Universidad Zaragoza
1542

Trabajo Fin de Grado

Visualización web y análisis de grafos para la interoperabilidad semántica en terminologías médicas

Web visualization and graph analysis for semantic interoperability in medical terminologies

Autor

Sofía Rodrigo Bonet

Directores

Carlos Tellería Orriols
Irene Sánchez Montejo

Titulación del autor

Grado en Ingeniería Informática
Departamento de Informática e Ingeniería de Sistemas

ESCUELA DE INGENIERÍA Y ARQUITECTURA
2025

RESUMEN

Visualización web y análisis de grafos para la interoperabilidad semántica en terminologías médicas

La interoperabilidad semántica en salud es clave para garantizar el intercambio y la comprensión consistente de los datos clínicos entre sistemas y regiones. Este Trabajo de Fin de Grado aborda la necesidad de crear y poblar una base de datos orientada a grafos que contenga diversos sistemas de clasificación médica (CIE-10-ES, CIAP-1, SNOMED-CT, etc.), así como la construcción de una herramienta visual e intuitiva para explorar y analizar dichos grafos, facilitando la interoperabilidad semántica y la normalización de datos en el Servicio Aragonés de Salud.

Utilizando estas clasificaciones, se diseña la ontología médica representada como un grafo en la base de datos OrientDB, con nodos que identifican conceptos médicos y aristas que representan relaciones como jerarquías o mapeos entre clasificaciones. Esta ontología incluye terminologías usadas tanto en atención primaria (CIAP-1) como en especializada (CIE-10, SNOMED-CT), entre otras. Un desafío abordado ha sido la falta de mapeos oficiales entre CIAP-1 y otras codificaciones, diseñando algoritmos que, a través de correspondencias intermedias, permitieron expandir las relaciones y completar el grafo.

Por otro lado, para superar las limitaciones de accesibilidad de la interfaz de OrientDB, se desarrolla una aplicación web interactiva utilizando Flask en el backend y Cytoscape.js en el frontend. La herramienta permite buscar conceptos por términos o códigos, visualizar relaciones jerárquicas, expandir nodos dinámicamente y exportar datos para análisis externo. Diseñada con un enfoque en usabilidad, la interfaz incorpora elementos visuales intuitivos, como colores distintivos por terminología, paneles emergentes y controles de navegación.

Las pruebas realizadas con usuarios técnicos y no técnicos destacan la utilidad de la herramienta, aunque también identifican áreas de mejora, como la optimización de la disposición del grafo en búsquedas extensas. En términos de impacto, la herramienta ofrece una solución práctica para mejorar la codificación médica, reducir errores y apoyar investigaciones basadas en datos clínicos.

Este proyecto no solo ha contribuido al avance de la informática médica, sino que también sienta las bases para futuras extensiones, como integrar más terminologías, incorporar análisis semánticos avanzados o validar su uso en otros entornos de salud.

Índice general

Capítulo 1

Introducción

1.1. Motivación

La interoperabilidad en el ámbito sanitario es fundamental para mejorar la calidad del sistema de salud, ya que permite el intercambio de información entre diferentes sistemas y servicios. Dentro de la interoperabilidad podemos distinguir 4 tipos: semántica, técnica, sintáctica y organizativa. Este trabajo se centra en la **interoperabilidad semántica**, que garantiza que los datos clínicos puedan ser intercambiados, interpretados y utilizados de manera coherente y precisa entre distintos sistemas, instituciones y regiones. Para asegurarla es preciso el uso de estándares o terminologías médicas.

Cada ámbito tiene su propia terminología. En el caso de diagnósticos se encuentran la *Clasificación Internacional de Enfermedades, 10ª Revisión (CIE-10-ES)* [1], la *Clasificación Internacional de Atención Primaria, 1ª Edición (CIAP-1)* [2], o la *Nomenclatura Sistemática de la Medicina – Términos Clínicos (SNOMED-CT)* [3], las cuales permiten codificar enfermedades y otros aspectos de la atención sanitaria con un lenguaje normalizado. Por otro lado, también se utilizan otras codificaciones especializadas en fármacos, como la *Clasificación Anatómica, Terapéutica y Química (ATC)* [4] y el sistema *RxNorm* [5] para la normalización de medicamentos, así como la base de datos de la *Agencia Española de Medicamentos y Productos Sanitarios (AEMPS)* [6], lo que permite la estandarización y gestión de medicamentos.

Estas codificaciones, utilizadas en el **Servicio Aragonés de Salud**, permiten registrar los casos de los pacientes en sistemas informáticos. Además, el registro de estos estándares es esencial para la realización de cohortes, grupos de pacientes con una misma patología. Sin embargo, este proceso no está exento de errores. En ocasiones, los médicos pueden introducir códigos incorrectos o imprecisos, lo que puede generar problemas de interpretación de registros médicos e incoherencias en estudios de investigación.

Para abordar esta problemática, en el *Instituto Aragonés de Ciencias de la Salud (IACS)* [7] se está diseñando una ontología de terminologías médicas basada en un grafo, el cual representa los conceptos médicos como nodos y las relaciones internas entre conceptos o correspondencias con otras terminologías como aristas. Este grafo tiene como objetivo, no solo facilitar la interoperabilidad semántica en la salud, sino también servir de apoyo al uso de modelos de Procesamiento de Lenguaje Natural (PLN) para la detección y corrección de errores en los registros, o nutrir de contexto a LLMs (Large Language Models). Este aspecto es crucial para mejorar la continuidad del cuidado del paciente y la calidad del dato clínico.

Para almacenar este grafo, se ha utilizado OrientDB [8], una base de datos especializada en grafos. Sin embargo, la interfaz nativa de OrientDB presenta serias limitaciones en términos de accesibilidad y usabilidad, ya que está diseñada para usuarios con conocimientos técnicos avanzados en bases de datos, lo que dificulta su adopción por parte de profesionales de la salud o investigadores no técnicos. Estos profesionales pueden aprovechar el grafo como una herramienta para la elección de la codificación del diagnóstico más apropiado en periodo de consulta.

Debido a esta deficiencia, surge la necesidad de desarrollar una herramienta web intuitiva que permita interactuar con el grafo de manera sencilla y eficiente. Esta herramienta no solo mejora la experiencia del usuario, sino que también habilita funcionalidades clave como la búsqueda de diagnósticos, la visualización de relaciones entre términos y la exportación de datos para análisis adicionales. En definitiva, este proyecto busca transformar una tecnología compleja y poco inaccesible, en un recurso práctico y funcional, adaptado a las necesidades de profesionales de la salud e investigadores.

1.2. Objetivos

El proyecto tiene como objetivo colaborar con la población del grafo y desarrollar una herramienta funcional, interactiva y visualmente comprensible que permita a profesionales de la salud, desarrolladores e investigadores explorar y analizar de manera eficiente las correspondencias entre sistemas de codificación médica.

En particular, se busca *poblar la base de datos* de forma estructurada y unificada, asegurando que los datos de diversas terminologías médicas sean consistentes y estén correctamente relacionados entre sí.

Además, se pone el foco en la *representación gráfica de las relaciones semánticas*, permitiendo buscar, visualizar y exportar la ontología médica de forma clara y eficiente. La interfaz debe incorporar estilos distintivos para diferenciar las terminologías y sus jerarquías, facilitando la interpretación de relaciones complejas y la comprensión de datos clínicos.

Es importante que la solución sea *escalable para futuros desarrollos*, adoptando un diseño portable y modular que permita integrar nuevas terminologías, ampliar funcionalidades o incluso soportar análisis avanzados mediante modelos de PLN. Esto garantizará que la herramienta pueda evolucionar y adaptarse a las necesidades cambiantes de la comunidad médica y tecnológica.

Por último, la herramienta debe *facilitar la investigación clínica* al permitir el análisis y la búsqueda de patrones en los datos médicos. Esto contribuirá significativamente a estudios sobre enfermedades, tratamientos y la identificación de posibles correlaciones entre distintas terminologías, fomentando avances en la investigación y la práctica clínica.

1.3. Estructura del documento

La presente memoria se estructura de la siguiente manera. En el **Capítulo 2**, se concretan las terminologías y se recopilan las herramientas existentes para la visualización de las mismas, así como sus carencias respecto a la solución propuesta. En el **Capítulo 3**, se detalla la estructura y construcción del grafo que sustenta la base del resto del proyecto. En el **Capítulo 4**, se describe la metodología seguida para el diseño e implementación de la herramienta. En el **Capítulo 5**, se presentan una serie de pruebas de usuario, acompañadas de ejemplos prácticos y propuestas de cambio de la herramienta desarrollada. El **Capítulo 6** recoge las conclusiones y propone líneas futuras de mejora. Finalmente, se incluye en el **Anexo** el código fuente relevante.

1.4. Tiempo dedicado

La Figura 1.1 muestra el tiempo invertido en el trabajo, desglosándolo por tareas más específicas.

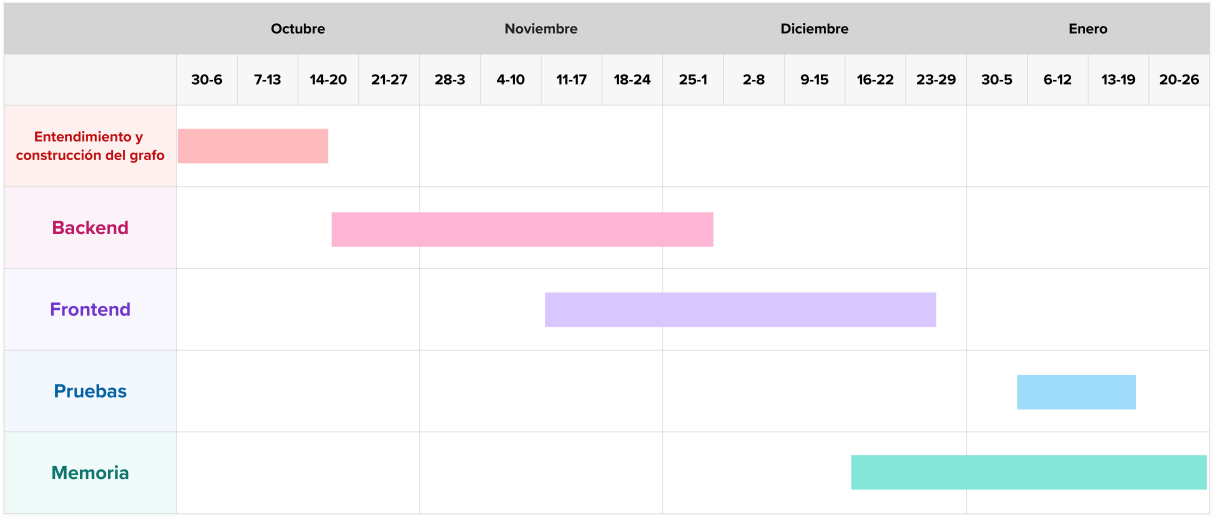


Figura 1.1: Diagrama de Gantt de la distribución temporal de las tareas del proyecto.

Capítulo 2

Estado del Arte

La interoperabilidad semántica de la información médica, especialmente cuando se combinan distintas codificaciones, representa un desafío clave para garantizar la consistencia y utilidad de los datos clínicos. *OMOP CDM (Observational Medical Outcomes Partnership Common Data Model)* [9] es un modelo de datos comunitario diseñado para estandarizar la estructura y el contenido de los datos observacionales. Dentro de OMOP, las categorías principales incluyen dominios clínicos como diagnósticos, procedimientos y medicamentos, organizados a través de los vocabularios estandarizados de *OHDSI (Observational Health Data Sciences and Informatics)*. Estas categorías incluyen a su vez diversas codificaciones médicas. Esta sección explora, en primer lugar, estos sistemas de codificación y, a continuación, las herramientas disponibles actualmente para su visualización.

2.1. Terminologías

La ontología enmarcada en este proyecto consta de las siguientes terminologías:

- **CIAP** (Clasificación Internacional de Atención Primaria): taxonomía de términos médicos publicada por la *WONCA (World Organization of Family Doctors)* [10]. Esta clasificación incluye aproximadamente 700 códigos organizados en 17 capítulos, cada uno de ellos asociado a un órgano, sistema o función biológica específica. Los códigos abarcan diagnósticos, problemas de salud, episodios de atención y motivos de consulta médica en atención primaria. Cada código está compuesto por un carácter seguido de dos dígitos; el carácter identifica el capítulo al que pertenece, mientras que los dígitos están asociados a una categoría conceptual, que se detalla en la descripción textual que acompaña al código.

  Existen dos versiones de CIAP, pero en este trabajo se utiliza su primera versión **CIAP-1**, que es la versión actualmente empleada en el Servicio Aragonés de Salud.

- **CIE-10** (Clasificación Internacional de Enfermedades, 10.ª Revisión): estándar publicado por la *OMS (Organización Mundial de la Salud)* [11] para la clasificación de enfermedades, procedimientos, oncología y problemas de salud, principalmente en el ámbito de la atención especializada. De forma más extensa que CIAP-1, está compuesta por unos 14.000 códigos únicos acompañados de una descripción textual y se organizan jerárquicamente en 22 capítulos, cada uno con secciones.

  El *National Center for Health Statistics (NCHS) de los Estados Unidos* [12] desarrolló la ICD-10-CM, que es la modificación clínica en inglés de la CIE-10. En España, esta versión es conocida como **CIE-10-ES** (Diagnósticos y Procedimientos).

- **SNOMED-CT** (Nomenclatura Sistemática de la Medicina – Términos Clínicos): terminología completa publicada y licenciada por *IHTSDO (International Health Terminology Standards Development Organisation)* [13]. Proporciona una cobertura y especificidad mucho mayor que CIE-10 y CIAP-1, ofreciendo una robusta ontología jerárquica de aproximadamente 1.200.000 relaciones, cada una con su identificador único y una descripción textual.

- **ATC** (Clasificación Anatómica, Terapéutica y Química): sistema internacional desarrollado por la *OMS* para clasificar medicamentos según el órgano o sistema en el que actúan, su propósito terapéutico y sus características químicas. Los medicamentos están organizados jerárquicamente en cinco niveles.
- **RxNorm**: sistema desarrollado por la *NLM (National Library of Medicine)* [14] de los Estados Unidos que proporciona una terminología estandarizada para los medicamentos clínicos, incluidos sus nombres genéricos, comerciales, dosis y formas farmacéuticas.
- **AEMPS** (Agencia Española de Medicamentos y Productos Sanitarios): sistema nacional en España encargado de la regulación, evaluación, autorización y control de medicamentos de uso humano y veterinario, así como de productos sanitarios.

A continuación, en el Cuadro 2.1 se ilustran algunos ejemplos de conceptos en las terminologías hasta ahora nombradas.

Terminología	**Código**	**Descripción**
CIAP-1	A03	Fiebre
CIAP-1	Z01	Pobreza, problemas económicos
CIE-10-ES	G30	Enfermedad de Alzheimer
CIE-10-ES	G30.1	Enfermedad de Alzheimer de comienzo tardío
SNOMED	44054006	Diabetes mellitus tipo 2
ATC	A01AA01	Fluoruro de sodio 1.1mg producto oral
RxNorm	1666106	Peróxido de magnesio
AEMPS	599926	Bisoprolol 10mg 60 comprimidos recubiertos

Cuadro 2.1: Ejemplos de conceptos médicos.

Todas estas codificaciones, a excepción de CIAP-1 y AEMPS, se enmarcan en el modelo de datos OMOP, es decir, están asociadas a un código estándar.

2.2. Herramientas de visualización existentes

A continuación, se detallan algunas de las herramientas actuales que permiten buscar y visualizar las terminologías médicas nombradas, así como los mapeos entre ellas.

- **Snow Owl** [15]: Un editor y servidor de terminologías médicas que soporta estándares como SNOMED-CT y otros. Aunque proporciona herramientas avanzadas para la edición de ontologías y la gestión de relaciones entre términos, su interfaz está dirigida principalmente a usuarios técnicos, lo que limita su uso en contextos clínicos.
- **i2b2 (Informatics for Integrating Biology and the Bedside)** [16]: Un marco de software para la integración y análisis de datos clínicos, diseñado para investigación traslacional, resultante de abordar las enfermedades desde diferentes áreas médicas. Aunque permite la integración de datos de diversas fuentes, no está optimizado para explorar relaciones específicas entre sistemas de codificación médica.
- **Herramientas basadas en sistemas de codificación**: Muchos hospitales y organizaciones utilizan soluciones personalizadas integradas en sus sistemas de *HCE (Historia Clínica Electrónica)* [17]. Estas soluciones suelen limitarse a búsquedas básicas de códigos y no proporcionan exploración visual ni funcionalidad avanzada de análisis.

- **Athena** [18]: Una herramienta en línea proporcionada por OHDSI que permite la búsqueda, exploración y descarga de terminologías médicas estandarizadas, incluyendo vocabularios estandarizados en OMOP, como SNOMED-CT, CIE-10-ES y RxNorm. Athena está diseñada para facilitar la interoperabilidad y estandarización en investigaciones clínicas y aplicaciones tecnológicas, pero no incluye una interfaz visual funcional, solo jerárquica, como la de la Figura 2.1.

Figura 2.1: Interfaz de Athena.

- **BioPortal** [19]: Una plataforma web de *NCBO (National Center for Biomedical Ontology)* que proporciona acceso a una amplia variedad de ontologías biomédicas y médicas. Permite la exploración, comparación y análisis de relaciones entre términos y terminologías, así como la integración de ontologías en aplicaciones mediante APIs. BioPortal destaca por su enfoque en la interoperabilidad y su facilidad de uso, pero carece de nuevo de una interfaz intuitiva, como muestra la Figura 2.2.

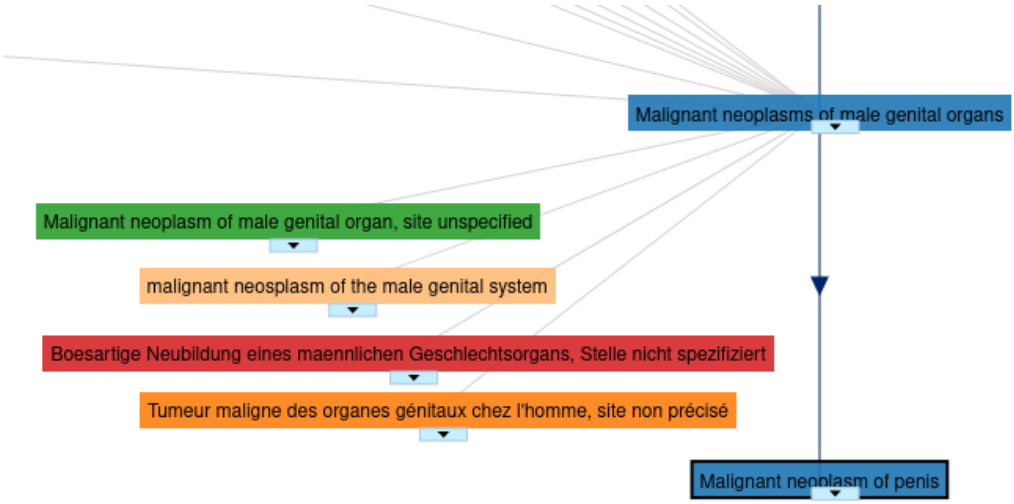


Figura 2.2: Interfaz de BioPortal.

A pesar de los avances de estas herramientas, ninguna combina de manera efectiva la exploración visual y jerárquica de relaciones entre terminologías con una interfaz accesible y funcional para usuarios no técnicos. Tampoco ofrece una visión inmediata de la descripción textual junto al código, o de las relaciones internas o mapeos de un concepto. Sobre todo, ninguna integra CIAP-1, utilizada en atención primaria en Aragón. Esto resalta la necesidad de una solución como la desarrollada en este proyecto, que ofrezca una manera de relacionar códigos CIAP-1 a otras terminologías estandarizadas, como CIE-10 o SNOMED. En este proyecto se propone una visualización de códigos, conceptos, relaciones internas y mapeos entre terminologías, mejorando el espectro actual e incluyendo codificaciones no exploradas por estas herramientas.

Capítulo 3

Construcción de la ontología

En este capítulo, se detalla la parte desarrollada del proceso de construcción y población del grafo, destacando sus características, los atributos de los nodos, y la creación de relaciones semánticas realizada entre los diferentes sistemas de codificación médica.

3.1. Estructura del grafo

La ontología diseñada en OrientDB consta de las terminologías nombradas en la sección 2.1 y sus interrelaciones de forma estructurada. Sin embargo, este proyecto se centra principalmente en diagnósticos (CIAP-1, CIE-10-ES, SNOMED-CT) y no tanto en fármacos (ATC, RxNorm, AEMPS).

Cada nodo en el grafo dirigido representa un concepto médico específico, mientras que las aristas entre nodos indican relaciones semánticas, como equivalencia, jerarquía o mapeo entre distintos sistemas de codificación. Los atributos más relevantes de cada nodo son:

- `@rid`: Identificador único del nodo en la base de datos OrientDB.
- `@class`: Sistema de codificación al que pertenece el nodo (CIAP-1, CIE-10-ES, SNOMED-CT, etc.).
- `concept_name`: Nombre del concepto en inglés.
- `concept_name_spanish`: Nombre del concepto en español, si está disponible.
- `synonymous_concept`: Sinónimos asociados al concepto.
- `concept_code`: Código asociado al concepto en su sistema de codificación.
- `concept_id`: Código OMOP asociado al concepto.

Las relaciones definidas entre los nodos:

- Estas relaciones pueden ser de carácter interno de la terminología como `include`, `hasFindingSite` y `isA`, que indican jerarquía o zonas donde se producen determinadas patologías, o mapeos con otras terminologías como `hasCIAP1`, `hasSNOMED` y `hasICD10`, que indican equivalencia.

3.2. Creación de relaciones entre terminologías

El mapeo entre clasificaciones es esencial para lograr la interoperabilidad semántica, ya que diferentes sistemas utilizan terminologías distintas, y en cualquier modelo de datos es necesario emplear un sistema único y completo de codificación. SNOMED-CT y CIE-10 han publicado **mapeos oficiales** entre ellos; aproximadamente 95.323 conceptos del primero están mapeados a 13.892 conceptos del segundo. Aprovechando estos avances, OHDSI también ha publicado un mapeo completo entre un amplio conjunto de terminologías para ser utilizado en su modelo común de datos OMOP. Todos estos mapeos están disponibles a través del portal web Athena. Sin embargo, el mapeo entre CIAP-1 y CIE-10 o entre CIAP-1 y SNOMED-CT no está incluido.

SNOMED-CT es la clasificación más extensa y completa, lo que la hace ideal para mapear. CIAP-1 no solo codifica conceptos relacionados con problemas de salud, como patologías, sino también razones de consulta médica que no necesariamente están vinculadas a problemas médicos específicos.

Por ejemplo, el código Z14 de CIAP-1 se refiere a 'Problemas con el cónyuge enfermo', un concepto que no tiene correspondencia en SNOMED-CT. Además, un concepto de CIAP-1 puede abarcar múltiples problemas médicos; por ejemplo, el código B81 'Anemia perniciosa' en CIAP-1 se mapea en SNOMED-CT al código 84027009 'Anemia perniciosa', pero también al código 83414005 'Anemia macrocítica'. Este ejemplo se detalla en la Figura 3.1.

Figura 3.1: Ejemplo mapeo 1-N CIAP-1 a SNOMED-CT en OrientDB.

Aproximadamente 491 conceptos de CIAP-1 se mapean a 4.879 conceptos de CIE-10, mientras que 643 conceptos de CIAP-1 se mapean a 518 conceptos de SNOMED-CT, dejando algunos códigos de CIAP-1 sin mapeo. Por lo tanto, para **lograr un mapeo completo** entre CIAP-1 y SNOMED-CT, el uso de CIE-10 podría ser una herramienta clave.

La función `mapsCIAPCIE` detallada en el Algoritmo 3.1, ejemplifica cómo se crean relaciones entre conceptos CIAP-1 y CIE-10-ES, a través de códigos intermedios de SNOMED y OMOP.

Además, se descubrió que el número de mapeos entre CIAP-1 y SNOMED-CT a priori utilizado en esta función era considerablemente bajo, por lo que se utilizaron posteriormente expresiones regulares sobre el `concept_name` para aumentarlos. Gracias a este enfoque, el número de conceptos mapeados se incrementó a 606 y a su vez esto facilitó la creación de relaciones indirectas entre CIAP-1 y CIE-10-ES.

Algorithm 1 Mapeo de CIAP-1 a CIE-10-ES

```
1:  function MAPSCIAPCIE
2:      conceptos_CIAP ← DB("SELECT @rid, concept_code, concept_id FROM CIAP1")
3:      for all ciap ∈ conceptos_CIAP do
4:          snomed ← DB("SELECT snomed FROM mapeos WHERE concept_id = {ciap_id}")
5:          codigos_CIE10 ← DB("SELECT cie10 FROM mapeos WHERE snomed = {snomed}")
6:          for all cie10 ∈ codigos_CIE10 do
7:              cie10_rid ← DB("SELECT @rid FROM CIE10 WHERE concept_id = {cie10_id}")
8:              if no EXISTE_RELACION(ciap_rid, cie10_rid, 'hasICD10') then
9:                  CREAR_RELACION(ciap_rid, cie10_rid, 'hasICD10')
10:                 REGISTRAR_LOG(archivo_log, "{ciap_rid} - {cie10_rid}")
11:             end if
12:         end for
13:     end for
14: end function
```

Algoritmo 3.1: Función de mapeo de CIAP-1 a CIE-10-ES.

Capítulo 4

Creación de la herramienta

En este capítulo se describe la metodología seguida para el diseño e implementación de una aplicación que permita la visualización e interacción del grafo construido. Se detallan las decisiones técnicas tomadas, el diseño arquitectónico del sistema y el desarrollo de cada componente, desde el backend y frontend hasta la integración.

4.1. Tecnologías utilizadas

Para el desarrollo de este proyecto, se han seleccionado tecnologías específicas que permiten almacenar, gestionar y visualizar de forma eficiente el grafo de terminologías médicas. A continuación, se describen brevemente estas herramientas y su papel en la solución.

OrientDB

OrientDB es una base de datos orientada a grafos que combina las características de una base de datos de documentos con las capacidades de manejo de relaciones propias de un grafo. Además, tiene también orientación a objetos, lo que permite almacenar tanto los nodos como las aristas de forma eficiente en distintas clases y establecer reglas de herencia entre dichas clases.

En este proyecto, OrientDB se utiliza como el motor principal para almacenar y utilizar el grafo de terminologías médicas. Su soporte para consultas con SQL extendido facilita la recuperación de datos complejos, como la búsqueda de nodos conectados por relaciones específicas o la expansión de nodos para visualizar sus conexiones. Sin embargo, su interfaz de usuario nativa es técnica y poco intuitiva, lo que motiva la creación de una solución gráfica más accesible. En particular, los nodos muestran por defecto el `@rid` y el nombre de las relaciones está predefinido en la base de datos, dificultando su entendimiento por parte de usuarios menos técnicos y generando un problema de accesibilidad. Esto se percibe en la Figura 4.1.

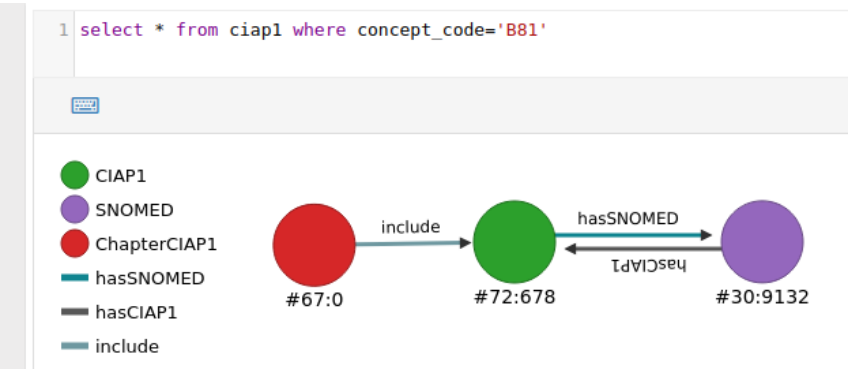


Figura 4.1: Interfaz de OrientDB.

Flask

Flask [20] es un framework ligero de Python para el desarrollo de aplicaciones web.

En este trabajo, Flask actúa como gestor del backend, encargado de procesar las consultas al grafo almacenado en OrientDB y proporcionar los resultados al frontend, a través de endpoints específicos para buscar nodos y relaciones, así como para expandir las conexiones de un nodo seleccionado. Esto permite que las consultas sean dinámicas y específicas, garantizando una experiencia fluida para los usuarios finales.

Cytoscape.js

Cytoscape.js [21] es una biblioteca de JavaScript diseñada para la visualización y el análisis de grafos interactivos. Proporciona una gran variedad de herramientas para personalizar la apariencia de los nodos y aristas, gestionar eventos de usuario, y aplicar algoritmos de disposición para organizar los elementos del grafo.

En este proyecto, Cytoscape.js se utiliza en el frontend, apoyado en HTML y CSS, para representar visualmente el grafo de terminologías. Los usuarios pueden buscar nodos, visualizar relaciones, y expandir conexiones con un alto grado de interactividad. Además, su flexibilidad permite personalizar aspectos como colores, etiquetas y estilos, facilitando la interpretación de los datos.

Docker

Docker [22] es una plataforma que permite empaquetar aplicaciones y sus dependencias en contenedores, asegurando que se ejecuten de manera consistente en diferentes entornos.

En este trabajo, Docker se utiliza para garantizar la portabilidad de la solución. Mediante un `Dockerfile` y un archivo `docker-compose.yml`, la aplicación puede desplegarse fácilmente en cualquier sistema, sin necesidad de configuraciones adicionales. Esto facilita la colaboración entre equipos y asegura que los usuarios finales puedan instalar y ejecutar la herramienta sin dificultades técnicas.

4.2. Diseño de la solución

El sistema desarrollado está compuesto por tres componentes principales: el frontend, el backend y la infraestructura de despliegue. A continuación en la Figura 4.2, se muestra mediante un diagrama cada uno de ellos y su interacción.

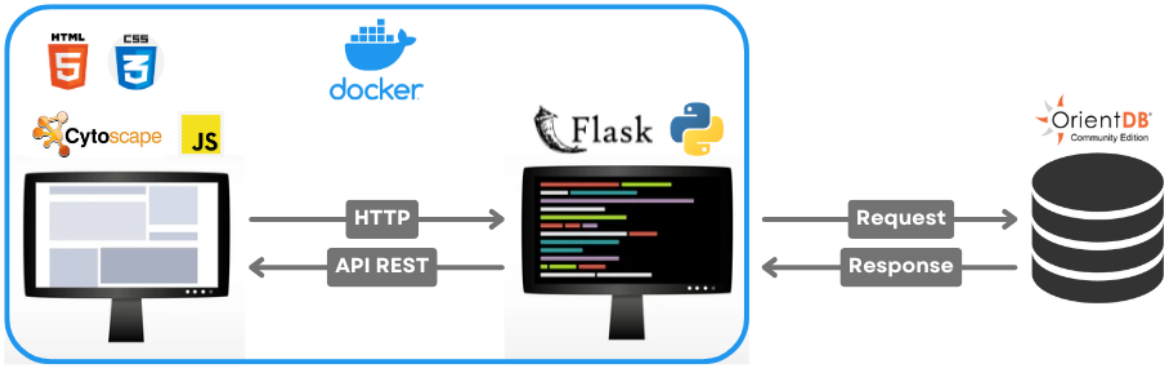


Figura 4.2: Diagrama de la arquitectura del sistema.

- El frontend envía solicitudes HTTP al backend para buscar diagnósticos (`search.js`) o expandir correspondencias de alguno de ellos (`graph.js`) mediante endpoints.
- El backend (`flaskAPI.py`) consulta la base de datos OrientDB utilizando SQL extendido y procesa los resultados para devolverlos en un formato que el frontend pueda utilizarlo (`nodes` y `edges`).
- Docker empaqueta la aplicación, asegurando que todos los componentes sean fácilmente desplegables y reproducibles.

Requisitos

Como ya hemos visto, OrientDB posee una interfaz poco funcional e intuitiva. En particular, las búsquedas se realizan mediante consultas SQL y los atributos de los nodos son difíciles de identificar y comprender. Además, no se puede visualizar de ninguna manera la disposición jerárquica de las terminologías, algo fundamental a la hora de comprender los datos clínicos. Es por ello que se ha desarrollado esta aplicación web, la cual debe cumplir con los siguientes requisitos funcionales:

1. *Búsqueda avanzada:* Implementar un sistema de búsqueda por términos o códigos, que permita a todo tipo de usuario localizar diagnósticos específicos o categorías relevantes dentro del grafo, sin necesidad de tener conocimientos en bases de datos. Incluir filtros para delimitar las búsquedas según criterios específicos como las terminologías.
2. *Visualización clara de los atributos:* Mostrar los atributos de los nodos y las relaciones de una forma clara, organizada y accesible, para que el usuario pueda entender rápidamente la información relevante.
3. *Incluir ventanas emergentes*, paneles laterales o herramientas similares que permitan ver, de forma rápida e intuitiva, datos como nombres de conceptos, códigos asociados y equivalencias entre terminologías, mejorando la experiencia de usuario.
4. *Exploración interactiva del grafo:* Permitir la expansión dinámica de nodos, mostrando las relaciones con otros términos y facilitando la navegación por el grafo. Ofrecer opciones para eliminar elementos, permitiendo una vista más organizada y personalizada según las necesidades del usuario.
5. *Exportación de datos:* Habilitar la posibilidad de exportar los nodos y relaciones visibles en el grafo a formatos estándar, como CSV, para su posterior análisis o integración con otras herramientas.

Como requisitos no funcionales:

1. *Portabilidad y facilidad de despliegue:* Utilizar Docker para empaquetar la aplicación y garantizar su portabilidad y facilidad de despliegue en diferentes entornos, asegurando que la herramienta sea accesible tanto en servidores locales como en entornos de red.
2. *Interfaz accesible para usuarios no especializados:* Diseñar la herramienta con un enfoque en la usabilidad, asegurando que profesionales de la salud sin experiencia en bases de datos o programación puedan utilizarla de manera eficiente. Proveer una documentación de ayuda clara y completa que facilite el uso.

4.3. Backend

El backend posee la lógica del sistema y es el responsable de procesar las consultas del frontend, comunicándose con la base de datos OrientDB. Está implementado en Flask de Python y ofrece los dos siguientes endpoints principales:

4.3.1. Endpoints

- `/api/search`

  Este endpoint permite buscar nodos en el grafo y las correspondencias entre ellos, utilizando filtros por término, código o terminología. Es ideal para localizar conceptos específicos o explorar categorías de terminologías.

  Mediante un método HTTP GET, el endpoint recibe como parámetros de entrada un término, un código o una terminología (o un conjunto de ellas), o una combinacion de todos ellos. Se recuperan entonces todas las entradas de la base de datos que coincidan parcialmente con esa búsqueda y se formatean como nodos y aristas en un formato JSON. Si no hay coincidencias o parámetros de entrada, se devuelve un grafo vacío.

  Su definición se puede consultar en el Anexo **A.1.1**.

- `/api/expand`

  Este endpoint permite expandir una o varias correspondencias de un nodo concreto del grafo. Es útil para explorar conexiones detalladas entre conceptos.

  Mediante un método HTTP GET, el endpoint recibe como parámetros de entrada el identificador del nodo y la lista de los identificadores de las relaciones a expandir. Se recuperan entonces todas las entradas de la base de datos que coincidan con esas relaciones y se formatean como nodos y aristas.

  Su definición se puede consultar en el Anexo **A.1.2**.

La documentación de la API se ha realizado mediante la herramienta Swagger [23] y puede consultarse en el Anexo **A.1.3**.

4.3.2. Disposición del grafo por niveles

Los nodos que se envían al frontend desde cualquiera de los dos endpoints tienen el siguiente formato:

```
node_data = {
    "data": {
        "id": str(node['id']),                # @rid
        "label": (label or "").capitalize(), # concept_name o sinónimos
        "code": node['code'],                 # concept_code
        "vocab": node['vocab'],               # @class
        "level": calculate_node_level(node)  # Nivel jerárquico calculado
        "in_relations": in_relations,         # Relaciones IN
        "out_relations": out_relations        # Relaciones OUT
    }
}
```

El atributo `level` se calcula en esta parte y se asigna a cada nodo del grafo para representar su jerarquía o distancia relativa respecto a los nodos raíz. Este nivel es esencial para organizar el grafo de manera lógica y visualmente comprensible, lo cual se detallará más en profundidad en la Sección 4.4.1.

4.4. Frontend

El frontend ha sido diseñado con el objetivo de proporcionar una experiencia de usuario intuitiva y eficiente, facilitando la interacción con el grafo de terminologías médicas. Está desarrollado con HTML, CSS y JavaScript, e integra la biblioteca `Cytoscape.js` para la visualización interactiva del grafo y `SweerAlert2` [25] para los pop-ups. A continuación, describimos detalladamente las funcionalidades principales implementadas y los principios de Interacción Persona-Ordenador aplicados.

4.4.1. Funcionalidades del Frontend

Para obtener una visión completa de todas las funcionalidades, se ha desarrollado un mapa de navegación, el cual puede observarse en el Anexo B.

Inicio

La visualización inicial de la herramienta web es la que muestra la Figura 4.3. Gran parte de la pantalla está destinada a espacio para el grafo ya que es el componente fundamental. El resto de componentes se detallarán en lo que sigue de sección.

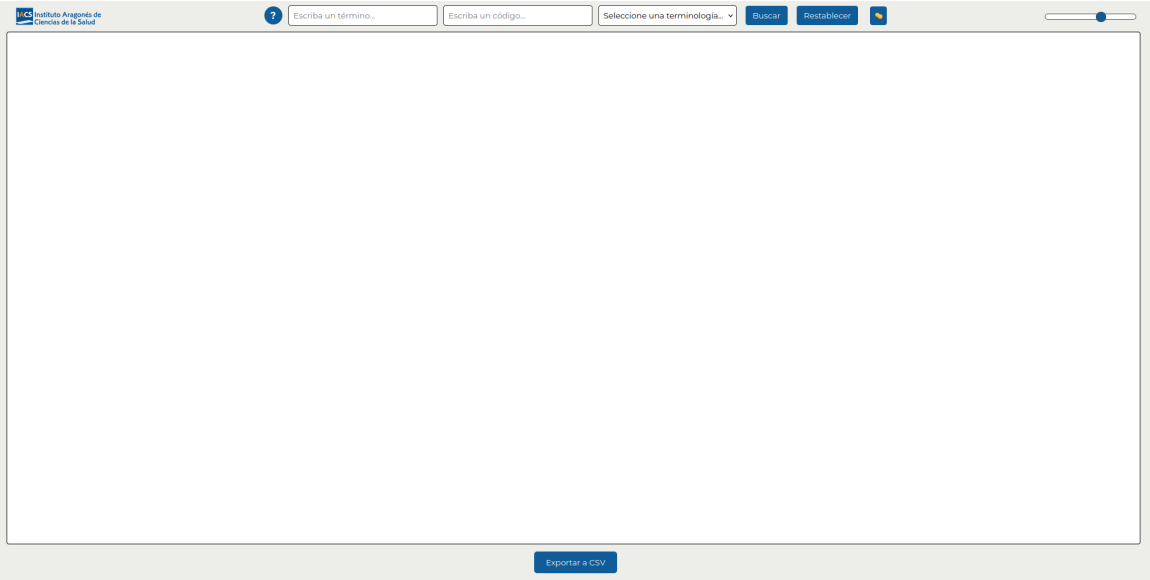


Figura 4.3: Visualización inicial de la herramienta.

Búsqueda con filtros

El usuario puede buscar diagnósticos y correspondencias del grafo utilizando términos y/o códigos exactos o parciales, teniendo también la posibilidad de filtrar por terminología mediante un menú desplegable (Figura 4.6). Esta funcionalidad se implementa con un formulario que envía las consultas al backend utilizando el endpoint `/api/search`. El botón *Restablecer* vacía el formulario y el grafo.

El código completo de esta funcionalidad se encuentra en el Anexo **A.2.1**, y un ejemplo de búsqueda se muestra en las Figuras 4.4 y 4.5.

Figura 4.4: Barra de búsqueda vacía.

Figura 4.5: Barra de búsqueda completada.

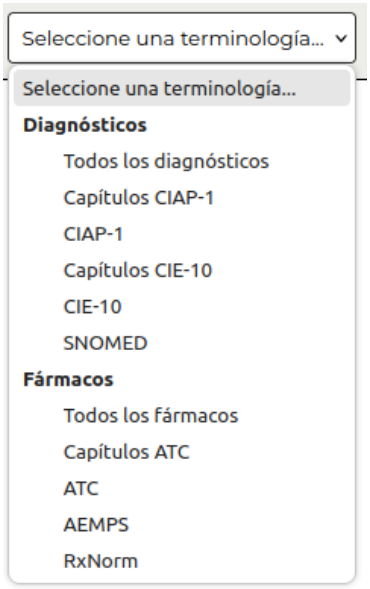


Figura 4.6: Menú desplegable de filtros para las terminologías.

Botón de ayuda

Un botón de ayuda a la izquierda de la barra de búsqueda proporciona al usuario información detallada sobre cómo utilizar la herramienta, tal y como muestra el pop-up de la Figura 4.7.

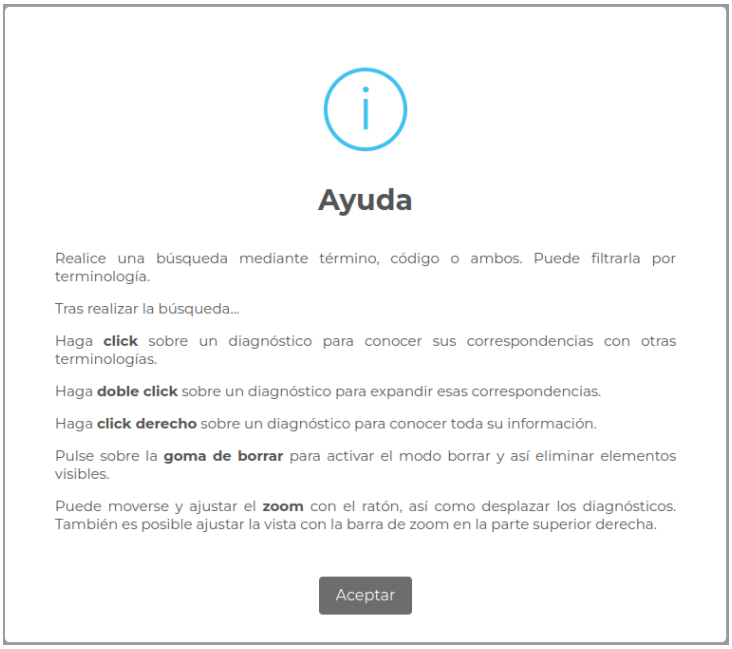


Figura 4.7: Pop-up de ayuda.

Disposición jerárquica del grafo y leyenda

Las disposiciones predeterminadas proporcionadas por la biblioteca `Cytoscape.js` resultan insuficientes para representar adecuadamente todas las características y propiedades inherentes al grafo. Por ello, ha surgido la necesidad de crear una función que organice y personalice la disposición de los nodos en el espacio, haciéndolo de manera jerárquica por niveles según la terminología. Esta función asigna unas coordenadas a cada nodo en función de la distancia al nodo raíz de su codificación, y de la dirección y el número de sus relaciones. El código para organizar el grafo se detalla en el Anexo **A.2.2**.

Además, se incluye una leyenda que indica el significado de los colores asignados a cada terminología, lo cual permite distinguir rápidamente los nodos. El contenido de la leyenda depende de si se ha buscado por las clases de diagnósticos o por las clases de fármacos.

La Figura 4.8 muestra la disposición jerárquica y la leyenda con la asignación de colores, en comparación a la Figura 4.9 equivalente en OrientDB.

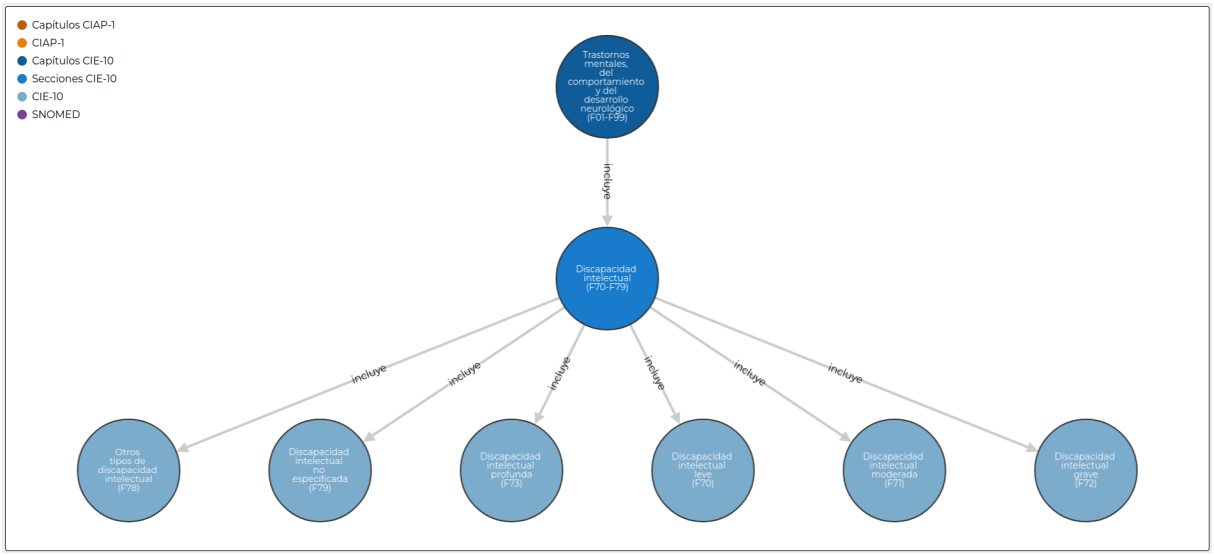


Figura 4.8: Disposición jerárquica del grafo y leyenda en la aplicación.

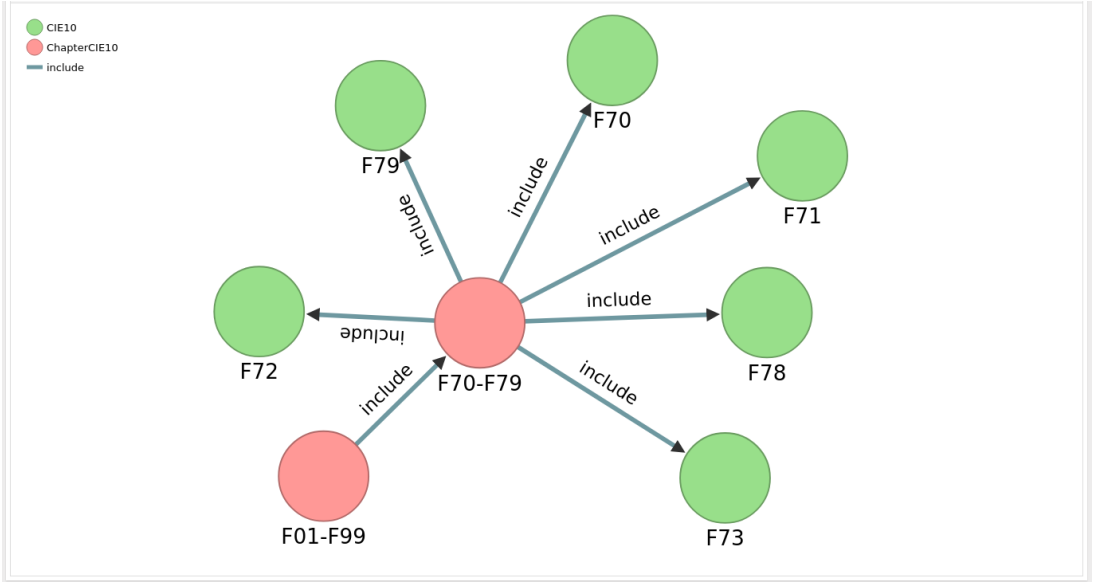


Figura 4.9: Disposición espacial del grafo y leyenda en OrientDB.

Interacción con el grafo mediante clics

Una vez realizada la búsqueda, la herramienta mostrará los diagnósticos encontrados y las relaciones entre ellos. A partir de entonces, podemos interactuar con los nodos de las siguientes formas:

- **Un clic:** Al hacer clic sobre un nodo, se muestra información sobre sus correspondencias entrantes y salientes. La nomenclatura de las relaciones definidas en la base de datos se modifica utilizando un mapeo definido en el código, para un mejor entendimiento por parte de la comunidad sanitaria.

  Las Figuras 4.10 y 4.11 muestran la acción descrita, siendo esta reversible mediante otro clic.

Figura 4.10: Nodo al inicio.

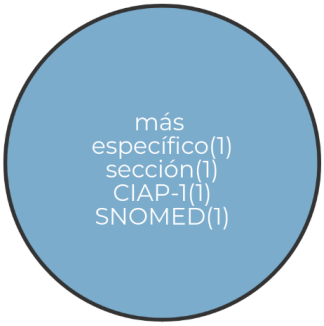


Figura 4.11: Nodo al hacer clic.

- **Doble clic:** Al realizar doble clic sobre el nodo, el sistema muestra un pop-up como el de la Figura 4.12 con las correspondencias del diagnóstico, dando a elegir cuáles de ellas se desea expandir y enviándolas al backend utilizando el endpoint `/api/expand`. El código de esta funcionalidad se encuentra en el Anexo **A.2.2**.

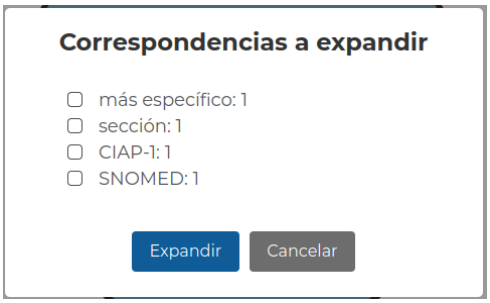


Figura 4.12: Pop-up al hacer doble clic.

- **Clic derecho:** Al hacer clic derecho sobre un nodo, se muestra un cuadro de información detallada con el nombre completo del diagnóstico, código y terminología como el de la Figura 4.13.

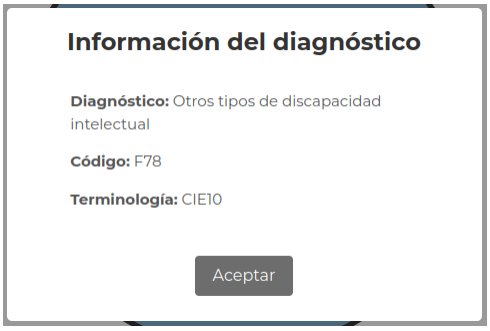


Figura 4.13: Pop-up al hacer clic derecho.

Modo borrar

Al hacer clic sobre el icono de borrar en la barra superior de la Figura 4.14, se muestra un cuadro que alerta de la activación del modo borrar como el de la Figura 4.15. En este modo, al hacer clic sobre cualquier nodo se elimina al instante, así como las relaciones que tenía. El botón se vuelve de color rojo para alertar al usuario de que está en este modo.

Figura 4.14: Botón de activar el modo borrar.

Figura 4.15: Pop-up al hacer clic en el botón.

Exportación de datos

En la parte inferior central de la pantalla, se sitúa un botón como el de la Figura 4.16 que permite exportar en formato CSV (Figura 4.17) los diagnósticos visibles para su análisis externo. Antes de producirse la exportación, el sistema avisa de la acción, tal y como muestra la Figura 4.18. Esta funcionalidad es crucial para garantizar que la información médica sea interpretada correctamente entre distintos sistemas, instituciones y regiones.

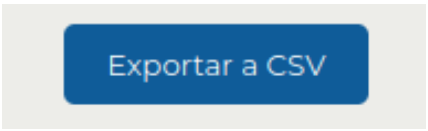


Figura 4.16: Botón exportar a CSV.

	A	B	C
1	Diagnóstico	Código	Terminología
2	Other genetic related intellectual disability	F78.A9	CIE10
3	Other genetic related intellectual disabilities	F78.A	CIE10
4	Syngap1-related intellectual disability	F78.A1	CIE10
5	Otros tipos de discapacidad intelectual	F78	CIE10

Figura 4.17: Fichero CSV exportado.

Figura 4.18: Pop-up de aviso pre-exportación.

Zoom y navegación

El usuario puede navegar por el grafo o ajustar el nivel de zoom mediante el ratón. Igualmente, se ha creado una barra de control de zoom como la de la Figura 4.19 para una mejor experiencia de usuario.

Figura 4.19: Barra de control de zoom.

4.4.2. Principios de Interacción Persona-Ordenador aplicados

- *Consistencia visual y funcional:* La interfaz mantiene estilos y colores uniformes para facilitar el aprendizaje del usuario.
- *Concordancia entre el sistema y el mundo real:* El botón rojo de eliminar o la goma de borrar concuerdan con el mundo del usuario.
- *Retroalimentación inmediata:* Las acciones del usuario, como clics y búsquedas, generan respuestas visuales o mensajes instantáneos.
- *Facilidad de uso:* Las opciones, como la búsqueda y los filtros, están diseñadas para minimizar la carga cognitiva del usuario.
- *Visibilidad del sistema:* La leyenda y la rueda de carga aseguran que el usuario siempre esté informado sobre el estado del sistema.
- *Flexibilidad:* Funciones como el zoom y la exportación permiten al usuario personalizar la interacción con el grafo según sus necesidades.
- *Estética y diseño minimalista:* La herramienta utiliza la misma gama de colores para diferentes funcionalidades y no existe ruido visual. Además, el tipo de letra está escogido para facilitar la lectura del usuario.
- *Prevención de errores:* El pop-up previo a la exportación a CSV de los elementos del grafo advierte al usuario de la acción que va a realizar.
- *Ayuda y documentación:* El botón de ayuda proporciona al usuario una documentación de las tareas que se pueden realizar en el sistema.

4.5. Integración con Docker

El código de la herramienta generado se ha subido al GitLab interno del IACS. Desde allí, se clona en los sistemas y se utiliza Docker para garantizar la portabilidad y facilidad de despliegue. Los siguientes archivos son clave en la configuración:

- `Dockerfile` define el entorno necesario para ejecutar el backend Flask: Usa una imagen base de Python, instala las dependencias necesarias, copia el código del proyecto y expone el puerto 5000 para las conexiones.
- `docker-compose.yml` simplifica el despliegue definiendo los servicios requeridos: El servicio `web` construye la imagen desde el `Dockerfile` y monta el código local en el contenedor. También se encarga de la exposición de puertos para permitir el acceso a la aplicación desde el navegador.

Por último, los comandos incluidos en el Anexo A.3 despliegan la aplicación.

Capítulo 5

Pruebas de usuario

5.1. Objetivos de las pruebas

El objetivo principal de las pruebas de usuario es evaluar la usabilidad, eficiencia, eficacia y satisfacción de los usuarios al interactuar con la herramienta de visualización del grafo. En particular, se busca:

- Determinar si los usuarios pueden realizar tareas específicas con éxito, como son las búsquedas, la exploración de relaciones y la exportación de datos.
- Identificar posibles problemas de usabilidad en la interfaz gráfica o en la interacción con el grafo.
- Recopilar retroalimentación de los usuarios sobre la experiencia general, claridad visual y accesibilidad de la herramienta.

5.2. Perfiles de usuario

Para garantizar resultados representativos, las pruebas se llevan a cabo con dos perfiles principales de usuarios:

- **Usuarios técnicos:** Desarrolladores con experiencia en tecnologías relacionadas con bases de datos o herramientas web.
- **Usuarios no técnicos:** Profesionales de la salud, como médicos, enfermeros o documentalistas clínicos, con conocimientos básicos de informática.

5.3. Pruebas diseñadas

Prueba 1

1. Buscar un diagnóstico específico por el término *'esquizofrenia'* y la terminología *'CIE-10-ES'*.
2. Identificar la disposición jerárquica.
3. Mirar que relaciones tiene el diagnóstico `F20` buscado.
4. Expandir la correspondencia SNOMED de ese diagnóstico.
5. Expandir la correspondencia CIAP-1 de ese SNOMED.
6. Expandir los capítulos CIE-10-ES y CIAP-1 que incluyen los diagnósticos.

Prueba 2

1. Restablecer la vista.
2. Buscar el código `'F45'`.
3. Eliminar de la vista los códigos que contengan la descripción "no especificado".
4. Exportar las relaciones visibles en el grafo a un archivo CSV.
5. Verificar la precisión de los datos exportados.

5.4. Análisis de resultados

Las pruebas se realizan en 6 usuarios, 3 técnicos y 3 no técnicos, combinando trabajadores del IACS y personal de salud externo. Para evaluar los resultados, se recopilan los siguientes datos cuantitativos y cualitativos:

Cuantitativo

- **Eficiencia:** Número de clics por persona requeridos para completar cada tarea.
- **Eficacia:** Número de tareas completadas con éxito y errores cometidos.

Cualitativo

- **Satisfacción:** Preguntas para evaluar la facilidad de uso, claridad y percepción estética, así como detectar patrones comunes en los errores y dificultades.
- **Gravedad:** Los problemas detectados se clasificaron por su gravedad (leves, moderados, críticos).

Los Cuadros 5.1 y 5.2 reflejan los datos cuantitativos obtenidos y a continuación se exponen y comentan los cualitativos.

Tarea	**Éxitos**	**Nºclics/persona**	**Errores detectados**
1. Buscar *esquizofrenia* en *CIE-10-ES*.	4	4	Filtrar por capítulo.
2. Identificar la jerárquica.	6	0	
3. Mirar relaciones.	6	1	
4. Expandir SNOMED.	6	3	
5. Expandir CIAP-1.	6	3	
6. Expandir los capítulos.	6	6	

Cuadro 5.1: Resultados de la Prueba 1

Tarea	**Éxitos**	**Nºclics/persona**	**Errores detectados**
1. Restablecer.	4	1	Recargar página.
2. Buscar F45.	5	2	Tecla Enter.
3. Eliminar nodos.	6	3	
4. Exportar.	6	2	
5. Verificar exportación.	5	0	Excel en Windows.

Cuadro 5.2: Resultados de la Prueba 2

Todos los usuarios participaron activamente en la propuesta de mejoras, aportando sugerencias valiosas para optimizar la herramienta.

Sugerencias

- *Filtro:* Los usuarios no técnicos señalaron que filtrar por capítulos resulta poco intuitivo, ya que es preferible hacerlo directamente por sistema de codificación (moderado).
- *Interfaz de usuario:* Incluir iconos en botones clave como 'Buscar' o 'Restablecer', lo cual podría mejorar la usabilidad (leve).
- *Funcionalidades adicionales:* Incorporar el uso de *CTRL+Z* para deshacer acciones y corregir errores de forma rápida (moderado) o permitir buscar con la tecla *Enter*, lo que haría el proceso más fluido (leve).
- *Orden y visualización:* Organizar los nodos por código de izquierda a derecha en el grafo y ordenar por especificidad al exportar los datos (leve).
- *Interacción con nodos:* Implementar la posibilidad de seleccionar y manipular múltiples nodos simultáneamente para moverlos o eliminarlos (leve), así como habilitar la opción de interactuar con la leyenda, permitiendo mostrar/ocultar nodos de una terminología específica al hacer clic en ella (leve).

Comentarios positivos

- **Colores:** La asignación de colores para cada terminología facilita la interpretación del grafo y mejora la comprensión de las relaciones entre los nodos.
- **Rueda de carga:** Es un elemento útil que informa al usuario sobre el estado actual del sistema y mejora la percepción de control durante la interacción.

En conclusión, las pruebas fueron recibidas de manera positiva, y los comentarios obtenidos proporcionan un camino claro para implementar mejoras.

Capítulo 6

Conclusiones

6.1. Resumen del impacto del trabajo

El desarrollo de este trabajo ha permitido abordar la necesidad de una herramienta web que facilite la visualización y exploración de grafos de diferentes terminologías médicas, promoviendo la interoperabilidad semántica y la comprensión de las relaciones internas y externas de diferentes sistemas de codificación (CIE-10-ES, SNOMED-CT, CIAP-1, etc.).

El proyecto partió de unas necesidades específicas, como la dificultad para comprender las relaciones entre terminologías y su impacto en la asignación de códigos por parte de la comunidad médica, debido también a la falta de herramientas accesibles e intuitivas para profesionales de la salud y desarrolladores. Además, la escasez de mapeos oficiales con terminologías no estandarizadas en OMOP, como CIAP-1, utilizada en atención primaria en Aragón, propició la necesidad de completar estas relaciones en la base de datos de OrientDB.

Entre las principales aportaciones del trabajo se encuentra, por un lado, la participación en el diseño y poblado de la base de datos, siendo necesario el diseño de algoritmos capaces de trabajar con grandes volúmenes de datos; y por otro, la implementación de una interfaz visual que facilite la búsqueda filtrada y la exploración de los conceptos médicos a través de grafos. La herramienta proporciona una experiencia clara y organizada gracias al uso de colores diferenciados para cada terminología, e incorpora funcionalidades como la interacción dinámica con los nodos mediante clics y la posibilidad de manipular y exportar elementos del grafo, adaptándose a las necesidades de todo tipo de usuario. Todo ello gracias al uso de tecnologías como Flask de Python o la librería Cytoscape de JavaScript.

La herramienta desarrollada ha demostrado ser útil y accesible, tal y como lo refleja el interés mostrado por los usuarios durante las pruebas realizadas. Actualmente, la aplicación está siendo utilizada por profesionales sanitarios y técnicos como complemento a sus herramientas laborales diarias, lo que denota su potencial impacto positivo en comparación a otras metodologías mucho más costosas en tiempo.

En términos generales, se considera que los objetivos iniciales del proyecto han sido cumplidos. Se ha obtenido un mayor conocimiento de las clasificaciónes biomédicas y como estas pueden ser utilizadas por científicos de datos. Además, se ha construido una herramienta funcional que responde a las necesidades planteadas, facilitando la comprensión y visualización de terminologías médicas de forma eficiente, y que puede ser utilizada gracias a su despliegue con Docker, tanto por profesionales de la salud como por desarrolladores del IACS interesados en mejorar la interoperabilidad semántica en el ámbito sanitario.

Por último, el tiempo dedicado en el IACS ha sido clave para el desarrollo de este trabajo. Durante este periodo, se han adquirido conocimientos prácticos en informática médica, además de habilidades en el manejo de tecnologías orientadas a grafos. La colaboración con los directores del TFG y otros profesionales que trabajan allí ha permitido superar los retos del proyecto y sentar las bases para futuras mejoras y aplicaciones prácticas.

6.2. Trabajo futuro

A pesar de los avances alcanzados, el proyecto es un punto de partida que deja abiertas diversas líneas de trabajo para ampliar y mejorar su funcionalidad. Entre las posibles extensiones y aplicaciones futuras se incluyen:

- Evaluar si la utilidad de las **sugerencias obtenidas en las pruebas** se ajusta al público objetivo y compensa su coste de implementación.
- Mejorar la **disposición en el espacio** de los nodos para aumentar la claridad en el caso de búsquedas poco específicas que generan una gran cantidad de nodos.
- Ampliar el número y la variedad de **terminologías** como versiones anteriores o complementarias a CIE-10, por ejemplo CIE-9 o CIE-10-ES (Procedimientos).
- Exportar los grafos generados a otros formatos estándar como **RDF (Resource Description Framework)** permitiendo, no solo la homogenización de los datos, sino también su integración en la web semántica, facilitando la creación de redes de datos enlazados y su reutilización por parte de otros sistemas y usuarios.
- Incorporar la capacidad de **importar datos desde CSV o RDF** para reconstruir el grafo existente, mejorando la compatibilidad con otras tecnologías y la continuidad del uso de la herramienta en el proceso de investigación.
- **Validar la herramienta en otros entornos**, como centros médicos y hospitales, para probar su utilidad práctica y recoger retroalimentación directa de los usuarios finales.
- Añadir atributos reales a los nodos del grafo, como un conteo del **número de episodios asociados a un diagnóstico** en un intervalo de tiempo específico, uniendo la ontología a bases de datos de registros clínicos.
- Incluir **pesos en los ejes del grafo** para representar la fuerza o relevancia de una relación, evaluando la similitud o distancia entre términos de la misma o de distintas codificaciones. Aplicando conceptos de Teoría de Grafos y explorando algoritmos para determinar estas métricas de proximidad, se pueden identificar los términos de mayor relevancia y afinidad semántica dentro de la ontología. Todo esto se enmarcará en el Trabajo de Fin de Grado de Matemáticas.

Apéndice A

Anexo: Código fuente

El código completo de la aplicación puede consultarse en el siguiente repositorio: `https://github.com/801467/tfg`

A.1. Backend: `flaskAPI.py`

A.1.1. Endpoint: `/api/search`

```
@app.route('/api/search', methods=['GET'])
def search_graph():
    term = request.args.get('term', '')
    code = request.args.get('code', '')
    vocab = request.args.get('vocab', '')

    # Si ninguno de los parametros tiene valor
    if not term and not code and not vocab:
        return jsonify({"nodes": [], "edges": []})

    try:
        # 1. Obtenemos nombre de todas las relaciones
        relations_query = """
            SELECT name
            FROM (SELECT expand(classes) FROM metadata:schema)
            WHERE superClass = 'E'
        """
        ...

        # 2. Obtenemos nodos (y relaciones de cada nodo)
        select_fields = [
            "@rid AS id",
            "concept_name AS label",
            "IFNULL(concept_name_spanish, '') AS label2",
            "IFNULL(synonymous_concept, '') AS label3",
            "concept_code AS code",
            "@class AS vocab"
        ]
        for relation in relations:
            select_fields.append(f"in_{relation} AS in_{relation}")
            select_fields.append(f"out_{relation} AS out_{relation}")

        node_query = f"SELECT {', '.join(select_fields)} FROM V"
        filters = []
        if term:
```

```
            filters.append(f"(concept_name LIKE '%{term}%' OR
                concept_name_spanish LIKE '%{term}%' OR
                synonymous_concept LIKE '%{term}%')")
        if code:
            filters.append(f"concept_code LIKE '%{code}%'")
        if vocab:
            vocab_list = [v.strip() for v in vocab.split(',')]
            filters.append(f"@class IN {vocab_list}")
        if filters:
            node_query += " WHERE " + " AND ".join(filters)
        node_query += " LIMIT -1"
        ...

        nodes = []
        for node in nodes_result:
            # tratamos label
            ...
            # tratamos relaciones
            ...

            node_data = {
                "data": {
                    "id": str(node['id']),
                    "label": (label or "").capitalize(),
                    "code": node['code'],
                    "vocab": node['vocab'],
                    "level": calculate_node_level(node, None),
                    "in_relations": in_relations,
                    "out_relations": out_relations
                }
            }
            nodes.append(node_data)

        # 3. Obtenemos relaciones entre esos nodos
        node_ids = [f"'{str(node['id'])}'" for node in nodes_result]

        if node_ids:
            edges_query = f"SELECT @rid AS id, out AS source, in AS
                target, @class AS label FROM E WHERE out IN [{', '.join(
                node_ids)}] AND in IN [{', '.join(node_ids)}] LIMIT -1"
            try:
                ...

            except requests.exceptions.HTTPError as e:
                print(f"Error en la obtencion de relaciones: {e}")
                edges = []
        else:
            edges = []

        return jsonify({"nodes": nodes, "edges": edges})

    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

Listing A.1: Definición del endpoint /api/search

A.1.2. Endpoint: /api/expand

```
@app.route('/api/expand', methods=['GET'])
def expand_node():
    node_id = request.args.get('node_id')
    node_level = request.args.get('node_level', type=int)
    edge_ids = request.args.get('edge_ids', '')
    if not node_id or not edge_ids:
        return jsonify({"nodes": [], "edges": []})

    try:
        # 1. Obtenemos nombre de todas las relaciones
        ...
        # 2. Obtenemos nodos relacionados (y relaciones de cada nodo)
        edge_ids_condition = ",".join([f"'{edge_id.strip()}'" for
           edge_id in edge_ids.split(',')])
        select_fields = [
            "@rid AS id",
            "concept_name AS label",
            "IFNULL(concept_name_spanish, '') AS label2",
            "IFNULL(synonymous_concept, '') AS label3",
            "concept_code AS code",
            "@class AS vocab"
        ]
        for relation in relations:
            select_fields.append(f"in_{relation} AS in_{relation}")
            select_fields.append(f"out_{relation} AS out_{relation}")

        node_query = f"""
            SELECT {', '.join(select_fields)} FROM V
            WHERE @rid IN (SELECT DISTINCT expand(bothV()) FROM E WHERE
               @rid IN [{edge_ids_condition}]) LIMIT -1
        """
        ...
        nodes = []
        for node in nodes_result:
            # tratamos label
            ...
            # tratamos relaciones
            ...
            node_data = {
                ...
            }
            nodes.append(node_data)

        # 3. Incluir relaciones seleccionadas
        edges_query = f"""
        SELECT @rid AS id, out AS source, in AS target, @class AS label
        FROM E
        WHERE @rid IN [{edge_ids_condition}] LIMIT -1
        """
        ...
        return jsonify({"nodes": nodes, "edges": edges})

    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

Listing A.2: Definición del endpoint /api/expand

A.1.3. Documentación API: swagger.json

```
{
    "swagger": "2.0",
    "basePath": "/",
    "paths": {
        "/api/expand": {
            "get": {
                "responses": {
                    "200": {
                        "description": "exito",
                        "schema": {
                            "$ref": "#/definitions/ExpandResponse"
                        }
                    }
                },
                "description": "Expande un nodo especifico para obtener
                   sus relaciones",
                "operationId": "get_expand_node",
                "parameters": [
                    {
                        "description": "Identificador del nodo que se
                           desea expandir",
                        "name": "node_id",
                        "type": "string",
                        "in": "query"
                    },
                    {
                        "description": "Nivel del nodo actual (opcional)
                           ",
                        "name": "node_level",
                        "type": "string",
                        "in": "query"
                    },
                    {
                        "description": "Identificadores de las
                           relaciones que se desean expandir, separados
                           por comas",
                        "name": "edge_ids",
                        "type": "string",
                        "in": "query"
                    }
                ]
            }
        },
        "/api/search": {
            "get": {
                "responses": {
                    "200": {
                        "description": "exito",
                        "schema": {
                            "$ref": "#/definitions/SearchResponse"
                        }
                    }
                },
                "description": "Busca nodos y relaciones en el grafo",
                "operationId": "get_search_graph",
                "parameters": [
```

```
                    {
                        "in": "query",
                        "description": "Sistema de codificacion a
                           filtrar",
                        "name": "vocab",
                        "type": "string"
                    },
                    {
                        "in": "query",
                        "description": "Codigo a buscar",
                        "name": "code",
                        "type": "string"
                    },
                    {
                        "in": "query",
                        "description": "Termino a buscar",
                        "name": "term",
                        "type": "string"
                    }
                ]
            }
        }
    },
    ...
    "definitions": {
        ...
        "Node": {
            "properties": {
                "id": {
                    "type": "string",
                    "description": "Identificador unico del nodo"
                },
                "label": {
                    "type": "string",
                    "description": "Etiqueta del nodo"
                },
                "code": {
                    "type": "string",
                    "description": "Codigo del nodo"
                },
                "vocab": {
                    "type": "string",
                    "description": "Sistema de codificacion al que
                       pertenece el nodo"
                },
                "level": {
                    "type": "integer",
                    "description": "Nivel del nodo"
                },
                "in_relations": {
                    "type": "object",
                    "description": "Relaciones entrantes del nodo"
                },
                "out_relations": {
                    "type": "object",
                    "description": "Relaciones salientes del nodo"
                }
            },
```

```
            "type": "object"
        },
        "Edge": {
            "properties": {
                "id": {
                    "type": "string",
                    "description": "Identificador de la relacion"
                },
                "source": {
                    "type": "string",
                    "description": "Nodo de origen"
                },
                "target": {
                    "type": "string",
                    "description": "Nodo de destino"
                },
                "label": {
                    "type": "string",
                    "description": "Tipo de relacion"
                }
            },
            "type": "object"
        },
        ...
    },
    ...
}
```

Listing A.3: Documentación de la API con Swagger.

A.2. Frontend

A.2.1. search.js

```
function performSearch() {
    const labelValue = document.getElementById('search-label').value.
       trim().toLowerCase();
    const codeValue = document.getElementById('search-code').value.trim
       ().toLowerCase();
    const classValue = document.getElementById('class-filter').value;

    let vocab = '';
    if (classValue === 'diagnosticos') {
        vocab = 'ChapterCIAP1,CIAP1,ChapterCIE10,CIE10,SNOMED';
    } else if (classValue === 'farmacos') {
        vocab = 'LevelATC,ATC,Aemps,RxNorm';
    } else {
        vocab = classValue;
    }
    updateLegend(classValue);
    if (labelValue || codeValue || vocab) {
        showLoadingSpinner();
        const searchParams = new URLSearchParams({
            term: labelValue,
            code: codeValue,
            vocab: vocab
        });

        fetch(`/api/search?${searchParams.toString()}`)
            .then(response => response.json())
            .then(data => {
                hideLoadingSpinner();
                if (data.nodes.length > 0) {
                    updateGraph(data);
                } else {
                    Swal.fire({
                        title: '!',
                        text: 'No se ha encontrado ninguna coincidencia.
                            ',
                        confirmButtonText: 'Aceptar',
                        confirmButtonColor: '#6d6d6d',
                    });
                }
            })
            .catch(error => {
                hideLoadingSpinner();
                console.error('Error en la busqueda:', error);
            });
    } else {
        Swal.fire({
            title: 'Campos vacios',
            text: 'Por favor, escriba en al menos uno de los campos de
               busqueda.',
        });
    }
}
```

Listing A.4: Función de búsqueda de diagnósticos.

A.2.2. graph.js

```
cy.on('dblclick', 'node', function (evt) {
    // pop-up
    ...
    }).then((result) => {
        if (result.isConfirmed) {
            const selectedEdgeIds = result.value;
            document.getElementById('loading-spinner').classList.remove(
                'hidden');
            const edgeIdsParam = selectedEdgeIds.map(id =>
                encodeURIComponent(id)).join(',');

            // Solicitud al servidor con los tipos de relaciones
                seleccionadas
            fetch(`/api/expand?node_id=${encodeURIComponent(nodeId)}&
                node_level=${encodeURIComponent(nodeLevel)}&edge_ids=${
                edgeIdsParam}`)
            .then((response) => response.json())
                .then((data) => {
                    if (data.nodes.length > 0) {
                        cy.add(data.nodes);
                        cy.add(data.edges);
                        organizeGraphByLevels();
                        cy.zoom(currentZoom);
                        cy.pan(currentPan);
                    } else {
                        Swal.fire({
                            title: 'Sin diagnosticos relacionados',
                            text: 'No se encontraron diagnosticos
                                relacionados para las correspondencias
                                seleccionadas.',
                            icon: 'info',
                            confirmButtonText: 'Aceptar',
                        });
                    }
                })
                .catch((error) => {
                    console.error('Error al expandir el nodo:', error);
                    Swal.fire({
                        title: 'Error',
                        text: 'Ocurrio un error al intentar expandir las
                            correspondencias del diagnostico.',
                        icon: 'error',
                        confirmButtonText: 'Aceptar',
                    });
                })
                .finally(() => {
                    document.getElementById('loading-spinner').classList
                        .add('hidden');
                });
        }
    });
});
```

Listing A.5: Función de doble clic sobre un diagnóstico.

```
function organizeGraphByLevels() {
    const levelSpacing = 150; // Vertical
    const horizontalSpacing = 150; // Horizontal
    const processedNodes = new Set(); // Set para evitar repetir nodo

    const levelMap = {}; // Agrupamos nodos por nivel y ordenamos
       niveles.
    cy.nodes().forEach(node => {
        const level = node.data('level');
        if (!levelMap[level]) {
            levelMap[level] = [];
        }
        levelMap[level].push(node);
    });
    const levels = Object.keys(levelMap).map(Number).sort((a, b) => a -
       b);

    // Asignamos posiciones a los nodos empezando desde el nivel mas
       bajo
    levels.forEach(level => {
        const nodesInLevel = levelMap[level];

        nodesInLevel.forEach(node => {
            if (processedNodes.has(node.id())) {
                return; // Saltamos nodos ya procesados
            }
            const parentEdges = node.connectedEdges().filter(edge => {
                const isIsaOrSynonym = edge.data('label') === 'Isa' ||
                   edge.data('label') === 'synonym';
                if (isIsaOrSynonym) {
                    return edge.source().id() === node.id(); // Relacion
                        Isa
                } else {
                    return edge.target().id() === node.id(); // Otras
                }
            });
            if (parentEdges.length > 0) { // Caso 1: Centrar los nodos
               hijos respecto al padre
                let parentNode;
                if (parentEdges[0].source().id() === node.id()) {
                    parentNode = parentEdges[0].target(); // Relacion
                       Isa o synonym
                } else {
                    parentNode = parentEdges[0].source();
                }
                const parentPosition = parentNode.position();
                const siblingEdges = parentNode.connectedEdges().filter(
                   edge => {
                    const isIsaOrSynonym = edge.data('label') === 'Isa'
                       || edge.data('label') === 'synonym';
                    if (isIsaOrSynonym) {
                        return edge.target().id() === parentNode.id();
                           // Relacion Isa o synonym
                    } else {
                        return edge.source().id() === parentNode.id();
                    }
```

```
                });
                const siblings = siblingEdges.map(edge => {
                    let siblingNode;
                    const isIsaOrSynonym = edge.data('label') === 'Isa'
                        || edge.data('label') === 'synonym';
                    if (isIsaOrSynonym) {
                        siblingNode = edge.source(); // Relacion Isa o
                            synonym
                    } else {
                        siblingNode = edge.target();
                    }
                    return siblingNode;
                });
                const siblingCount = siblings.length;
                const totalSiblingWidth = (siblingCount - 1) *
                    horizontalSpacing;
                const offsetX = parentPosition.x - totalSiblingWidth /
                    2;
                const siblingIndex = siblings.findIndex(sibling =>
                    sibling.id() === node.id());
                const xPosition = offsetX + siblingIndex *
                    horizontalSpacing;
                const yPosition = parentPosition.y + levelSpacing;
                node.position({ x: xPosition, y: yPosition });

            } else { // Caso 2: Nodo raiz (no tiene padre)
                const indexInLevel = levelMap[level].indexOf(node);
                let xPosition;
                if (level === 1 && nodesInLevel.length === 2) {
                    xPosition = indexInLevel * horizontalSpacing*40; //
                        Espaciado especial para nivel 1
                } else {
                    xPosition = indexInLevel * horizontalSpacing; //
                        Espaciado normal
                }const yPosition = level * levelSpacing;
                node.position({ x: xPosition, y: yPosition });
            }
            processedNodes.add(node.id());
        });
    });

    cy.fit(cy.elements(), 50);
}
```

Listing A.6: Función para organizar el grafo por niveles.

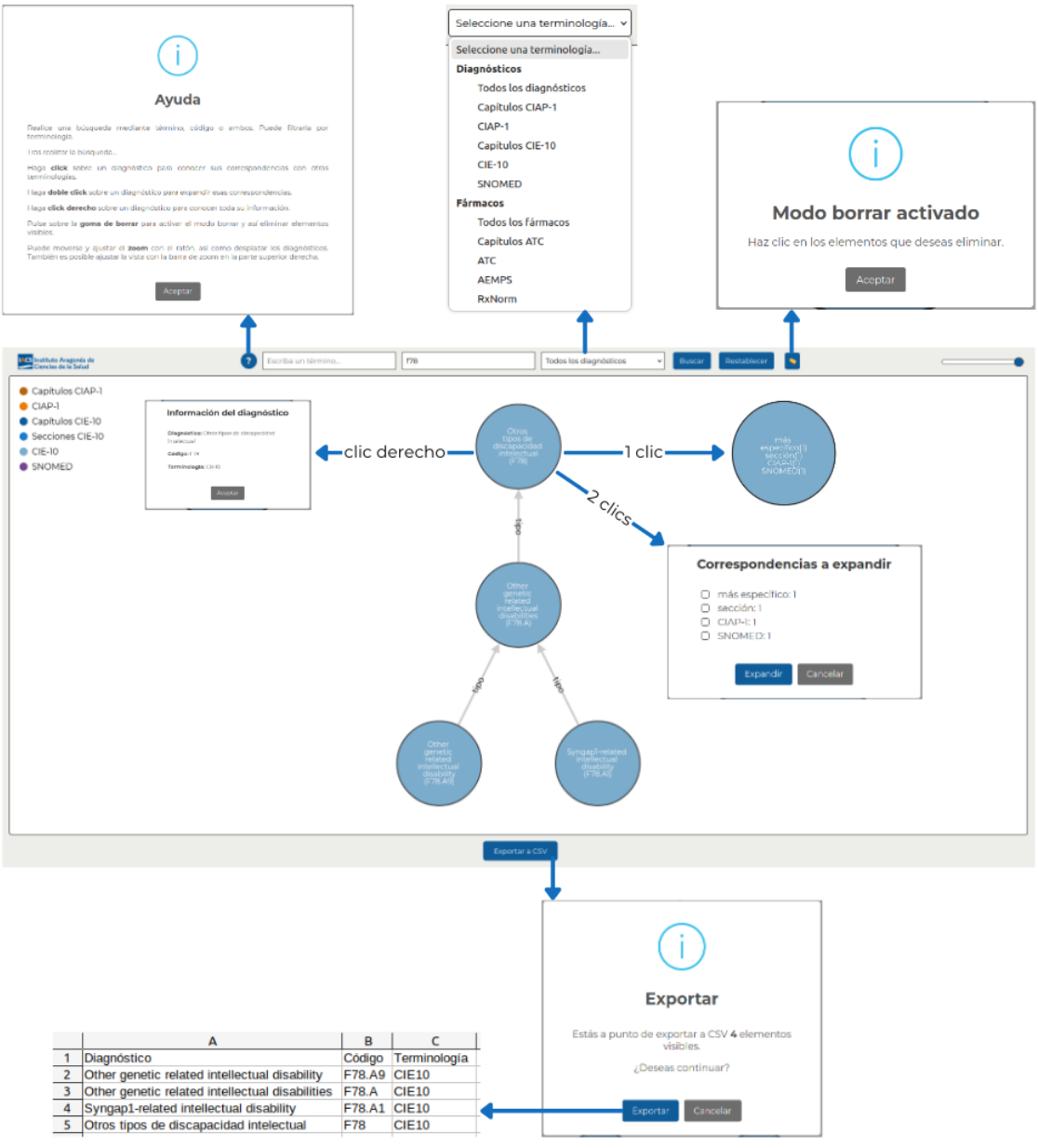
A.3. Despliegue

```
# Acceso a la carpeta que contiene el repositorio Git clonado
cd <carpetaConGit>

# Actualizar el repositorio local con los últimos cambios
git pull origin master

# Listar todas las imágenes de Docker disponibles en el sistema
docker images

# Mostrar todos los contenedores de Docker
docker ps -a

# Detener el contenedor con el nombre "interface_ontology" (si está corriendo)
docker stop interface_ontology

# Eliminar el contenedor y la imagen
docker rm interface_ontology
docker rmi interface_ontology:latest

# Reconstruir la imagen de Docker con el Dockerfile
docker build -t interface_ontology .

# Regresar y levantar el servicio usando docker-compose
cd ..
docker compose up interface_ontology -d

# Comprobar el estado del contenedor
docker ps -a
```

Listing A.7: Script de Bash para desplegar la herramienta web.

Apéndice B

Anexo: Mapa de navegación



	A	B	C
1	Diagnóstico	Código	Terminología
2	Other genetic related intellectual disability	F78.A9	CIE10
3	Other genetic related intellectual disabilities	F78.A	CIE10
4	Syngap1-related intellectual disability	F78.A1	CIE10
5	Otros tipos de discapacidad intelectual	F78	CIE10

Figura B.1: Mapa de navegación de la aplicación.

Bibliografía

[1] CIE-10-ES, *Clasificación internacional de enfermedades, 10.ª edición, Modificación Clínica en Español*, https://www.sanidad.gob.es/estadEstudios/estadisticas/normalizacion/CIE10/2024/Manual_Codificacion_CIE10ES_Diagnosticos_5_Edicion_1774004053538742055.pdf.

[2] CIAP-2, *Clasificación Internacional de Atención Primaria, 2.ª edición*, https://www.who.int/standards/classifications/other-classifications/international-classification-of-primary-care.

[3] SNOMED-CT, *Nomenclatura Sistemática de la Medicina – Términos Clínicos*, https://snomedsns.es/.

[4] ATC, *Clasificación Anatómica, Terapéutica y Química*, https://www.sanidad.gob.es/estadEstudios/estadisticas/estadisticas/estMinisterio/SIAP/5FARMACOS_CONSUMO.pdf.

[5] RxNorm, https://www.nlm.nih.gov/research/umls/rxnorm/index.html.

[6] AEMPS, *Agencia Española de Medicamentos y Productos Sanitarios*, https://www.aemps.gob.es/.

[7] IACS, *Instituto Aragonés de Ciencias de la Salud*, https://www.iacs.es/.

[8] OrientDB Community Edition, http://orientdb.org/.

[9] OMOP CDM, *Observational Medical Outcomes Partnership Common Data Model*, https://www.ohdsi.org/data-standardization/.

[10] WONCA, *World Organization of Family Doctors*, https://www.globalfamilydoctor.com/espanol.

[11] OMS, *Organización Mundial de la Salud*, https://www.who.int/es.

[12] NCHS, *National Center for Health Statistics*, https://www.cdc.gov/nchs/index.html.

[13] IHTSDO, *International Health Terminology Standards Development Organisation*, https://www.snomed.org/.

[14] NLM, *National Library of Medicine*, https://www.nlm.nih.gov/.

[15] B2i Healthcare, *Snow Owl Docs*, https://docs.b2ihealthcare.com/snow-owl.

[16] i2b2, *Informatics for Integrating Biology and the Bedside*, https://www.i2b2.org/.

[17] International Organization for Standardization, *Electronic health records*, https://www.iso.org/healthcare/electronic-health-records.

.

[18] OBSERVATIONAL HEALTH DATA SCIENCES AND INFORMATICS, *Athena*, https://athena.ohdsi.org/search-terms/start.

[19] NATIONAL CENTER FOR BIOMEDICAL ONTOLOGY, *BioPortal*, https://bioportal.bioontology.org/.

[20] FLASK, *Flask Documentation*, https://flask.palletsprojects.com/en/stable/.

[21] JAVASCRIPT, *Cytoscape.js*, https://js.cytoscape.org/.

[22] DOCKER, https://www.docker.com/.

[23] SMARTBEAR, *Swagger*, https://swagger.io/.

[24] CANVA, https://www.canva.com/.

[25] SWEETALERT2, https://sweetalert2.github.io/.