícita es la visión global de la taxonomía, es decir, su cohesión semántica. Como ejemplo, en la Figura 2.2, se podría incluir el término *Ornitorrinco* debajo del nodo *Reptil*, por su característica común de poner huevos, que en ese contexto los asemeja. Sin embargo, si se intenta responder a la pregunta «¿A qué reino animal pertenece el ornitorrinco?», la respuesta lógica debería ser *Mamífero*, por lo que habría que resolver esta inconsistencia en la taxonomía.

Como este aspecto es importante a tratar para asegurarse de que todos las entidades pertenecen al mejor lugar posible, el algoritmo HiExpan propone un módulo de optimización global de la taxonomía. El objetivo consiste en ajustar los niveles contiguos de la taxonomía dos a dos (es decir, ascendentes y descendentes por pares) de forma que se busque el mejor nodo «padre» para cada uno de los hijos de un subárbol. El proceso, en mayor detalle se basa en dos hipótesis. Primera, las entidades que tienen un padre común son similares y forman un mismo «set» coherente y, segunda, cada entidad es más similar a su padre «verdadero» que al resto de entidades del mismo padre «verdadero».

Por lo tanto, se asume que hay m nodos «padre» en el nivel superior de los dos y n nodos «hijo» en el nivel inferior. A partir de ahí, se denomina $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ para representar las similitudes entidad-entidad de los «hermanos» y se denomina $\boldsymbol{Y^c} \in \mathbb{R}^{n \times p}$ para representar

la similitud «padre-hijo» para cada par de entidades. Por lo tanto, $\boldsymbol{W_{i,j}} = sim_{sib}(ei, ej)$ si $i \neq j$, si no $\boldsymbol{W_{i,j}} = 0$. De forma análoga,$\boldsymbol{Y^c_{i,j}} = sim_{par}(\langle ei, ej \rangle)$.

A mayores, se define otra matriz de $n \times p$, llamada $\boldsymbol{Y^s}$ con $\boldsymbol{Y^s_{i,j}} = 1$ si un nodo «hijo» e_i está debajo del nodo «padre» e_j y $\boldsymbol{Y^s_{i,j}} = 0$ en caso contrario. Esta matriz representa la configuración actual de los nodos de ascendentes y descendentes. Se usará $\boldsymbol{F} \in \mathbb{R}^{n \times p}$ para definir la asignación teórica de nodos «hijo» a los «padres» que se busca obtener. Se plantea, por lo tanto, el siguiente problema de optimización, que refleja las hipótesis planteadas:

$$\min_F \sum_{i,j}^{n} W_{ij} \| \frac{F_i}{\sqrt{D_{ii}}} - \frac{F_j}{\sqrt{D_{ii}}} \|_2^2 + \mu_1 \sum_{i=1}^{n} \| F_i - \frac{Y_i^c}{\|Y_i^c\|_1} \|_2^2 + \mu_2 \sum_{i=1}^{n} \| F_i - Y_i^s \|_2^2, \tag{2.11}$$

donde $\boldsymbol{D}_{ii}$ es la suma de la fila i de la matriz $\boldsymbol{W}$, y μ_1 y μ_2 son dos parámetros de ajuste no negativos. El primer término de la expresión corresponde a la primera hipótesis planteada y modela la similitud de dos entidades «hermanas», de forma que si dos entidades son similares (tienen un $\boldsymbol{W}_{ij}$ alto), deberían tener un «padre» similar. El segundo término corresponde a la segunda hipótesis, que modela la similitud «padre-hijo». Por último, el último término es el suavizado respecto a las hipótesis, que representa la configuración de la taxonomía después de aplicar el proceso de expansión, previo a la optimización. Esto, en conjunto con los parámetros de ajuste, permite ponderar la importancia de asumir que la estructura previa a optimización es robusta en contraposición al posible factor de que un nodo «padre» tenga un candidato mejor a esta posición entre los nodos «hijo» del nivel inferior (del mismo o de distinto subárbol).

Para solventar este problema de optimización, se deriva respecto de $\boldsymbol{F}$, proporcionando la siguiente solución en forma cerrada[9]:

$$\boldsymbol{F} = (\boldsymbol{I} - \alpha S)^{-1} \cdot (\beta_1 \boldsymbol{Y}^c + \beta_2 \boldsymbol{Y}^s), S = D^{-1/2} W D^{-1/2}, \tag{2.12}$$

donde $\alpha_1 = \frac{1}{1+\mu_1+\mu_2}$, $\beta_1 = \frac{\mu_1}{1+\mu_1+\mu_2}$ y $\beta_2 = \frac{\mu_2}{1+\mu_1+\mu_2}$. A partir de esta optimización, se obtienen los nodos reorganizados con las mejores posiciones para cada par de niveles.

[9] Solución en forma cerrada: Se dice que una ecuación es una solución en forma cerrada si resuelve un problema dado en términos de funciones y operaciones matemáticas elegidas de un conjunto limitado y generalmente aceptado.

Capítulo 3

Aspectos tecnológicos

A la hora de abordar un problema y construir una solución, un aspecto crítico es la elección adecuada de tecnologías y herramientas que influirán tanto en el flujo del proceso de desarrollo como en la implementación de aspectos técnicos dentro de éste. En este capítulo, se detallarán las herramientas empleadas y los aspectos tecnológicos destacados para abordar dicha solución.

Primero, se explicarán los elementos esenciales para el manejo del módulo de construcción de taxonomías y búsqueda facetada. A continuación, se expondrán los aspectos estructurales relacionados con la persistencia, la aplicación web para la explotación de resultados («back-end» y «front-end») y el despliegue conjunto de los módulos de la aplicación. Por último, se comentarán aquellas herramientas de apoyo al desarrollo y que influyen transversalmente en todo el proceso software.

3.1 Módulo de construcción de taxonomías

El módulo de construcción de taxonomías es muy diverso, ya que su propósito es servir peticiones controladas que permitan control de errores centralizado, dado que hay muchos posibles puntos de fallo. Algunos de estos aspectos tecnológicos son, realmente, dependencias directas de la solución adoptada. Otras, sin embargo, son derivadas de decisiones de diseño o estrategias derivadas del diseño de alto nivel de la arquitectura software, que se expondrá en subsiguientes capítulos.

3.1.1 AutoPhrase

AutoPhrase es una herramienta de extracción de frases, cuya finalidad es ofrecer información de mayor calidad que los «n-gramas»[1] para diversos fines como la búsqueda, extracción

[1] «n-grama»: Secuencia de n términos consecutivos.

de características sintácticas, procesamiento de lenguaje natural, etc. A diferencia de otros métodos de obtención de frases, este es automático y no requiere de esfuerzo humano y valora su validez en base a su popularidad (la frecuencia con la que se obtiene la frase), concordancia (la forma en la que se colocan los tokens y su significado), capacidad informativa (si habla de un tema o un concepto concreto) y completitud (que la frase en cuestión sea completa en significado en el contexto del documento).

3.1.2 Python3, SciPy, NumPy y SpaCy

Python3 es un lenguaje de programación multiparadigma conocido por su influencia en el ámbito científico gracias a bibliotecas como SciPy y NumPy. El lenguaje, en conjunto con ambas bibliotecas, pone a disposición del desarrollador un entorno completo de funcionalidades matemáticas. NumPy es una biblioteca para la gestión de «arrays» n-dimensionales eficientes. SciPy es un ecosistema para el desarrollo matemático, científico e ingeniería, compuesta por diversos algoritmos enfocados a computación técnica y eficiente.

Por otro lado, SpaCy es una biblioteca para Python/Cython para el procesamiento de lenguaje natural[2]. Esta biblioteca resulta esencial para el etiquetado de los términos obtenidos del corpus, y dicho etiquetado servirá para la detección de sustantivos y descarte de aquellas palabras que no son relevantes para el contexto del problema a resolver.

3.1.3 Gensim y word2vec

Gensim es una biblioteca para el procesado de documentos, extracción de temáticas y procesado de lenguaje natural. Utiliza aprendizaje no supervisado para la extracción de temáticas, anotación de términos y, especialmente, la obtención de «term embeddings», representaciones matemáticas de términos en forma de vector. Para ello, se utilizará el modelo «word2vec», un modelo basado en «SkipGrams», una técnica de aprendizaje no supervisado para la extracción de términos similares a partir de otra distinta.

3.1.4 Probase

Probase es una base de conocimiento probabilística para la obtención de tipos para una entidad. Esta base de conocimiento pretende reflejar el conocimiento subyacente en el sentido común del ser humano a través de millones de entidades y relaciones organizados en una taxonomía probabilística que permite proporcionar etiquetado de entidades respecto a todas las conexiones almacenadas en este proyecto. A través de esta estructura, se pueden obtener los «entity types», que proporcionan información de valor semántico para el entendimiento

[2] Conocido comúnmente como «Natural Language Processing» (NLP)

correcto de términos dentro del dominio específico, debido a la tremenda variedad semántica e informativa que puede ofrecer una única palabra según una contextualización dada.

3.1.5 Flask, Celery, Redis y Gunicorn

Flask [10] es una biblioteca para Python especializada en la creación de contenedores API REST. Contiene las herramientas necesarias para la realización de «endpoints», gestión de peticiones HTTP, parámetros de URI, contenido en el cuerpo de la petición, etc. Es una biblioteca sencilla de usar y fácilmente configurable, lo que suponía una buena opción para el despliegue de llamadas remotas para la gestión del módulo de taxonomías, aprovechando un diseño por contrato que ofrezca un enfoque claro para la comunicación entre módulo y aplicación web.

Para la construcción de taxonomías, se ha dispuesto de Celery, una biblioteca que proporciona una cola distribuida para la ejecución de tareas. Permite delegar las tareas de mayor envergadura en un hilo asíncrono, sin interferir en la comunicación con el usuario.

Para poder mantener información del hilo ejecutor, su estado y comunicación, es necesario un «message broker»[3]. Éste permitirá comunicar el módulo con el hilo ejecutor de tareas asíncronas. Existen varias opciones para la comunicación, siendo una de ellas Redis, un motor de base de datos en memoria, que cubre perfectamente las necesidades y es ligero y fácil de utilizar.

Para terminar de cubrir el paquete completo de funcionalidad del módulo, es preferible hacer uso de una aplicación «Web Server Gateway Interface» (WSGI), un envoltorio de aplicaciones web para Python que ofrece un punto de entrada y salida para las conexiones. Para este contexto, se ha utilizado Gunicorn, un WSGI sencillo de usar y ligero.

3.1.6 Docker

Uno de los puntos clave para el despliegue del módulo de construcción es el uso de Docker, un proyecto de automatización del despliegue en contenedores, con los recursos ajustados a las necesidades del software que contiene. Esto facilita la tarea de virtualización, generalmente engorrosa y que consume demasiados recursos para las necesidades de un software, además de generar imágenes estables de mucho mayor tamaño e inflexibilidad. Con esta tecnología, se pueden lanzar contenedores, monitorizarlos y volcar su configuración en una imagen que siempre devuelva un contenedor idéntico del que se partió.

Asimismo, como se requiere desplegar varios contenedores a la vez que son intercomunicados, se ha hecho uso de Docker Compose, una herramienta que permite, a través de una configuración de entorno y un fichero de definición en formato YAML. De este modo, a través

[3] Generalmente, un «message broker» es un programa intermediario que se encarga de traducir los mensajes de un sistema a otro

de un único fichero se configurarán todos los contenedores que sean necesarios y se podrá configurar una red privada para su comunicación interna sin necesidad de exponer todos los contenedores.

Gracias a esta tecnología, se ha podido conseguir un despliegue de todos los contenedores que han sido necesarios para la implementación no sólo del módulo de construcción de taxonomías, sino para la gestión de operaciones asíncronas y el lanzamiento de nodos de motores de búsqueda de Elasticsearch –ver: 3.2.2–.

3.2 Persistencia

Para poder mantener un registro de los usuarios, los proyectos, directorios y las construcciones de las taxonomías que se van a realizar, es necesaria una buena gestión de la persistencia para la información normalizada en atributos pertenecientes entidades consistentes. Es por esto por lo que se ha optado por un sistema de gestión de bases de datos relacional.

3.2.1 MySQL y H2

MySQL es un conocido sistema de gestión de bases de datos relacionales, que cumple en gran medida con estándar SQL ISO/IEC, tiene gestión de la concurrencia (ideal para una aplicación con múltiples usuarios) y tiene una gran cantidad de documentación accesible para el desarrollador. Gestiona la persistencia en disco, con una zona reservada de memoria para el acceso y la escritura rápidas a datos, que se sincroniza periódicamente con el almacenamiento. Esta base de datos cubre con las necesidades del sistema en entornos de pre-producción y producción.

Por otro lado, H2 es un sistema de gestión de bases de datos codificado en Java [11], con gestión de la persistencia en la memoria volátil. Su principal característica, además de su volatilidad, es su facilidad para la integración en aplicaciones Java, sin necesidad de establecer una conexión a través de «sockets». A pesar de su poca utilidad en el caso de una aplicación que gestione persistencia y ésta sea de acceso a largo plazo –en caso de que la aplicación en cuestión sufra una caída, la información sería irrecuperable–, es de mucha utilidad a la hora de realizar pruebas de unidad, integración y sistema, ya que esa información no debe pervivir en una base de datos y la realización de una base de datos exclusivamente para estas pruebas puede resultar problemático, en cuanto al coste en su mantenimiento a la par de una base de datos de pre-producción y otra de producción. Mediante este tipo de base de datos integrada, la gestión de la integración continua se vuelve más ágil

3.2.2 Elasticsearch y elasticsearch-py

Elasticsearch es un software de búsqueda basado en Apache Lucene, implementada en lenguaje Java. Provee herramientas completas de indexación con un motor de búsqueda en texto, con capacidad de tenencia múltiple[4] con una interfaz web REST que se comunica a través de objetos JSON. Otra de las características llamativas de este software es que permite, de forma sencilla, adoptar una arquitectura distribuida con múltiples nodos que forman parte de un «cluster». De este modo, el procesado de las peticiones se distribuye entre los distintas instancias para evitar sobrecargas.

Para poder acceder al «cluster» de instancias de este software, existen alternativas al acceso mediante peticiones HTTP puras. En este caso, se ha dispuesto de elasticsearch-py, una biblioteca cliente de Python3 para Elasticsearch, que contiene las herramientas necesarias para hacer peticiones de forma cómoda a través de objetos. De esta forma, el trabajo de indexación y búsqueda será tan solo un módulo más dentro del sistema completo.

3.2.3 Flyway

A la hora de actualizar una base de datos incrementalmente, existen muchos puntos de flaqueza. En primer lugar, es complicado mantener la integridad de los datos a la hora de modificar el modelo relacional ya aplicado a una versión actual del sistema en un entorno de producción. Por otro lado, es complicado llevar un registro claro de la evolución de la persistencia, aún con la información que proporciona un sistema de control de versiones.

Flyway es una herramienta de migración de bases de datos en código abierto, muy útil para solventar este tipo de problemáticas dadas en la forma de desarrollo convencional, donde existe una separación muy definida entre los integrantes del grupo de desarrollo y los integrante del grupo operacional. Esta herramienta es, simplemente, un paso más para aproximarse a una filosofía DevOps, a través de la cual la gestión de la base de datos está mucho mas integrada dentro del proceso de desarrollo, proporcionando responsabilidades para una integración adecuada y elegante a los desarrolladores. Posee un sistema de líneas base, para aquellas migraciones estables del sistema que hayan alcanzado un hito en la vida de un proyecto (p.ej.: cada «release» del mismo), y una entidad de persistencia que lleva el registro de la evolución de la base de datos a lo largo de las migraciones, permitiendo identificar los distintos «scripts» de migración a través de su número de versión y una breve descripción.

[4] Tenencia múltiple: Se conoce como el principio de la arquitectura del software por el cual una sola instancia de una aplicación puede servir a diversos clientes u organizaciones

3.2.4 Hibernate y JPA

Hibernate es una herramienta de software libre de «Object-Relational Mapping» (ORM) para Java que establece un puente entre una base de datos relacional y el paradigma de objetos de una aplicación, añadiendo una capa de abstracción sobre JDBC y ocultando los detalles de la implementación de la misma, a través de anotaciones o archivos XML. De esta forma, en lugar de trabajar sobre las entidades de la base de datos directamente, se trabaja sobre los objetos, tratándolos como objetos persistentes. Esta abstracción también permite ocultar los detalles de implementación del sistema de gestión de bases de datos concreto sobre el que se esté trabajando, dando flexibilidad a la hora de cambiar entre sistemas sin modificar la implementación. Además, implementa la API de «Java Persistence API» (JPA), un «framework» para la plataforma Java EE que permite utilizar lenguajes de consulta de alto nivel (JPQL), metadatos objeto-relacional y un conjunto de funcionalidades para equiparar el funcionamiento de los objetos como entidades persistentes, sin perder las ventajas del desarrollo orientado a objetos. Como Hibernate basa su implementación en JPA, no solo ofrece una funcionalidad más pulida, sino que, además, tiene un propio lenguaje extendido de JPQL, llamado HQL, sobre el que se harán las consultas, principalmente.

3.3 Aplicación web

El desarrollo de una aplicación web, a día de hoy, es un proceso interesante en cuanto a la cantidad de «frameworks» y bibliotecas que colaboran entre sí para dar lugar a un sistema complejo realizado cada vez de una forma más sencilla. Aunque este ecosistema sea amigable al desarrollador a la hora de contemplar y fabricar soluciones, es importante echar un vistazo detallado a aquellas tecnologías más relevantes que han permitido que la construcción completa del sistema haya sido satisfactoria, haciendo hincapié en las de mayor impacto (o más estructurales) y en las que más valor didáctico han aportado.

3.3.1 Back-end

Spring y Spring Boot

Spring [12] es un ecosistema para el desarrollo de aplicaciones, basado en IoC o Inversión del Control. Su principal característica es la inclusión de un contenedor específico para la instanciación, inicialización y conexión de los diversos objetos de una aplicación, gestionando su tiempo de vida y su disposición en función de los eventos proporcionados. Los objetos de un contenedor («beans») suelen estar configurados a través de XML o anotaciones. Estos se pueden obtener a través de búsqueda de dependencias (cuando se le pide al contenedor un objeto con un nombre especifico o de un tipo concreto) o por inyección de dependencias (es

el contenedor el que suministra los objetos a otros objetos que lo declaren como dependencia), siendo el que se usará en el desarrollo.

Aunque esta es su filosofía principal, este es sólo uno de los muchos módulos que presenta este ecosistema. De Spring se aprovechará también el uso del módulo de acceso a datos, el módulo de gestión de transacciones, el módulo del «Modelo Vista Controlador» (MVC) para la gestión de APIs REST y HTTP «servlets» que permitan gestionar peticiones entre la vista y el modelo, módulo de autenticación y autorización y módulo de «testing» para asegurar la correcta ejecución de las diversas pruebas a través del contenedor y permitiendo ejecutar la lógica de negocio sin comprometer la independencia de las pruebas.

De forma adicional, se dispone de Spring Boot [13], una herramienta para la construcción sencilla de aplicaciones Spring con un servidor auto-contenido, configuración inicial consistente (en cuanto a dependencias necesarias) y personalizable. Asismismo, permite simplificar el complejo sistema de anotaciones del ecosistema Spring con anotaciones más sencillas, permitiendo resolver interdependencias de las diferentes bibliotecas de forma transparente al desarrollador. Se usará esta herramienta para agilizar el proceso de construcción inicial de la aplicación, reduciendo tiempo de desarrollo asociado a la resolución de conflictos y estructuración del proyecto.

Retrofit

Retrofit es una biblioteca cliente para la declaración «type-safe»[5] de peticiones HTTP a través de interfaces y anotaciones en Java y Android. Permite definir un cliente HTTP API simplemente esbozando los métodos, los parámetros y tipo de la «HTTP request» y la URL sobre la que realizarán las peticiones. Permite definir una implementación secuencial mediante «builders»[6], estableciendo una URL absoluta sobre la que posteriormente se irán construyendo URL relativas en cada uno de los métodos de la interfaz que implementará el cliente. Además, permite transformar cómodamente las respuestas JSON en objetos Java.

Junit, Mockito, MockMVC, JaCoCo

A la hora de desarrollar, el «testing» es uno de los aspectos más importantes para garantizar la validez y la exactitud de un proyecto software. Para poder mantener un conjunto de pruebas bien definido e integrado en la aplicación, facilitando la tarea de implementación, se ha utilizado JUnit [14], el «framework» de pruebas más utilizado para Java. Asimismo, Mockito es una biblioteca para la simulación de comportamiento de diversos objetos, para poder

[5] «Type safety»: Principio de las ciencias de la computación que define el alcance con el que un lenguaje de programación previene o restringe errores de tipo.

[6] «Builder»: Patrón creacional de diseño software que simplifica la construcción de objetos complejos a través de la construcción de partes individuales que permite construir representaciones diversas del mismo objeto en función de la diversidad de su clase abstracta constructora.

realizar pruebas de unidad sobre componentes que tengan interdependencias. MockMVC es una biblioteca para la prueba de «endpoints» creada para Spring, con el fin de hacer pruebas sobre los controladores de la aplicación.

Por otro lado, JaCoCo es una biblioteca para la medición de cobertura en código Java. Se combina bien con JUnit, y permite generar reportes en ficheros para su posterior lectura, aspecto que vendrá bien para favorecer la lectura automatizada de métricas.

3.3.2 Front-end

HTML5, CSS y JS

HTML es un estándar de lenguaje de marcado para la elaboración de páginas web establecido por el «World Wide Web Consortium» (W3C), siendo el estándar más conocido por los navegadores web. HTML5 es la quinta revisión de este estándar, que incorpora nuevos elementos para los navegadores modernos.

CSS, por otro lado, es un lenguaje de diseño gráfico para definir la presentación visual de un documento escrito en un lenguaje de marcado, habitualmente HTML. La idea nace de la necesidad de segregar la estética de la estructura de contenedores que permite separar la estructura y la presentación de páginas web, de forma que pueda variar la presentación del documento en función de múltiples factores como el tamaño de la ventana de visualización, el tipo de dispositivo (p.ej.: móvil u ordenador), el tipo de navegador y el estado interno de la página web, entre otras cosas.

Por último, JavaScript es un lenguaje de programación interpretado caracterizado por la programación basada en prototipos[7] y funciones de primera clase [8]. Es un lenguaje multiparadigma, cuyo estándar es ECMAScript, y sobre el cual existen diversos «frameworks» de alto nivel para optimizar la actualización del árbol «Domain Object Model» (DOM).

Node.js, Babel, ESLint y Yarn

Node.js es un marco de aplicaciones multiplataforma de código abierto, basado en JavaScript, asíncrono y con una arquitectura orientada a eventos[9]. Su motivación surge de la necesidad de crear aplicaciones que se comunican a través de la red altamente escalables, que permitan la gestión de peticiones asíncronas, la incorporación de módulos básicos y la posibilidad de extender esos módulos para la creación de aplicaciones web de alto nivel basadas en JavaScript. Está será la base sobre la que se trabajará para realizar el frontal de la solución.

[7] Programación basada en prototipos: Estilo de programación orientada a objetos en el que las clases no se definen explícitamente, sino que se derivan de agregar propiedades y métodos a una instancia de otra clase.

[8] Función de primera clase: Aquella función que es tratada como cualquier otra variable, habitualmente funciones anónimas

[9] Arquitectura orientada a eventos: Patrón de arquitectura software que promueve la producción, detección y consumo de eventos, o cambios significativos en una aplicación.

Babel es una herramienta de compilación para JavaScript que permite transformar el código JS más moderno en un código apto para cualquier navegador o versión de Node.js. Funciona a través de un sistema de plugins que permite definir qué es lo que se quiere que se transforme y en qué resultado, desde estándares como el ECMAScript 2015 hasta extensiones del lenguaje como JSX (se verá más adelante).

ESLint es un analizador de código estático que permite localizar errores, problemas y realiza asistencia en la resolución de conflictos que tienen solución automática. Permite, también, el añadido de reglas, analizadores personalizados y preprocesar código. Este analizador será de utilidad para facilitar el desarrollo del frontal de la aplicación web.

Por último, Yarn es un instalador de paquetes JavaScript y gestor de dependencias lanzado por Facebook en colaboración con otros desarrolladores. Es un tipo más novedoso de gestión de dependencias, sustituyendo a «npm» (el gestor de dependencias por defecto de Node.js) por un gestor de mayor rendimiento, con mayor velocidad para la obtención de dependencias y mayor estabilidad en la resolución de conflictos de dependencias.

ReactJS, JSX, MaterialUI

ReactJS [15] es una biblioteca de JavaScript que permite construir interfaces de usuario de forma sencilla y con gestión del estado interno por cada vista. Mediante estas vistas declarativas, esta biblioteca permite encapsular el estado de cada componente dinámico dentro del mismo, favoreciendo actualizaciones eficientes en el árbol DOM a través del cálculo de diferencias entre el árbol de componentes de esta biblioteca y el árbol del navegador. De esta forma, la vista sólo se actualizará cuando haya un cambio de estado y sólo se actualizarán aquellos componentes que estén subordinados a ese estado, haciendo la gestión óptima del costoso proceso que es actualizar el árbol del navegador web.

Una de características más conocidas de ReactJS es su orientación a componentes, que permite crear clases que extiendan al componente base de esta biblioteca, habilitando la gestión de propiedades (aquellas que vienen dadas de entrada o por herencia) y la gestión del estado (aquel que es intrínseco a la clase en sí). De este modo, no solo es fácil segregar el estado entre los distintos componentes, sino que incorporar aspectos de la programación orientada a componentes encaja muy bien con el estilo de arquitectura orientada a eventos, permitiendo establecer un compromiso de Cadena de responsabilidad

Otra de las características beneficiosas de esta biblioteca es la posibilidad de usar técnicas avanzadas, como la creación de Componentes de Orden Superior[10], los cuales permiten reutilizar la lógica, decorando otros componentes y añadiéndoles propiedades y funcionalidad. Mientras un componente transforma propiedades en interfaz de usuario, un componente de orden superior permite transformar un componente en otro. Esta técnica no forma parte del

[10] Por sus siglas en inglés: «Higher Order Component» (HOC).

API de React, pero es un patrón que surge de la naturaleza composicional de este. Asimismo, un HOC es una función pura[11], lo que incrementa la eficiencia a la hora de actualizar el árbol DOM (una función con efectos colaterales puede provocar más actualizaciones en el árbol del estado de React, provocando efectos indeseados como un decremento de la «performance» del navegador).

Por otro lado, una técnica avanzada más reciente y novedosa es el uso de «React Hooks». Los «Hooks» surgen como una alternativa al uso de clases para la composición de componentes y lógica, cambiando por completo el paradigma objetual por un sistema de funciones que se «enganchan» al estado de React y al ciclo de vida de los componentes funcionales, proporcionándoles funcionalidad adicional. Los «Hooks» funcionan por sí solos, sustituyendo a las clases, pero, a pesar de superponerse en lógica con los HOC, no los sustituyen por completo. Existen casuísticas que permiten que se utilice una técnica sobre la otra, y no son incompatibles entre sí: se puede tener dentro del mismo proyecto Componentes de Orden Superior y «Hooks» sin ningún tipo de problema.

Para poder trabajar cómodamente con los elementos que renderizará React a través de sus componentes, se ha decidido apoyar el desarrollo con JSX, una extensión de JavaScript que permite definir elementos intrínsecos de HTML en el propio código JavaScript, de forma que quedan entrelazadas las expresiones JSX con ambos lenguajes combinados. Aunque al principio puede sonar contraproducente, ya que rompe con la filosofía arquitectural del MVC, simplemente es una forma de abordar el problema desde otro punto de vista. Partiendo del diseño orientado a componentes de React, que acepta de base que la lógica de renderizado está estrechamente ligada a la lógica de la interfaz de usuario. Esta separación no es una separación convencional entre maquetado y control, sino que es una separación de intereses[12].

Finalmente, MaterialUI [16] es una biblioteca que ofrece componentes de React, agilizando el desarrollo con componentes base que cubren necesidades básicas del desarrollo web, como barras de navegación, superficies, listas o menús desplegables. De esta forma, se facilita el proceso de diseño, decorando estos componentes base y los temas de colores que proporciona para darle un aspecto fresco a la aplicación web.

Redux

Si React es una biblioteca para la gestión del estado de un componente, Redux es su análoga en lo que a estado de la aplicación respecta. Es una biblioteca para la gestión de un contenedor

[11] Función pura: Aquella que dada una misma entrada, siempre devuelve una misma salida, sin efectos colaterales.

[12] Separación de intereses: Principio del diseño software que busca separar las distintas partes de un software en intereses, es decir, conjuntos de información que afectan al código del programa que lo compone.

de acceso a nivel de aplicación, basado en el concepto de reducer[13]. Gracias a Redux, se puede definir estado que permita la comunicación, a nivel global de la aplicación, entre componentes.

Este «middleware»[14] permite definir un conjunto de acciones, las cuales tienen un tipo concreto y describen que algo ha ocurrido. A partir de las acciones y sus tipos, el «reducer» se encarga de actualizar el estado en consecuencia, notificando a todos los componentes que están suscritos solo a esa parte de la información que se ha modificado, permitiendo de esta forma no interferir con la filosofía de React de no actualizar más elementos del árbol DOM de los necesarios. Para poder mantener suscribir los componentes tan solo a los cambios de estado necesarios para cumplir con su interés, Redux proporciona el uso de selectores, funciones que acceden al contenedor del estado y proporcionan la lógica seccionada de forma coherente.

3.4 Herramientas de desarrollo

Apache Maven

Hoy en día, la construcción del software es un proceso esencialmente repetitivo y fácil de automatizar, así como su gestión, su configuración, la ejecución de sus pruebas y su despliegue. Es por ello que es difícil concebir un proyecto sin tener en cuenta una herramienta adecuada a estas necesidades que permita acelerar en gran medida todos estos procesos rutinarios, contribuyendo, a su vez, a la reducción de errores, tiempos y costes y a la estandarización de procedimientos, gracias a su automatización.

Apache Maven es una herramienta para la gestión y construcción automática de software Java. Se basa en una configuración de la construcción basada en XML, a través de uno o más ficheros «Project Object Model» (POM), que no sólo definirá el ciclo de construcción del proyecto, sino que también permite definir su estructura, generar, validar y gestionar documentación, establecer dependencias del proyecto, hacer uso de «plugins» para su uso a lo largo del ciclo de vida de la construcción y facilitar el despliegue y diversificar la configuración condicionalmente a través de la creación de perfiles.

Para este proyecto, será esencial hacer uso de esta herramienta, ya que permitirá definir varios perfiles en base al entorno (desarrollo o producción) y en base al tipo de sistema de gestión de bases de datos que se quiere utilizar. Además, facilitará la inyección de bibliotecas de dependencias en el proyecto, proporcionará un arquetipo sobre el que comenzar a trabajar, agilizará la ejecución de pruebas y la obtención de estadísticas resultado (p.ej.: cobertura de código) y permitirá la rápida construcción del software.

[13] Reducer: En programación funcional, el reducer es el encargado de, a partir de información distribuida, aglutinar el resultado dada una petición concreta

[14] «Middleware»: Software que, generalmente, se encuentra entre dos aplicaciones para servir intercambios de información y mediar la comunicación lógica entre ambos

PyCharm, Eclipse, VSCode

Los «Integrated Development Environment» (IDE) son un elemento indispensable para la agilización en el proceso de desarrollo, tanto a la hora de disponer de herramientas para aumentar la eficiencia en la codificación como la posibilidad de tener un abanico de herramientas auxiliares que intervienen a lo largo de todo el desarrollo («debugging», ejecución, compilación y resaltado de sintaxis en tiempo real, gestión de paquetes, dependencias y contenedores, integración con sistemas de gestión de la configuración…). Es por eso que se debe disponer de un buen IDE adecuado a las necesidades del lenguaje y el «framework» que se vaya a utilizar, ya que facilita el entendimiento, la modificación y el mantenimiento de proyectos de gran envergadura minimizando las pérdidas de tiempo. Debido a la naturaleza multimodular de este proyecto, se ha optado por el uso de tres entornos distintos, cada uno el más adecuado a su lenguaje.

PyCharm es un entorno de desarrollo utilizado especialmente para Python, desarrollado por JetBrains. Es el IDE más utilizado para este lenguaje y su aspecto más cuidado es la asistencia inteligente a la hora de obtener sugerencias para completar el código, lo que reduce el tiempo de desarrollo y facilita la tarea de comprensión del desarrollador.

Eclipse, por otro lado, es un entorno de desarrollo multilenguaje basado en espacios de trabajo, ampliable mediante un sistema de «plugins» para aumentar sus funcionalidades. Es el entorno predominante para el desarrollo de proyectos Java, siendo el más popular y del cual se obtiene más información, por poseer una mayor comunidad de desarrolladores.

Por último, VSCode es un editor de código fuente[15] multiplataforma, con soporte para la depuración, la integración con un sistema de control de versiones, finalización inteligente de código, resaltado de sintaxis y refactorización, entre otras cosas. Es compatible con diversos lenguajes de programación, y permite personalización a través de extensiones, así que su versatilidad se ha hecho idónea para la edición del «front-end».

Git, GitLab y GitFlow

El versionado del software es un aspecto esencial a la hora de realizar un desarrollo. No solo es necesario que haya prácticas asociadas a la gestión de la configuración del software[16], sino que disponer de un sistema de control de versiones[17] es útil en diversos aspectos, como la comunicación entre desarrolladores del mismo proyecto, el etiquetado de versiones y archivos

[15] A pesar de no ser un IDE propiamente dicho, posee un subconjunto de las funciones de uno, sirviendo a un propósito similar, por lo que se ha decidido agruparlo entre los entornos.

[16] En inglés: «Software Configuration Management» (SCM), conjunto de prácticas que contribuyen a la elaboración de código fuente por varios desarrolladores simultáneamente, el seguimiento del estado de las fases del desarrollo del software (sus versiones), y sus cambios (el control de versiones) y la conducción de la integración de las partes del software en un único producto.

[17] En inglés: «Version Control System» (VCS)

que las componen, el control de cambios de un proyecto y la personalización de entornos de pruebas para facilitar la gestión de errores a través de un sistema de incidencias.

Git es un software de control de versiones desarrollado por Linus Torvalds, nacido de la necesidad de un control de versiones distribuido que desempeñase un buen papel en cuanto a rendimiento y simultaneidad en la distribución, muy popular y con una comunidad muy extendida a día de hoy. La característica principal de este VCS es su potencial de control distribuido, que facilita el trabajo de diversos desarrolladores de forma simultánea a través de ramas y con un sistema de etiquetado basado en «commits», que contienen el estado del software en un momento del cambio determinado. Además, cuenta con la posibilidad de mantener un registro de los cambios de forma «offline», pudiendo sincronizar con un repositorio remoto solo cuando se considere oportuno o se posible. Se ha escogido GitLab como el servicio web sobre el que se realizará el proyecto para la gestión del versionado del software.

Por otro lado, en muchas ocasiones, a pesar de tener la posibilidad de trabajar con un VCS distribuido, es difícil trabajar de forma simultánea si no se establecen unas pautas para la ramificación del código, el etiquetado de versiones o la identificación y resolución de funcionalidades y «bugs». El establecer acuerdos o flujos de trabajo es esencial para evitar que el caos se apodere del proceso de desarrollo de software. La vertiente principal del flujo de trabajo se denomina GitFlow.

GitFlow es un envoltorio de Git, publicado por Vincente Driessen[18]. Propone un modelo de creación de ramas, cada una con un propósito propio que sirve a satisfacer las necesidades mencionadas anteriormente. Consiste en el flujo dividido de cinco ramas:

- *Feature branches*: En cada una de estas ramas se encontrará una funcionalidad completa del software.

- *Develop branch*: A partir de esta rama se crean las ramas anteriores y es donde sucede el proceso de integración de cada una de las funcionalidades cuando hayan finalizado su desarrollo.

- *Release branch*: A partir de la rama *develop*, una vez que se acerca el momento de liberar una versión con el conjunto de funcionalidades finalizadas, se crea esta rama, para la ejecución de pruebas y resolución de posibles errores, tanto de configuración como de funcionalidad. Una vez se finaliza una versión, se integra tanto en *develop* como en *master*.

- *Master branch*: Esta rama contendrá el código listo para su puesta en producción, con una versión estable proviniente de una *release branch*. Opcionalmente, se puede etiquetar convenientemente con el número de versión.

[18] https://nvie.com/posts/a-successful-git-branching-model/

- *Hotfix branch*: Finalmente, si se detecta un error en producción (es decir, proviniente de *master*, se creará este tipo de rama con un ciclo de vida breve para solucionar el error en el menor tiempo posible, integrándola cuanto antes tanto en *master* como en *develop*.

A pesar de que este flujo es una muy buena práctica para mantener estandarizado parte del proceso de gestión de la configuración software, es impensable que un único desarrollador pueda mostrar todas sus ventajas poniéndolo en práctica, ya que gran parte de este potencial de flujo de trabajo (e incluso del sistema de control de versiones) se basa en el trabajo en equipo. A su vez, este proceso se puede complementar a través del control de la integración de las distintas ramas, mediante «merge requests». Las «merge requests» son elementos de control a la hora de integrar una funcionalidad en una rama de desarrollo común (p. ej.: volcar una *feature* en la rama *develop*), que permiten hacer revisión de los cambios y construcciones, descargas de la versión candidata, corrección de errores y realización de comentarios, hasta finalmente llegar a la aceptación o rechazo de la integración de los cambios. Sin embargo, en el contexto de este trabajo, se utilizará GitFlow con fines didácticos y por facilidad organizativa a la hora de asociar los cambios a las funcionalidades y «releases».

Figura 3.1: Ejemplo de flujo de trabajo con GitFlow en GitLab

Jenkins

Uno de los aspectos más delicados del desarrollo del software es la integración, a menudo puesta en jaque por la división del trabajo entre los miembros de un equipo de desarrollo, que

implementarán diversos aspectos del mismo sistema. Esto puede derivar, a la hora de integrar la funcionalidad resultante, en éxitos en pruebas a nivel de unidad, pero fracasos en pruebas a nivel de integración, resultando en un sobreesfuerzo inesperado. Estas integraciones, idealmente, se pueden dar a diario, e incluso varias veces por día, lo que lleva a la necesidad de detectar los errores de forma prematura y a tener una gestión para la integración continua.

La integración continua[19] es una práctica de desarrollo software que permite mantenerlo actualizado de forma constante, de forma que permita la detección de errores, la trazabilidad de las construcciones, la gestión del despliegue y la comunicación entre miembros del equipo de desarrollo. De esta forma, y gracias a la disponibilidad de construcciones estables de software, se puede saber detalladamente qué cambios han provocado un posible fallo y se puede tener una medida de la estabilidad del software en base a las construcciones más recientes.

Jenkins es una herramienta de automatización «open source» escrita en Java, enfocada a CI y que facilita ciertos aspectos de la entrega continua. Permite la ejecución de proyectos de diversos tipos, entre ellos proyectos Maven. Como añadido, admite sincronización con herramientas de control de versiones, como Git, y posee un sistema de «plugins» para la comunicación con otras herramientas. Entre otras cosas, muestra la tendencia de las ejecuciones, enlaces permanentes a las ejecuciones más relevantes (última ejecución, última ejecución estable y última ejecución fallida), información sobre la ejecución de los pasos adicionales a la ejecución (como el uso de «plugins» o el despliegue) y el histórico de construcciones –casi todo visible en la Figura 3.2–. Esta herramienta será idónea para enlazar con el sistema de control de versiones y sincronizar la construcción del software a medida que se va agregando funcionalidad al software.

Taiga

Taiga es un gestor de proyectos de carácter ágil de código abierto. Permite la gestión de distintos tipos de proyectos, donde se puede hacer un registro del avance del proyecto desde una perspectiva funcional (este aspecto y otros relacionados se comentarán en el capítulo posterior). Integra la funcionalidad necesaria para la gestión de proyectos basada en Scrum, a través de la definición de un *backlog* que permitirá representar las necesidades funcionales del proyecto a lo largo del desarrollo.

A través de las entradas del *backlog*, Taiga ofrece formas de representar la carga, dividir el trabajo y la asignación, estimar el avance y elaborar gráficas que permiten definir aspectos característicos del desarrollo ágil para determinar la velocidad a la que avanza el proyecto y la determinación de trabajo a realizar en siguientes iteraciones[20], entre otras características.

[19] En inglés: «Continuous Integration» (CI)

[20] En inglés: «velocity forecasting», una funcionalidad del gestor que permite determinar qué tareas son oportunas para satisfacer el ritmo de la carga de trabajo de *sprints* anteriores.

SonarQube

Otro aspecto importante a la hora de asegurar la calidad del proyecto durante el desarrollo es la inspección continua[21]. La definición de calidad es muy ambigua: se podría hablar de cómo el software se adecúa a las necesidades del cliente (calidad funcional o externa). Sin embargo, el tipo de calidad que busca la práctica de la inspección continua es una calidad más técnica o interna, a menudo descuidada o revisada superficialmente, que describe cómo de bueno es un software respecto a aspectos como la seguridad, el rendimiento, la robustez o la facilidad de mantenimiento.Esta práctica permite que todos los intervinientes en el proceso de desarrollo sean partícipes activos y necesarios del avance de la calidad, desde el primer momento en el que se comienza a desarrollar, permitiendo detectar amenazas de forma prematura y facilitando la resolución de una forma rápida y ágil.

SonarQube es una herramienta diseñada con este fin, para asegurar la calidad interna de un proyecto software. Incluye métricas como el número de «bugs», «code smells», vulnerabilidades, problemas de seguridad, deuda técnica, el porcentaje de cobertura del código y el porcentaje de líneas duplicadas. Además, proporciona una visión detallada de los problemas que pueda tener el software, una vista de métricas que permite saber cuáles son las partes del proyecto más susceptibles de perder calidad, una visión de código donde se puede observar con detalle sobre qué parte de la implementación se han detectado esos problemas –basados en conjuntos extensos de reglas– y un apartado donde se puede observar el histórico de ejecuciones del escáner de SonarQube.

Para este proyecto, se ha agregado un paso adicional al proceso de integración continua con Jenkins, al que, en caso de ejecutar con éxito una construcción, se realizará un escáner de SonarQube (a través de un «plugin») que permitirá analizar el software de la construcción concreta y recibirá un reporte de cobertura de código resultado de ejecutar las pruebas en el proceso de integración.

[21] En inglés «Continuous Inspection» (CI)

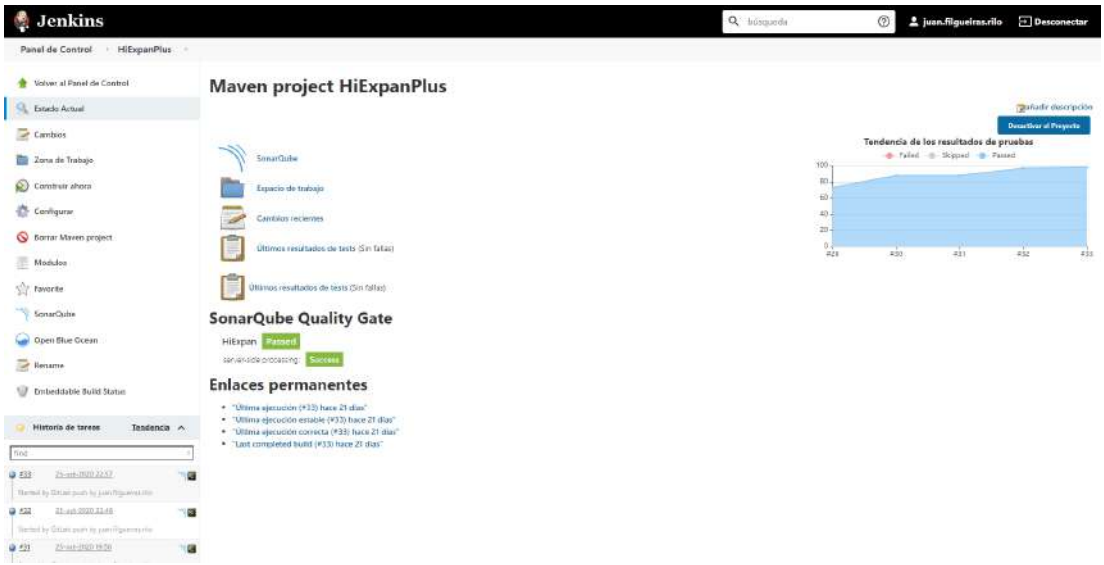

Figura 3.2: Ejemplo de Integración Continua - Interfaz principal de Jenkins

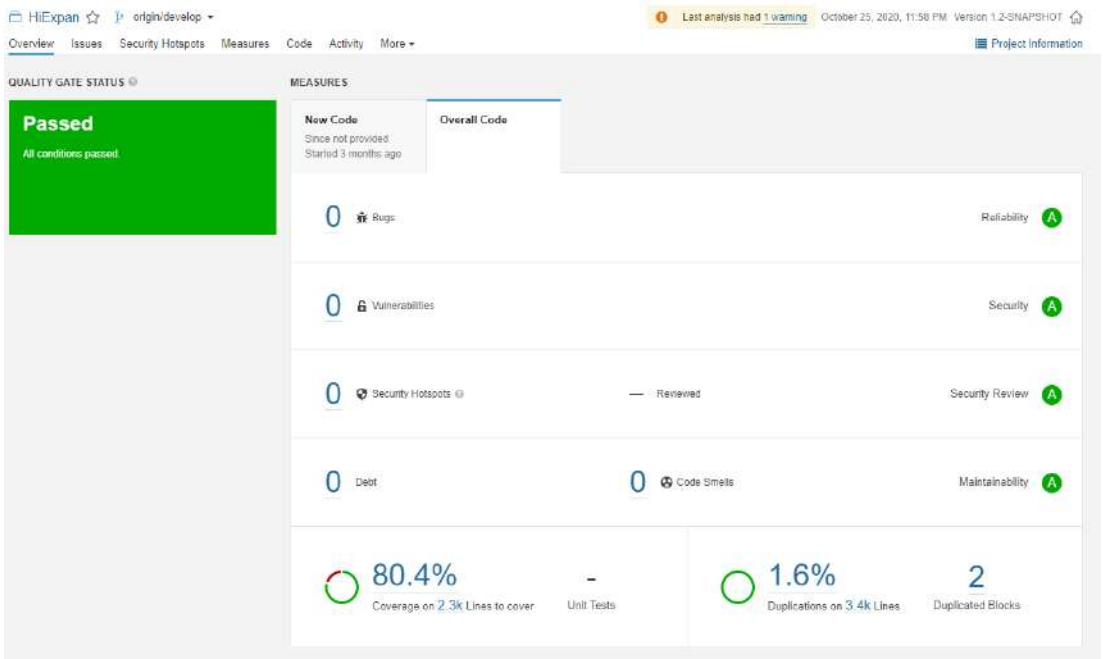

Figura 3.3: Ejemplo de Inspección Continua - Interfaz Principal de Sonar

Capítulo 4

Metodología y Gestión de proyecto

En este capítulo se tratará lo relativo a la metodología de desarrollo que se ha aplicado a lo largo de la realización de este trabajo. Primero, se explicará la metodología escogida y se expondrán las razones de su uso y adaptación al proyecto, para finalmente abarcar otros aspectos relacionados a la gestión de proyectos como la estimación de esfuerzo, tiempo y coste y la gestión de riesgos.

4.1 Metodología

La metodología de desarrollo constituye el corazón de un proyecto, indicando, de forma rigurosa, el proceder a la hora de determinar las acciones a realizar en cualquier punto del desarrollo (de forma más o menos explícita). Esta elección marcará, desde el comienzo del proyecto, las pautas esenciales para la adecuada construcción de la solución de acuerdo a los objetivos establecidos. La base metodológica, por lo tanto, ha de apoyarse en motivos no sólo de conocimiento de la situación contextual y los objetivos del proyecto, sino que deberá valorarse también su oportunidad estratégica y la viabilidad de su aplicación a este trabajo.

4.1.1 Elección de la metodología

La existencia de diversas metodologías hace que el proceso de elección de la metodología sea el el resultado de años de estudio sobre diferentes formas de gestionar el flujo del proceso software. Desde las metodologías más predictivas (a menudo basadas en el Proceso Unificado de Desarrollo de Software (PUDS)), hasta las más adaptativas, existen muchos criterios de selección que, sobre la balanza, no pone a ninguna como predominante sobre las demás. Sin embargo, a la hora de escoger, existen factores determinantes que pueden posicionar a algunas metodologías por encima de otras por ser más adecuadas al contexto.

A la hora de analizar la situación inicial, se presenta la realización de un trabajo académico, con un dominio específico, pero de difícil comprensión inicial para el alumno. Como la

visión del dominio es vaga, es importante tener en cuenta el grado de afectación que podría tener el adquirir conocimientos del dominio durante el proceso del desarrollo, pues podría desviar su foco por completo. Por lo tanto, el estudio preliminar del contexto del trabajo no es suficiente, puesto que la profundización en él llevaría demasiado tiempo como para afrontarlo desde el principio, y existe la posible necesidad de revisar en periodos breves de tiempo el avance del desarrollo para tomar decisiones a medida que se van adquiriendo conocimientos sobre el dominio. Por último, existe una funcionalidad inicial relativamente acotada, que puede complementarse *in medias res* a medida que se va avanzando, en función de la oportunidad y la viabilidad.

En resumen:

- Existe una necesidad de adaptación a cambios en los requerimientos y es preferible una buena gestión del cambio.
- El riesgo si se adoptan objetivos a largo plazo de forma predictiva es alto, puesto que el dominio es denso y completamente desconocido para el alumno.
- Hay una funcionalidad mínima, sobre la que conviene añadir, de forma compartimentada, funcionalidad que complemente el proyecto.
- Importa la comunicación con los tutores, ya que de ellos se obtiene la indicación de si el desarrollo está orientado correctamente o hay que tomar decisiones de cambio.

Tras haber estudiado estos puntos, es bastante evidente que practicar la agilidad es una opción acertada, ya que se observa la necesidad de realizar avances iterativos en periodos breves de tiempo, con construcción incremental y mucha reactividad ante el cambio. Es por ello que se ha adoptado una metodología ágil, especialmente basada en Scrum. No obstante, antes de comenzar a explicar qué es Scrum, se comentarán los fundamentos del Manifiesto Ágil.

4.1.2 Manifiesto Ágil

El Manifiesto Ágil es un documento redactado en 2001 en el que se propone un cambio completo de visión respecto a la forma tradicional de desarrollar software. En aquel momento, nace el término «Métodos Ágiles» para definir a aquellos métodos que estaban surgiendo como alternativa a las metodologías formales, por su excesiva pesadez y rigidez en cuanto al carácter normativo que poseen, sumado a la fuerte dependencia de planificaciones detalladas previas al desarrollo. Está basado en los siguientes 4 valores:

1. Individuos e interacciones sobre procesos y herramientas.
2. Software en funcionamiento sobre documentación exhaustiva.

3. Colaboración del cliente sobre negociación contractual.
4. Respuesta ante el cambio sobre seguir un plan.

Asimismo, estos cuatro valores derivan los 12 principios que conforman la base procedimental para establecer un marco de trabajo ágil:

1. Prioridad en satisfacer al cliente mediante la **entrega continua y temprana** de software **con valor**.
2. Aceptación de los **cambios**, incluso en etapas tardías del desarrollo, de forma que los procesos Ágiles aprovechen el cambio para proporcionar **ventaja competitiva** al cliente.
3. Entrega **frecuente** del software **funcional**.
4. Responsables de negocio y desarrolladores **trabajan juntos** de forma cotidiana durante todo el proyecto
5. **Motivación** en el trabajo en base a confianza, apoyo y un buen entorno.
6. **Conversación cara a cara** como método más eficiente y efectivo de comunicar información al equipo de desarrollo y entre sus miembros.
7. **Software funcionando** como medida principal de progreso.
8. Capacidad de mantener el desarrollo a un **ritmo constante** de forma indefinida.
9. Atención continua a la **excelencia técnica** y al **buen diseño**.
10. **Simplicidad**. Mantener al máximo la cantidad de trabajo no realizado.
11. Equipos **auto-organizados**.
12. **Reflexión periódica** sobre como mejorar la efectividad y modificación del comportamiento en consecuencia.

Como ya se puede deducir por el enfoque comunicativo en el equipo de desarrollo (debido a que este equipo solo estará formado por el alumno) y por la constancia en la dedicación en un proyecto (a menudo interrumpida por motivos académicos), algunos de estos principios del Manifiesto Ágil no son aplicables al contexto de este trabajo. No obstante, reiterando la dificultad que supone una comprensión profunda del dominio desde el primer momento, una metodología ágil sigue siendo predominantemente útil si se busca comunicación constante con «feedback» basado en funcionalidad entregada. A la hora de escoger una metodología, se ha recurrido a una conocida en detalle por el alumno: Scrum.

4.1.3 Scrum

Scrum [17] es un marco de trabajo para el abordaje de problemas complejos y adaptativos y para la entrega de productos con el máximo valor posible productiva y creativamente. El máximo valor productivo se alcanza en base a la búsqueda de la obtención de funcionalidad con el mayor aporte estratégico en cuanto a la oportunidad.. El máximo valor creativo surge de la colaboración entre el equipo de desarrollo y la organización para la consecución de objetivos nacidos de la necesidad de abordar el problema de la mejor forma posible.

Scrum es una metodología ligera y fácil de entender, pero muy difícil de dominar. Esto se debe, entre otras cosas, a la dificultad que supone dominar la gestión de las relaciones entre miembros, no sólo de un equipo, sino entre el equipo y la organización, llevando a la sincronización constante y al apoyo en la toma de decisiones frente al cambio, que son la base motivacional del trabajo en equipo en este contexto.

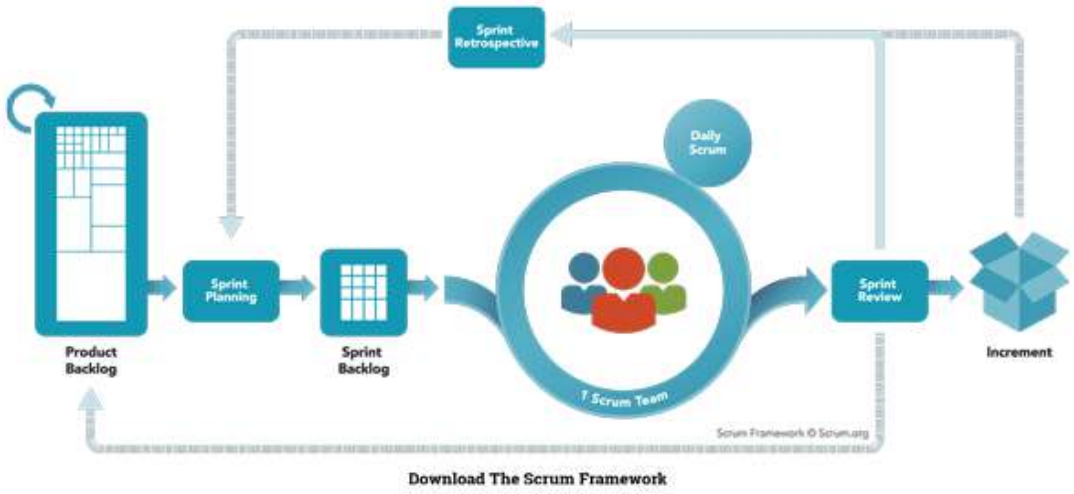

Figura 4.1: Diagrama del proceso de Scrum

Este marco de trabajo se basa en la teoría de control de procesos empírica o empirismo. A través del empirismo, es decir, el conocimiento venido de la experiencia de aquellos que practican Scrum, se hace la toma de decisiones. Además, para mejorar el control del riesgo y la predictibilidad, se emplea un enfoque iterativo e incremental. Existen tres pilares conceptuales que sirven de cimiento para la práctica adecuada del empirismo:

- *Transparencia*: Todo aspecto que sea significativo para el proceso debe ser **visible** para aquellos que son responsables del resultado. Esto requiere que haya un **estándar común** para facilitar el entendimiento. Uno de los conceptos que debe estar definido bajo estándar es la definición de **«Terminado»**[1].

[1] La definición de «Terminado» es un aspecto clave en Scrum, ya que puede ser muy variable entre distintos

- *Inspección*: Todo miembro del equipo Scrum debe **inspeccionar** con frecuencia los artefactos de Scrum y estar pendiente del avance hacia un objetivo común, lo suficiente para poder tomar **medidas preventivas** de forma reactiva si se detectan variaciones, pero no lo demasiado frecuente como para **entorpecer** el ritmo de trabajo.

- *Adaptación*: En el caso de que se produzcan desvíos significativos en torno a unos **límites aceptables**, se deberá **ajustar** el proceso o el material empleados. Este pilar está estrechamente comunicado con el anterior, ya que el ajuste debe realizarse en base a una buena inspección.

Scrum proporciona una serie de roles, eventos y artefactos con los que se puede trabajar, de forma que permiten alcanzar los pilares mencionados.

Roles

Un equipo Scrum define unos roles, que permitirán formar equipos auto-organizados, en cuanto a que todo el equipo es dirigido por sí mismo y la dirección del trabajo es interna y multifuncionales, en cuanto a que poseen todas las competencias necesarias para realizar el trabajo sin dependencias externas. Estos roles son:

- *Product Owner*: El *Product Owner* es el responsable de maximizar el valor del producto y del trabajo del equipo de desarrollo. Representa los intereses del cliente, y gestiona los elementos del *Product Backlog*, es decir, los expresa claramente y los ordena por prioridad, asegurándose de que tanto la pila del producto como los elementos que la componen se entienden claramente (favoreciendo la transparencia). Sus decisiones han de ser respetadas y debe asegurarse de que es una única persona. Constituye un baluarte entre las necesidades de la organización y el equipo de desarrollo para evitar desvíos de atención a la hora de captar necesidades beneficiosas para la organización.

- *Equipo de Desarrollo*: El *Equipo de Desarrollo* consiste en los profesionales que desempeñan el trabajo de entregar un Incremento de producto «Terminado», que potencialmente se pueda (y se deba) poner en producción a final de cada *Sprint*(se verá más adelante). Las características principales del *Equipo de Desarrollo* son su auto-organización; que les da una percepción de equipo de miembros sin un líder concreto (todos toman decisiones al unísono), su multifuncionalidad –en cuanto a la disponibilidad de habilidades necesarias para realizar el desarrollo– y la compartición de responsabilidades como un equipo único. Por último, el tamaño del equipo debe ser limitado, entre tres y nueve

equipos Scrum y su puesta en común proporciona mucha transparencia al proceso, a la hora de guiar, realizar, evaluar y expresar cuándo una unidad funcional del proyecto está lista para su liberación, sin malos entendidos entre miembros del equipo ni entre equipo y organización.

miembros, lo suficientemente pequeño para mantenerse ágil y lo suficientemente grande para desarrollar una cantidad de trabajo significativa.

- *Scrum Master*: El *Scrum Master* es el papel encargado de asegurarse que se entiende la lógica subyacente a Scrum. Colabora con las personas externas al equipo para entender qué interacciones pueden ser de relevancia para el equipo y cuáles no. Realiza un trabajo de mediación constante para garantizar la correcta comunicación bajo los estándares establecidos, evitando los malentendidos. Por un lado, presta servicio al *Product Owner* para ayudar gestionar el *Product Backlog*. También colabora para favorecer la gestión de eventos que sean necesarios. Por otro lado, también presta servicio al *Equipo de Desarrollo*, guiándolo en la auto-organización y multifuncionalidad.

Eventos

Los eventos de Scrum son bloques de tiempo («time boxes») que permiten asegurar la inspección y adaptación continuas, con el fin de minimizar el número de reuniones no definidas y cambios no deseados. Todos los eventos tienen una duración máxima, que se deberá respetar con el fin de mantener el rigor y evitar desviaciones. Existen cuatro eventos Scrum y un evento que es el contenedor de los demás:

- *Sprint*: El corazón de Scrum es el *Sprint*, un bloque de tiempo de entre quince días y un mes, sobre el cual se realiza un incremento de producto (que estará «Terminado»), oportuno para su liberación inmediata. Lo más habitual es que los *Sprints* tengan la misma duración a lo largo del desarrollo. Dentro de un *Sprint*, no se realizan cambios que puedan afectar al objetivo, y los objetivos de calidad no disminuyen. Todos los *Sprints* constan de un objetivo, un diseño y un plan flexible que guiará la construcción, el trabajo y el producto resultante.

- *Reunión de planificación del Sprint*: Esta reunión se realiza al comienzo de cada *Sprint*, con ocho horas máximas de duración. En esta reunión se define el objetivo (común para focalizar la atención en su cumplimiento por parte del equipo), cómo se alcanzará –es decir, con qué funcionalidad se adquiere el cumplimiento del objetivo– y se realiza un diseño de alto nivel y una planificación flexible.

- *Daily Scrum*: Esta reunión, de duración máxima de quince minutos, es de carácter diario y tiene como objetivo la sincronización de actividades por parte del *Equipo de Desarrollo*. A través de preguntas, se actualiza el estado del objetivo respecto a su grado de completitud. Las preguntas más comunes son: «¿Qué he hecho ayer que ayudó al Equipo a lograr el Objetivo del *Sprint*?», «¿Qué voy a hacer hoy para ayudar al Equipo a lograr

el Objetivo del *Sprint*?», «¿Existe algún impedimento que alguien de nosotros o todo el Equipo impida que logremos el Objetivo del *Sprint*?».

- *Sprint Review*: Reunión que se realiza al final de cada *Sprint*, de máximo cuatro horas (para *Sprints* de un mes), para inspeccionar el incremento de funcionalidad que cumple el objetivo. Es una reunión más informal, donde están invitados los «stakeholders»[2] y se realiza una demostración de lo conseguido, permitiendo revisar el *Product Backlog* para realizar cambios sobre él si fuese necesario. De la misma forma, se indica qué se ha «Terminado» y se reevalúa la pila de producto.

- *Sprint Retrospective*: Se trata de una reunión de carácter más filosófico (duración máxima tres horas para *Sprints* de un mes), previa a la siguiente reunión de planificación del *Sprint*, donde el Equipo Scrum se reúne para inspeccionarse a sí mismo y compartir experiencias. Esencialmente, se busca facilitar la comodidad del entorno, evitar riesgos, aumentar la calidad en el proceso Scrum, revisar personas, relaciones, procesos y herramientas, e incluso revisar la definición de «Terminado». De esta forma, se favorece la adaptación y la inspección del equipo.

Artefactos

Los artefactos de Scrum representan trabajo o valor en diversas formas que son útiles para mantener la transparencia y dar oportunidades de inspección y adaptación. Estos artefactos están diseñados de forma que se asegura que todos los miembros del equipo tienen el mismo entendimiento del artefacto a través de la maximización de la transparencia. Estos artefactos son:

- *Product Backlog*: El *Product Backlog* es una lista ordenada con todo lo que podría ser necesario en el producto y es la única fuente de requisitos para cualquier cambio a realizarse en el producto. Aquí se establecen todas las funcionalidades, cambios y mejoras que deben realizarse en un futuro. Este artefacto nunca está completo, evoluciona constantemente y lo hace en función del estado actual del producto. El orden de los elementos de esta lista representa la prioridad con la que se deben completarlos, en base a los criterios del *Product Owner*, que se encarga de mantenerlo actualizado y ordenado de forma que se garantice la transparencia.

- *Sprint Backlog*: La lista del *Sprint Backlog* conforma un subconjunto de elementos del *Product Backlog*,escogidos para la coherente consecución de un objetivo (y su correspondiente incremento funcional «Terminado») bajo el marco de un *sprint*. Su cometido es visibilizar el trabajo pendiente asociado al incremento, con el suficiente detalle como

[2] «Stakeholder»: Aquel que está interesado o está involucrado en que el proyecto avance exitosamente.

para que el *Equipo de Desarrollo* sepa exactamente qué hacer y se puedan sincronizar en base a un artefacto transparente común, permitiendo obtener una instantánea del avance en unidades funcionales.

- *Sprint Burndown charts*: Otro artefacto interesante para observar el avance de un *sprint* es esta gráfica, que permite observar el trabajo pendiente de forma descendente a lo largo del tiempo. No sólo es una observación sencilla del avance, sino que, al final, permite obtener una medida de la *velocity*, que es la cantidad de trabajo que un equipo puede acometer en un *Sprint*. Cuanta más continuidad se observe en la bajada de este gráfico, más fiable será la medida de la *velocity*.

4.1.4 Historias de usuario y puntos de historia

A la hora de definir los elementos del *Product Backlog*, la definición de requerimientos y su posterior refinamiento en requisitos se vuelve un proceso demasiado laborioso que no se corresponde con la agilidad que se pretende practicar en este contexto presente. Es por eso que la alternativa ágil para definir una unidad funcional de requisito es la *Historia de Usuario*[3]. Esta es la unidad mínima de entrega funcional que recoge un deseo del cliente desde una perspectiva del usuario que va a hacer uso de una característica del sistema. Consta del siguiente formato:

Como soy <ROL> quiero <ACCIÓN> para <MOTIVACIÓN>

Como el punto de vista es el del cliente, es evidente que este formato de representar un requisito es idóneo para mantener al actualizado un *Product Backlog* que satisfaga a los intereses del usuario y sea transparente para el desarrollador.

Para poder realizar una estimación del esfuerzo que se necesita para cumplimentar una *Historia de Usuario*, se utilizan los *Puntos de Historia*, una representación abstracta del esfuerzo necesario en base a todos los aspectos necesarios para completar una historia. Estos puntos pueden representar un amplio espectro de factores como la complejidad de las tareas necesarias para completar la historia, la cantidad de trabajo necesario, dificultad del diseño, consideraciones tecnológicas y operativas, riesgos y aspectos desconocidos, entre otras cosas. Para asignar estos puntos adecuadamente es necesaria una escala de asignación y un método para la estimación. La escala puede ser muy diversa, teniendo en cuenta que la asignación no es arbitraria ni toma valores fuera de ésta. Una opción muy extendida es el uso de una escala basada en la sucesión de Fibonacci, debido a que, a medida que el esfuerzo es mayor, más complicado es estimar con exactitud y más necesario es establecer un «peor caso» con margen de adaptación. Esta escala es idónea para esta situación, dejando distancia cada vez mayor entre valores sucesivos..

[3] En inglés: «User Story»

4.1.5 Scrum adaptado al proyecto

Como ya se ha comentado anteriormente, la utilidad completa de la metodología Scrum sólo se alcanza en el contexto de un equipo multidisciplinar de varias personas y realizando *sprints* continuamente sin interrupciones, que permiten la práctica de la agilidad a través de inspección y adaptación. Sin embargo, debido a que el equipo de desarrollo es constituido por una persona, y debido a sus obligaciones académicas y la situación producida por el Estado de Alarma respecto a la crisis sanitaria producida por la COVID-19, se han producido ciertas limitaciones que llevaron a la adaptación de la metodología con el fin de seguir obteniendo el beneficio de utilizar una metodología ágil para el desarrollo donde el entendimiento del dominio aumenta conforme se avanza en la entrega de funcionalidad útil.

Teniendo en cuenta la siguiente asignación de roles de Scrum:

- El rol de *Product Owner*, que representará las necesidades del proyecto, será ejercido por uno de los tutores de este proyecto, en concreto por Álvaro Barreiro García.

- El rol de *Scrum Master*, quien se asegurará de favorecer los eventos Scrum y se asegurará de que se realiza Scrum, lo ejercerá el otro director de este proyecto, Javier Parapar López.

- El *Equipo de Desarrollo* estará formado por el alumno, Juan Luis Filgueiras Rilo.

Se establecen las siguientes limitaciones:

- La duración de un *sprint* es de quince días y la cantidad de esfuerzo de cada uno será variable, de acuerdo con la necesidad de ajustar la *velocity* a lo largo del desarrollo, con una media aproximada de cuarenta y cinco puntos de historia/*sprint*.

- A pesar de que no se sigue la continuidad entre *sprints*, la duración interna del mismo sí que se mantendrá constante. Sin embargo, es posible que la carga de trabajo diaria no sea equilibrada a lo largo de la realización del mismo.

- Como el equipo de desarrollo está conformado por una única persona, las reuniones *Daily Scrum* se ven comprometidas por la falta de sentido en cuanto a la sincronización. No obstante, seguirá habiendo inspección diaria como parte del trabajo del alumno.

- Del mismo modo, el juicio de expertos no es viable al tratarse de una única persona en el equipo de desarrollo, por lo que la estimación de puntos de historia se realizará en base a criterios de experiencia propia.

4.2 Gestión del proyecto

Una vez justificada la metodología, es necesario establecer una gestión adecuada del proyecto para realizar una estimación, planificación y gestión de recursos, costes y riesgos. De esta forma, se ajustarán todos estos parámetros respecto a una expectativa que nos permitirá obtener una visión global del estado del proyecto en cualquier momento del proceso de desarrollo.

4.2.1 Estimación

Para realizar una adecuada estimación, es primordial establecer un valor aproximado del punto de historia en base a la cantidad de esfuerzo que supone un proyecto como este. Hay que tener en cuenta que esta aproximación no es necesaria en el marco de desarrollo Scrum. pero sí es importante para la proyección del coste de un proyecto. En este caso, se ha tomado la siguiente suposición:

1 punto de historia ≃2 horas de esfuerzo

De esta forma, se puede transformar la cantidad de esfuerzo de un *sprint* en horas-persona. Así pues, un *sprint*:

45 puntos de historia (promedio) = 90 horas-persona

Y si se divide entre los 15 días efectivos[4] del *sprint*, se tiene un total de 6 horas-persona de esfuerzo diarias, las cuales representan un trabajo equilibrado si se tiene en cuenta la naturaleza de doble mención que tiene este Trabajo de Fin de Grado. Evidentemente, y como ya se ha mencionado en la Sección 4.1.5, estas horas de esfuerzo diarias son una referencia, puesto que los primeros *sprints* son más ligeros para ir adoptando *velocity* y luego se ha llegado a abarcar incluso más puntos de usuario de los habituales, debido a que la situación era adecuada para una dedicación completa a este trabajo.

4.2.2 Planificación

La planificación se realiza en base a un número de *sprints*, cada uno con una razón propia para su existencia: su objetivo. En este proyecto se han alcanzado a realizar un total de siete *sprints*, con los siguientes objetivos:

1. **Búsqueda y organización de la ejecución del algoritmo de creación de taxonomías**

[4] Se contabilizarán los 15 días totales, sin tener en cuenta la naturaleza laborable o no de éste, puesto que la dedicación tuvo esta misma naturaleza a efectos prácticos.

2. **Despliegue del módulo de construcción**
3. **Desarrollo del soporte base de las operaciones mínimas de gestión de la aplicación**
4. **Gestión de peticiones al generador de taxonomías y visualización de resultados**
5. **Control del historial de llamadas al constructor de taxonomías**
6. **Obtención de datos y control del módulo de construcción de taxonomías**
7. **Búsqueda facetada por términos de la taxonomía**

4.2.3 Recursos

De nuevo, los recursos humanos están expresados en la Sección 4.1.5, siendo:

- Product Owner: Álvaro Barreiro García.
- Scrum Master: Javier Parapar López.
- Equipo de desarrollo: Juan Luis Filgueiras Rilo.

Los recursos materiales vienen proporcionados tanto por el alumno como por los directores. Los más destacables son:

- *Portátil Acer Aspire E15*: Con procesador AMD A10 7300, 16GB DDR3 de Memoria RAM, 500GB HDD. Utilizado principalmente para el desarrollo y pruebas, proporcionado por el alumno y valorado en **545€**.
- *Ordenador sobremesa*: Con procesador Intel i5-7400, 32GB DDR3 de Memoria RAM, 256 SSD, 1,5TB HDD. Utilizado para demostraciones, pruebas con «datasets» de la infraestructura multimodular de contenedores, proporcionado por el alumno y valorado en **970€**.
- *Servidor Jenkins*: Para la Integración Continua, proporcionado por los directores.
- *Servidor SonarQube*: Para la Inspección Continua, proporcionado por los directores.
- *Servidor Taiga*: Para la planificación Scrum, proporcionado por los directores.
- *Servidor GitLab*: Para la gestión del versionado del proyecto, proporcionado por los directores.

4.2.4 Costes

Para calcular los costes de los recursos humanos y materiales anteriormente declarados, se ha dispuesto del *Guia Salarial Sector TI Galicia* [18] y se añadirá el factor de coste social asociado a la producción del bien, multiplicando por 1,32: Se estimará que la dedicación de

Rol	Coste base (€/hora)	Factor de coste social	Coste total (€)
Desarrollador Junior	8,26	1,32	10,90
Director Proyecto	20,83	1,32	27,50

Tabla 4.1: Estimación de coste para recursos humanos del proyecto

cada Director de Proyecto es de 4 horas-persona por *sprint*, teniendo en cuenta *Sprint Planning*, *Sprint Review* y posibles reuniones «ad hoc». Por otro lado, se cuenta con 7 *sprints*, cada uno con 45 puntos de historia de media, resultando en un total de 80 horas-persona. Teniendo en cuenta dos Directores y un Desarrollador en el equipo: Para calcular el coste asociado de

Rol	Tiempo(h/sprint)	Nº miembros	Nº *sprints*	Coste (€/h)	Total (€)
Desarrollador	80	1	7	10,90	6 104
Director	4	2	7	27,50	1 540
				Subtotal	7 644€

Tabla 4.2: Desglose del coste de los recursos humanos

los recursos materiales, se asumirá unaa vida útil de los materiales que estén valorados de 48 meses, y se dividirá entre los meses totales de tiempo desde el comienzo del proyecto –20 de Febrero 2020– hasta fin de proyecto –26 de Octubre 2020–, de forma que: Este es el desglose

Equipo	Valor (€)	Vida útil (meses)	Uso (meses)	Coste total (€)
Portátil Acer Aspire E15	545	48	8,6	97,65
Ordenador Sobremesa	970	48	6,2	125,29
			Subtotal	222,94€

Tabla 4.3: Desglose de coste de los recursos materiales

total del coste de recursos humanos para el proyecto. En cuanto al coste material, las licencias son de software libre y los equipos de trabajo y servidores dispuestos no son específicos del proyecto, por lo que su vida útil escapa del contexto de este. Por lo tanto, el coste total será el coste de recursos humanos (establecido en 4.2), es decir, 7 866,94€.

4.2.5 Gestión de Riesgos

Un último aspecto esencial a la hora de realizar la planificación (y quizá el más importante para asegurar la estabilidad de lo desarrollado en secciones anteriores) es la gestión de riesgos.

Los riesgos son problemas potenciales que pueden ocurrir y afectan a diversos aspectos del proyecto, perjudicándolo y desviándolo de su planificación en esfuerzo, tiempo y coste.

Aunque las metodologías ágiles ya minimizan los riesgos a través de sus principios para la inspección y adaptación reactiva, es importante identificar, clasificar y, en caso necesario, prevenir, inspeccionar y asimilarlos.

Identificación y clasificación

El primer paso es identificar los posibles riesgos para el contexto de este proyecto, independientemente de si están bajo inspección de esta metodología o no. Además de identificarlos, se establecerá una probabilidad de aparición, un impacto en caso de manifestarse el problema, y una exposición temporal al mismo. Para identificarlos, se ha prestado atención a la complejidad del dominio y a la revisión de planificación y estimaciones: A pesar de

Código	Nombre	Probabilidad	Impacto	Exposición
R-DIS	Diseño deficiente	Media	Alto	Media
R-IMP	Implementación deficiente	Baja	Alto	Baja
R-REC	Insuficientes recursos	Baja	Medio	Media
R-TEC	Inadecuada tecnología	Media	Medio	Media
R-DOM	Dominio complejo	Media	Alto	Alta
R-REQ	Requisitos imprecisos	Baja	Alto	Baja

Tabla 4.4: Identificación y clasificación de riesgos

que ciertos riesgos mencionados (*R-DIS*, *R-TEC*, *R-DOM*, *R-REQ*) están, en parte, en observación por la metodología, es importante tomar medidas para todos los presentes pues son las amenazas más visibles dentro del proyecto.

Prevención

Para aquellos riesgos que presentan una exposición alta, es necesario establecer un plan de contingencia que permita minimizar la posibilidad de fracaso del proyecto. Estos riesgos son analizados desde el principio del proyecto, antes de comenzar el primer *sprint*, de forma que si aparecen temprano, la contingencia será mayor y se previene su aparición en etapas tardías del proyecto, donde su impacto puede suponer el fracaso del proyecto. A continuación, se presenta el plan de contingencia para el único riesgo de exposición alta:

- **R-DOM - Dominio complejo**: Teniendo en cuenta la complejidad del dominio, es posible que los conocimientos adquiridos sobre Recuperación de Información2.1 sean insuficientes para profundizar rápidamente en el dominio propuesto. Es por eso que se

hará verdadero hincapié en la revisión periódica de toda la documentación asociada al tema principal, sin dejar de centrar el foco en el dominio acotado desde un principio.

Seguimiento

En aquellos riesgos de exposición media, lo ideal es hacer un seguimiento periódico para impedir que su exposición pase a ser alta y, por lo tanto, amenazante cuanto más avanzado está el proyecto:

- **R-DIS - Diseño deficiente**: Se realizará una revisión del diseño en cada *sprint*, prestando atención a la evolución de este y minimizando la complejidad, refactorizando su estructura al menor indicio de dificultad en su entendimiento.
- **R-TEC - Inadecuada tecnología**: En este caso, la cantidad de nuevas tecnologías va supeditada a su propio desarrollo, por lo que será importante seleccionar tecnologías bien documentadas y robustas, con gran comunidad de desarrolladores y compatibles entre sí.
- **R-REC - Insuficientes recursos**: En este caso, los recursos forman parte del contexto limitante de este proyecto al que hay que ceñirse, por lo que se hará hincapié en limitar el esfuerzo para no sobrecargarlos.

Asimilación

Los riesgos cuya exposición es más baja simplemente son aceptados, ya que la gestión de riesgos también supone un coste y, en muchas ocasiones, no compensa la gestión sobre la probabilidad de aparición de estos riesgos:

- **R-IMP - Implementación deficiente**: En este caso, la implementación deficiente puede verse asociada al mal uso de las tecnologías novedosas. Sin embargo, la tecnología principal sobre la que se desarrollará el proyecto es bien conocida por el alumno, por lo que gran parte del proyecto irá bien apoyada en un proceso de implementación adecuado y acotado adecuadamente, además, por las tareas de cada *Historia de Usuario*.
- **R-REQ - Requisitos imprecisos**: Para este riesgo, los requisitos son gestionados a través de los directores cumpliendo esa función ausente del cliente, pudiendo establecer un objetivo a largo plazo con unos requisitos laxos que sirvan de guía desde el principio del proyecto (y se vean plasmados en el *Product Backlog*).

Capítulo 5

Desarrollo

En este capítulo se abordará todo lo relativo al proceso de desarrollo de este proyecto. Antes de entrar en detalle, se describirán los requisitos del proyecto resultantes –más adelante se verá como evolucionan a lo largo del desarrollo–. Después, se explicará la arquitectura y el modelo de datos del sistema. Por último se detallarán los detalles de cada uno de los *sprints*.

5.1 Análisis de requisitos

Un requisito es una necesidad o expectativa de un sistema, a menudo expresada por el cliente. Este requisito se obtiene, típicamente, de la interacción directa con el cliente, que expresará sus necesidades a través de requerimientos, para su posterior estudio. Los requisitos son representados en las diferentes historias de usuario, las cuales van evolucionando dentro del *Product Backlog* (como ya se ha comentado en la Sección 4.1.3, desde la agregación o el borrado de historias hasta el desgranamiento y modificación de las mismas.

En esta sección se exponen no sólo los requisitos funcionales del sistema, expresados de esta forma, sino que también se hará un estudio de los requisitos no funcionales, aquellos requisitos que son necesarios para el correcto funcionamiento del sistema, pero no se identifican con una unidad de comportamiento del sistema. Su estudio es relevante, tanto para la correcta elección de tecnologías –detalladas en el Capítulo 3– como de las decisiones arquitecturales.

5.1.1 Requisitos funcionales

Como ya se ha explicado previamente, los requisitos representan las funcionalidades a las que debe dar soporte el sistema. En la Tabla 5.1 se podrán observar el listado final.

5.1.2 Requisitos no funcionales

Para poder deducir las restricciones inherentes al sistema, se ha hecho un estudio de los requisitos no funcionales al comienzo del proyecto, para prevenir posibles inconsistencias en

ID	Historia de Usuario	Puntos
#2	**Como** desarrollador, **quiero** buscar una biblioteca de partida para la construcción automatizada de taxonomías **para** valorar su optimización tecnológica	18
#3	**Como** desarrollador, **quiero** disponer de un «pipeline» de ejecución en un único paso **para** su posterior integración en aplicaciones	18
#9	**Como** usuario, **quiero** disponer de un sistema en línea de generación de generación de taxonomías **para** permitir su uso desde aplicaciones e invocaciones remotas	33
#18	**Como** usuario, **quiero** poder autenticarme **para** identificar y crear proyectos propios, pudiendo exponer algunos de ellos al resto de usuarios	20
#15	**Como** usuario, **quiero** disponer de un asistente para la subida de corpus al «framework» de construcción de taxonomías	26
#39	**Como** usuario, **quiero** poder dar de baja cualquier proyecto que haya creado, junto a su información	17
#16	**Como** usuario, **quiero** ejecutar la petición de construcción de taxonomías, con parámetros de personalización	15
#17	**Como** usuario, **quiero** acceder a los resultados de las taxonomías creadas **para** poder explorar las distintas entidades incluidas y los parámetros por los que fueron escogidas	31
#54	**Como** usuario, **quiero** poder llevar un historial de llamadas al constructor de taxonomías	21
#58	**Como** usuario, **quiero** que el sistema sugiera nombres de taxonomías ya empleados en otras construcciones **para** sobrescribir resultados de taxonomías	16
#40	**Como** usuario, **quiero** conocer el grado de avance del proceso de construcción de taxonomías	30
#60	**Como** usuario, **quiero** descargar los resultados de la construcción **para** poder explotar los datos en bruto	28
#59	**Como** usuario, **quiero** tener a mi disposición una herramienta de búsqueda por facetas **para** visualizar los documentos que influyeron a la selección de los términos en cada paso	38
#72	**Como** usuario, **quiero** visualizar el detalle de los documentos resultantes de la búsqueda facetada	19

Tabla 5.1: Requisitos funcionales al final del proyecto

la aceptación del producto software. Los requisitos no funcionales extraídos son los siguientes:
La mayoría de estos requisitos surgen de la necesidad de una arquitectura software que per-

Código	Nombre	Descripción
RNF-1	Disponibilidad	El sistema debe ofrecer la funcionalidad cuando se necesita.
RNF-2	Tolerancia a fallos	El sistema debe operar según lo previsto ante la presencia de fallos.
RNF-3	Integridad de datos	El sistema debe prevenir cambios no autorizados en los datos del sistema.
RNF-4	Fiabilidad	El sistema debe ofrecer la funcionalidad como se necesita, minimizando el número de fallos y la probabilidad de que se de un fallo.
RNF-5	Usabilidad	El sistema debe fácil de entender y atractiva para el usuario, con curva de aprendizaje leve.
RNF-6	Escalabilidad	El sistema debe soportar una creciente carga de trabajo, siendo sus recursos fácilmente escalables ante este suceso.
RNF-7	Capacidad de recuperación	El sistema debe poder restaurarse a un estado deseado en caso de interrupción y fallo.

Tabla 5.2: Requisitos no funcionales del sistema

mita procesar cantidades ingentes de datos que varían en formato y generan más información (estructurada y no estructurada) con cierta tolerancia y facilite el proceso de construcción, que de por sí ya es laborioso. Además, existen diversos puntos de fallo que se detectarán a lo largo del desarrollo y dan lugar a algunas de las necesidades expresadas en la Tabla 5.2.

5.2 Arquitectura

Un correcto diseño de la arquitectura es de suma importancia para poder satisfacer los requisitos. Como se trata de la aplicación de una metodología ágil, la arquitectura ha ido evolucionando de un diseño inicial a uno más complejo y consolidado en función del nacimiento de nuevos requisitos, funcionales y no funcionales. En esta sección se mostrará el resultado final de la arquitectura, basado en el Modelo C4 [19], una aproximación para la realización de diagramas de arquitectura que expresan, a diferentes niveles, la interacción entre componentes y usuarios. Este modelo propone una vista en tres niveles de granularidad: una vista de contexto, que se centra en establecer los límites del sistema de este proyecto; una vista de contenedores, que delimita los elementos agrupables en unidades que sirven al mismo propó-

sito; y una vista de componentes, que permite indagar más en detalles internos de tecnología, para cada contenedor.

En la Figura 5.1 se puede observar el diagrama de contexto, donde se detallan los sistemas internos y externos, el punto de entrada del usuario, y las interacciones con el sistema de este proyecto.

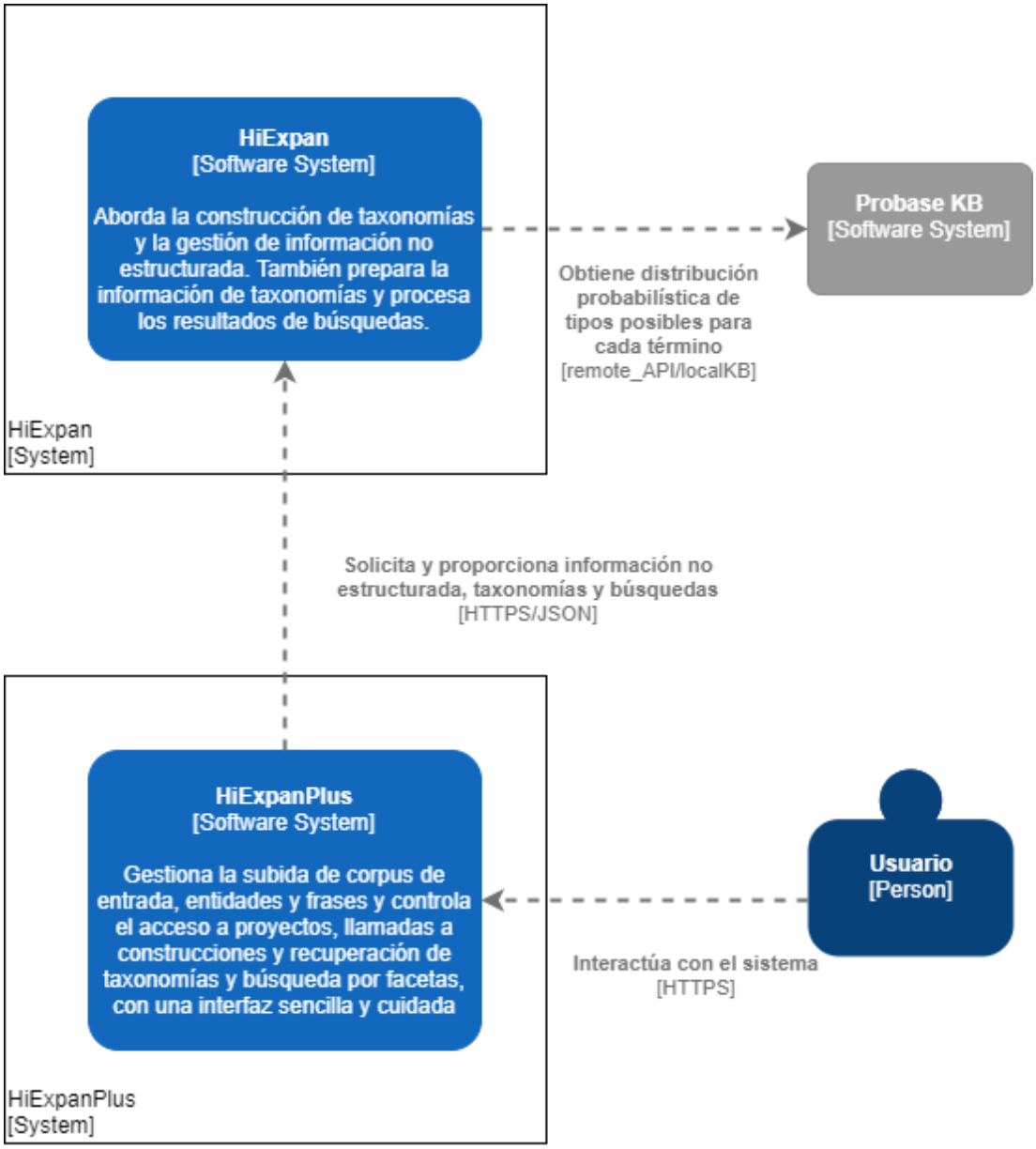

Figura 5.1: Diagrama de contexto de la aplicación completa

En el diagrama de contenedores (Figura 5.2) se puede esbozar la arquitectura del sistema a grandes rasgos, una arquitectura cliente-servidor distribuida que favorece la escalabilidad

(*RNF-6*) de recursos de forma selectiva –puesto que un servidor para el «front-end» no consumirá recursos de la misma forma que un servidor para el «back-end»– y la disponibilidad (*RNF-1*). El servidor de esta arquitectura, a su vez, es cliente de una API Rest que realiza la gestión de los datos no estructurados y la construcción de taxonomías. Esto se realiza por tres motivos:

- Se favorece la tolerancia a fallos (RNF-2), permitiendo que un fallo sea perceptible desde el «back-end» y poder preparar una respuesta estructurada ante este fallo, minimizando el impacto de un error en la API (que por naturaleza de su especificación, es el punto de fallo principal).

- Se garantiza de mejor forma la integridad de datos (RNF-3), permitiendo que el API trabaje con los datos no-estructurados y dejando al servidor la responsabilidad de los datos estructurados, que son más cercanos al usuario.

- Se busca que módulo para la construcción de taxonomías pueda ser accesible desde fuera de los límites de la aplicación, planteando un supuesto en el que varias aplicaciones realizan la explotación de datos desde diversas interfaces y con distintos propósitos, conteniendo este proyecto una de esas aplicaciones de explotación.

El diagrama de componentes de la Figura 5.3 detalla la disposición interna del servidor, una división en capas compuesta de controladores de peticiones REST, que se comunican con el cliente (y con la API RESTful) mediante patrones «Data Transfer Object» (DTO)[1], fachadas de lógica de negocio, y patrones «Data Access Object» (DAO)[2].

Por último, el diagrama de componentes del módulo remoto de construcción de taxonomías –Figura 5.4– es, a su vez, la expresión de una arquitectura «Pipe and Filter», caracterizada por el procesamiento y la transformación de los datos en cada uno de los componentes. Para complementar esta estructura de «pipeline» con un procesado lento y costoso, se ha añadido una facha para la selección del punto de entrada (se comentará con más detalle en la Sección 5.4.1). Por sencillez, no se busca entrar en más detalle en esta sección.

5.3 Modelo de datos

En la Figura 5.5 se detalla el modelo de datos de aquella información estructurada del sistema. Este modelo se corresponde al que manejará la aplicación servidor, para comunicarse eficientemente con la API y realizar la gestión de datos por parte del usuario.

[1] Patrón DTO: Patrón de arquitectura enfocado a la comunicación estándar entre procesos, facilitando la invocación remota de métodos.

[2] Patrón DAO: Patrón de arquitectura que segrega la lógica de negocio de la lógica acceso a datos, mediante objetos que encapsulan y definen el modo de conectarse a la base de datos.

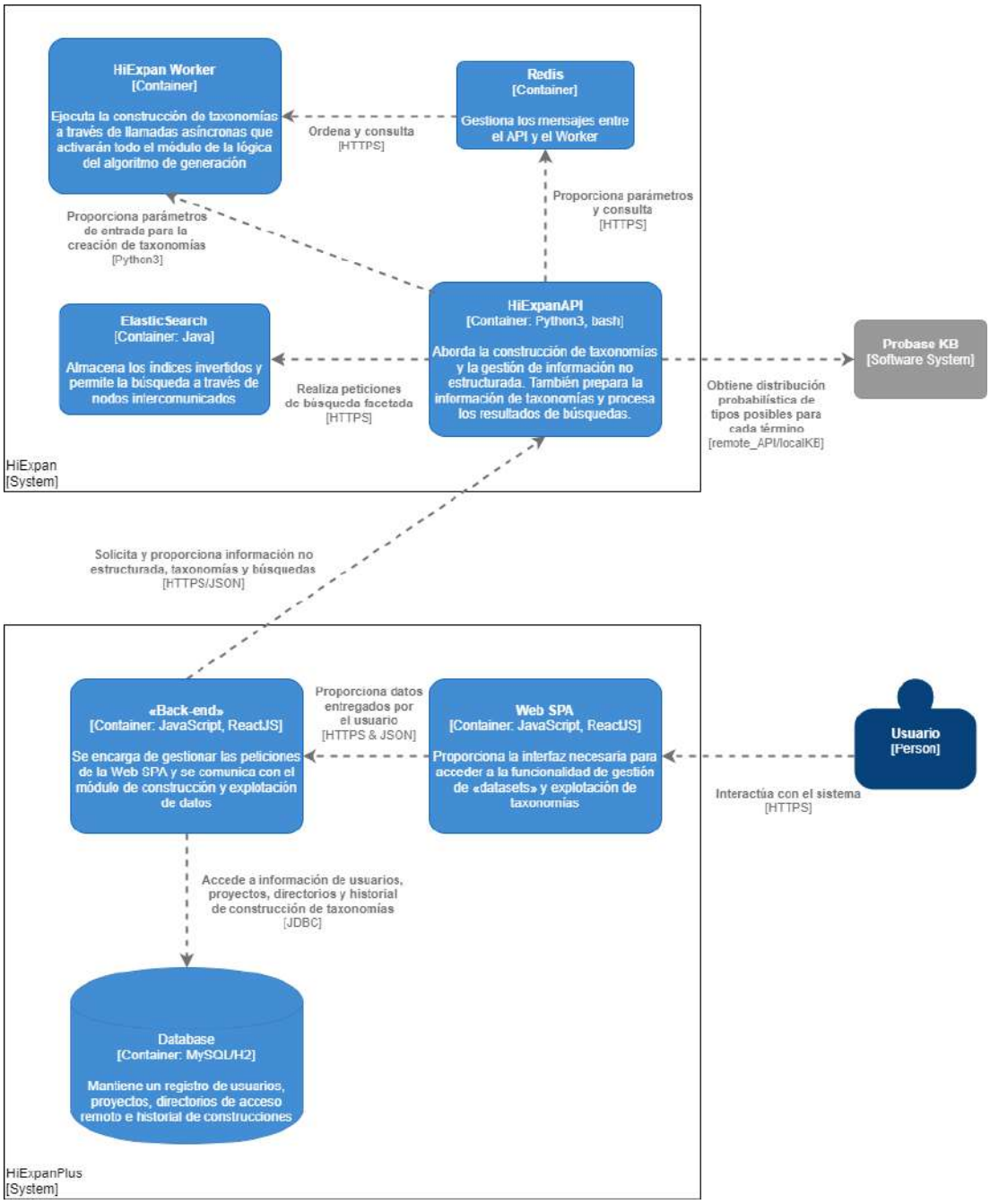

Figura 5.2: Diagrama de contenedores de la aplicación completa

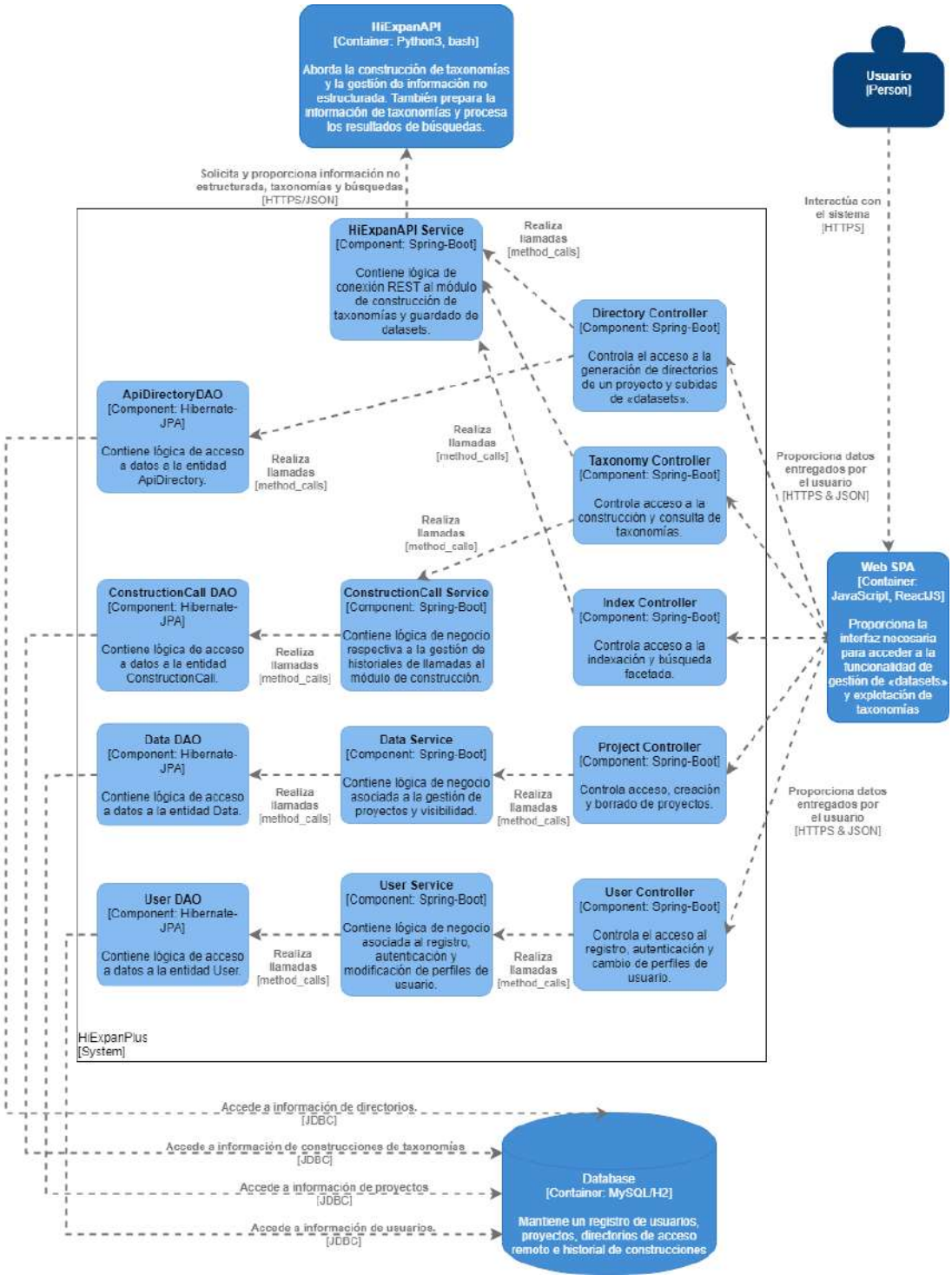

Figura 5.3: Diagrama de componentes de la aplicación WEB

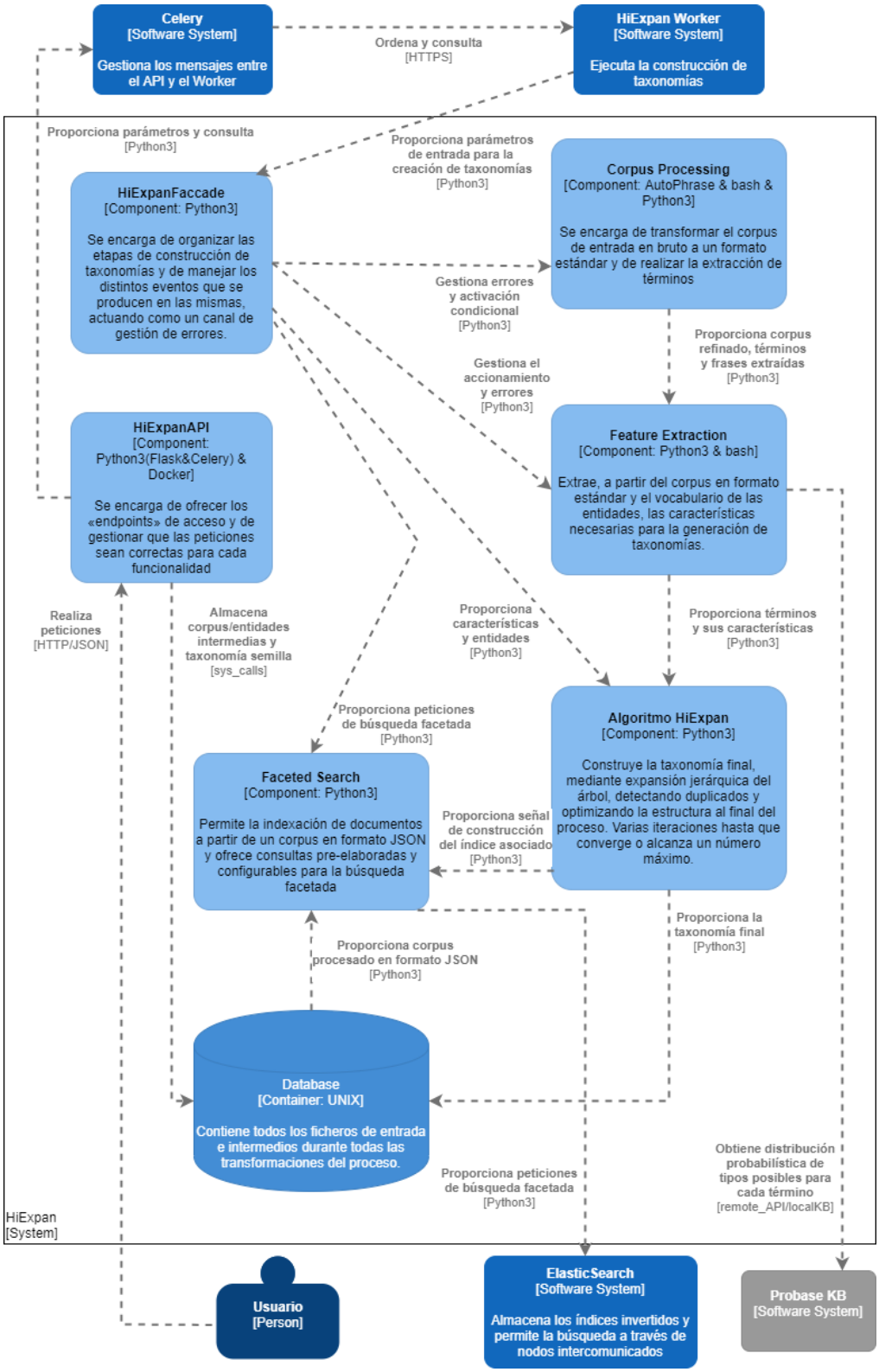

Figura 5.4: Diagrama de componentes del API de construcción

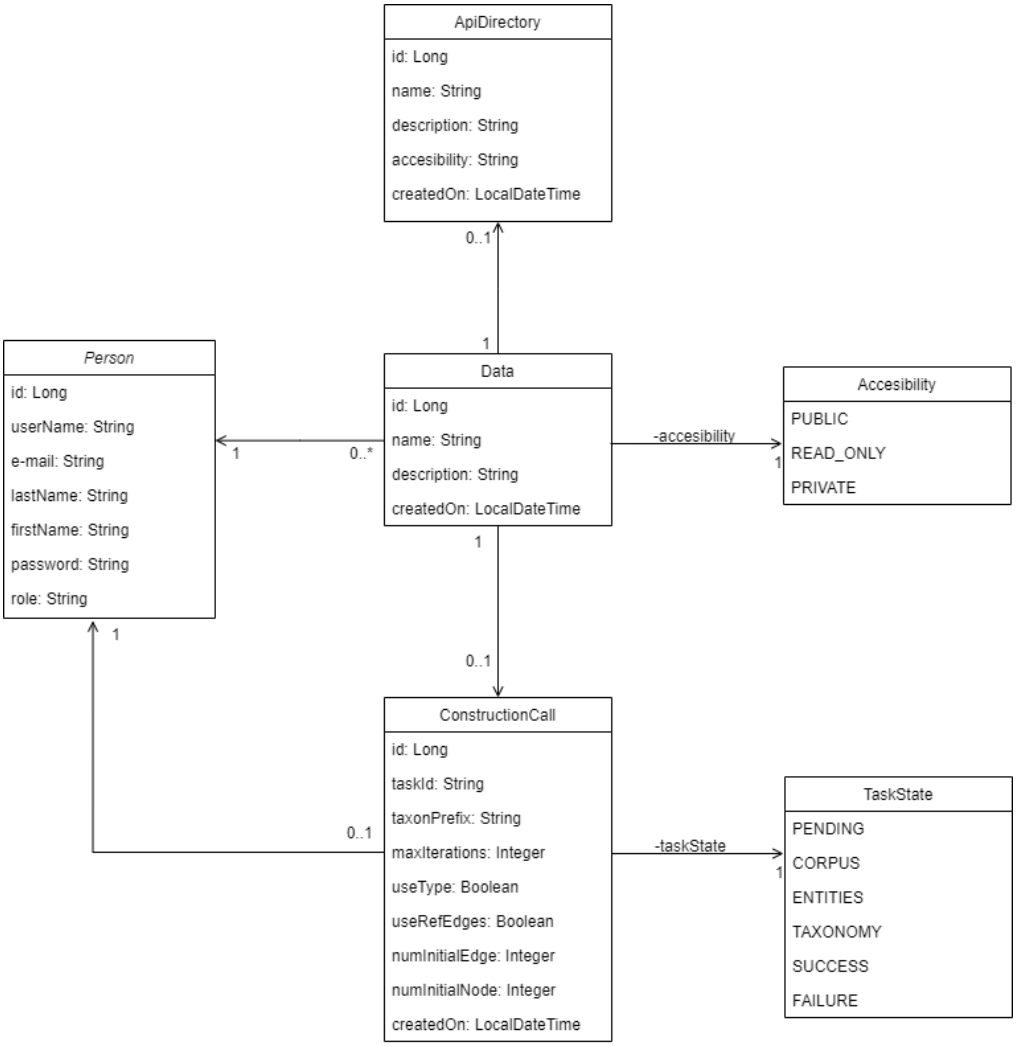

Figura 5.5: Modelo de datos de la Aplicación Web

5.4 Desarrollo

En esta sección se detalla, iteración a iteración, el proceso de desarrollo, explicando el abordaje de las historias de usuario, la evolución en la arquitectura, la evolución en el diseño y detalles del proceso de implementación. Al comienzo del proyecto, las historias de usuario establecidas inicialmente (Tabla 5.1.1) fueron las siguientes: #2 (18 puntos), #3 (18 puntos), #9 (33 puntos).

5.4.1 Sprint 1: Búsqueda y organización de la ejecución del algoritmo de creación de taxonomías

En este primer *sprint* el objetivo es abarcar las historias de usuario *#2:**Como** desarrollador, **quiero** buscar una biblioteca de partida para la construcción automatizada de taxonomías **para** valorar su optimización tecnológica* y *#3:**Como** desarrollador, **quiero** disponer de un «pipeline» de ejecución en un único paso **para** su posterior integración en aplicaciones*, con un total de puntos de historia objetivo de 36. Cabe recordar, como ya se comentó en la Sección 4.1.5, que los *sprints* empezarán abarcando menos puntos de historia al comienzo, para facilitar el aumentar la «velocity» en posteriores proyectos.

Para comenzar a abarcar estas historias de usuario, se procedió a realizar un estudio del estado del arte de las posibilidades, para poder tener un punto de partida a nivel de implementación. Tras una búsqueda de un algoritmo de construcción versátil –en lo que a dominio de aplicación se refiere–, se ha encontrado el «framework» HiExpan, que realiza la generación de taxonomías guiada por tarea a través de la expansión del árbol jerárquico (como ya se ha comentado en el Capítulo 2). Afortunadamente, este artículo cuenta con un proyecto de código con unas bibliotecas de código «ad hoc» para la comprensión y ejecución del algoritmo.

Una vez realizado un estudio de este «framework», se ha realizado un esbozo preliminar del flujo –Figura 5.6–, habiendo observado que la ejecución consiste en tres pasos esenciales que realizan compartición de datos de forma parcial. Primero, se realiza una extracción de frases y términos, que se etiquetarán gramaticalmente y se extraerán los términos clave (*Corpus Processing*). Segundo, se extraerán las diferentes características: «skip-patterns», «type features» (a través de llamadas a una API externa de extracción o de forma local con una base de conocimiento) y «term embeddings» *Feature Extraction*. Finalmente, y con todos estos datos extraídos y almacenados, se realiza la construcción de taxonomías, a través de una taxonomía semilla definida en un fichero y todos los ficheros intermedios resultantes de los dos primeros pasos *HiExpan*).

Cada uno de estos pasos contiene una configuración propia, utiliza bibliotecas diferentes, posee parámetros de invocación e incluso está pensado para su sustitución de componentes internos, de forma que cada «script» presente en los distintos pasos pueda realizar modifica-

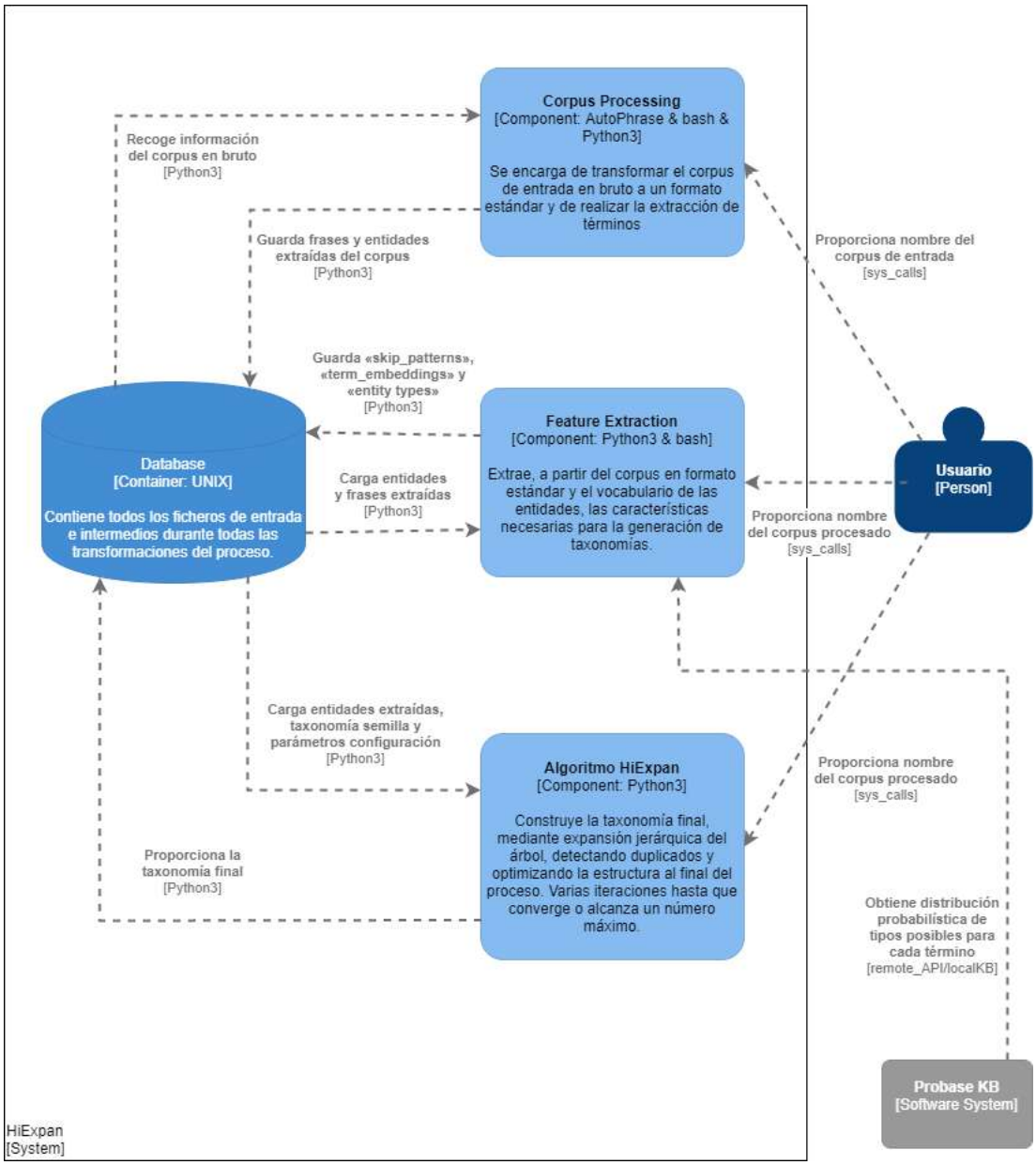

Figura 5.6: Estado preliminar del «framework» HiExpan

ciones internas, siempre y cuando las entradas y salidas entre componentes sean del mismo formato convenido. Para ello, se ha propuesto la aplicación del Patrón Factoría, pudiendo agilizar el proceso de construcción e incluso proponer alternativas en un futuro. Esta decisión está motivada por la constante entropía que sufre el estado del arte de la Recuperación de la Información, área que está en constante predisposición a los cambios.

Una vez refactorizado cada submódulo, es importante interconectarlos de forma que, en cuanto una de las partes haya realizado su trabajo, informe a la siguiente de que puede seguir la cadena de procesamiento (con toda la gestión de errores que se pueda dar de por medio). Esto no es más que la aplicación de una arquitectura de «pipeline», ya comentada. De esta forma, y estableciendo dónde tiene que acceder cada submódulo para obtener los ficheros que le corresponde procesar, se obtiene una ejecución en paso único que informa el avance de cada paso, otorgando un único punto de entrada.

No obstante, tras analizar el artículo, se detecta que también es posible que, a partir de ciertos archivos intermedios, un usuario pueda saltarse tanto el paso de *Corpus Processing* como el de *Feature Extraction*. Por lo tanto, y debido a la notable ganancia de tiempo que supone no realizar alguno de estos pasos, se decide utilizar un Patrón Fachada, que permita exponer al usuario una forma cómoda de invocar llamadas que le convengan con la opción de saltarse pasos o especificar parámetros de personalización. En concreto, se debe permitir:

- Saltarse el procesado del corpus, siempre y cuando existan los archivos intermedios resultantes de este paso.
- Saltarse la extracción de las características, siempre que ya existan las características de entrada de la construcción de taxonomías.
- El nombre del taxón resultado (por defecto *toy*).
- Número de iteraciones del proceso de expansión del árbol.
- La opción de realizar la expansión en profundidad como se relata en la Sección 2.4.4, o utilizar las aristas de dos alturas del árbol contiguas como referencia.
- La opción de activar o desactivar el uso de «type features» –ver Sección 2.4.2– como medida de similitud.
- El número de relaciones que se usarán como referencia para el cálculo de la expansión en profundidad de un nodo.
- El número de nodos iniciales que se escogerán para una expansión en profundidad.

Por lo tanto, no solo se ha de tener en cuenta que la entrada a este «pipeline» es un corpus y una taxonomía semilla, sino que se debe considerar la gestión de estos parámetros de forma

que el proceso de construcción sea personalizado y permita el ahorro de tiempo de elaboración.

Tras realizar todo el análisis anterior, se procede a realizar un diagrama de arquitectura inicial que reflejará el flujo del proceso. En este caso, solo se ha realizado un diagrama de contenedores –Figura 5.7–, que reflejará la arquitectura planteada. Posteriormente, se ha procedido a crear un proyecto de Python en PyCharm y a configurar un entorno virtual para la gestión de dependencias. A continuación, se ha creado un proyecto en Git y se ha vinculado a otro proyecto de Jenkins, a través de un «webhook»[3] que estará suscrito a los eventos de actualización del repositorio Git. Por cada rama de GitFlow que se actualice, se realizará una integración y un análisis posterior en SonarQube. Por el momento, solo se han realizado pruebas empíricas del «pipeline», con datos no estructurados sencillos de procesar (en concreto, un corpus «doc per line» de 5 000 líneas, donde cada línea es una documento del corpus) para comprobar el correcto funcionamiento y el cambio en caliente de parámetros de configuración a través de factorías.

Conclusiones del *sprint*

En este *sprint* el avance del diseño es decisivo, así como la decisión tomada a la hora de estudiar el algoritmo de construcción de taxonomías. Esta será la base que permitirá sustentar al resto de iteraciones con funcionalidad añadida que le de completitud funcional a este sprint y permita ampliar el alcance del sistema al objetivo deseado. Por el momento, se obtiene un incremento funcional que incluye un «pipeline» estudiado de construcción de taxonomías, configurable y que permita una ejecución en un único paso. El resultado del sprint deja el *backlog* con una sola Historia de Usuario, la nº #9:.

A partir de la revisión del *sprint*, surgen nuevas Historias de Usuario: la #15: ***Como** usuario, **quiero** disponer de un asistente para la subida de corpus al «framework» de construcción de taxonomías*; la #16: ***Como** usuario, **quiero** ejecutar la petición de construcción de taxonomías, con parámetros de personalización*; la #17: ***Como** usuario, **quiero** acceder a los resultados de las taxonomías creadas **para** poder explorar las distintas entidades incluidas y los parámetros por los que fueron escogidas* y la #18: ***Como** usuario, **quiero** poder autenticarme **para** identificar y crear proyectos propios, pudiendo exponer algunos de ellos al resto de usuarios*, que se añadirán al «backlog» con una estimación preliminar y una vaga descripción –pues se volverá sobre ellas en posteriores iteraciones–.

[3] «Webhook»: Retrollamada HTTP que reacciona ante ciertos eventos específicos.

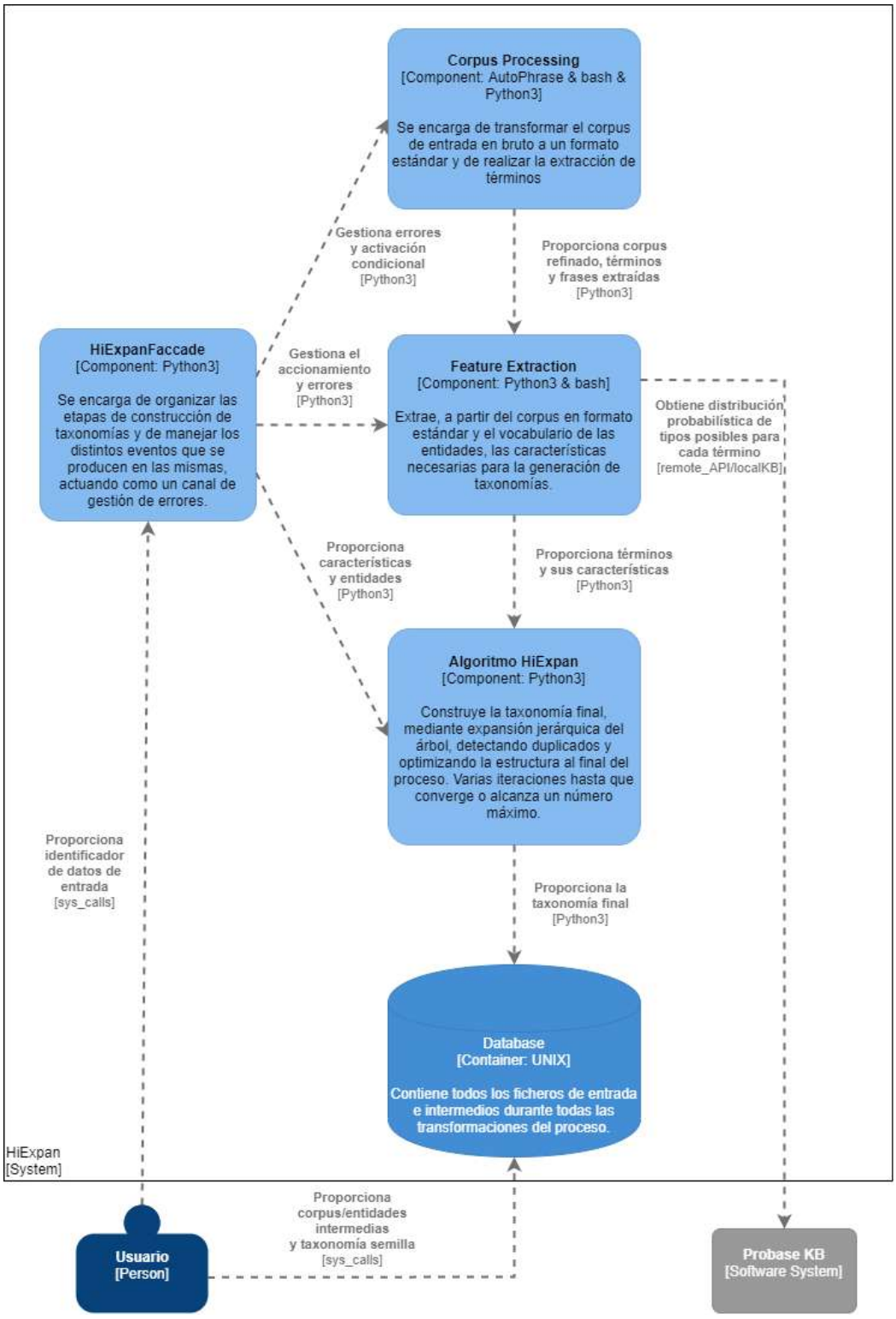

Figura 5.7: Diagrama de contenedores - Sprint 1

5.4.2 Sprint 2: Despliegue del módulo de construcción

El objetivo de este segundo *sprint* tiene un carácter más operacional, pero es esencial para la concepción del módulo de construcción como un API de acceso remoto. Primero, se decide establecer un diseño de la arquitectura de alto nivel, para observar la evolución buscada en esta iteración. Para cumplir este objetivo, se ha contado con la historia de usuario #9: ***Como** usuario, **quiero** disponer de un sistema en línea de generación de taxonomías **para** permitir su uso desde aplicaciones e invocaciones remotas*, que consta de 33 puntos, siendo el cómputo total de puntos de historia para este *sprint*.

En la Figura 5.8 se observa el planteamiento de la arquitectura, que busca establecer un contenedor propio que reciba peticiones REST mediante diferentes «endpoints» de una API expuestos para la recepción de peticiones. Para la subida de ficheros la lógica subyacente es sencilla, con la comprobación adecuada de formato, contenido y corrección de los ficheros que se reciben.

En cuanto a la creación de la taxonomía, dado que se trata de un proceso muy costoso y un usuario no debería estar dispuesto a gestionar una construcción que puede durar horas (e incluso días) de forma síncrona, se ha dispuesto de una biblioteca de gestión de tareas asíncronas para la APIs RESTful: Celery (ver: 3.1.5). A través de esta biblioteca y de un servidor de gestión de mensajes para comunicar API con uno o más nodos «worker» –encargados de la construcción de la taxonomía, en este caso–, se puede realizar múltiples construcciones sin comprometer el módulo de construcción para otras operaciones más elementales como la subida de archivos o la recuperación de taxonomías. Para esta última operación, se ha tenido que realizar una conversión del fichero de salida (la taxonomía) en un objeto JSON, fácil de comunicar. Este «work-around» es importante una eficiente comunicación con aplicaciones consumidoras y encamina a una favorable presentación de los datos.

Finalmente, para conseguir obtener una imagen estable del «pipeline», el API para acceder a datos y uno o más nodos «worker» con su servidor propio de comunicación, se ha utilizado Docker-compose. Docker-compose ha permitido englobar el despliegue de todas las unidades independientes como una única unidad, que ofrezca un punto de entrada de forma segura y éste de acceso al API, permitiendo que el resto de la arquitectura interna no se vea afectado por el diseño por contrato de la firma de los métodos del API.

Conclusiones del *sprint*

Para esta iteración, el alcance operacional obtenido está más definido y se obtiene una estabilidad en cuanto al módulo de construcción, permitiendo obtener una unidad funcional completa que se podría desplegar por sí sola. Esto establece un punto de partida para la realización de la aplicación web, que buscará comunicarse y explotar la información del módulo

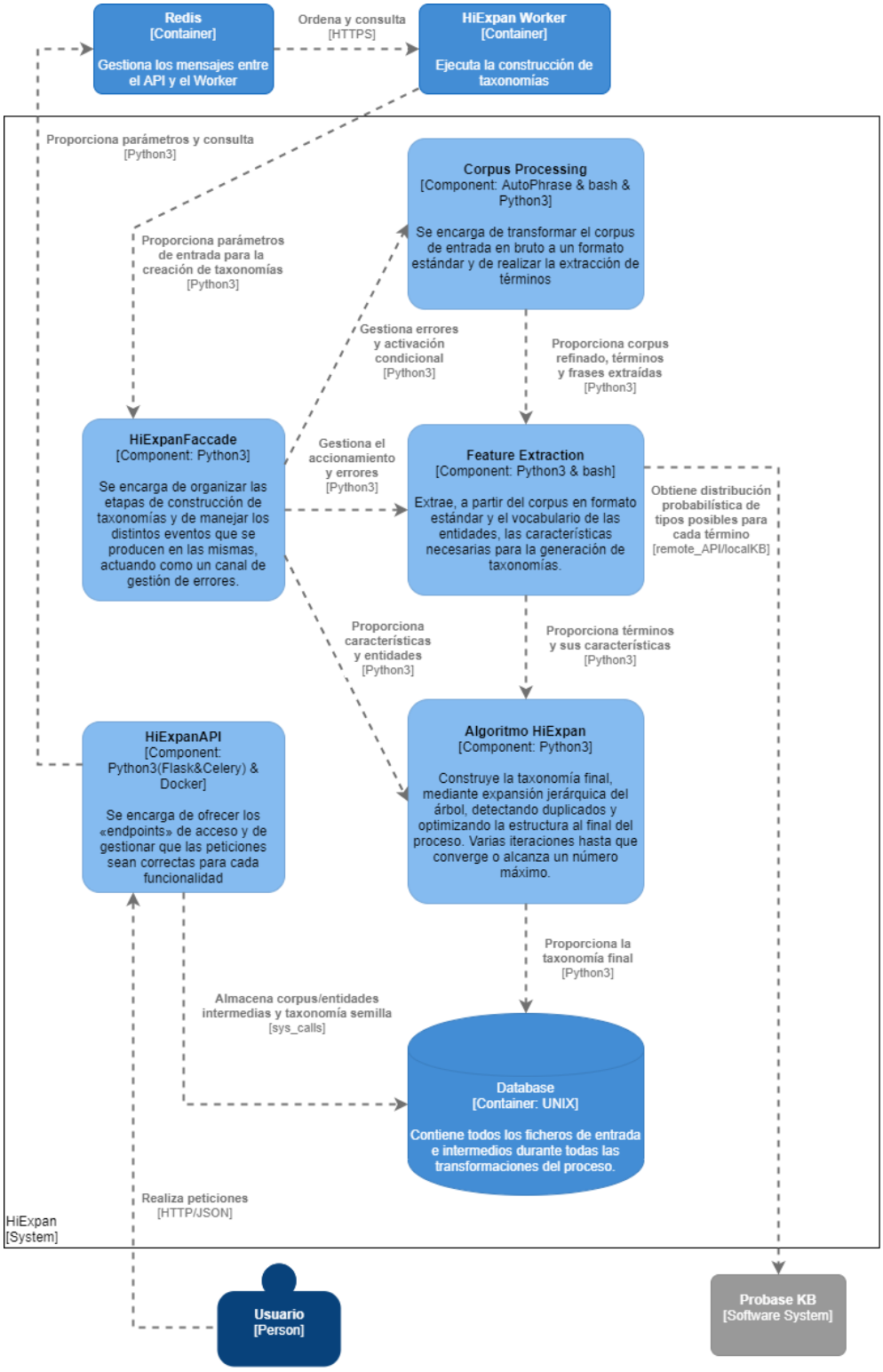

Figura 5.8: Diagrama de contenedores - Sprint 2

remoto que se acaba de consolidar en esta iteración. En la revisión del *sprint*, solo ha surgido una nueva necesidad, la historia de usuario #39: ***Como** usuario, **quiero** poder dar de baja cualquier proyecto que haya creado, junto a su información*, tras observar la importancia de que un usuario pueda dar de baja un «dataset» de su propiedad. Después, se procedió a refinar las historias de usuario ya existentes para una estimación concreta y la definición de subtareas para cumplir con cada uno de estos requisitos.

5.4.3 Sprint 3: Desarrollo del soporte base de las operaciones mínimas de gestión de la aplicación

Tal y como está expresado en la formulación del objetivo, se busca la realización del soporte de aquellas operaciones base previas a la construcción de taxonomías. Estas operaciones son las incluidas en las historias de usuario #18: ***Como** usuario, **quiero** poder autenticarme **para** identificar y crear proyectos propios, pudiendo exponer algunos de ellos al resto de usuarios*; #15: ***Como** usuario, **quiero** disponer de un asistente para la subida de corpus al «framework» de construcción de taxonomías* y #39: ***Como** usuario, **quiero** poder dar de baja cualquier proyecto que haya creado, junto a su información*; sumando un total de 63 puntos de historia. A pesar de la mayor carga de trabajo que supone este *sprint* respecto a los anteriores, estas historias cuentan con el condicionante de que no podrían encajar en la anterior iteración, por la pérdida de coherencia que supondría establecer un objetivo que englobe el objetivo del *sprint* anterior y alguna funcionalidad desvinculada de éste.

El diagrama que corresponde a este *sprint* es el diagrama de contenedores final, que expresa el funcionamiento en la Sección 5.2.

En el caso del «back-end», se procederá a crear un proyecto Maven en Eclipse a partir de un arquetipo para el desarrollo con Spring-Boot. A partir de este arquetipo, se crearán las entidades pertinentes, la configuración de seguridad para la autenticación, autorización y control de acceso –mediante «JSON Web Token» (JWT), un estándar para la creación de «tokens» de acceso que permiten la propagación de identidad y privilegios–, el acceso a datos y los servicios de lógica de negocio y los controladores. Antes de comenzar con la implementación, se realizó una configuración del POM del proyecto, de forma que admita dos perfiles, uno para el desarrollo y otro para producción y, por otro lado, dos perfiles asociados al uso de bases de datos. Para el uso de la aplicación en desarrollo, se usará la gestión de la persistencia a través de MySQL. Para el «testing» se utilizará H2, un sistema de gestión de bases de datos en memoria que permite ahorrar la configuración de una base de datos extra y la integración sencilla en Jenkins de cada rama. Por otro lado, se ha configurado el «plug-in» de JaCoCo en conjunto a JUnit para permitir realizar reportes de cobertura de código y su posterior uso en el análisis de la calidad técnica del proyecto en SonarQube.

Para el desarrollo en «front-end», se generó un prototipo a partir de npm, sobre el cual se

construirán los componentes principales para cubrir las necesidades del proyecto, basándose en los componentes de MaterialUI. Se han implementado una página de inicio, registro y autenticación Finalmente, una vez autenticado en la aplicación, se puede proceder a construir un proyecto donde se incluirá el nombre, una breve descripción del mismo y un marcador de visibilidad. Este marcador de visibilidad permitirá definir de qué forma los usuarios que no han creado un proyecto pueden interactuar con uno, bajo las siguientes premisas:

- Sólo el usuario que ha creado el proyecto puede modificar el «dataset» subido.
- Un proyecto de accesibilidad «PUBLIC» es accesible por cualquier usuario y permite la ejecución de construcción de taxonomías y consulta de cualquier taxonomía generada.
- Un proyecto de accesibilidad «READ_ONLY» es accesible por cualquier usuario, permitiendo la consulta de cualquier taxonomía, pero sin permitir la ejecución de construcción de taxonomías.
- Un proyecto de accesibilidad «PRIVATE» sólo será accesible desde el usuario que lo creó.

Una vez se ha creado un proyecto, se podrá ver el listado de proyectos a los que un usuario puede acceder, mostrando primero los más recientes. A través de este listado, se podrá visualizar el detalle y se podrá eliminar a través de un icono habilitado para ello. En el mismo lugar se hará una zona de subida de ficheros que permite subir, o bien un corpus en formato «.txt», o bien dos documentos: uno de frases en formato «.json» y otro de entidades asignadas a IDs en formato «.txt». En cuanto un documento es subido a través de alguna de estas opciones, se actualizará el detalle, que expresará la información de la última hora de subida y el nombre del usuario que subió la documentación.

Conclusiones del *sprint*

En este *sprint* se ha conseguido un incremento funcional sobre la aplicación final que servirá una forma cómoda de trabajar con el módulo de construcción de taxonomías. Este es un buen primer paso, y consolida una iteración muy completa con muchas funcionalidades básicas para asentar nuevos *sprints*. A partir de la revisión de este *sprint* se definen las historias de usuario #40: ***Como** usuario, **quiero** controlar en vivo cómo de completa está la construcción de mis taxonomías*, #60: ***Como** usuario, **quiero** descargar los resultados de la construcción **para** poder explotar los datos en bruto*. Posteriormente, se realizará un refinamiento de aquellas historias previstas para ser realizadas en un plazo próximo. Estas dos historias surgen de observar que un usuario, desde la aplicación web, debe poder observar el estado del proceso de construcción en el momento que decida consultarlo y, una vez termine, debería poder aprovechar la información de la taxonomía en bruto para su ámbito aplicativo.

5.4.4 Sprint 4: Gestión de peticiones al generador de taxonomías y visualización de resultados

Esta iteración producirá un incremento decisivo para un uso funcional completo del módulo de construcción de taxonomías como un servicio accesible desde la aplicación principal. Para alcanzar el objetivo anterior, se han abarcado las historias de usuario #16 ***Como*** *usuario,* ***quiero*** *ejecutar la petición de construcción de taxonomías, con parámetros de personalización* y #17 ***Como*** *usuario,* ***quiero*** *acceder a los resultados de las taxonomías creadas* ***para*** *poder explorar las distintas entidades incluidas y los parámetros por los que fueron escogidas*, para un total de 46 puntos de historia.

El primer paso consiste en la actualización del diseño de clases para plasmar el alcance del sistema una vez se entregue el siguiente incremento. En este caso, no se percibirán actualizaciones relevantes en el diagrama de la arquitectura respecto a la iteración anterior.

En lo que respecta al «back-end», se añade la lógica necesaria para la comunicación con el módulo de construcción a través de llamadas REST con la biblioteca Retrofit, empleada en la abstracción de este tipo de llamadas en una sencilla interfaz personalizable (ver: Sección 3.3.1). Después, se complementa con la creación de una entidad *constructionCall* que albergará la información registrada de una ejecución de construcción. A continuación, se realiza la lógica de negocio necesaria, con comprobación de errores en cuanto a la conexión del API, gestión de permisos respecto a quién realiza la petición y la visibilidad del proyecto y comprobación de que sólo se está realizando una única construcción a la vez por proyecto. Esto último se realizará para mantener la integridad de los ficheros intermedios respectivos a cada proyecto, teniendo en cuenta que puede haber inconsistencias en las entradas y salidas de dos ejecuciones, en las distintas del «pipeline», sobre el mismo directorio de ficheros.

Respecto al «front-end», se añade al detalle de un proyecto el acceso a la funcionalidad de construcción de la taxonomía. Los parámetros de configuración ya se han mencionado en la Sección 5.4.1, junto con un editor interactivo de la taxonomía «semilla», que garantizará que el usuario envía la información de forma consistente, al tratarse de una estructura compleja de datos. Este editor interactivo contará con el nodo raíz, al que se le pueden añadir nodos de forma incremental. Sobre un nodo creado, que no sea el nodo raíz, está la posibilidad de cambiar su nombre o borrarlo, en conjunto al árbol subyacente a ese nodo. Se validarán los datos de entrada, tanto en el lado cliente como en el lado servidor, de forma que el nombre de la taxonomía no está vacío, los parámetros numéricos son enteros positivos y la «semilla» no está vacía. Una vez la taxonomía se haya construido y se haya finalizado el proceso de construcción, se podrá consultar la última taxonomía generada de un proyecto, a través de un enlace dentro del detalle. Este enlace no forma parte de la funcionalidad principal, sino que es temporal, ya que su objetivo es completar un incremento completo listo para su despliegue, y fue necesaria una revisión «ad hoc» de los requisitos pendientes en el medio de la iteración

para aclarar la forma de recuperar las taxonomías construidas a partir del punto actual del proyecto. Respecto a la visualización de los resultados, es necesario buscar una forma intuitiva para el usuario en cuanto a la vista de una taxonomía, una estructura que, de por sí, es bastante compleja. Es por esto que el objetivo pasa por proponer una estructura en forma de árbol, que permita, de forma dinámica, expandir u ocultar los diferentes subárboles de los nodos, para focalizar la visualización de puntos de interés del usuario. Asimismo, también se muestran –en un cuadro que se verá al pasar el puntero del ratón por los nodos– las características de cada entidad de forma que muestra su ID, el nivel al que se encuentra y la «Confidence Score», o parámetro que define la certeza de que una entidad pertenece al «set» en el que se ha colocado.

Conclusiones del *sprint*

Como ya se ha mencionado, durante la ejecución de este *sprint* ha sido necesaria una revisión del *Product Backlog*, para definir la necesidad de mostrar el historial de todas las *constructionCall* que se han ido haciendo y así poder acceder a todas las taxonomías creadas para un proyecto. A su vez, se busca también conocer qué etiquetas se han utilizado para la construcción de taxonomías, de forma que no se borre ninguno de forma accidental. De esta forma, nacen las historias de usuario #54: ***Como** usuario, **quiero** poder llevar un historial de llamadas al constructor de taxonomías* y #58: ***Como** usuario, **quiero** que el sistema sugiera nombres de taxonomías ya empleados en otras construcciones **para** sobrescribir resultados de taxonomías*, de forma detallada y priorizadas en la lista, ya que son funcionalidades de gran aporte para el punto actual del proyecto.

5.4.5 Sprint 5: Control del historial de llamadas al constructor de taxonomías

Tras la revisión de la iteración previa, las tareas #54 y #58 son de suma importancia para añadir coherencia al flujo lógico de operación por parte del usuario. Por lo tanto, serán los componentes de este *sprint*, sumando un total de 37 puntos. Uno de los puntos fuertes de haber diseñado la *constructionCall* en la iteración anterior es que los cambios a realizar en el «back-end» son mínimos, permitiendo centrarse en la realización del frontal web. Para este caso, simplemente se ha implementado la lógica de negocio asociada a la obtención del historial de llamadas de una taxonomía, cuyos nombres tendrán enlace a las taxonomías vigentes para ver su detalle. A mayores, se ha diseñado la lógica para la obtención de todos los nombres distintos de taxonomías de un proyecto.

Respecto al «front-end», lo que se realizará es una lista –accesible desde el detalle de un proyecto– de construcciones con los enlaces pertinentes a cada detalle de taxonomía a través de enlaces. Por otro lado, a la hora de construir una taxonomía, se coloca un componente de

autocompletado para la recomendación o el recordatorio de las distintos nombres de taxonomías que ya han sido utilizados, de forma que no es necesario buscar en la lista del historial qué nombres se han utilizado.

Conclusiones del *sprint*

Este *sprint* ha resultado encajar perfectamente con la funcionalidad previa, debido a lo necesario que era cumplir el objetivo que se planteaba en esta iteración para entregar un incremento un poco más cohesivo respecto al soporte de las funciones que ofrece. Como resultado de la revisión del *sprint*, se ha ido esbozando – sin detallar – la historia de usuario #59: ***Como** usuario, **quiero** tener a mi disposición una herramienta de búsqueda por facetas **para** visualizar los documentos que influyeron a la selección de los términos en cada paso*, para no olvidar que forma parte de los objetivos iniciales y es el momento de empezar a tenerla en cuenta.

5.4.6 Sprint 6: Obtención de datos y control del módulo de construcción de taxonomías

Antes de desarrollar la búsqueda por facetas, es importante tener a disposición la descarga de las taxonomías y buscar una forma de solicitar información al nodo trabajador que realiza la construcción de taxonomías, para que pueda proporcionar información del progreso de la ejecución asíncrona. Es por esto que, en este *sprint*, se incluyen las historias de usuario #40: ***Como** usuario, **quiero** controlar en vivo cómo de completa está la construcción de mis taxonomías* y #60: ***Como** usuario, **quiero** descargar los resultados de la construcción **para** poder explotar los datos en bruto*, sumando un total de 58 puntos de historia.

Para poder abordar el problema de la consulta del estado, se ha hecho un estudio en profundidad de la biblioteca Celery (ver: Seccion 3.1.5), para poder averiguar si existe una forma de consultar al «worker», que ejecuta una tarea, su estado, a través de lo que se conoce como «message broker», el intermediario que hará de mediador entre ambas partes comunicándose con objectos JSON. De esta forma, desde el API del módulo de construcción se puede solicitar el estado de la tarea asíncrona, simplemente proporcionando un ID de tarea único y autogenerado. Para poder añadir el estado de completitud, se ha proporcionado a la fachada la posibilidad de modificar el estado entre llamadas de las etapas del «pipeline», gestionando un mensaje de error en caso de fallo irrecuperable. Tanto para el módulo como para la aplicación web, se han definido los estados: *CORPUS*, *ENTITIES*, *TAXONOMY*, *SUCCESS*, *FAILURE*, con distintos mensajes asociados. Mediante Retrofit, se realizará la consulta al «endpoint» expuesto de Flask que consultará al servidor y le avisará de notificaciones de estado junto con un valor de progreso porcentual. Aquellas llamadas que hayan terminado o fracasado persistirán su situación final en un nuevo campo del modelo de datos, junto al nuevo ID de tarea

asíncrona para la consulta. Para la descarga de archivos también se ha habilitado un punto de acceso para la devolución de taxonomías resultado y la lógica de negocio asociada.

Respecto al «front-end», se ha habilitado un botón que aparecerá para el usuario autenticado y mostrará las distintas tareas en proceso de ejecución o terminadas, si es la primera vez que observa la notificación de su finalización con éxito o fallo. También se mostrará una barra de progreso. Para la descarga de la taxonomía, se habilita un enlace sencillo en el detalle de la taxonomía.

Conclusiones del *sprint*

Para esta iteración, la lógica asociada a la consulta del estado de las llamadas ha supuesto bastante esfuerzo que, afortunadamente, se ha ceñido a los puntos de historia estimados, que iban con holgura por el desconocimiento a la hora de abordar este requisito. Por otro lado, se revisa y detalla la historia de usuario que queda, relativa a la búsqueda facetada (#59). Asimismo, se ha decidido que debería ir acompañada de un acceso al detalle de un documento, motivado por la necesidad de plasmar información más diversa en un listado a la hora de ejecutar la búsqueda y la comodidad de no mostrar todo el contenido de un documento dentro de un listado, pues no favorece a la experiencia de usuario. De este modo, surge la historia de usuario #72: ***Como** usuario, **quiero** visualizar el detalle de los documentos resultantes de la búsqueda facetada.*

5.4.7 Sprint 7: Búsqueda facetada por términos de la taxonomía

Una vez se ha conseguido, a través de todos los incrementos funcionales previos, obtener el objetivo principal de este proyecto –desarrollar una aplicación multi-modular que soporte la generación, exportación y visualización de taxonomías– sólo queda desarrollar este último incremento funcional, basado en la búsqueda por facetas sobre un índice invertido construido sobre el mismo corpus que ha servido de entrada para la obtención de la taxonomía. De este modo, en este *sprint* se han incluido las dos historias de usuario restantes: la #72 –expresada en la anterior iteración– y la #59: ***Como** usuario, **quiero** tener a mi disposición una herramienta de búsqueda por facetas **para** visualizar los documentos que influyeron a la selección de los términos en cada paso*, con un total de 58 puntos de historia para esta iteración.

La búsqueda facetada estará basada en los términos del propio árbol resultado, pero también contemplará la búsqueda a través de texto. En el resultado, se quiere mostrar los documentos en forma de listado, en conjunto a distintas agregaciones por cada entidad que exista en la taxonomía. De esta forma, será necesario almacenar qué entidades de la jerarquía final se encuentran en los documentos del corpus, y así poder filtrar por términos y agrupar los resultados por las entidades que se encuentren en el cuerpo del documento. Para la búsqueda

del detalle completo de un documento, simplemente bastará con proporcionar los IDs en el listado y permitir la búsqueda por ID dentro del índice.

Para poder abordar adecuadamente esta funcionalidad, se ha abordado primero la realización de los cambios de diseño relativos al módulo de construcción. Lo más importante es acomodar la arquitectura de una forma que no impacte negativamente a lo que ya está organizado. Se ha hecho uso de Elasticsearch (ver: Sección 3.2.2), un software de búsqueda basado en Apache Lucene, que aumenta el nivel de abstracción que ofrece y es fácil de desplegar de forma modular. Generalmente, se suele desplegar más de una máquina, a modo de «cluster» de nodos, que permita no sólo la resiliencia respecto a caídas, sino que pueda distribuir el trabajo de indexación y búsqueda en función de la carga de trabajo de cada nodo. Afortunadamente, la documentación oficial de Elastic[4] ya pone a disposición imágenes estables de este servidor, por lo que la adaptación dentro de la arquitectura del servidor no se verá afectada. De este modo, se añaden tres nodos de Elasticserch intercomunicados a la composición de Docker existente. La razón de conjuntar el servidor de indexación y búsqueda con el de construcción de taxonomías es la coherencia. La indexación es, de forma previsible, un paso condicionado por la construcción satisfactoria de una taxonomía, de forma que sólo tiene sentido crear un índice de aquellos corpus que dieron lugar a una ejecución correcta del generador de ontologías.

Una vez realizada la configuración y comprobado que el nodo principal es accesible y está operativo, se realizan los cambios en el módulo de construcción para la indexación en caso de éxito. Primero, se reacondiciona el corpus en el paso inicial, de forma que devuelva un corpus en formato JSON listo para su indexación. Más adelante, en el paso de extracción de entidades a partir de las frases, se creará otro fichero intermedio que llevará el registro de las entidades y un listado de documentos en las que aparecen. Otra alternativa consistiría en esperar a la obtención de los nodos de la taxonomía final y realizar una búsqueda de esos nodos en el conjunto de frases para extraer los documentos que los contienen. Sin embargo, esta segunda aproximación no es muy acertada, puesto que los nodos de la taxonomía sufren ciertas transformaciones –«stemming»[5], cambios en mayúsculas y minúsculas, términos compuestos unidos por una «_» en vez de un espacio, etc.– respecto al término original almacenado en las frases o los documentos, haciendo que los nombres no coincidan y la información se vea desvirtuada.

El proceso es el siguiente: Primero se extraen los términos a analizar de las frases (funcionalidad que se aprovecha del módulo en el estado actual). Durante esta extracción, el término pasa por un proceso de transformación (como el que se ha mencionado previamente) para dar lugar al nombre estable de la entidad candidata. En este momento, se le asigna un ID a

[4] https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html

[5] «Stemming»: En Recuperación de la Información, derivación de un término a su raíz léxica, realizado en el paso de análisis de un sistema de este ámbito.

la entidad o, si bien ya existe, se incrementa el número de ocurrencias de la entidad. En este punto es cuando se asigna el ID del documento al listado por ID de entidad, de forma que se trazará de qué documento se ha obtenido la entidad en cuestión. De esta forma, no importan las transformaciones que sufra un término, solo importa el resultado al que da lugar después de las transformaciones y añadir el documento al listado.

Después de llevar la traza entidad-documento se realizarán los métodos relativos a la construcción del índice, de forma que se almacenarán dos campos: El cuerpo del documento y una lista de palabras clave, que serán los términos de la taxonomía final. La subida al índice se hará por bloques de documentos, a través de un API de Elasticsearch que permite la subida múltiple de documentos, incrementando la velocidad de indexación. Para la búsqueda, se hará una consulta conjuntiva en base a un texto opcional, que filtre los resultados por las distintas facetas, o nodos de la taxonomía. A su vez, devolverá las facetas de los documentos resultantes, de forma que se pueda seguir profundizando.

Finalmente, se habilitan los «endpoints» del API del módulo de construcción para proporcionar transparencia y mantener el principio de punto de acceso único para las funcionalidades de este módulo. A continuación se amplían los servicios y los controladores del «back-end» –las entidades del modelo de datos no requieren de ampliación, por ser información no estructurada ya almacenada sobre la que se busca–, de forma que se proporcionan los puntos de acceso para el «front-end» que permitan acceder al módulo y construir la interfaz de búsqueda. El componente de búsqueda va asociado al componente de visualización, como es de esperar. Desde la taxonomía se pueden añadir los distintos nodos según se navega y, adicionalmente, un texto de búsqueda. El resultado será en formato lista con paginación de una búsqueda, ordenada por el «score» de cada documento computado respecto al texto de búsqueda –el filtrado por facetas no proporciona «score» por ser una decisión binaria–. A la izquierda se encuentra una lista con las agregaciones devueltas y una opción para añadirlas y aumentar la navegabilidad. Asimismo, se muestran las etiquetas por cada documento de la lista y, de modo global, el «score» más alto y el número total de documentos que satisfacen la búsqueda. Si se accede al enlace «Ver más…», se mostrará el documento completo, donde aparecerá el ID del documento y su contenido (cuerpo y etiquetas).

Conclusiones del *sprint*

Esta iteración fue la más multidisciplinar, en cuanto a la diversidad de cambios que hizo falta implementar para satisfacer los requisitos. En consecuencia, el resultado es muy satisfactorio, dando lugar a un incremento muy completo y de utilidad para la comprobación de resultados. Asimismo, es útil para detectar anomalías en el corpus o incoherencias en la taxonomía, de forma que permita el estudio de calidad de las entradas y suponga un aliciente a más y mejores construcciones, produciendo resultados cada vez más enriquecidos por el

conocimiento del usuario. Como se trata del último *sprint* dentro del marco de este proyecto, la revisión del mismo consistirá en la comprobación del correcto funcionamiento y la prueba con «datasets» para verificar la correcta consistencia. Como resultado, el balance final del proyecto es satisfactorio, dando lugar a una aplicación completa en todos sus incrementos y con oportunidad de uso, puesto que el planteamiento es novedoso y proporciona una oportunidad de avance en el dominio de la construcción de taxonomías.

5.4.8 Calidad del proyecto

Un aspecto omiso en el planteamiento del desarrollo, pero que ha sido una constante a lo largo del mismo, es el concepto de calidad técnica, que se ha podido medir a través de las herramientas de integración e inspección continuas: Jenkins y SonarQube (ver Secciones 27 y 29). A lo largo del proyecto, se han comentado aspectos de la configuración, pero no se ha incidido en la realización de pruebas. Esto se debe a que, para todas las iteraciones, se han realizado pruebas que se han ido integrando a través de Jenkins en cada actualización del repositorio Git, y, a su vez, SonarQube exige una calidad del código constante, estableciendo objetivos mínimos sobre el código nuevo (cobertura, duplicaciones, bugs, vulnerabilidades...) cada vez que se realiza una integración satisfactoria de Jenkins. En la Figura 5.9 se puede observar el gráfico de las últimas integraciones con Jenkins, de forma que se puede ver la evolución progresiva de las pruebas realizadas y el total de las mismas en el punto final del proyecto. En el eje horizontal se puede ver el número de integración correspondiente.

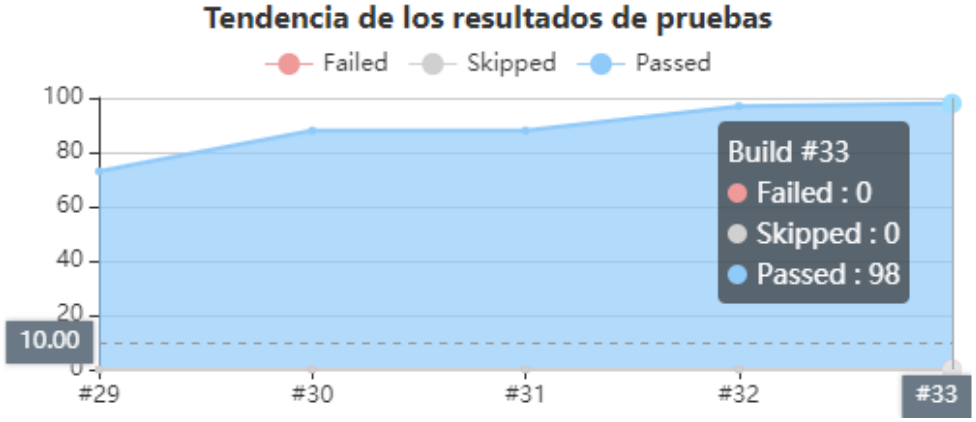

Figura 5.9: Resultados de ejecución de Jenkins al final del desarrollo

En la figura 5.10 se puede observar la vista general de análisis del código donde se pueden apreciar buenas métricas respecto a la calidad técnica global del proyecto.

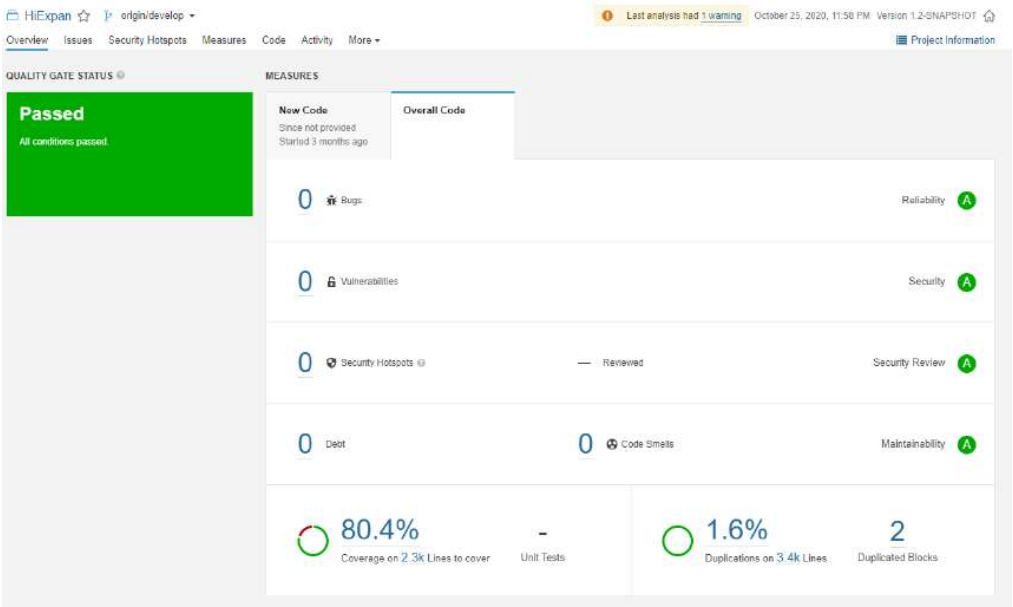

Figura 5.10: Resultados de SonarQube al final del desarrollo

Figura 5.11: Gráfico «burn-down» del total de sprints, en Taiga

Capítulo 6

Conclusiones y trabajo futuro

En la situación actual, las taxonomías son de gran aporte en contextos muy diversos dentro y fuera del área de la Recuperación de la Información. Sin embargo, su construcción supone un despliegue de esfuerzo, tiempo y coste que no es trivial, teniendo en cuenta la importancia del conocimiento experto para construir una buena jerarquía dependiente del dominio. Este proyecto es un punto de inflexión para la construcción automatizada, ofreciendo una solución más completa que otras ya existentes, puesto que el proceso de construcción es guiado por tarea, representada por una taxonomía inicial incompleta que se expandirá de forma jerárquica. Del mismo modo, este sistema ofrece un modo de interacción muy dinámico con el usuario.

6.1 Conclusiones

Este desarrollo tenía un objetivo muy amplio: desarrollar toda una investigación y desarrollo para la construcción y explotación de taxonomías y elaborar una aplicación marco que permita trabajar de forma eficiente con ellas. En este aspecto, se han desarrollado todas las funcionalidades que se habían esbozado desde el principio, cumpliendo con los objetivos preestablecidos. Por un lado, se buscaba partir de un algoritmo de construcción automática de taxonomías, entenderlo y adaptarlo para poder ejecutarlo. Por el otro, se necesitaba una aplicación que permitiese aprovechar este módulo de construcción, permitiendo proporcionarle datos y explotar los resultados de una forma sencilla para el usuario, gestionando estos datos a través de proyectos de construcción y explotación, con cómodas interfaces y un diseño fresco.

Uno de los mayores retos de este proyecto fue congeniar todas las tecnologías entre sí, no sólo por el aprendizaje requerido para su uso, sino que fue muy importante que el diseño de la arquitectura no permitiese dejar cabos sueltos en cuanto a su disponibilidad. Si bien la complejidad es alta, el resultado es satisfactorio, puesto que la disponibilidad es a medida entre cada tecnología diferente. Por otro lado, el dominio de este proyecto, siendo totalmente desconocido para el alumno ha permitido profundizar en lo que supone la construcción de

taxonomías para el estado del arte actual.

Gracias a este proyecto, se ha podido entender desde una perspectiva más práctica todo el proceso educativo que ha supuesto el paso por el grado. De la mención de Computación, se amplían conocimientos de Recuperación de la Información, demostrando que es un campo al que aún le queda mucho en lo que profundizar y que tiene diversidad de ámbitos de aplicación, además de adquirir conocimientos técnicos sobre el uso de tecnologías modernas para las Ciencias de la Computación y formas avanzadas de construcción de índices y motores de búsqueda, desde una perspectiva más operativa. De la mención de software, se ha comprendido mejor la dificultad de abordar la gestión de un proyecto, desde la elección de una metodología ágil, hasta el cálculo de costes, estimación y gestión de riesgos. Este aspecto es relevante para interiorizar todo el proceso, que es complejo y requiere de mucha experiencia, incluso con conocimiento teórico. Del mismo modo, es necesario tener una visión clara del avance del desarrollo y una proyección previa de cómo se va a dar este desarrollo, gracias a la fase de diseño. Asimismo, inspeccionar la calidad del código, tanto a nivel funcional como a nivel técnico, se vuelve un aspecto más interesante cuando se participa de lleno en el proceso.

6.2 Trabajo futuro

Respecto al trabajo futuro, este proyecto tiene diversos frentes abiertos. Por un lado, se puede ampliar el módulo de construcción de taxonomías, para añadir diferentes estrategias de construcción de forma dinámica a través de la aplicación, permitiendo probar distintas formas de construcción que pudieren surgir en un futuro y comparar cualitativamente sus resultados. Por otro lado, se puede buscar ofrecer más funcionalidades a través del módulo de construcción y explotación, permitiendo operaciones más complejas como la obtención de una taxonomía en los distintos pasos de ejecución intermedios del «framework» HiExpan. Asimismo, y relacionado con lo anterior, se puede incluir una llamada para la comparativa de resultados de un mismo proyecto, teniendo en cuenta las diferencias de parámetros y las diferencias en los distintos pasos de ejecución, pudiendo observar la influencia de los parámetros de configuración en cada uno de estos pasos intermedios.

Respecto al marco de la aplicación, una proyección de ampliación podría ser la modificación de una taxonomía devuelta, de forma que permita correcciones que favorezcan su uso, puesto que los resultados pueden estar sujetos a pequeñas incoherencias. Esta modificación puede suponer un cambio importante para poder usar satisfactoriamente la taxonomía en ámbitos reales. A su vez, también se valora la implementación de una funcionalidad que permita la carga de una taxonomía de referencia con propósitos como la comprobación de versiones o la certificación de que una taxonomía está bien construida respecto a una taxonomía de calidad que sirva como referencia.

Apéndices

Apéndice A

Documentación de planificación y resultados de implementación

En este anexo, se mostrará el aspecto visual de la aplicación y los resultados de las iteraciones en la aplicación Taiga, así como dos diagramas de clase que se consideran resultados directos entregables de los dos primeros *sprints*:

A.1 Sprint 1

La estimación de las historias de usuario corresponden a la Figura A.1.

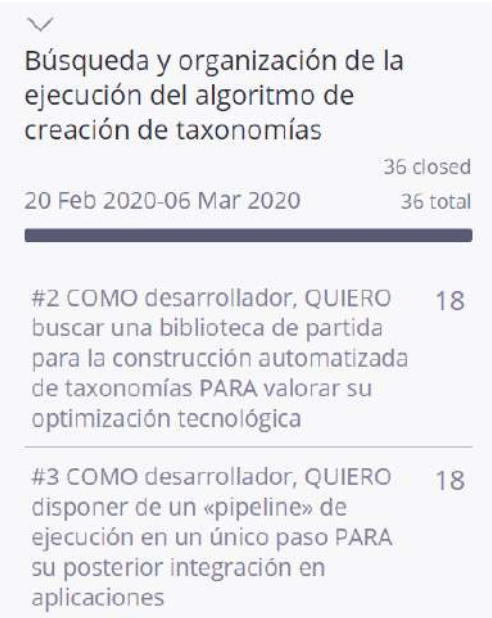

Figura A.1: Estimación Sprint 1

El detalle del primer *sprint* con todas sus subtareas se corresponde a la Figura A.2.

Figura A.2: Detalle Sprint 1

La Figura A.3 muestra lo comentado respecto al uso del patrón factoría y del patrón fachada para la selección configurable de clases procesadoras de información.

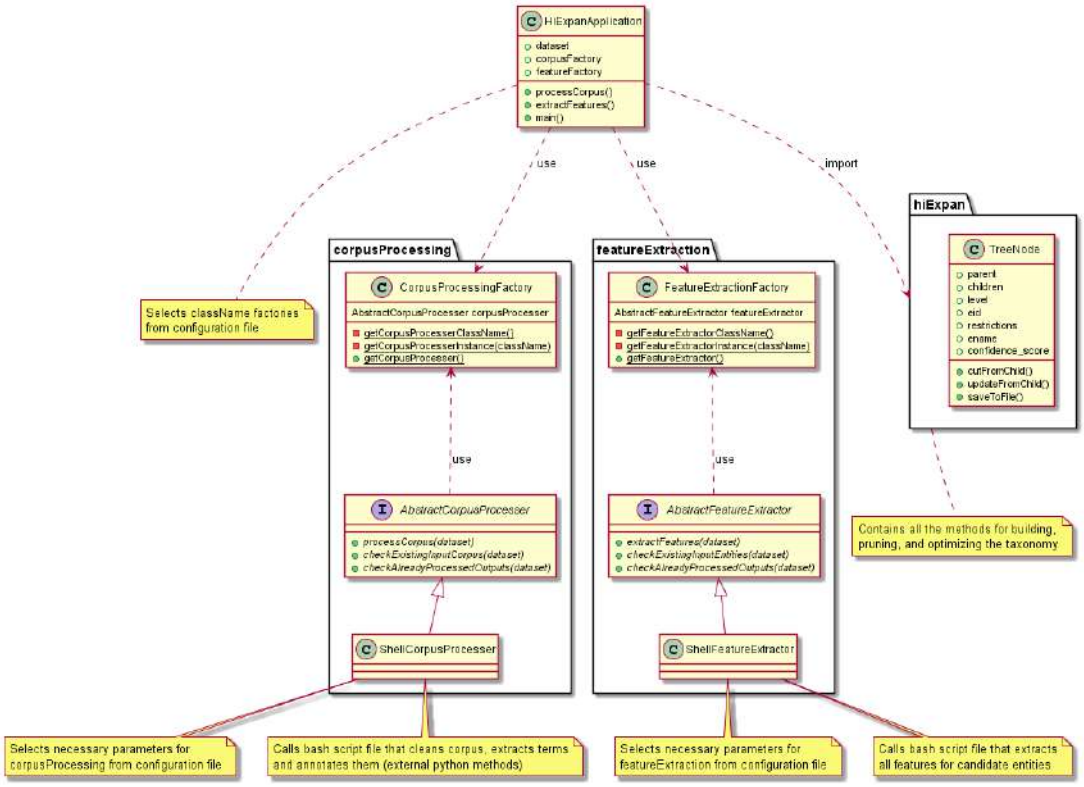

Figura A.3: Aplicación del patrón factoría y fachada en el módulo de construcción de taxonomías

A.2 Sprint 2

La estimación de las historias de usuario corresponden a la Figura A.4

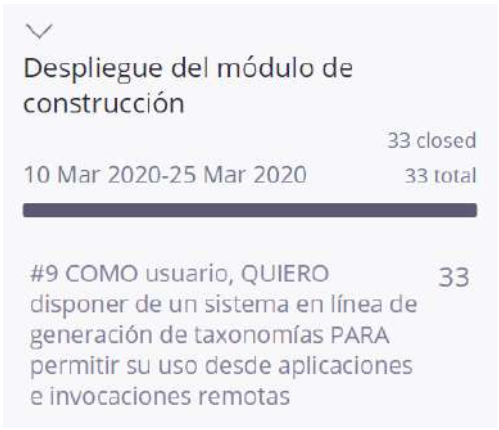

Figura A.4: Estimación Sprint 2

El detalle del segundo *sprint* con todas sus subtareas se corresponde a la Figura A.5. Para este *sprint*, se puede destacar el surgimiento de una tarea sin historia asignada (la configuración de Jenkins y Sonar), que ha surgido de una reunión «ad hoc» entre directores y alumno. De esta forma, también se tiene en cuenta como parte del desarrollo y su esfuerzo de configuración relacionado.

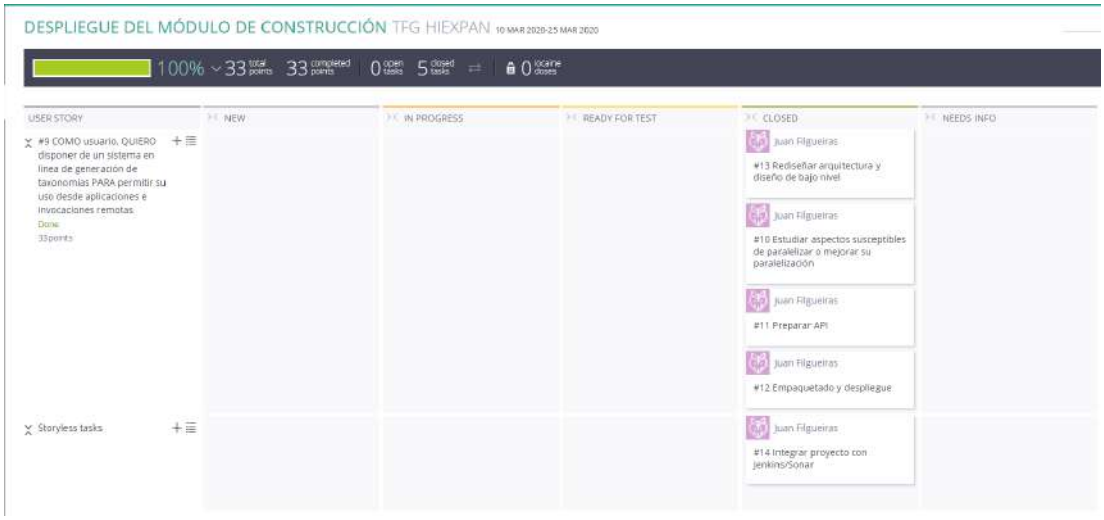

Figura A.5: Detalle Sprint 2

La Figura A.6 muestra el diagrama de clases con la nueva clase API que gestionará la lógica subyacente.

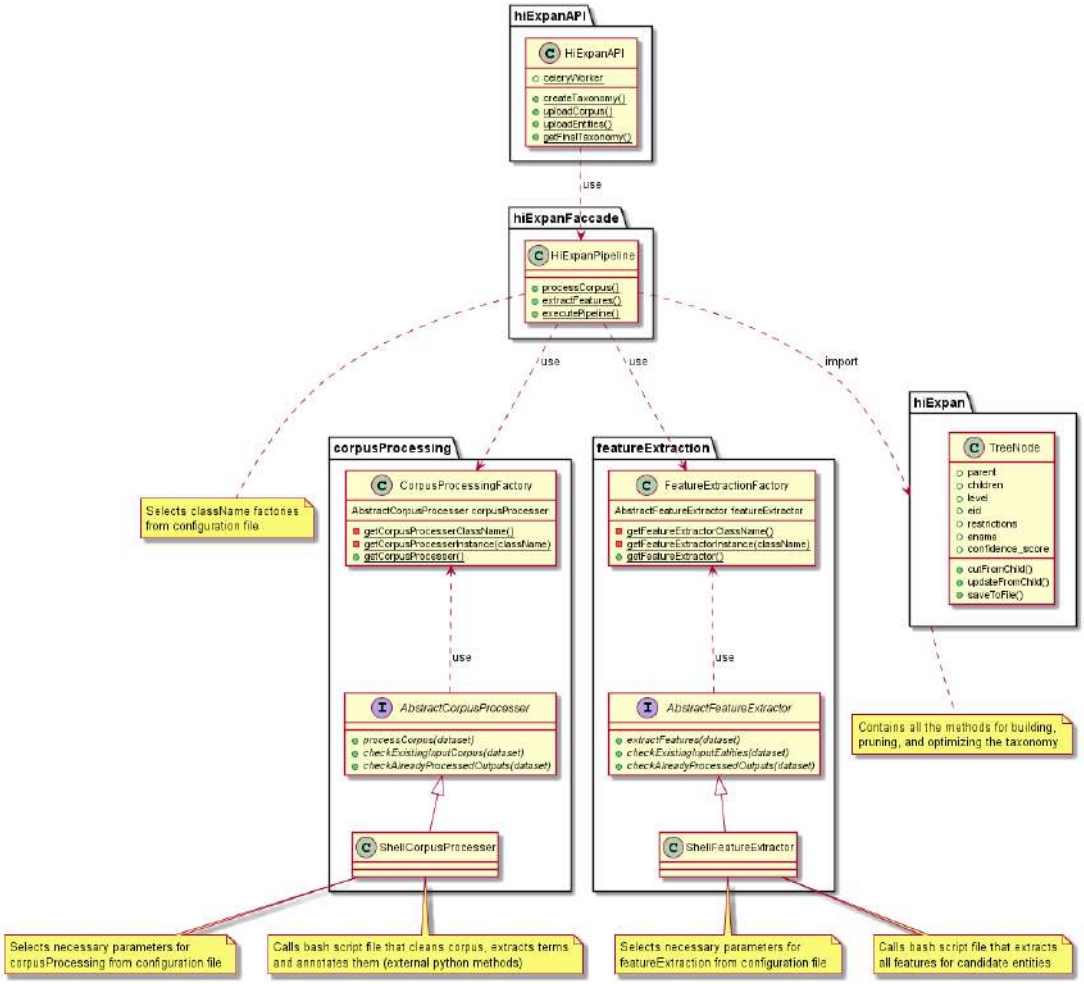

Figura A.6: Diagrama de clases resultado con el API de entrada

A.3 Sprint 3

La estimación de las historias de usuario corresponden a la Figura A.7 Por comodidad, se ha omitido el detalle de este sprint, puesto que ocupa mucho espacio de imagen y no aportan información excesivamente relevante, puesto que la mayoría son historias de usuario básicas.

La *landing page* se localiza en la Figura A.8.

Figura A.8: Landing Page

La web de *login* se encuentra en la Figura A.9.

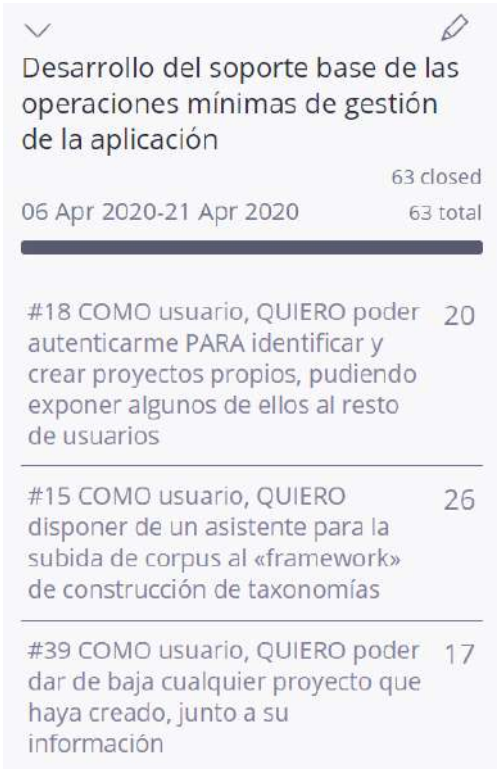

Figura A.7: Estimación Sprint 3

Figura A.9: Login

La web de registro de usuarios se puede ver en la Figura A.10.

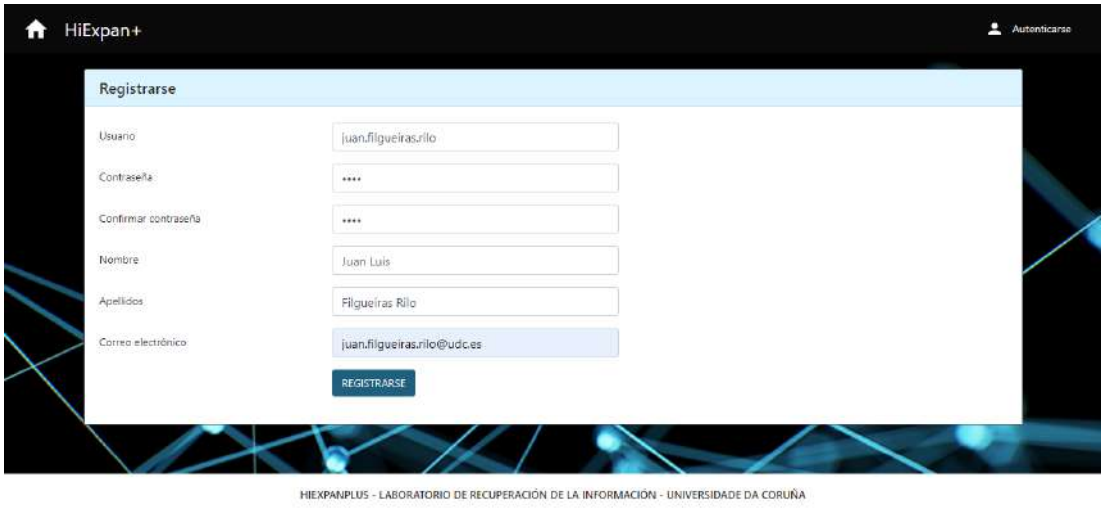

Figura A.10: Register

La web donde se muestran las opciones de usuario una vez se ha autenticado se encuentra en la Figura A.11.

Figura A.11: Opciones tras la autenticación

El formulario de creación del proyecto se encuentra en la Figura A.12.

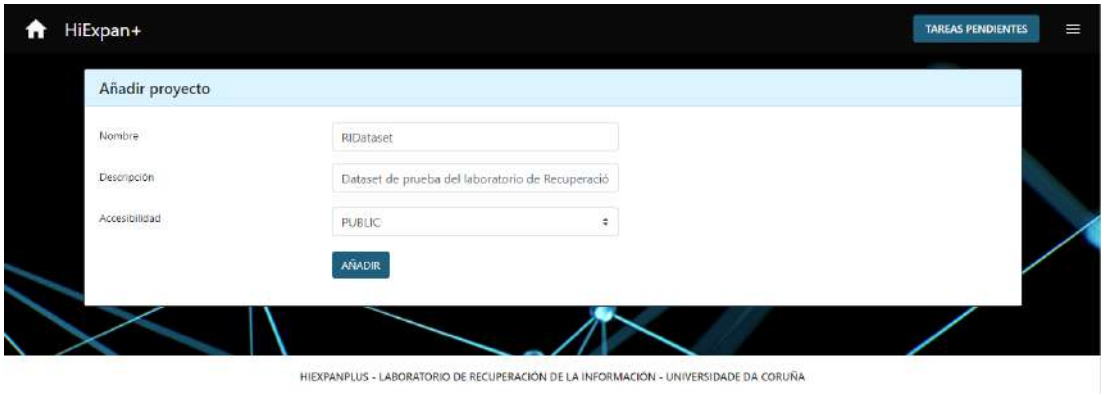

Figura A.12: Crear Proyecto

El listado de un proyecto por orden descendente de creación se puede observar en la Figura A.13.

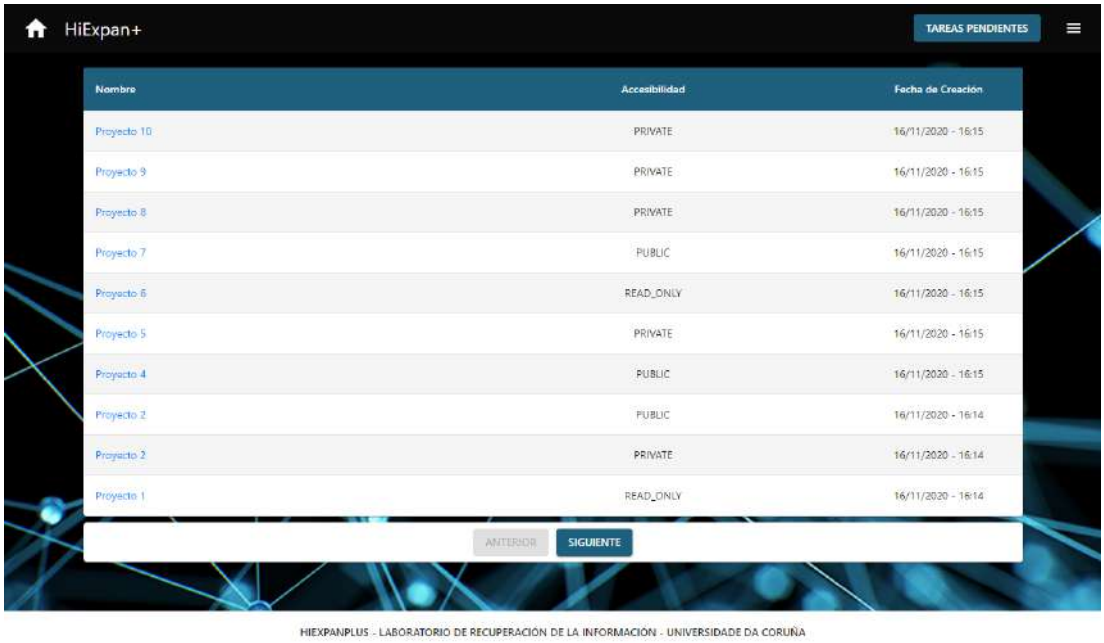

Figura A.13: Listado Proyectos

El detalle de un proyecto con la opción de su borrado se puede observar en la Figura A.14.

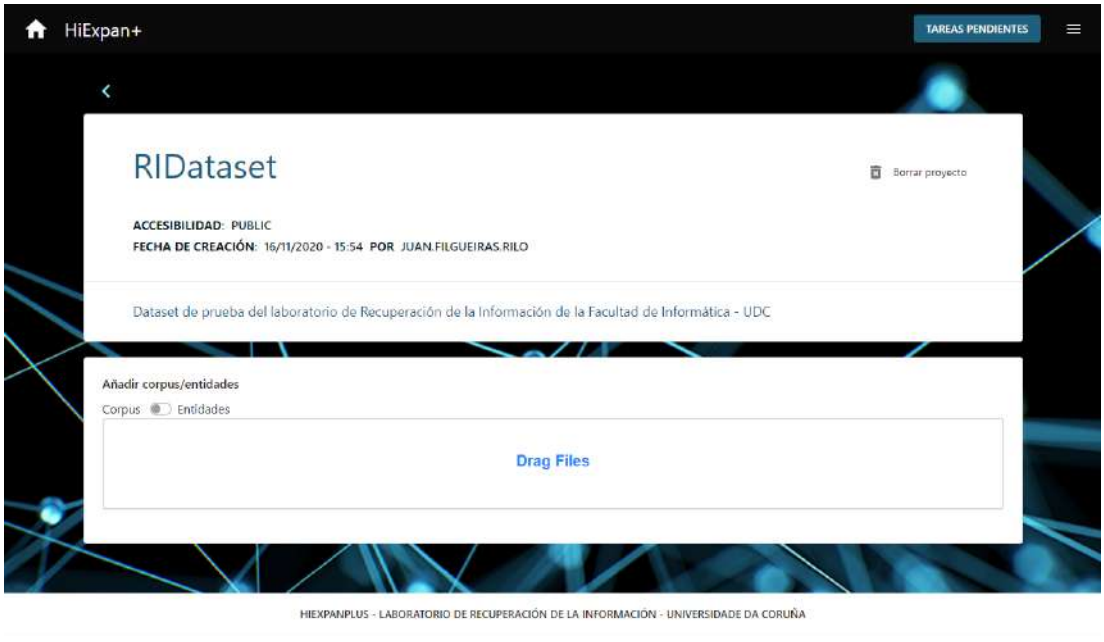

Figura A.14: Detalle Proyecto

A.4 Sprint 4

La estimación de las historias de usuario corresponden a la Figura A.15.

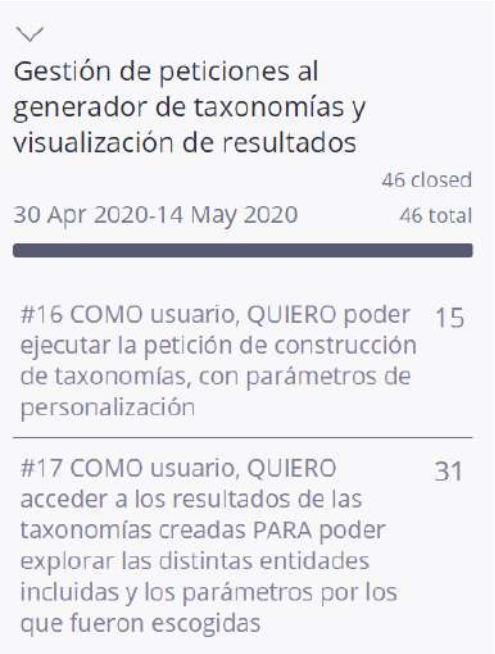

Figura A.15: Estimación Sprint 4

El detalle del *sprint* se puede observar en la Figura A.16.

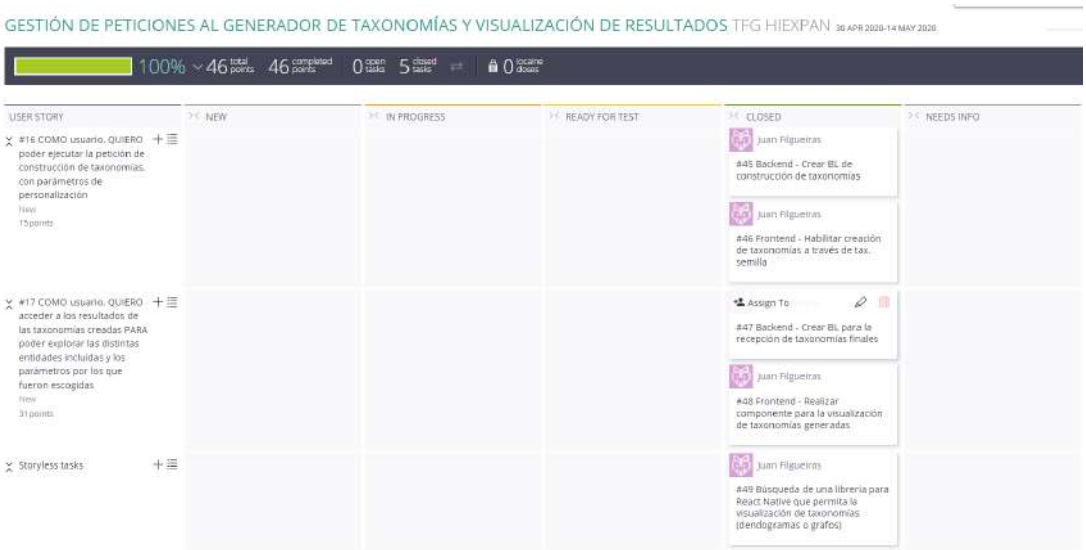

Figura A.16: Detalle Sprint 4

Respecto a la creación de la taxonomía, el formulario se puede ver en la Figura A.17, y el detalle de la edición de los nodos se puede observar en la Figura A.18.

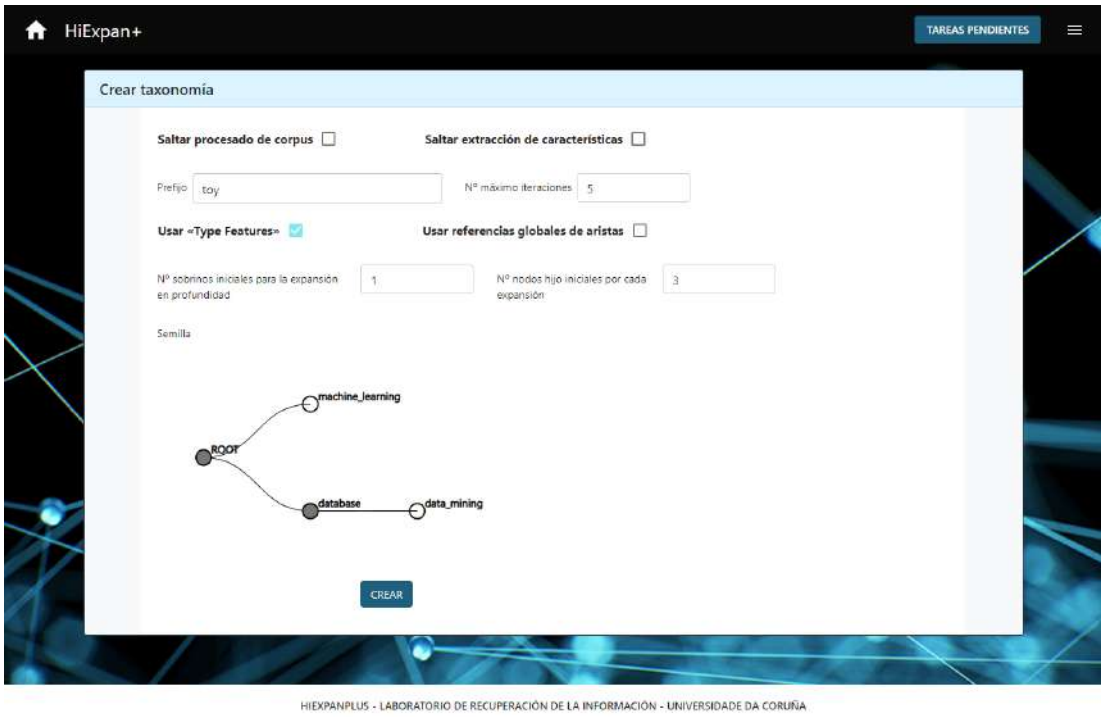

Figura A.17: Formulario creación de la taxonomía

Figura A.18: Edición de los nodos de la taxonomía semilla

El detalle de una taxonomía después del proceso de construcción se encuentra en la Figura A.19.

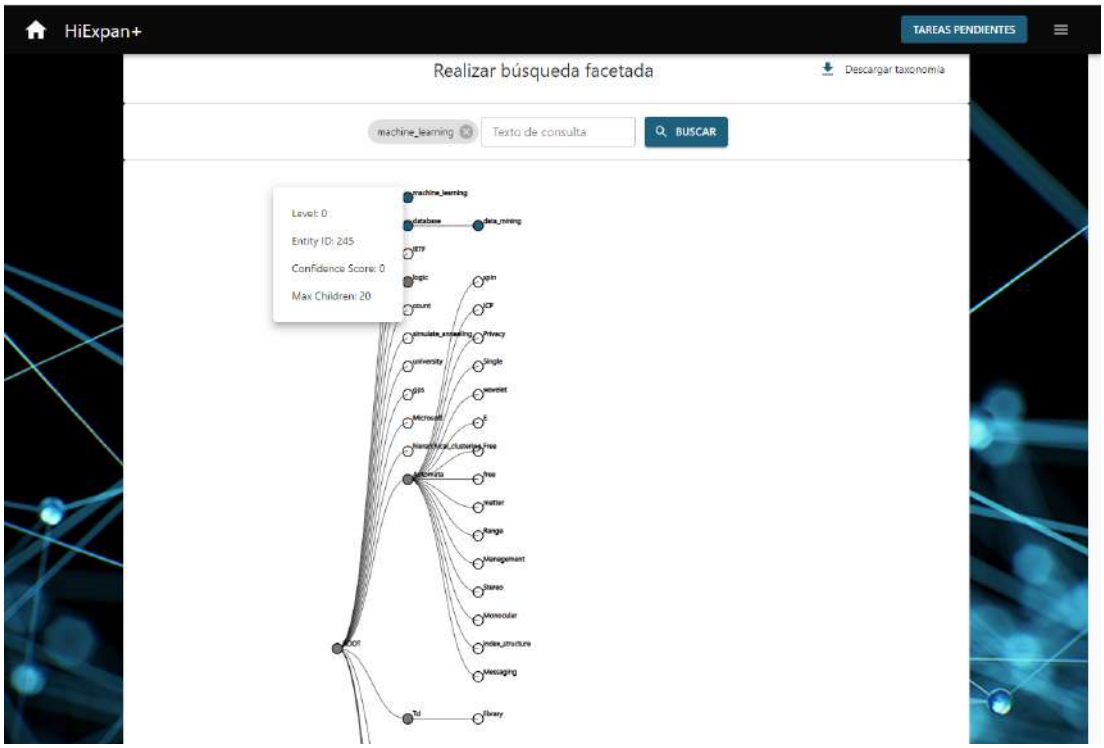

Figura A.19: Detalle Taxonomía

A.5 Sprint 5

La estimación de las historias de usuario corresponden a la Figura A.20.

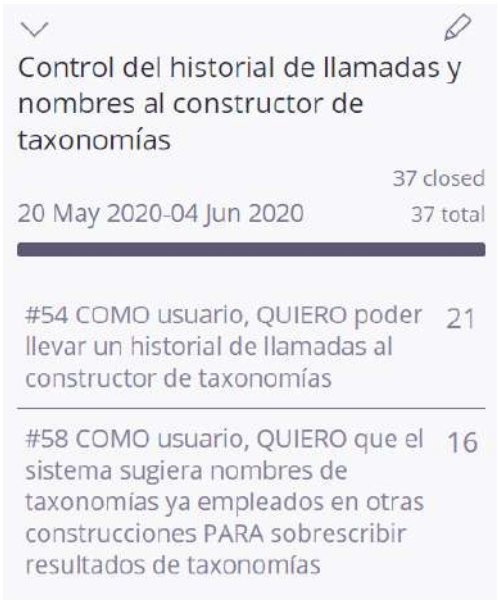

Figura A.20: Estimación Sprint 5

El detalle del *sprint* según lo planificado se puede observar en la Figura A.21.

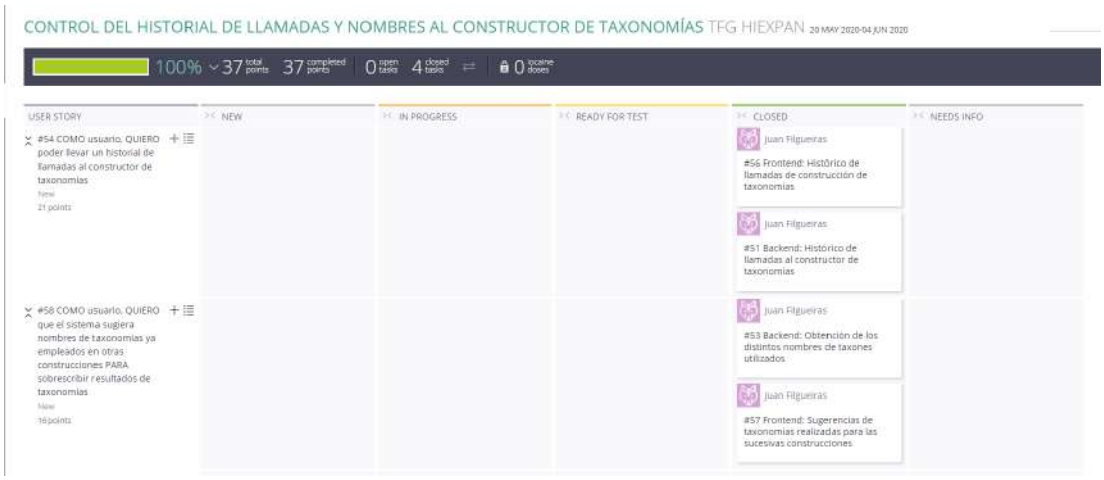

Figura A.21: Detalle Sprint 5

El listado de las distintas taxonomías, en concreto de la que se acaba de construir, se localiza en la Figura A.22.

El detalle del autocompletado, que se localiza en el formulario de creación de una taxonomía, se localiza en la Figura A.23.

A.6 Sprint 6

La estimación de las historias de usuario corresponden a la Figura A.24.

El detalle de este sprint se localiza en la Figura A.25. Respecto al botón de descarga de la taxonomía resultado, se puede apreciar en la Figura A.19.

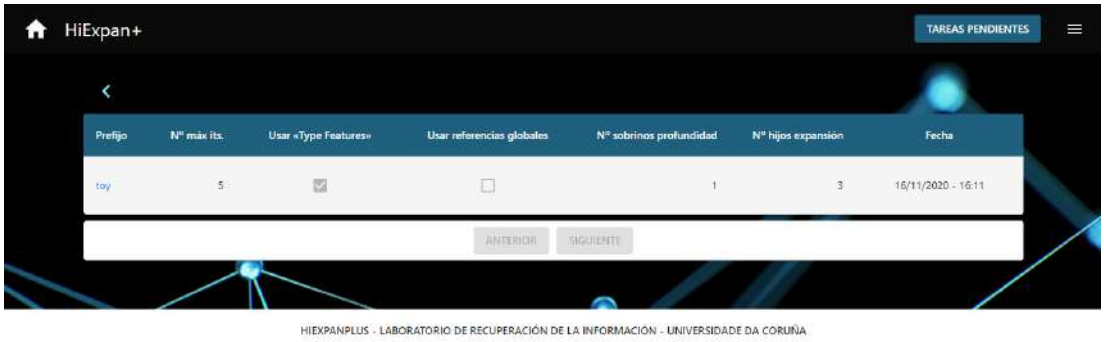

Figura A.22: Listado de llamadas de construcción

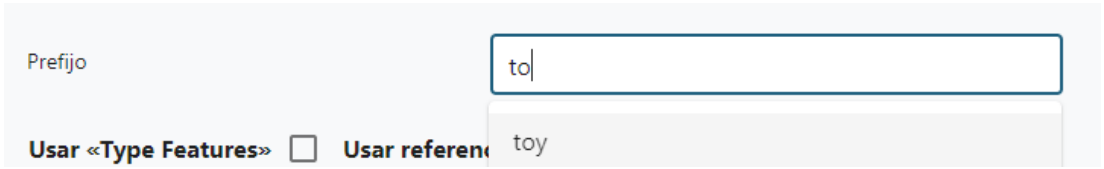

Figura A.23: Autocompletado a partir de las taxonomías construidas

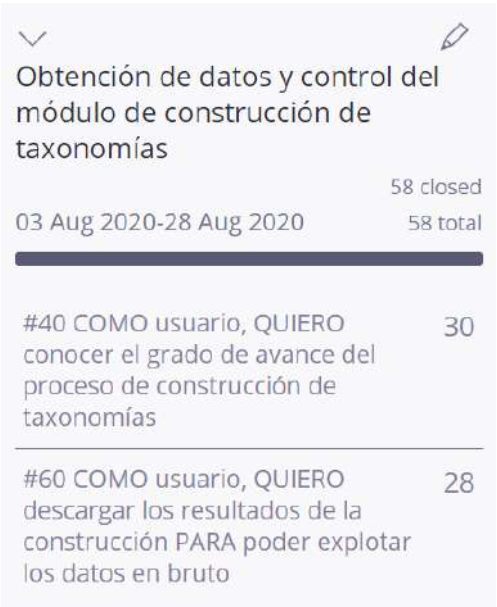

Figura A.24: Estimación Sprint 6

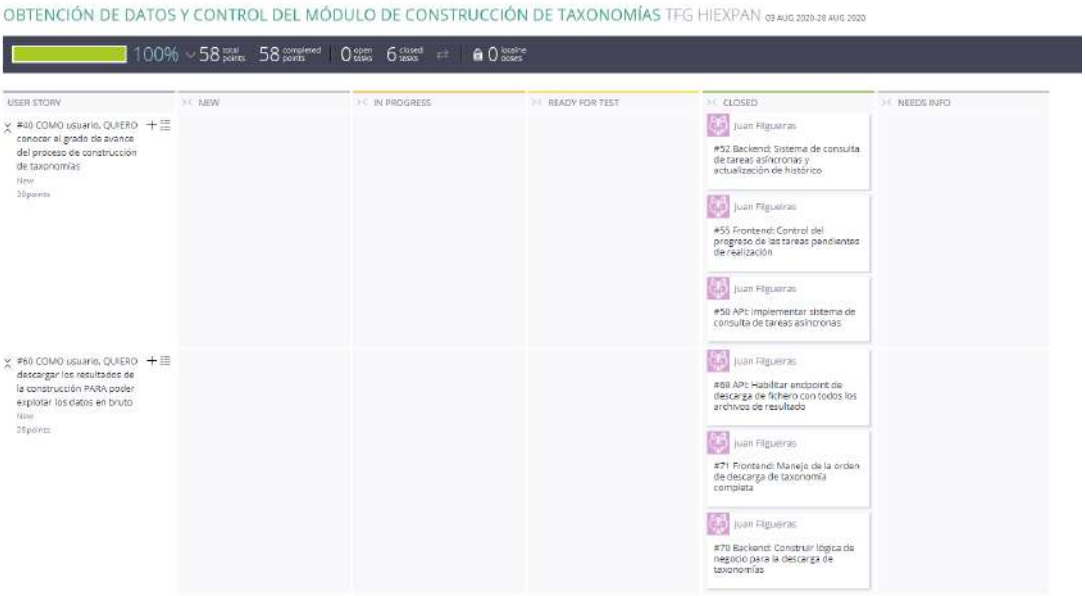

Figura A.25: Detalle Sprint 6

Los distintos estados del proceso de construcción de taxonomías se pueden observar en las Figuras A.26, A.27, A.28, A.29, A.30 y A.31.

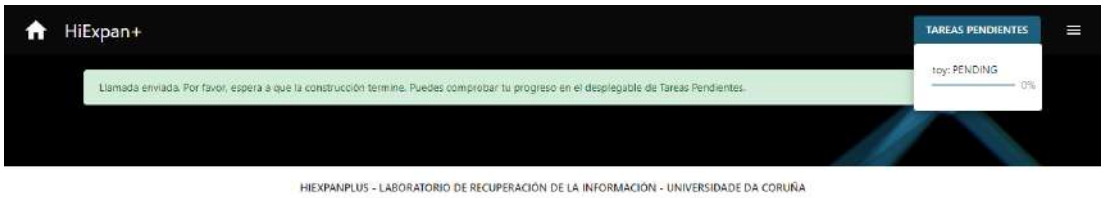

Figura A.26: Estado de espera de la construcción

Figura A.27: Estado en procesamiento del corpus

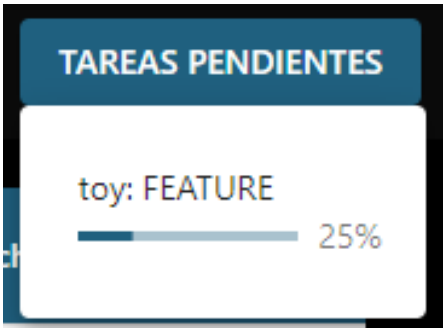

Figura A.28: Estado en extracción de las características intermedias

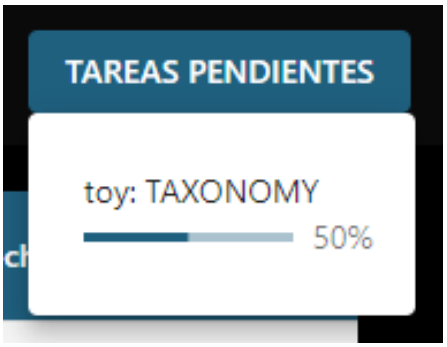

Figura A.29: Estado en construcción de la taxonomía

Figura A.30: Estado en construcción del índice

Figura A.31: Estado de creación completa y satisfactoria

A.7 Sprint 7

La estimación de las historias de usuario corresponden a la Figura A.32.

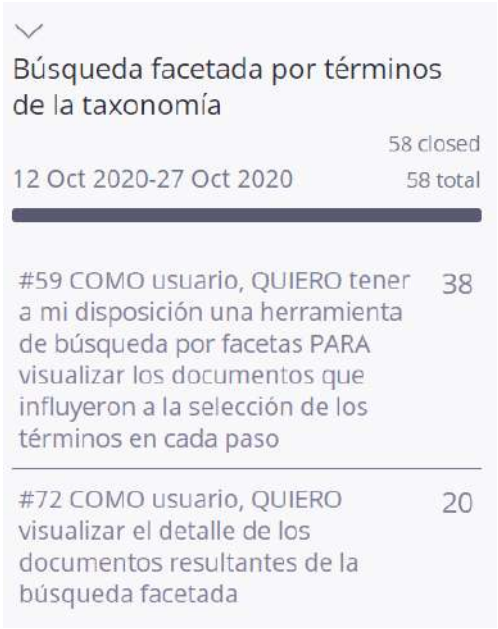

Figura A.32: Estimación Sprint 7

El detalle del *sprint* se encuentra en la Figura A.33.

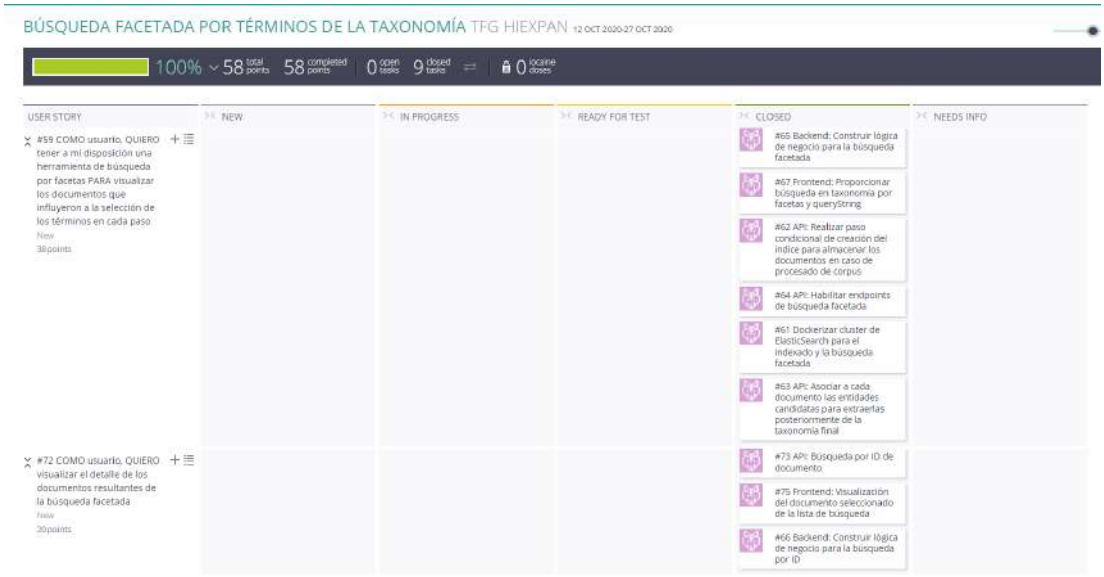

Figura A.33: Detalle Sprint 7

El componente para realizar la búsqueda facetada se encuentra en el detalle de la taxonomía, en la Figura A.19.

El resultado de una búsqueda facetada se puede observar en la Figura 2.1.1.

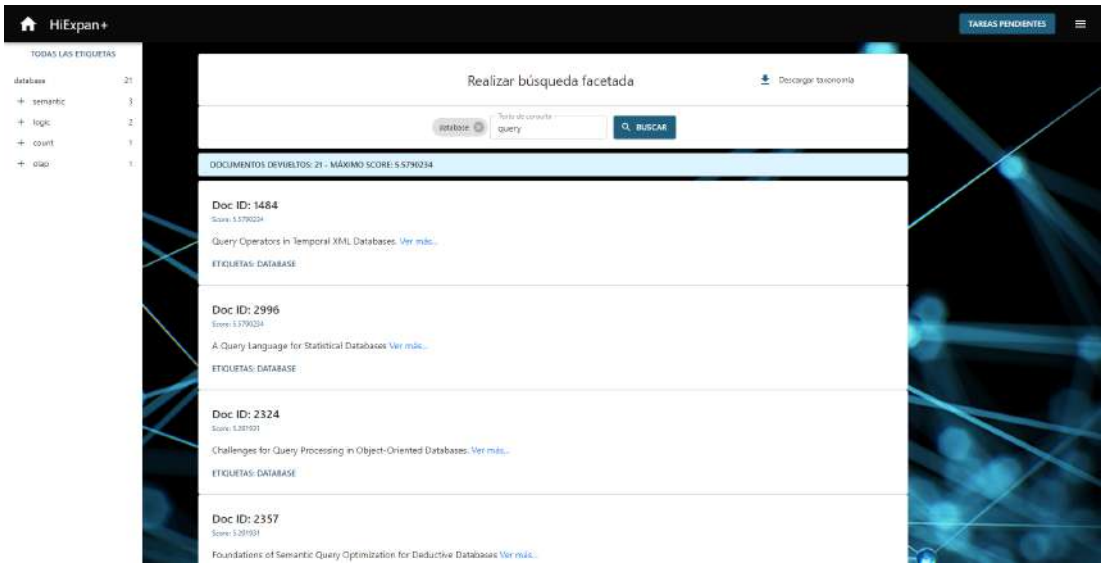

Figura A.34: Resultado de la búsqueda facetada

El detalle de un documento se puede observar en la Figura A.35.

Figura A.35: Detalle del documento

Lista de acrónimos

CI «Continuous Inspection». 36

CI «Continuous Integration». 35

DAO «Data Access Object». 57

DOM «Domain Object Model». 28, 29, 31

DTO «Data Transfer Object». 57

HOC «Higher Order Component». 29, 30

IDE «Integrated Development Environment». 32

IoC «Inversion of Control». 26

JPA «Java Persistence API». 26

JWT «JSON Web Token». 69

mrr «Mean Reciprocical Rank». 16

MVC «Modelo Vista Controlador». 27, 30

NLP «Natural Language Processing». 22

NN Sustantivo Singular. 12

NNP Nombre Propio. 12

NNS Sustantivo Plural. 12

ORM «Object-Relational Mapping». 26

POM «Project Object Model». 31, 69

POS «Part-of-Speech». 12

PUDS Proceso Unificado de Desarrollo de Software. 39

SCM «Software Configuration Management». 32

VCS «Version Control System». 32, 33

W3C «World Wide Web Consortium». 28

WSGI «Web Server Gateway Interface». 23

YAML «YAML Ain't Markup Language». 23

Glosario

Cadena de responsabilidad Patrón de diseño que favorece la creación de manejadores para eventos o solicitudes de forma que cada manejador tenga una única responsabilidad y se separe de la responsabilidad de un componente.. 29

clase semántica Relativo a la lingüística, todas las palabras que tienen un aspecto de significado común son de la misma clase semántica.. 10

code smells Síntoma de un proyecto software que, si bien no es un error en sí, puede indicar un problema de mayor nivel, como una deficiencia en el diseño software que, con el tiempo, inducirá a un error o dificultades en el desarrollo.. 36

deuda técnica Concepto que indica el esfuerzo adicional necesario para reconducir una solución que, por sencilla, contribuye negativamente al desarrollo de un proyecto, introduciendo alguno de los aspectos negativos a la calidad del software mencionados.. 36

DevOps Práctica de ingeniería de software que tiene como objetivo unificar el desarrollo de software (Dev) y la operación del software (Ops). Defiende la automatización y el monitorizado en todos los pasos de la construcción del software, desde la integración, las pruebas, la liberación hasta la implementación y la administración de la infraestructura, apuntando a ciclos de desarrollo más cortos. 25

Hiperonimia Relación semántica que vincula a una determinada unidad léxica con otras de significado más específico por las que puede ser sustituida.. 1, 2, 9

hiperónimo Palabra cuyo significado engloba el de otras.. 8

hipónimo Cada una de las palabras englobadas por el hiperónimo.. 8

Inversión del Control Principio del diseño del software que busca invertir el flujo de programación imperativo clásico, consistente en la llamada procedimental de métodos que

marcan el flujo de forma inevitable, sustituyéndolo por un estilo más reactivo a eventos y situaciones. Busca que el desarrollador solo indique las respuestas deseadas a las situaciones, y que una biblioteca o «framework» externo realice las llamadas a dichas respuesta cuando sea necesario.. 26

recall En Recuperación de la Información, es la fracción de documentos relevantes que se han recuperado frente al total de documentos recuperados.. 9, 12

similitud de coseno Medida de la similitud existente entre dos vectores en un espacio que posee un producto interior con el que se evalúa el valor del coseno del ángulo comprendido entre ellos.. 9

sucesión de Fibonacci Secuencia ordenada de números cuyo cálculo se computa a partir de la suma de los dos números anteriores en la secuencia, siendo los dos primeros elementos de valor 1. Ejemplo: 1, 1, 2, 3, 5, 8, 13, 21…. 46

Bibliografía

[1] A. Gupta, R. Lebret, H. Harkous, and K. Aberer, "Taxonomy induction using hypernym subsequences." [En línea]. Disponible en: https://arxiv.org/abs/1704.07626

[2] M. A. Hearst, "Automatic acquisition of hyponyms from large text corpora," 1992.

[3] C. Aebeloe, I. Keles, G. Montoya, and K. Hose, "Star pattern fragments: Accessing knowledge graphs through star patterns," 2013. [En línea]. Disponible en: https://arxiv.org/abs/2002.09172

[4] R. Fu, J. Guo, B. Qin, W. Che, H. Wang, and T. Liu, "Learning semantic hierarchies via word embeddings," 2013. [En línea]. Disponible en: http://ir.hit.edu.cn/~car/papers/acl14embedding.pdf

[5] J. Shen, Z. Wu, D. Lei, J. Shang, X. Ren, and J. Han, "Setexpan: Corpus-based set expansion via context feature selection and rank ensemble," 2013. [En línea]. Disponible en: https://arxiv.org/abs/1910.08192

[6] J. Shen, Z. Wu, D. Lei, C. Zhang, X. Ren, M. T. Vanni, B. M. Sadler, and J. Han, "Hiexpan: Task-guided taxonomy construction by hierarchical tree expansion." [En línea]. Disponible en: https://arxiv.org/abs/1910.08194

[7] J. Shang, J. Liu, M. Jiang, X. Ren, C. R. Voss, and J. Han, "Automated phrase mining from massive text corpora." [En línea]. Disponible en: https://arxiv.org/abs/1702.04457

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," 2013. [En línea]. Disponible en: https://arxiv.org/abs/1301.3781

[9] M. Qu, X. Ren, Y. Zhang, and J. Han, "Weakly-supervised relation extraction by pattern-enhanced embedding learning." [En línea]. Disponible en: https://arxiv.org/abs/1711.03226

[10] G. Dwyer, S. Aggarwal, and J. Stouffer, *Flask: Building Python Web Services*. Packt Publishing, 2017. [En línea]. Disponible en: https://books.google.es/books?id=n6HBAQAACAAJ

[11] J. Gosling, B. Joy, G. L. Steele, G. Bracha, and A. Buckley, *The Java Language Specification, Java SE 8 Edition*, 1st ed. Addison-Wesley Professional, 2014.

[12] C. Walls, *Spring in Action*, 4th ed. USA: Manning Publications Co., 2014.

[13] ——, *Spring Boot in Action*, 1st ed. USA: Manning Publications Co., 2016.

[14] T. Kaczanowski, *Practical Unit Testing with JUnit and Mockito*. POL: Tomasz Kaczanowski, 2013.

[15] A. Freeman, *Pro React 16*, 1st ed. APress, 2019.

[16] A. Boduch, *React Material-UI Cookbook: Build captivating user experiences using React and Material-UI*. Packt Publishing, 2019. [En línea]. Disponible en: https://books.google.es/books?id=m86PDwAAQBAJ

[17] J. Sutherland and K. Schwaber, "The scrum guide," 2017. [En línea]. Disponible en: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf

[18] "Guia salarial sector TI Galicia," 2016. [En línea]. Disponible en: https://es.scribd.com/document/288511179/Guia-Salarial-Sector-TI-Galicia-2015-2016

[19] "C4 Model." [En línea]. Disponible en: https://c4model.com/