PROYECTO

Captura de datos públicos de la Web, almacenamiento y procesamiento.

Clinic Cloud

CICLO FORMATIVO DE GRADO SUPERIOR

Desarrollo de Aplicaciones Multiplataforma

I.E.S. «Venancio Blanco» SALAMANCA

AUTOR*[ES]*

Rubén García Rodríguez

Licencia

Resumen

Clinic Cloud es un proyecto de software libre y gratuito que tiene como objetivo recopilar, almacenar y procesar datos médicos públicos disponibles en la web. A través de una arquitectura basada en microservicios, esta aplicación permitirá centralizar una gran cantidad de estudios médicos en una base de datos SQL, facilitando la búsqueda y el acceso a información relevante para profesionales de la salud, investigadores y cualquier persona interesada en la investigación médica.

En la actualidad, la información médica y científica se encuentra dispersa en múltiples plataformas y bases de datos especializadas, muchas de las cuales requieren suscripciones de pago o conocimientos avanzados en la materia para acceder a resultados relevantes. Esto supone una barrera para médicos, estudiantes e investigadores que necesitan acceder a estudios relevantes sin perder tiempo navegando por documentos extensos y múltiples páginas web. Por ello, es importante desarrollar una solución a esta problemática, ofreciendo un sistema automatizado que recopile, organice y ponga a su disposición la información de manera estructurada y resumida.

El núcleo del proyecto radica en un motor de búsqueda avanzado basado en procesamiento de lenguaje natural (NLP), que permitirá a los usuarios realizar consultas en términos coloquiales y recibir resultados relevantes sin necesidad de utilizar terminología específica o conocimientos sobre consultas avanzadas. Para ello, se emplearán técnicas de NLP y pgvector en PostgreSQL para optimizar la búsqueda. Además, se utilizará el modelo de Transformers BART (en Python) en la fase de inferencia, permitiendo generar resúmenes automáticos de los estudios recopilados y proporcionando una visión rápida y clara de su contenido sin necesidad de leer el documento completo.

La arquitectura del sistema estará basada en microservicios en la nube, cada uno con una función específica. Un primer módulo se encargará del web scraping, recopilando información de diversas fuentes públicas de estudios médicos. Posteriormente, un motor de inferencia procesará los textos y generará resúmenes automáticos, los cuales serán almacenados en una base de datos optimizada en PostgreSQL con pgvector, facilitando búsquedas rápidas y eficientes. Otro microservicio se encargará del procesamiento de consultas en lenguaje natural, permitiendo que los usuarios obtengan resultados precisos sin necesidad de utilizar términos técnicos. El último microservicio será la interfaz web, desarrollada en React, donde primará la accesibilidad, ya que cualquier persona deberá poder realizar búsquedas de manera sencilla y recibir un listado de estudios médicos con su respectivo resumen y enlace a la fuente original.

El proyecto está dirigido principalmente a médicos e investigadores que requieren acceso fácil y rápido a estudios médicos relevantes para su trabajo, instituciones académicas interesadas en contar con una herramienta centralizada para la consulta de estudios científicos, estudiantes de medicina en busca de información, y cualquier persona interesada en el ámbito de la investigación médica.

Cabe destacar, que Clinic Cloud no solo busca facilitar el acceso a información médica pública, sino democratizar el conocimiento al eliminar barreras, tanto económicas como técnicas, en el acceso a información médica.

La modularidad y escalabilidad del sistema en la nube permitirán que pueda mantenerse y evolucionar a largo plazo, comenzando con una implementación en contenedores Docker y, en caso de necesidad, migrando hacia una infraestructura orquestada con Kubernetes para una mayor capacidad de procesamiento y disponibilidad.

Este proyecto representa un avance significativo en la manera de acceder y organizar la información médica pública, proporcionando una herramienta eficaz y gratuita que pone la tecnología al servicio de todos.

Índice de contenido

Índice de figuras

Necesidades del sector productivo y de la organización de la empresa

Clinic Cloud es un proyecto de iniciativa individual desarrollado bajo un modelo de software libre, lo que significa que no está respaldado por una empresa o entidad comercial, sino que nace como un esfuerzo personal para cubrir una necesidad localizada en el campo de la información médica. Este camino supondrá un gran reto para hacerse hueco en el mercado, pues este desarrollo competirá directamente en un sector altamente especializado y en constante evolución. Una evolución impulsada por la digitalización y el crecimiento del mercado de la tecnología aplicada a la salud.

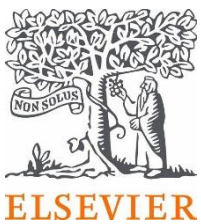
Sin embargo, gran parte de los recursos en el sector se encuentran restringidos tras modelos de suscripción costosos o barreras de acceso institucionales lo que provoca una falta de equidad en términos de salud a nivel global y pone en riesgo los derechos humanos de millones de personas (The Lancet, 2021).

Análisis del Sector y Empresas Relacionadas

El sector de la tecnología aplicada a la salud ha experimentado un crecimiento significativo en los últimos años, impulsado por la necesidad de digitalización en la investigación médica y la gestión de información sanitaria. Se estima que el mercado global alcanzó un valor de 222.680 millones de dólares en 2022 y se proyecta que registre una tasa de crecimiento anual del 27,2% en el período, hasta 2032 (EmergenResearch, 2023). Esto es un claro indicador de la expansión del sector, en el cual, este proyecto buscará su oportunidad.

En este contexto, el proyecto se situaría entre dos tipos de proveedores:

- Empresas de software académico y bases de datos científicas:
    - ***Elsevier y Springer***: Son editoriales académicas que ofrecen acceso a una gran cantidad de revistas y libros científicos. Sus plataformas, como ScienceDirect (Elsevier) y SpringerLink, proporcionan herramientas de búsqueda avanzadas, aunque requieren suscripciones de pago, lo que limita su disponibilidad, dejando fuera a millones de usuarios. Como es el caso de las universidades españolas o el CSIC que han visto restringido su acceso en el último año (elDiario, 2025).
    - ***PubMed***: Mantenida por la Biblioteca Nacional de Medicina de EE.UU., es una base de datos gratuita que incluye referencias y resúmenes de artículos de investigación biomédica. Aunque es accesible sin costo, su interfaz puede resultar compleja para usuarios sin formación digital, la curva de aprendizaje para un uso rápido y efectivo es muy pronunciada (PubMed Wiki, 2024).



3 - Logo Editorial Elsevier

2 - Logo PubMed

1 - Logo Editorial Springer

- Aplicaciones de inteligencia artificial en la búsqueda de documentos científicos:
    - ***Semantic Scholar**:* Desarrollada por el Allen Institute for Artificial Intelligence, es una plataforma utiliza técnicas modernas de procesamiento de lenguaje natural para mejorar la búsqueda y comprensión de artículos académicos. Ofrece resúmenes generados automáticamente en una sola frase y destaca los artículos más influyentes en un campo determinado (Semantic Scholar Wiki, 2023).
    - ***SciSpace***: Desarrollada por Typeset, SciSpace es una plataforma que emplea inteligencia artificial generativa para facilitar la lectura y comprensión de documentos científicos. Permite a los usuarios realizar preguntas sobre el contenido de un artículo y obtener respuestas contextuales, además de generar resúmenes automáticos y explicar términos complejos (SciSpace Wiki, 2025).

SEMANTIC SCHOLAR

5 – Logo Semantic Scholar

SCISPACE

4 – Logo SciSpace

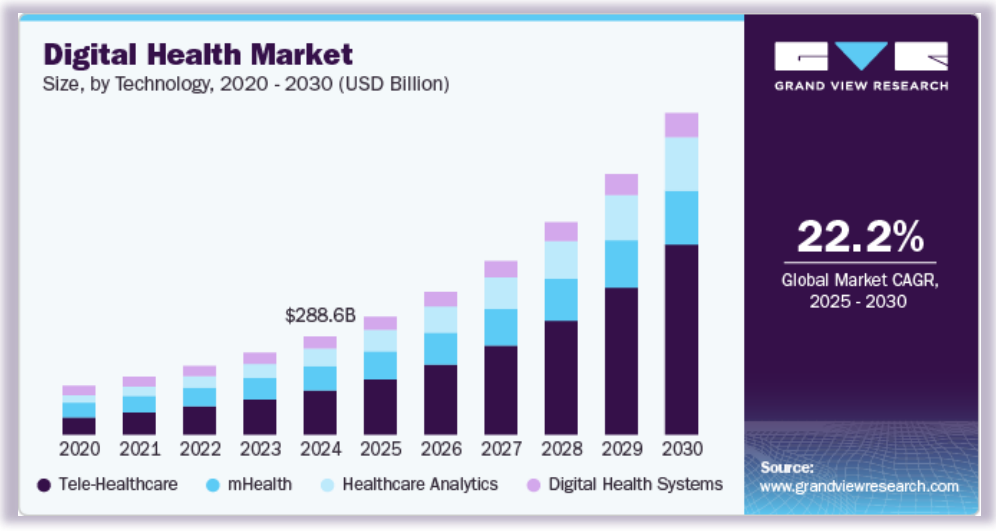
Aunque estas últimas no estén especializadas en el campo de la medicina es una aproximación, a una mayor escala, de una de las claves del proyecto.

En este contexto, una aplicación que combine la accesibilidad de PubMed con las capacidades avanzadas de búsqueda y análisis de la inteligencia artificial aplicadas por SciSpace o Semantic Scholar sería una solución integral para todos aquellos que necesitan acceder a información médica de manera eficiente y comprensible.

Clinic Cloud busca llenar ese vacío al ofrecer una alternativa, aunque a menor escala, de acceso libre y gratuito con una experiencia de usuario simplificada.

Oportunidades de Negocio y Justificación del Proyecto

La oportunidad de negocio de este proyecto no se basa en una valoración objetiva o monetaria, sino que lo hace en su valor como recurso accesible para la comunidad médica y cualquiera que lo necesite. Además, la creciente digitalización de la información médica, el auge de la inteligencia artificial, así como el crecimiento de la digitalización del sector médico, forman una oportunidad única para este desarrollo.



6 - Gráfico del Tamaño del Sector de la Medicina Digital

Entre los factores que justifican su viabilidad cabe destacar:

La democratización del acceso a información médica: Muchas investigaciones y estudios médicos son de difícil acceso por la enorme dispersión en la red. Contar con una aplicación libre que los centralice y resuma supondría un gran beneficio social, ya que aportaría gran accesibilidad al conocimiento.

El uso de inteligencia artificial para optimizar las búsquedas: La implementación de un motor de búsqueda basado en procesamiento de lenguaje natural ayudaría en gran medida a los usuarios a encontrar estudios relevantes sin necesidad de conocimientos técnicos en la formulación de consultas a bases de datos.

El modelo de código abierto y la colaboración comunitaria: Al ser un proyecto open source, puede beneficiarse del apoyo de la comunidad, permitiendo mejoras continuas sin depender de una única persona, entidad o empresa.

Dado que el proyecto no tiene una estructura comercial, su éxito dependerá de su difusión y adopción en la comunidad médica y académica. Considerando el crecimiento del mercado y el aumento del uso de nuevas tecnologías en el ámbito de la salud, su viabilidad resulta prometedora.

Consideraciones Legales, Fiscales y Financieras

Clinic Cloud no es un proyecto comercial, ni tampoco hay una empresa detrás del proyecto, las obligaciones fiscales y laborales son mínimas, por no decir nulas, en su desarrollo. Sin embargo, es importante considerar ciertos aspectos legales y financieros en caso de que el proyecto crezca y requiera financiamiento externo u otras formas de organización.

- **Cumplimiento de regulaciones**:

Aunque en la base de datos no almacenará información personal ni datos clínicos privados, sino únicamente estudios médicos de acceso público, es esencial garantizar que las técnicas de web scraping utilizadas respeten los términos de uso de las fuentes de datos y no infrinjan regulaciones como el Reglamento General de Protección de Datos (RGPD) en Europa o la Ley de Portabilidad y Responsabilidad de Seguros de Salud (HIPAA) en Estados Unidos (iapp, 2024).

El RGPD establece que la recopilación y procesamiento de datos personales deben basarse en una justificación legal válida. Aunque la información esté disponible públicamente en Internet, no implica necesariamente que pueda ser utilizada libremente sin restricciones. Por ejemplo, la Agencia Española de Protección de Datos ha señalado que, aunque la información publicada en una página web esté disponible para su consulta por parte de cualquier usuario, los datos personales contenidos en estos no constituyen datos de libre uso (metricson, 2024).

Por lo tanto, es importante que revisar y cumplir con los términos de servicio de cada fuente de datos y asegurarse de que las prácticas de web scraping se ajusten a las regulaciones aplicables para evitar consecuencias legales.

- **Opciones de financiamiento:**

Existen programas de subvenciones y financiamiento para proyectos de software libre y de impacto social. Por ejemplo, la Fundación Shuttleworth financia ideas innovadoras que logren conectar la tecnología, el conocimiento y el aprendizaje, que aporten en la construcción de un mundo mejor. Ofrece becas anuales que cubren salarios, viajes y gastos operativos, permitiendo a los beneficiarios desarrollar sus ideas durante un año (gestionandote.org, 2021).

En este caso, podría ser interesante Goteo, una plataforma de crowdfunding que se centra en proyectos que generan un bien común, así como software de código abierto y/o contenido de libre acceso. Esta plataforma permite aportaciones monetarias y colaboraciones de voluntarios en los proyectos, está gestionada por la Fundación Goteo, una organización sin ánimo de lucro con sede en Barcelona (Goteo Wiki, 2024).

7 - Logo Fundación Goteo

8 - Logo Fundación Shuttleworth

- **Licencia del software:**

Para garantizar que Clinic Cloud se mantenga como un proyecto de código abierto adoptará la licencia GNU General Public License (GPL), la misma en la que se basa Linux. Esta licencia asegura que el software y todas sus modificaciones o derivados permanezcan libres y accesibles para la comunidad, evitando que terceros puedan apropiarse del código para usos privados (Cristian Palau, 2024).

Al elegir la licencia GPL, quedarán establecidas las siguientes condiciones:

1. **Libertad de uso y distribución**: Cualquier persona o institución puede utilizar y distribuir el software sin restricciones, siempre que respete los términos de la licencia.
2. **Acceso al código fuente**: Garantiza que el código estará siempre disponible para su estudio, modificación y mejora por parte de la comunidad.
3. **Software libre**: Cualquier software derivado de Clinic Cloud deberá conservar la misma licencia GPL, impidiendo que versiones modificadas sean privatizadas o distribuidas bajo términos más restrictivos.

Esta elección está basada en el compromiso del proyecto con la accesibilidad y la colaboración, asegurando que la aplicación y sus futuras versiones beneficien a toda la comunidad en lugar de quedar restringidas a intereses comerciales.

9 - Logo de General Public License

- **Sostenibilidad del proyecto:**

Dado que la sostenibilidad del proyecto también dependerá de su escalabilidad y facilidad de implementación, es clave considerar desde el inicio una arquitectura flexible y eficiente que permita su crecimiento progresivo.

Si bien el proyecto comienza como una iniciativa personal, su modularidad y arquitectura basada en microservicios permitirán que pueda escalar con el tiempo. En una fase inicial, será desplegado utilizando contenedores Docker para simplificar su implementación y mantenimiento y minimizar los costes. Si la demanda crece y se requiere mayor capacidad de procesamiento, se podrá migrar a una infraestructura orquestada con Kubernetes, que con una mayor inversión y optimizando la gestión de recursos, permitirá un flujo de usuarios mucho mayor y su despliegue en entornos más robustos.

En cuanto a la necesidad de montar una empresa u organización, si el proyecto logra atraer la atención de la comunidad médica y tecnológica, se podrá considerar la formación de un equipo colaborativo que contribuya con mejoras y mantenimiento del software. Esto podría incluir desarrolladores interesados en el proyecto, así como profesionales de la salud que validen la calidad de los resultados del motor de búsqueda.

10 - Representación Ficticia de Microservicios en la Nube

Planificación del proyecto

La planificación del proyecto parte con la vista puesta en conseguir un producto software libre que aporte su granito de arena en la democratización del acceso a la información médica, aprovechando las tecnologías emergentes como inteligencia artificial o la arquitectura modular que facilitan los microservicios en la nube.

Actualmente, la dispersión de estudios científicos y bases de datos especializadas supone una barrera de acceso para muchos usuarios, ya sea por la falta de formación técnica en la búsqueda de información o por los modelos de pago que restringen su disponibilidad. Como se mencionó en el apartado anterior, el proyecto busca ofrecer una alternativa accesible y gratuita que centralice la información pública disponible, facilitando su consulta mediante herramientas de búsqueda avanzadas basadas en procesamiento de lenguaje natural y la generación automática de resúmenes.

Los principales objetivos del proyecto son:

- ✓ Desarrollar una plataforma que permita la búsqueda eficiente de estudios médicos en fuentes de acceso público.
- ✓ Implementar un sistema de inferencia basado en modelos de procesamiento de lenguaje natural para mejorar la búsqueda y recuperación de información.
- ✓ Desarrollar una funcionalidad de generación automática de resúmenes que permita a los usuarios obtener una visión rápida y comprensible de los estudios científicos.
- ✓ Garantizar una arquitectura escalable y modular basada en microservicios para optimizar el rendimiento y la expansión futura del sistema.
- ✓ Mantener el proyecto bajo una licencia de código abierto (GPL) para asegurar que continúe siendo gratuito y evolucione con la contribución de la comunidad.

Esto es una pequeña planificación de los objetivos, los cuales se definirán de forma más detallada en la fase de diseño del sistema.

Identificación de Fuentes de Datos

Para garantizar la fiabilidad y utilidad del motor de búsqueda, es fundamental seleccionar fuentes de datos que proporcionen información médica rigurosa, accesible y de alta calidad. En este sentido, existen diversas bases de datos y repositorios científicos que cumplen con estos criterios y que permiten la integración mediante API o métodos de acceso estructurados.

2 - Logo PubMed

PubMed: Gestionada por la Biblioteca Nacional de Medicina de EE.UU., es una de las bases de datos más completas en biomedicina. Contiene millones de citas y resúmenes de artículos científicos provenientes de revistas revisadas por pares. Además, proporciona una API oficial que facilita la extracción de datos y su integración en aplicaciones externas.

11 - Logo MedlinePlus

MedlinePlus: También gestionada por la Biblioteca Nacional de Medicina de EE.UU., ofrece información médica en un lenguaje accesible tanto para profesionales de la salud como para el público general. También cuenta con una API oficial para el acceso al contenido.

12 - Logo Epistemonikos

Epistemonikos: Una base de datos colaborativa con una gran cantidad de contenido sobre medicina y diagnóstico. Esta plataforma también cuenta con una API oficial, lo que facilita el acceso al contenido.

13 - *Logo OMS*

Biblioteca de la Organización Mundial de la Salud: Contiene un extenso repositorio de documentos, informes y datos de relevancia global en salud.

14 - Logo Google Académico

Google: También existe interés en integrar datos provenientes de Google Académico. Sin embargo, esta alternativa es la más complicada técnicamente. Además, las políticas de uso pueden limitar el scraping de datos.

Esta selección de fuentes está basada en una búsqueda de información médica de calidad, que sea accesible y que permita el uso por parte de terceros. A su vez, esta selección no es invariable, durante el desarrollo pueden incorporarse nuevas fuentes o alguna de las nombradas puede ser descartada por dificultades técnicas o del tipo que sean, pues siempre se tendrán en cuenta las implicaciones legales y técnicas que surjan con el uso de cada fuente.

Selección de Tecnologías y Herramientas

El proyecto se desarrollará principalmente en Python, ya que tiene un gran número de bibliotecas y frameworks adecuados para las distintas partes del proyecto. Desde tecnologías avanzadas de procesamiento de lenguaje natural, hasta frameworks eficientes para web scraping o desarrollo de API. En principio, se ha planeado usar las siguientes herramientas software:


16 - Logo Python

Python: Será el lenguaje de programación principal debido a su ecosistema amplio y bien soportado, que incluye una gran variedad de bibliotecas especializadas accesibles a través de pip, lo que permite integrar múltiples frameworks para tareas como el procesamiento de lenguaje natural, web scraping o la implementación de API.


17 - Logo FastAPI

FastAPI: Framework de Python para el desarrollo del backend, optimizado para crear API REST con soporte asíncrono.


15 - Logo PostgreSQL

PostgreSQL: Sistema de gestión de bases de datos del proyecto. Se complementará con la extensión de código abierto pgvector para realizar la vectorización de la base de datos.

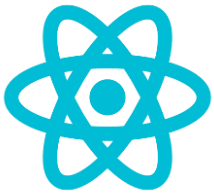

18 - Logo Hugging Face

Hugging Face Transformers: Biblioteca de Python permite la implementación de modelos de NLP avanzados. Se utilizará específicamente BART para la generación de resúmenes.

Scrapy: Un framework de Python para el web scraping, es decir, para extraer información estructurada de sitios web de forma sencilla.

19 - Logo Scrapy

React: Framework de JavaScript para el desarrollo del frontend, no será necesario un gran conocimiento del mismo ya que la interfaz está planeada para ser muy sencilla.



21 - Logo React

Docker: Plataforma en la que se crearán los contenedores donde se alojarán los microservicios de forma independiente, facilitando su gestión y despliegue

20 - Logo Docker

Kubernetes: Opción para la orquestación de contenedores en caso de que el proyecto escale y requiera una infraestructura más robusta.

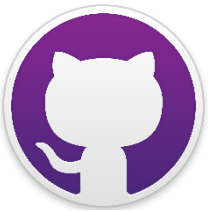
22 - Logo Kubernetes

Visual Studio Code: Entorno de desarrollo principal del proyecto. Desde su terminal se podrán lanzar los servicios, y gracias a su ecosistema de extensiones, ofrece soporte para todas las tecnologías y herramientas necesarias.

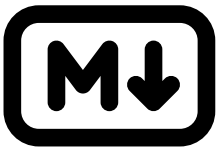
23 - Logo Visual Studio Code

GitHub: Será la herramienta principal para la gestión del desarrollo. Se utilizarán las siguientes funcionalidades:

- *GitHub Projects*: Para el seguimiento de tareas y organización del trabajo mediante tableros Kanban.
- *Issues y Milestones*: Para registrar problemas, mejoras y definir hitos en cada fase del desarrollo.
- *Markdown en README*: Para la documentación del proyecto directamente en el repositorio, facilitando el acceso y comprensión del código por parte de la comunidad.
- *GitHub Desktop*: Herramienta para la gestión del control de versiones con Git, facilitando la portabilidad del proyecto y el seguimiento de cambios.



24- Logo GitHub Desktop



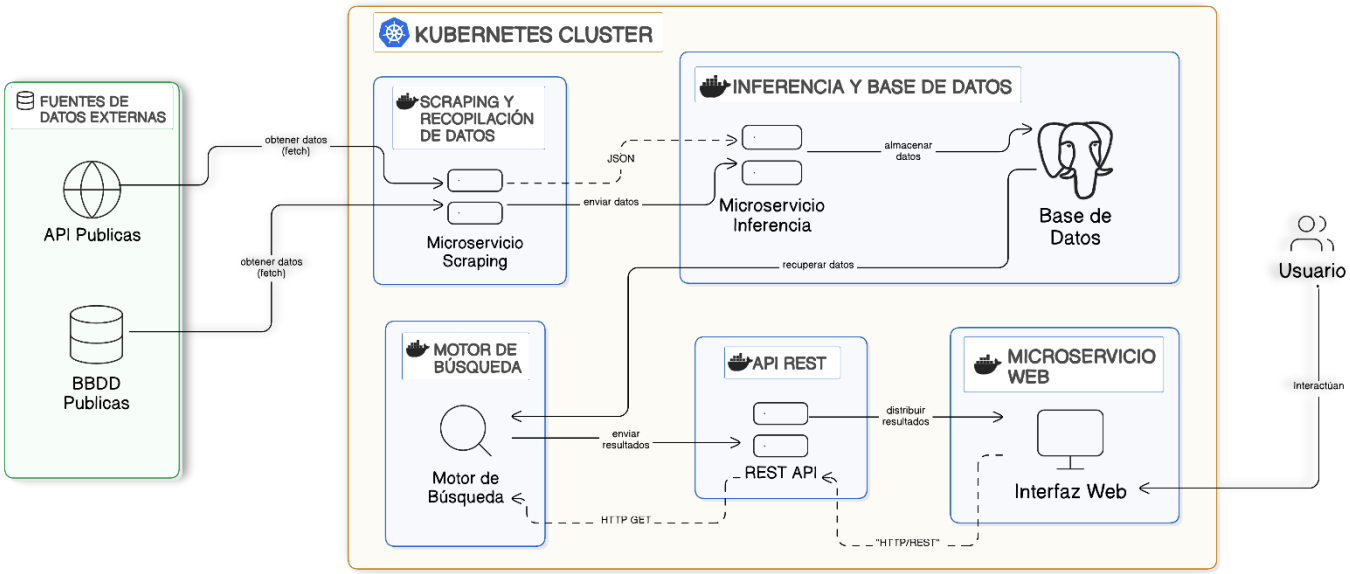
25 - Logo Markdown

Planificación de la Arquitectura

El plan de la arquitectura del sistema está basado en microservicios, de manera que cada componente funcione de manera independiente, favoreciendo la escalabilidad y el mantenimiento del proyecto. La estructura en contenedores permite que cada parte/servicio pueda ser desarrollado, desplegado y modificado de manera autónoma.

El sistema se compondrá de cinco microservicios. Aunque todos los microservicios funcionan como servidores dentro de la infraestructura, su relación sigue un modelo cliente-servidor dependiendo de la dirección de las solicitudes y las dependencias entre ellos.

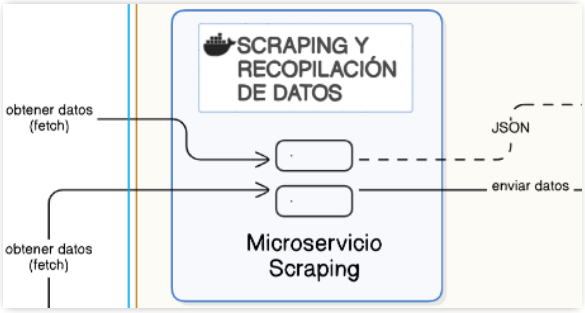
De ser necesario, se migrará a un clúster de Kubernetes, que actuará como la infraestructura de todo el sistema gestionando la ejecución y escalabilidad de los cinco microservicios. Cada microservicio se ejecutará en un pod dentro del clúster, lo que permitirá su despliegue independiente y la asignación dinámica de recursos según la carga de trabajo. También asegurará la información de la base de datos, ya que estará respaldada por volúmenes persistentes de Kubernetes.



26 - Diagrama de la arquitectura del sistema

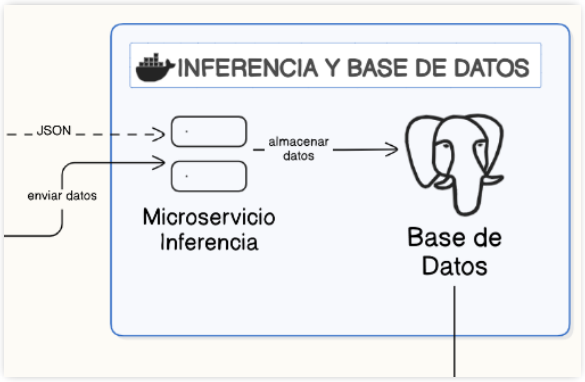
1. Microservicio de Scraping y Recopilación de Datos

Este microservicio se encargará de obtener la información de las fuentes previamente identificadas. Se encargará tanto de la integración con API públicas como de la extracción de datos mediante web scraping. Una vez recopilada la información, esta se enviará al Microservicio de Inferencia y Base de Datos para su procesamiento y almacenamiento.



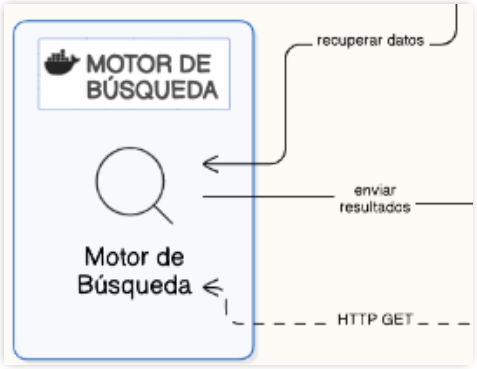
2. Microservicio de Inferencia y Base de Datos

Una vez recibidos los estudios médicos desde el servicio de Scraping, este microservicio se encarga de su procesamiento. Se encargará de generar resúmenes automáticos condensando la información clave de cada documento. Además de realizar la inferencia sobre los datos, este servicio administrará la base de datos del sistema. Actuará como servidor respondiendo a solicitudes del motor de búsqueda.



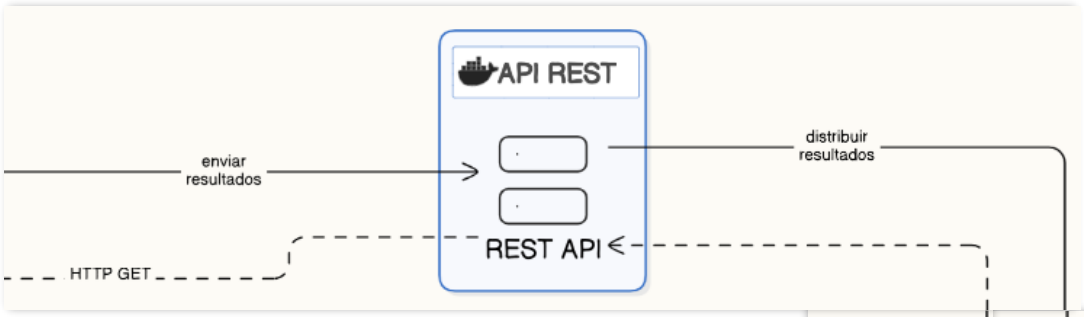
3. Microservicio del Motor de Búsqueda

Será el encargado de procesar las búsquedas realizadas por los usuarios. Cuando reciba una consulta de la API, el motor la procesará, recuperando los documentos más adecuados de la base de datos, enviando los resultados a la API REST.



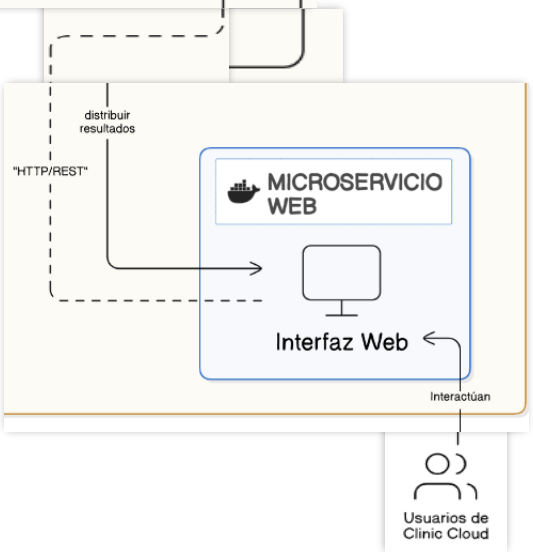
4. Microservicio API REST

Su principal función será exponer los endpoints a través de los cuales los usuarios y otros servicios podrán enviar solicitudes y obtener respuestas de manera estructurada. Es la puerta de comunicación que funciona como intermediario entre los distintos servicios y la interfaz web, es decir, gestionará las solicitudes que llegan desde la web y las distribuirá al motor de búsqueda.



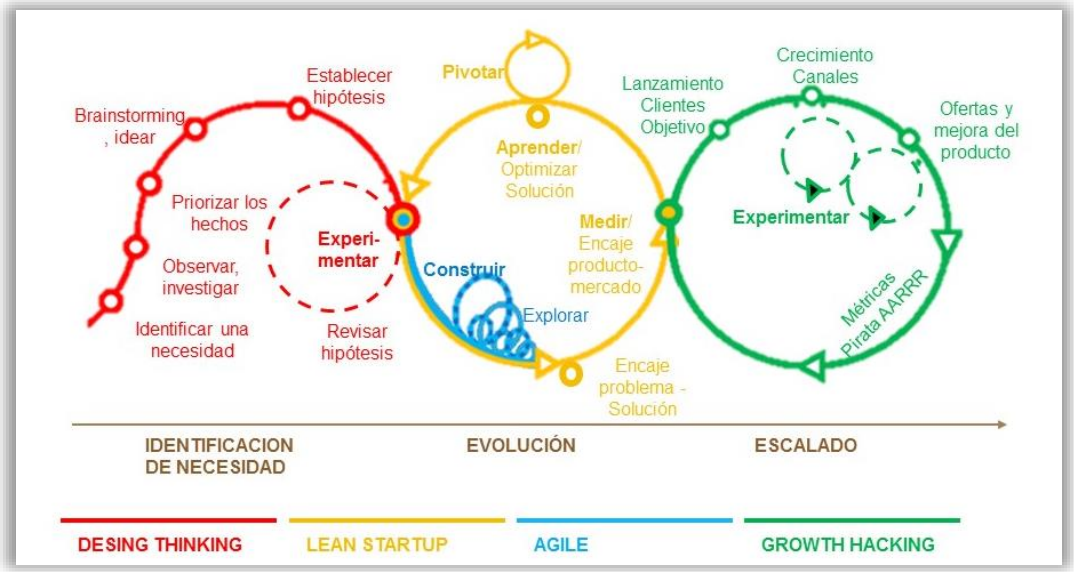
5. Microservicio Web

El usuario final podrá interactuar con el sistema a través de una interfaz web sencilla e intuitiva. Este microservicio se encargará de gestionar las peticiones de los usuarios y comunicarlas a la API REST, además de mostrar los resultados obtenidos de forma estructurada y comprensible.



Cronograma y Metodología de Trabajo

El proyecto se desarrollará siguiendo una metodología ágil, parecida a Lean Startup, con fases concretas que permitan validar cada etapa del proyecto. Esta elección se debe a que el desarrollo será llevado a cabo por un solo desarrollador, por lo que no tiene sentido hacer uso de metodologías diseñadas para equipos. Se ha definido un cronograma inicial en cinco fases:



27 - Representación de la Metodología de Trabajo del Proyecto

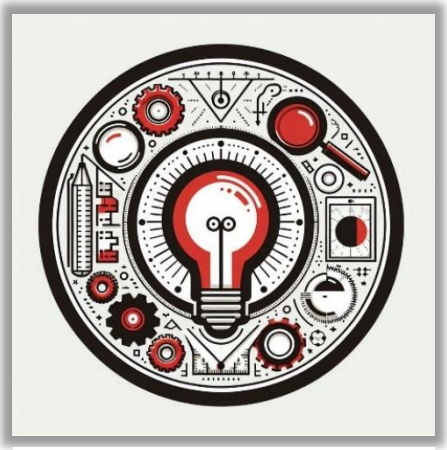
1. Idea y Planificación del Proyecto (10 días - finalizado)

En esta fase inicial se han definido los objetivos generales, el alcance y las tecnologías a utilizar. También se han seleccionado las fuentes de datos que se usarán, así como tecnologías y herramientas que serán necesarias en la fase de desarrollo. Por último, se ha definido como será la arquitectura del sistema, la estructura de microservicios y las conexiones entre ellos.

2. Diseño del Sistema (10 días)

En esta fase se detallará el diseño del sistema y se definirán los requisitos tanto funcionales como de datos, asegurando una base sólida antes de la implementación:

- Definición de objetivos específicos: Identificación de las funcionalidades clave y los resultados esperados del sistema.
- Requisitos no funcionales: Identificación de las características de calidad del sistema y restricciones sobre cómo debe funcionar.
- Requisitos funcionales: Definición de las principales funcionalidades del sistema y redacción de los casos de uso.



28 - Figura Fase 1

- Diseño de datos: Modelado de la base de datos, estableciendo las tablas, índices y relaciones necesarias.
- Diseño de la interfaz de usuario: Creación de bocetos o mockups de la interfaz web.

29 - Figura Fase 2

3. Desarrollo del Producto Mínimo Viable (MVP) (1-2 Meses)

En esta etapa se construirán los componentes esenciales del sistema, permitiendo una versión inicial funcional:

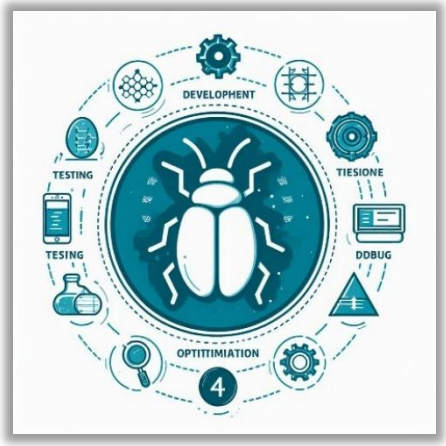
- Implementación del mecanismo de scraping básico.
- Creación de la base de datos en PostgreSQL.
- Desarrollo del motor de búsqueda.
- Desarrollo del motor de inferencia.
- Desarrollo de la API REST.
- Creación de una interfaz de usuario básica, permitiendo la interacción inicial con el sistema.
- Implementación de contenedores Docker.

30 - Figura Fase 3

4. Pruebas y Optimización (1-2 Meses)

Con el MVP funcional, se procederá a la fase de pruebas y mejoras con el objetivo de optimizar el rendimiento y la usabilidad del sistema:

- Evaluación del rendimiento del motor de búsqueda y optimización de los algoritmos de recuperación de información.
- Mejora de la generación de resúmenes, ajustando modelos y parámetros para aumentar la calidad de la inferencia.
- Implementación de mecanismos de validación de datos para mejorar la fiabilidad del scraping y prevenir errores en el almacenamiento.
- Ajustes en la seguridad del proceso de recolección y almacenamiento de datos, garantizando el cumplimiento de buenas prácticas en el manejo de información médica.



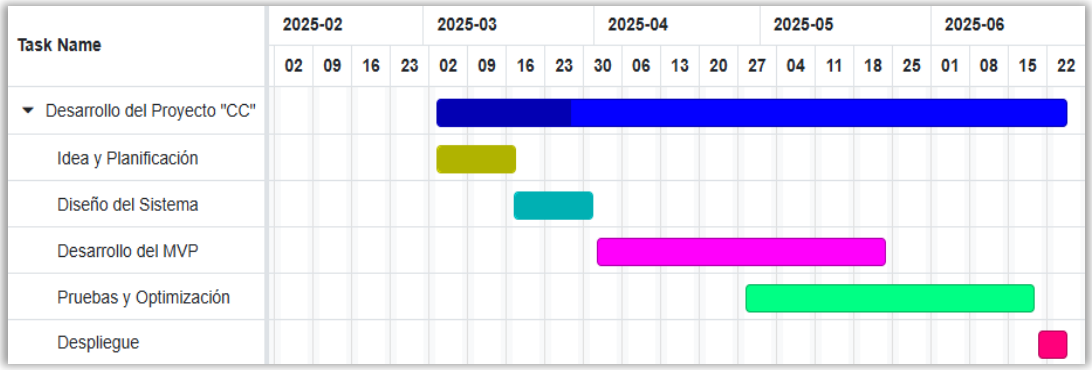
31 - Figura Fase 4

5. Despliegue y Crecimiento del Sistema (Indefinido)

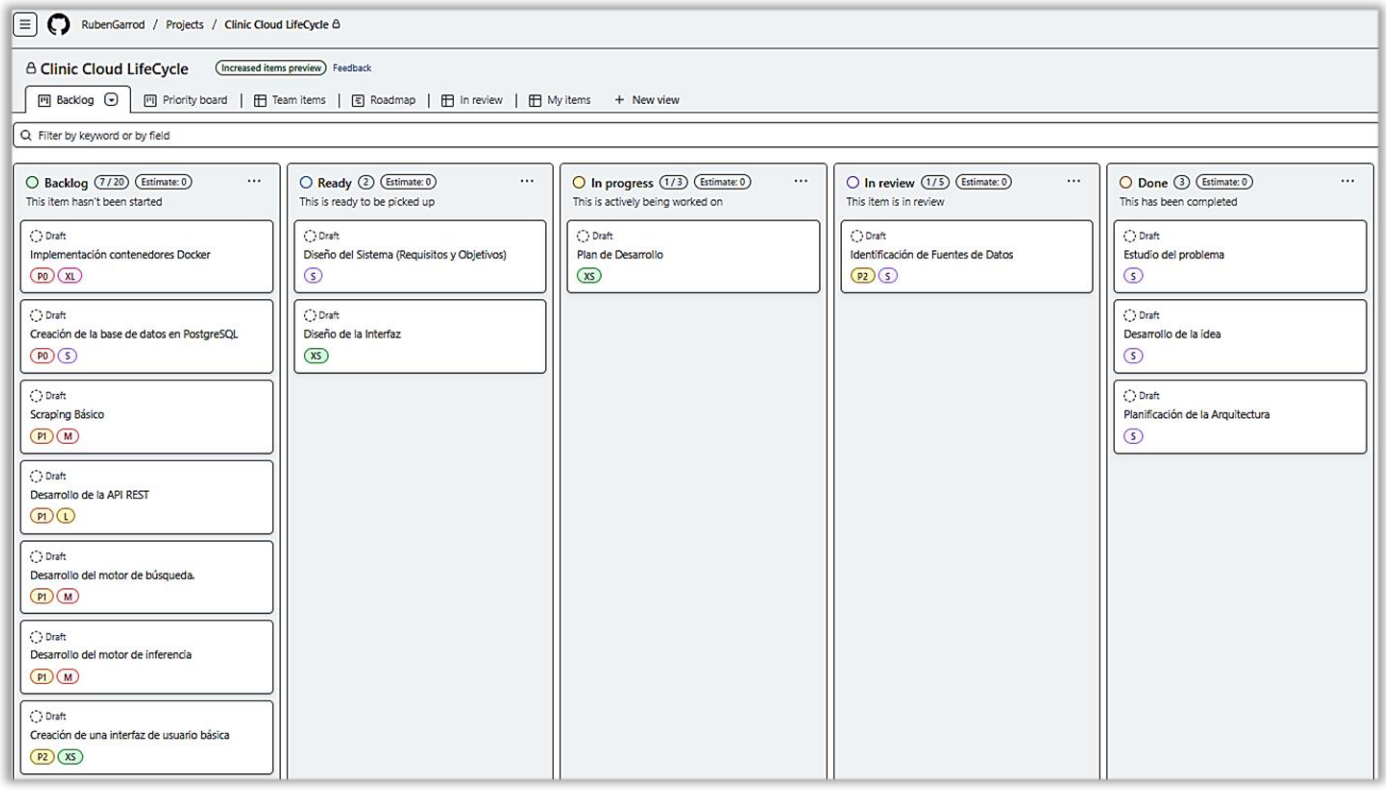
Una vez validado el sistema, lanzamiento y expansión:

- Publicación de la versión inicial del sistema.
- Difusión en comunidades médicas y académicas para incentivar su uso y recibir retroalimentación.
- Despliegue completo en Kubernetes.
- Ciclo de mejora continua, recopilando sugerencias de los usuarios para optimizar el sistema de manera progresiva.

32 - Figura Fase 5



34 - Diagrama de Gantt del Ciclo de Desarrollo del Proyecto

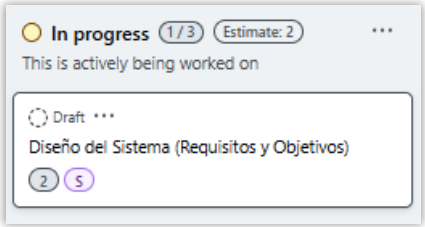


33 - Tablero Kanban del Proyecto en GitHub Projects

Diseño del proyecto

El diseño del sistema es una fase clave en el desarrollo de software, en la cual se define la estructura técnica y funcional del proyecto. En esta etapa se establecerán los requisitos funcionales y no funcionales, los objetivos y se plantearán los elementos clave del sistema.

Además, se realizará el diseño de la base de datos y un mockup de la interfaz de usuario, asegurando que todos los componentes estén alineados en la dirección de cumplir con los objetivos.



35 - Tarea de diseño añadida a "In progress"

Objetivos

Los objetivos representan las metas específicas que se deben alcanzar durante el desarrollo para cumplir con los requerimientos del sistema. Este apartado presenta los objetivos organizados en tablas con información concisa, agrupados en secciones que van desde los fundamentos iniciales hasta los aspectos más refinados del sistema. Este enfoque proporciona una hoja de ruta estructurada para facilitar un desarrollo eficiente y bien organizado.

Fundamentos del sistema:

- ✓ Microservicios básicos y contenerización: Estas son las bases de todo el sistema, asegurando que la recolección y almacenamiento de datos funcione correctamente.

OBJ 01	Creación de la base de datos.
Versión	0.1.0
Descripción	Crear y configurar la base de datos PostgreSQL con las tablas necesarias y sus relaciones.
Subobjetivos	- Vectorización de los datos - Modelado de tablas - Implementar índices - Conexiones con otros microservicios
Importancia	Crítica.
Comentarios	

OBJ 02	Implementación del mecanismo de scraping.
Versión	0.1.1
Descripción	Desarrollo de un sistema para obtener datos de forma automatizada de las fuentes identificadas.
Subobjetivos	- Crear el scraper básico - Conexión con API externas - Validar datos recolectados
Importancia	Crítica.
Comentarios	Atención a las consideraciones legales de cada fuente.

OBJ 03	Motor de inferencia.
Versión	0.1.2
Descripción	Desarrollo el motor para generar los resúmenes a partir de los datos obtenidos y procesados.
Subobjetivos	- Crear el pipeline de inferencia - Validar con casos reales
Importancia	Media.
Comentarios	

OBJ 04	Contenerización.
Versión	0.2.1
Descripción	Crear contenedores Docker para cada componente del sistema.
Subobjetivos	- Configurar los docker-compose.yml - Probar despliegue en ambiente controlado
Importancia	Necesaria.
Comentarios	Testear la compatibilidad en un clúster de prueba en Kubernetes.

Integración y Comunicación

- ✓ Una vez que los fundamentos están listos, es esencial habilitar la comunicación entre los diferentes componentes.

OBJ 05	Integración entre microservicios.
Versión	0.3.1
Descripción	Establecer una comunicación eficiente y segura entre los diferentes microservicios.
Subobjetivos	- Pruebas de interoperabilidad - Configurar protocolos de comunicación
Importancia	Crítica.
Comentarios	

OBJ 06	Desarrollo de la API REST.
Versión	0.3.2
Descripción	Crear una API REST para gestionar la comunicación entre la interfaz web y los servicios del backend.
Subobjetivos	- Diseñar los endpoints - Pruebas funcionales
Importancia	Necesaria.
Comentarios	

Función Principal

- ✓ Con los componentes comunicándose correctamente, se puede desarrollar la funcionalidad principal del sistema, el motor de búsqueda.

OBJ 07	Motor de búsqueda.
Versión	0.4.1
Descripción	Desarrollo y optimización del motor de búsqueda para conseguir una rápida recuperación de los datos.
Subobjetivos	- Implementar algoritmo inicial - Optimizar consultas - Evaluar rendimiento
Importancia	Crítica.
Comentarios	Realizar pruebas con data sets de panda genéricos.

Interfaz y Experiencia del Usuario

- ✓ La interacción del usuario final se prioriza una vez que la funcionalidad base del sistema está establecida.

OBJ 08	Creación de interfaz de usuario.
Versión	1.0.0
Descripción	Desarrollo de la interfaz gráfica con React y despliegue en su contenedor.
Subobjetivos	- Conexión con la API REST - Implementación del diseño básico - Pruebas de Funcionalidad
Importancia	Necesaria.
Comentarios	Revisar accesibilidad una vez sea funcional.

OBJ 09	Experiencia del usuario.
Versión	1.5.0
Descripción	Mejorar la usabilidad y accesibilidad de la interfaz web para usuarios finales.
Subobjetivos	- Implementar diseño responsivo - Realizar pruebas de usabilidad - Optimizar tiempos de carga - Internacionalización (i18n). - Localización (l10n)
Importancia	Alta.
Comentarios	Recoger feedback de la comunidad sobre la usabilidad y la accesibilidad.

Optimización y Escalabilidad

- ✓ Finalmente, se implementan mejoras para el rendimiento, mantenimiento y crecimiento del sistema.

OBJ 10	Seguridad en la API REST.
Versión	1.5.1
Descripción	Garantizar la seguridad de las solicitudes y respuestas.
Subobjetivos	- Utilizar HTTPS - Realizar pruebas de penetración
Importancia	Media.
Comentarios	

OBJ 11	Escalabilidad del sistema
Versión	2.0.0
Descripción	Diseñar una infraestructura que permita escalar los servicios según la demanda de usuarios.
Subobjetivos	- Configurar Kubernetes - Probar escalabilidad horizontal - Implementar balanceadores de carga
Importancia	Baja.
Comentarios	Atención a las consideraciones legales de cada fuente.

OBJ 12	Monitoreo y mantenimiento.
Versión	2.0.0
Descripción	Crear un sistema para monitorear el estado y rendimiento de los microservicios.
Subobjetivos	- Alertas automáticas - Generar informes
Importancia	Baja.
Comentarios	Atención a las consideraciones legales de cada fuente.

Requisitos No Funcionales

Los requisitos no funcionales (RNF) definen las características de calidad que debe cumplir el sistema una vez sea desplegado. Estos requisitos no describen funcionalidades específicas, sino que establecen las condiciones bajo las cuales el sistema debe operar, garantizando que sea confiable, eficiente y adaptable a las necesidades del usuario. A continuación, se detallan los RNF organizados en tablas, proporcionando información clave sobre su propósito, importancia y relación con otros objetivos y requisitos del sistema.

RNF 01	**Rendimiento.**
Versión	1.0.0
Descripción	El sistema, en condiciones normales, debe procesar y devolver los resultados en pocos segundos.
Objetivos relacionados	OBJ-06, OBJ-07, OBJ-08
Importancia	Alta.
Comentarios	

RNF 02	**Escalabilidad.**
Versión	1.0.0
Descripción	El sistema debe manejar un crecimiento gradual de usuarios concurrentes sin afectar el rendimiento.
Objetivos relacionados	OBJ-04, OBJ-05, OBJ-11
Importancia	Alta.
Comentarios	Validar mediante simulaciones de tráfico de usuarios.

RNF 03	**Tolerancia a fallos.**
Versión	1.0.0
Descripción	El sistema debe continuar funcionando en caso de fallos menores en los microservicios.
Objetivos relacionados	OBJ-05, OBJ-06, OBJ-12
Importancia	Media.
Comentarios	

RNF 04	**Compatibilidad multiplataforma.**
Versión	1.0.0
Descripción	El sistema debe ser accesible desde navegadores modernos como Chrome, Firefox, Safari o Edge, desde cualquier pantalla o dispositivo.
Objetivos relacionados	OBJ-08, OBJ-09
Importancia	Alta.
Comentarios	Probar en diferentes dispositivos y resoluciones.

RNF 05	**Accesibilidad.**
Versión	1.0.0
Descripción	El sistema debe ser accesible, asegurando que personas con discapacidad puedan utilizarlo.
Objetivos relacionados	OBJ-08, OBJ-09
Importancia	Alta.
Comentarios	Realizar pruebas con usuarios con diferentes niveles de accesibilidad.

RNF 06	**Reutilización de Componentes.**
Versión	1.0.0
Descripción	El sistema debe tener una arquitectura modular para facilitar su ampliación y reutilización en otros proyectos.
Objetivos relacionados	OBJ-04, OBJ-05, OBJ-12
Importancia	Media.
Comentarios	Garantizar que la modularidad permita integraciones y mejoras por parte de la comunidad open source.

RNF 07	**Documentación Técnica.**
Versión	0.0.1
Descripción	El sistema debe contar con una documentación clara y completa para facilitar la colaboración y ampliación del proyecto.
Objetivos relacionados	OBJ-01, OBJ-05, OBJ-12
Importancia	Alta.
Comentarios	Mantener la documentación actualizada y accesible para cualquier desarrollador interesado en contribuir.

Requisitos Funcionales

Los requisitos funcionales describen las capacidades específicas del sistema desde la perspectiva de los actores involucrados (usuarios finales, componentes automatizados del sistema y administradores). Estos requisitos son clave para garantizar que el sistema cumpla con las funcionalidades esperadas de manera efectiva y se presentan mediante casos de uso documentados.

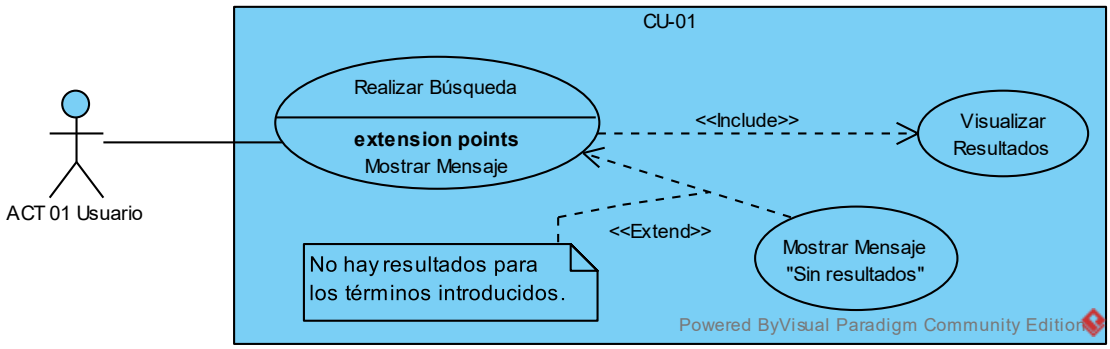
A continuación, se presentan los actores principales del sistema, definidos y organizados para facilitar la referencia en los casos de uso:

Actor		Descripción.
ACT 01	Usuario	Persona que utiliza el sistema para realizar búsquedas, consultar resultados, y acceder a documentos.
ACT 02	Administrador	Responsable del sistema. Gestiona la base de datos, supervisa reportes y mantiene la calidad del sistema.
ACT 03	Sistema	Componente encargado de procesar documentos, generar resúmenes, clasificar temáticas, realizar búsquedas...

Los casos de uso están organizados según las interacciones específicas de los actores con el sistema. Cada caso incluye los pasos necesarios para lograr un objetivo concreto y detalla las condiciones previas y posteriores para garantizar un flujo de operación correcto. A continuación, se presentan los casos de uso:

CU 01	Realizar una búsqueda simple.
Actor Principal	**ACT 01**
Descripción	El usuario introduce términos en la barra de búsqueda para encontrar documentos relevantes.
Precondiciones	- El usuario tiene acceso al sistema (conexión estable a Internet). - Hay documentos almacenados en la base de datos.
Flujo Principal	1. El usuario accede a la página principal. 2. Introduce términos en la barra de búsqueda. 3. El sistema procesa la consulta mediante el motor de búsqueda. 4. Los resultados relevantes se presentan ordenados por relevancia.
Flujos Alternativos	Si no hay resultados, se muestra un mensaje.
Resultado Esperado	El usuario visualiza una lista de resultados relacionados con los términos de búsqueda ingresados.

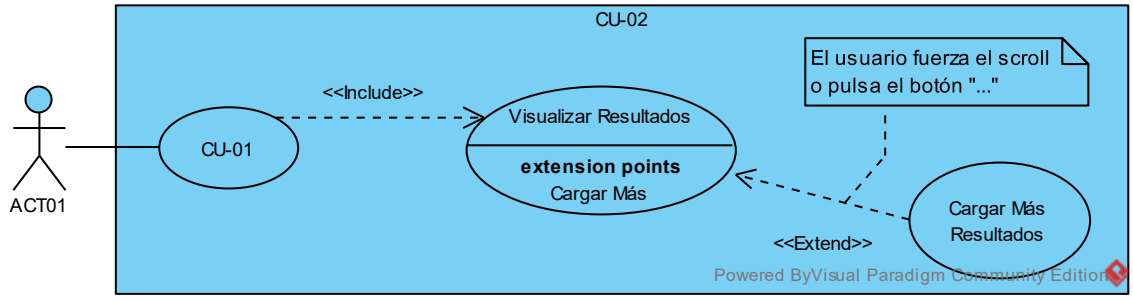
Este caso de uso describe cómo el sistema permite a los usuarios realizar búsquedas para encontrar documentos relacionados con la misma. El proceso se centra en la interacción del usuario con la barra de búsqueda y cómo el motor de búsqueda procesa los términos ingresados, devolviendo resultados ordenados según su relevancia.



36 - Diagrama del Caso de Uso 01

CU 02	Visualizar resultados de búsqueda.
Actor Principal	**ACT 01**
Descripción	El usuario visualiza una lista de resultados relacionados con sus términos de búsqueda.
Precondiciones	- La búsqueda ya se ha realizado. - Existen documentos relevantes en la base de datos.
Flujo Principal	1. El sistema presenta los resultados en forma de lista. 2. Cada entrada muestra el título del documento, un breve resumen y el enlace para acceder al documento completo. 3. El usuario puede hacer scroll y cargar más resultados.
Flujos Alternativos	Si no hay resultados, se muestra un mensaje.
Resultado Esperado	El usuario puede identificar fácilmente los documentos que le interesen en función del título y el resumen.

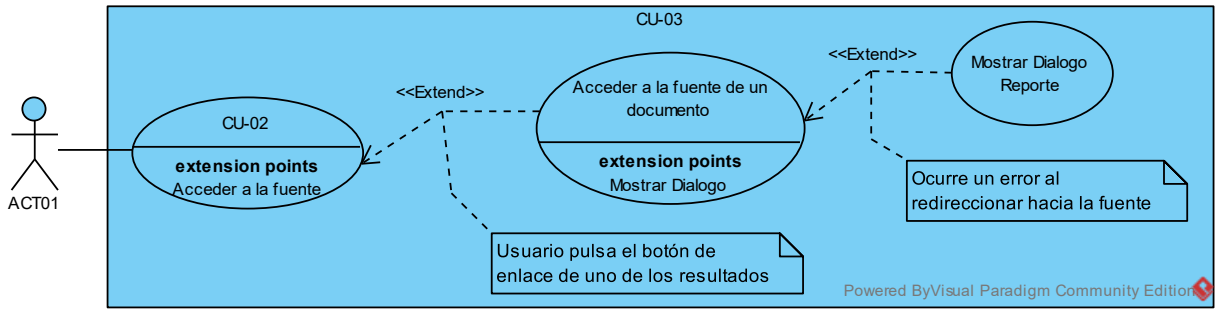
En este caso de uso se detalla cómo los usuarios pueden visualizar los resultados tras realizar una búsqueda. Cada entrada de la lista proporciona información clave como el título, un breve resumen, y un enlace al contenido, permitiendo al usuario identificar rápidamente el documento de interés.



37 - Diagrama del Caso de Uso 02

CU 03	Acceder a un documento desde los resultados.
Actor Principal	**ACT 01**
Descripción	El usuario selecciona un documento de los resultados para visualizarlo desde su fuente original.
Precondiciones	- El usuario ha realizado una búsqueda previa. - Los resultados están cargados en pantalla.
Flujo Principal	1. El usuario hace clic en el botón "Ir a la fuente". 2. El sistema redirige al usuario al enlace original del documento.
Flujos Alternativos	Si el enlace está roto, se muestra un mensaje de error.
Resultado Esperado	El usuario es dirigido correctamente al documento en su fuente original.

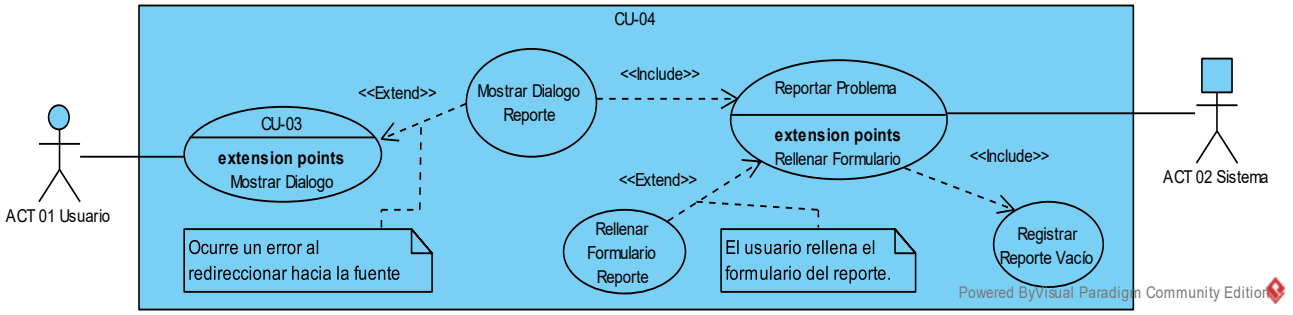
El propósito de este caso de uso es describir el proceso mediante el cual los usuarios pueden acceder a los documentos listados en los resultados. Esto incluye interactuar con enlaces directos para ser redirigidos al documento en su fuente original.



38 - Diagrama del Caso de Uso 03

CU 04	Reportar un problema en un resultado (enlace roto).
Actor Principal	**ACT 01**
Descripción	El usuario informa que un enlace de un documento está roto o no es accesible.
Precondiciones	- El usuario está visualizando la lista de resultados. - Uno de los enlaces no funciona correctamente.
Flujo Principal	1. El sistema genera automáticamente el reporte al detectar el error. 2. El usuario puede (opcionalmente) añadir una descripción o comentario.
Flujos Alternativos	Si no desea describir el problema, simplemente envía el reporte básico.
Resultado Esperado	El sistema registra el problema para que pueda ser revisado posteriormente por el desarrollador.

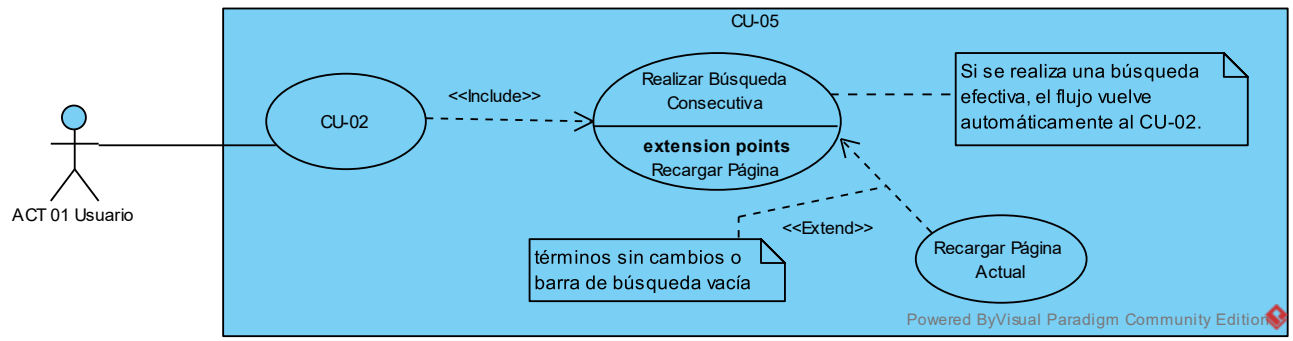
Este caso de uso muestra cómo los usuarios pueden reportar problemas en los resultados de búsqueda (enlaces rotos). Cuando el usuario intente acceder a un enlace y esté caído se le dará la opción de rellenar un formulario de reporte para facilitar la calidad y la eficiencia del servicio. Cuando el usuario no rellene el formulario el reporte se registrará automáticamente, estos reportes tendrán menos prioridad.



39 - Diagrama del Caso de Uso 04

CU 05	Búsquedas consecutivas sin recargar la página.
Actor Principal	**ACT 01**
Descripción	El usuario puede realizar varias búsquedas consecutivas sin necesidad de recargar la página principal.
Precondiciones	- El usuario tiene acceso a la barra de búsqueda en cualquier momento.
Flujo Principal	1. El usuario realiza una búsqueda inicial. 2. El usuario introduce nuevos términos en la barra de búsqueda. 3. El sistema procesa y reemplaza los resultados en la misma página.
Flujos Alternativos	Si el usuario introduce términos vacíos, se refresca la página actual.
Resultado Esperado	El usuario puede realizar varias búsquedas de manera fluida sin interrupciones.

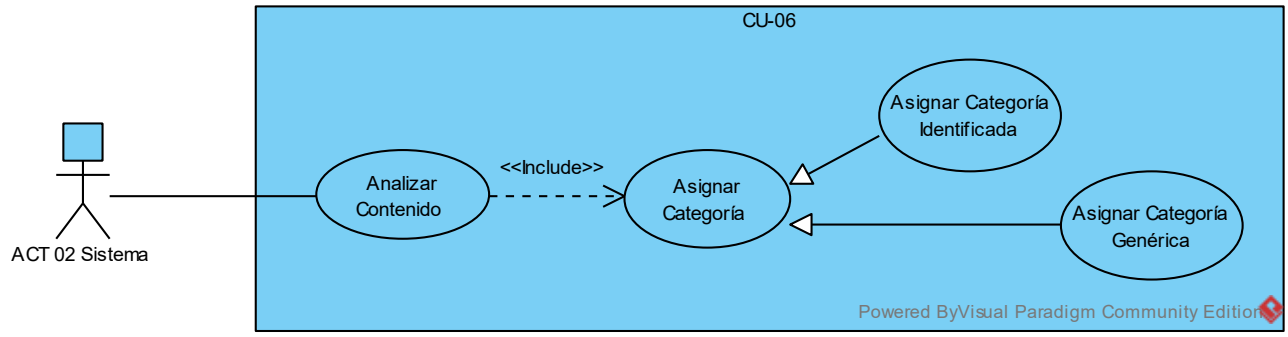
Aquí se describe la posibilidad de realizar múltiples búsquedas de manera fluida, sin necesidad de recargar la página. Este caso de uso resalta la capacidad del sistema para actualizar dinámicamente los resultados en base a nuevas consultas ingresadas por el usuario.



40 - Diagrama del Caso de Uso 05

CU 06	Clasificar automáticamente los documentos.
Actor Principal	**ACT 02.**
Descripción	El sistema analiza el contenido de un documento recién almacenado y asigna una categoría principal según su temática.
Precondiciones	- El documento ha sido registrado en la base de datos. - El modelo de clasificación de categorías está entrenado y funcional.
Flujo Principal	1. El sistema procesa el contenido del documento utilizando el modelo de clasificación. 2. El sistema identifica la categoría más relevante basada en patrones semánticos y lo relaciona.
Flujos Alternativos	Si no se puede determinar la categoría, el sistema asigna una categoría genérica (Ej.” Sin Categoría”) para su revisión posterior.
Resultado Esperado	El documento queda asignado a la categoría más adecuada según su contenido.

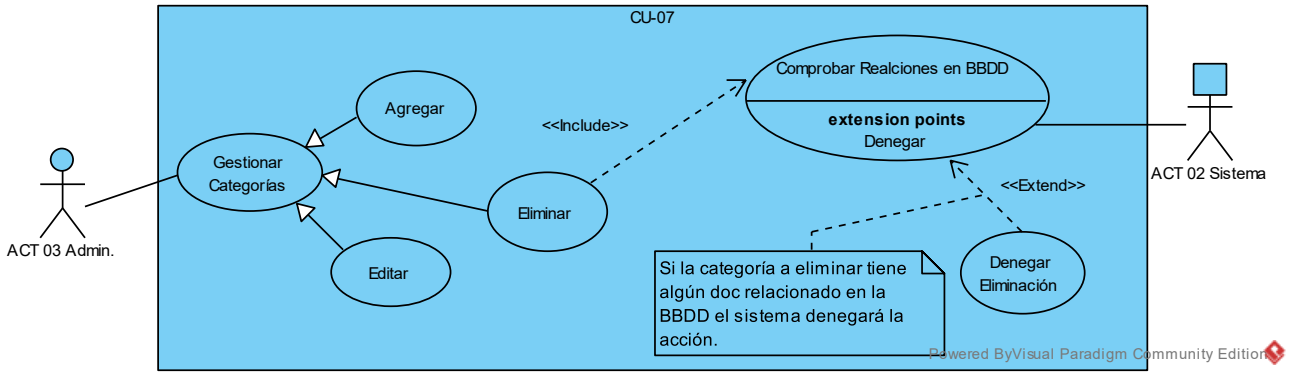
En este caso de uso se detalla cómo el sistema clasifica automáticamente los documentos en categorías específicas basadas en su contenido. Esto facilita búsquedas más precisas y asegura que la base de datos esté bien organizada.



41- Diagrama del Caso de Uso 06

CU 07	Gestionar categorías.
Actor Principal	**ACT 03.**
Descripción	El administrador añade, edita o elimina categorías para mantener actualizado el sistema.
Precondiciones	- El administrador tiene acceso al sistema con permisos suficientes para modificar las categorías.
Flujo Principal	1. El administrador accede a la base de datos. 2. Selecciona la opción para añadir, modificar o eliminar una categoría. 3. Se actualiza la base de datos.
Flujos Alternativos	Si se intenta eliminar una categoría que tiene documentos asociados, el sistema no lo permite.
Resultado Esperado	Las categorías se actualizan correctamente en el sistema.

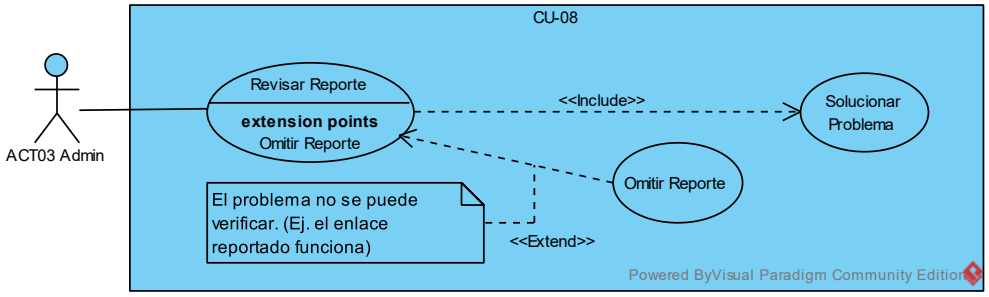
Este caso de uso describe cómo el administrador puede gestionar las categorías del sistema. Esto incluye agregar, modificar o eliminar categorías, permitiendo mantener la estructura temática del sistema actualizada, mejorando la eficiencia y usabilidad. En el caso de intentar eliminar una categoría con relaciones en la base de datos, no se le permitirá llevar a cabo la acción de forma automática.



42 - Diagrama del Caso de Uso 07

CU 08	Revisar reportes.
Actor Principal	**ACT 03**
Descripción	El administrador revisa los reportes enviados por los usuarios sobre problemas en los resultados y aplica las correcciones.
Precondiciones	- Hay reportes registrados en el sistema. - El administrador tiene acceso a los reportes.
Flujo Principal	1. El administrador visualiza la lista de reportes pendientes. 2. Selecciona un reporte y revisa el detalle del problema descrito. 3. Realiza las acciones necesarias.
Flujos Alternativos	Un reporte no puede ser verificado o el problema no existe, el administrador lo omite.
Resultado Esperado	Los problemas reportados son resueltos o gestionados adecuadamente.

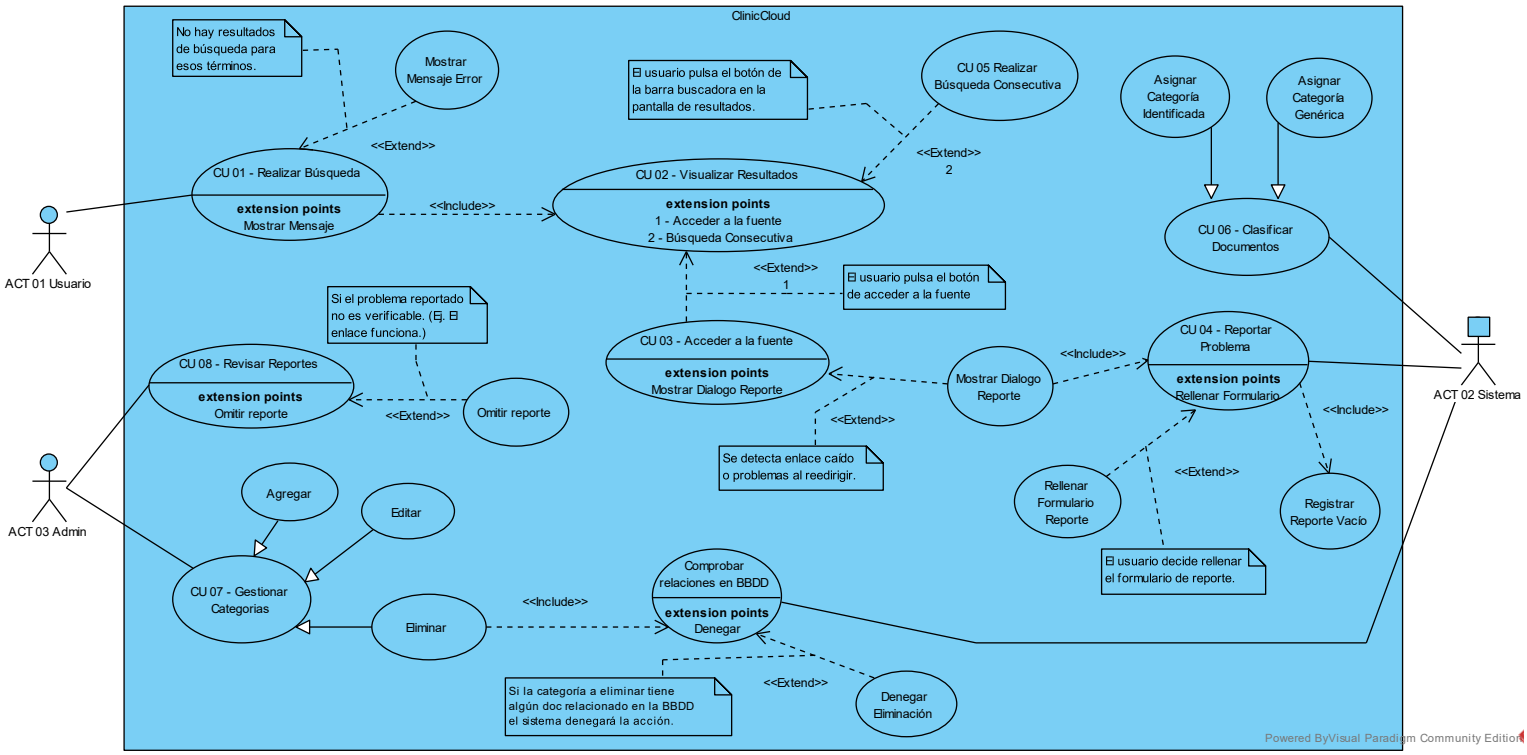
En este caso se aborda cómo los administradores revisan y resuelven los reportes enviados por los usuarios. Esto garantiza la calidad de los resultados y permite tomar medidas correctivas como corregir enlaces rotos. Si al revisar un reporte no puede verificar que existe un problema, se archivará el reporte y no se tomará ninguna acción correctiva.



43 - Diagrama del Caso de Uso 08

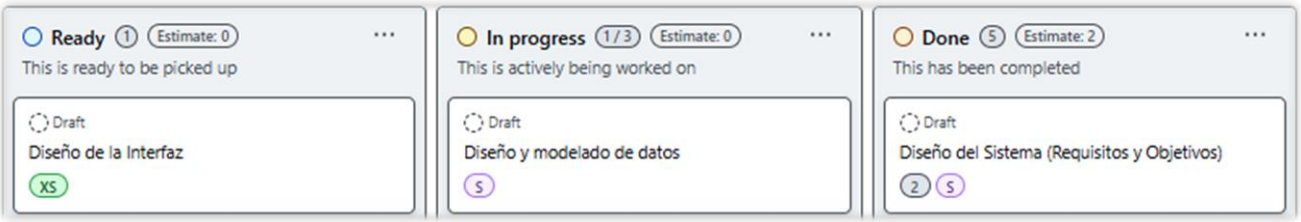
Diagrama General de Casos de Uso

El siguiente diagrama ilustra las interacciones generales entre los actores (Usuario, Administrador y Sistema) y los casos de uso principales del sistema. Este diagrama busca proporcionar una vista global de las funcionalidades clave del sistema, organizadas según las acciones específicas realizadas por los actores. A través de este diagrama, se puede ver cómo los actores interactúan con el sistema para llevar a cabo las tareas de búsqueda, así como el administrador lleva a cabo las de clasificación de documentos y gestión de reportes, mientras que el sistema cuenta con mecanismos que aseguran una operación eficiente y acorde a los requisitos funcionales definidos.



44 - Diagrama de Casos de Uso del Sistema

Diseño y Modelado de la Base de Datos



45 - Actualización del Backlog del Proyecto (1)

En este apartado se definirá el diseño y estructura de la base de datos, que constituye uno de los pilares fundamentales del sistema. Su propósito es almacenar, organizar y gestionar eficientemente la información recopilada, permitiendo integrar el motor de búsqueda basado en procesamiento de lenguaje natural (NLP) y la generación automática de resúmenes.

La base de datos se ha diseñado utilizando PostgreSQL como sistema de gestión, aprovechando la extensión pgvector para la vectorización y búsqueda avanzada de datos. Esta estructura es clave para garantizar el rendimiento y la escalabilidad del sistema, al tiempo que permite clasificar los documentos según su categoría principal. Este diseño optimiza las consultas del usuario, ofreciendo resultados relevantes de forma rápida y eficaz.

Clasificación Temática: Organización a través de Categorías

Para optimizar el rendimiento del sistema y facilitar las búsquedas específicas, los documentos recopilados serán clasificados automáticamente en categorías basadas en su contenido principal. Esta clasificación es esencial para dos aspectos clave del sistema:

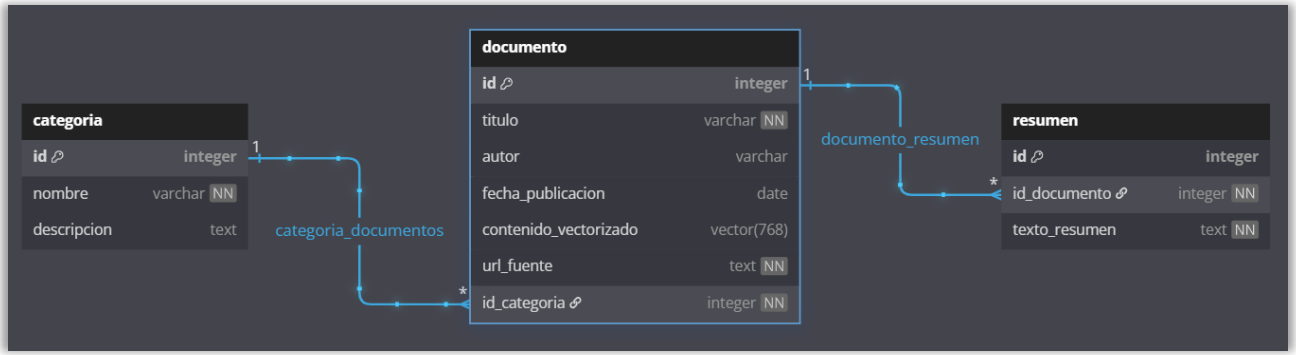
- Eficiencia en las búsquedas: Al asignar cada documento a una categoría específica, podemos filtrar rápidamente los datos relevantes para una búsqueda determinada, reduciendo el tiempo de respuesta y la carga sobre el motor de búsqueda.
- Mejora en la organización: Las categorías sirven como un punto de referencia para organizar grandes volúmenes de datos, asegurando que el sistema sea escalable y que los nuevos documentos sean integrados de manera eficiente.

Cómo se Clasifican los Documentos

La asignación de categorías se realizará mediante un pipeline automatizado que combine técnicas de procesamiento de lenguaje natural (NLP) con modelos de clasificación:

- Análisis de contenido: El texto de cada documento es analizado mediante un modelo de *embeddings* para extraer representaciones vectoriales que reflejan su significado.
- Clasificación temática: Un modelo entrenado previamente identifica la categoría principal del documento (por ejemplo, "Oncología" o "Cardiología") basándose en patrones semánticos del contenido.
- Almacenamiento estructurado: El documento se inserta en la base de datos, incluyendo la referencia a la categoría correspondiente mediante su id en la tabla categoría.

Modelo de Datos: Tablas y Campos



46 - Diagrama Entidad Relación

Tabla Categoría

La tabla categoría contiene las temáticas principales que clasifican los documentos del sistema. Cada categoría actúa como un filtro para organizar y buscar documentos de manera más eficiente, permitiendo al motor de búsqueda centrarse en grupos específicos de contenido.

Campo:	Tipo:	Descripción:
id	SERIAL, clave primaria	Identificador único de cada categoría.
nombre	VARCHAR, único y no nulo	Nombre de la categoría: "Oncología", "Cardiología"…

Tabla Documento

La tabla documento almacena toda la información relevante de los archivos recuperados por el sistema de scraping, incluyendo detalles básicos como título, fecha, y el contenido vectorizado para realizar búsquedas avanzadas. Esta tabla estará conectada directamente con la tabla categoría a través del campo id_categoria, permitiendo filtrar documentos según la categoría en las búsquedas.

Campos:	Tipo:	Descripción:
id	SERIAL, clave primaria	Identificador único de cada documento.
titulo	VARCHAR, no nulo	Título descriptivo del documento.
autor	VARCHAR, opcional	Autor del documento, si está disponible.
fecha_publicacion	DATE, opcional	Fecha de publicación del documento.
contenido_vectorizado	VECTOR(768), pgvector	Embedding vectorial generado a partir del contenido del documento.
url_fuente	TEXT, no nulo	Enlace al documento original para referencia.
id_categoria	INTEGER, clave foránea	Referencia a la categoría del documento en la tabla categorías.

Tabla Resumen

La tabla resumen almacena los resúmenes generados automáticamente para cada documento, optimizando el proceso de recuperación de datos al cargar únicamente los textos asociados a los documentos relevantes.

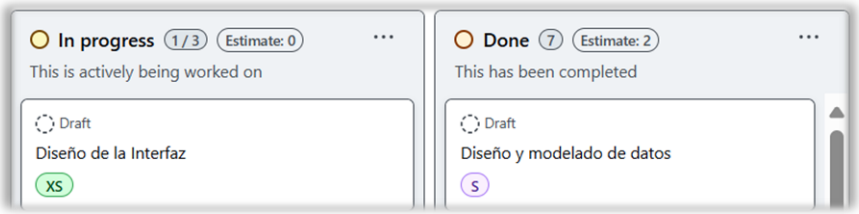
Campos:	Tipo:	Descripción:
id	SERIAL, clave primaria	Identificador único del resumen.
id_documento	INTEGER, clave foránea	Referencia al documento correspondiente en la tabla documento.
texto_resumen	TEXT, no nulo	Contenido del resumen generado automáticamente.

La relación entre resumen y documento se establece mediante el campo id_documento, asegurando que cada resumen esté vinculado a su documento único. Esto permite que solo los resúmenes de los documentos encontrados se carguen y se muestren al usuario.

Optimización del Diseño

Con la separación entre documentos, categorías y resúmenes se busca que el sistema sea:

- Eficiente: Las búsquedas se realizan en dos pasos para reducir la carga del sistema. Primero, los documentos relevantes se recuperan, y luego, sus resúmenes se cargan desde la tabla correspondiente.
- Escalable: La base de datos puede crecer independientemente en cada tabla, facilitando la incorporación de nuevos documentos, categorías y resúmenes.
- Modular: Las estructuras independientes permiten actualizaciones y regeneración de datos sin afectar otras partes del sistema.



47 - Actualización del Backlog (2)

Diseño de la Interfaz de Usuario

En este apartado se abordará el diseño visual de las pantallas principales del sistema. Para ello, se han elaborado mockups utilizando la herramienta Figma, los cuales representan tanto la página inicial como la pantalla de resultados. Se han desarrollado versiones adaptadas para escritorio y smartphone, asegurando que el diseño sea responsivo y accesible en diferentes dispositivos. Estas representaciones sirven como guía para el desarrollo, asegurando que el diseño sea consistente, accesible y centrado en las necesidades del usuario.

48 - Logo de Figma

Objetivos del Diseño

El diseño debe estar enfocado en ofrecer una experiencia intuitiva y funcional para los usuarios, simplificando el proceso de búsqueda y visualización de resultados. La prioridad es conseguir una interfaz accesible y bien estructurada (en línea con los objetivos OBJ-08 y OBJ-09), que permita a los usuarios interactuar con el sistema de manera directa, sin pasos innecesarios o complicaciones. Además, el diseño garantiza la adaptabilidad a diversos dispositivos, manteniendo la funcionalidad y accesibilidad en todos los formatos.

Pantalla Principal

La pantalla principal incluye una barra de búsqueda centrada en el diseño, junto con el logotipo de la aplicación y el botón de búsqueda. Esta pantalla no contiene elementos distractores y está optimizada para facilitar el ingreso de consultas por el usuario.

49 - Mockup Pantalla Principal

Elementos Clave:

Barra de búsqueda: Centrada como elemento principal, con un placeholder profesional y claro: "Buscar documentación clínica…"

Botón de búsqueda: Adyacente a la barra, con un diseño claro y accesible, respetando la paleta de colores.

Logotipo: Ubicado en la parte superior, representando la identidad visual del sistema.

Mockups:

Versión para Escritorio: Diseñada para pantallas de mayor tamaño, con suficiente espacio para un diseño limpio y centrado.

Versión para Smartphone: Optimizada para dispositivos móviles, asegurando que los elementos sean funcionales y legibles.

Pantalla de Resultados de Búsqueda

La pantalla muestra una lista ordenada de resultados, cada uno con el título del documento, resumen breve y enlace al contenido. En la versión de escritorio se mostrará una vista previa del documento en el lateral, posiblemente a través de un iframe, lo que le dará una mayor armonía visual al diseño. En ambas versiones, se mantiene en pantalla la barra de búsqueda para poder realizar búsquedas consecutivas.

50 - Mockup de Pantalla de Resultados

Elementos Clave:

Título de la Página: "Resultados de Búsqueda", ubicado en la parte superior para indicar el propósito de la pantalla.

Subtítulo: La búsqueda realizada por el usuario, presentada justo debajo del título para dar contexto a los resultados mostrados.

Logo de la App**:** En la versión de escritorio ubicado en la esquina superior derecha, en la de móvil se añade en los ítems de la lista de resultados. En ambos casos para reforzar la identidad del sistema.

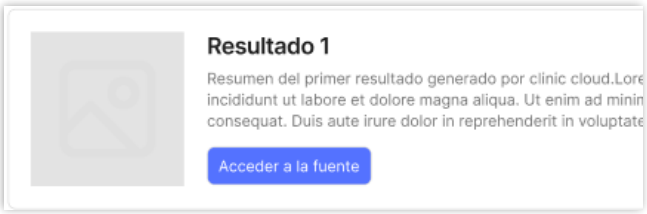
Lista de Resultados:

- Título de cada ítem que refleja el documento encontrado.
- Breve resumen del contenido del documento.
- Botón para acceder al documento completo en su fuente original.
- En la versión de escritorio una vista previa del documento en la parte izquierda.

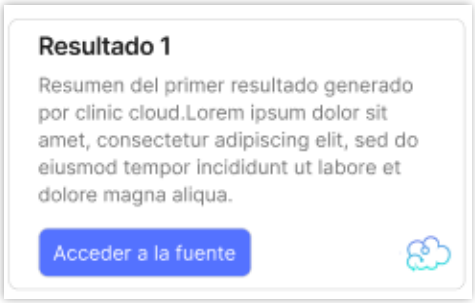
Mockups:

Versión para Escritorio: Incluye la vista previa y mayor espacio para la lista de resultados.

Versión para Smartphone: Adaptada para mostrar los resultados en una lista vertical, sin vista previa para optimizar el espacio.



51 - Ejemplo de Resultado en Escritorio



52 - Ejemplo de Resultado en Móvil

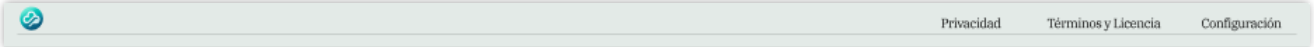
Elementos Comunes

Ambas pantallas incluyen elementos comunes que refuerzan la identidad y navegación del sistema:

Encabezado: El encabezado es simple y funcional, con enlaces que ofrecen información adicional sobre la aplicación y su código abierto.

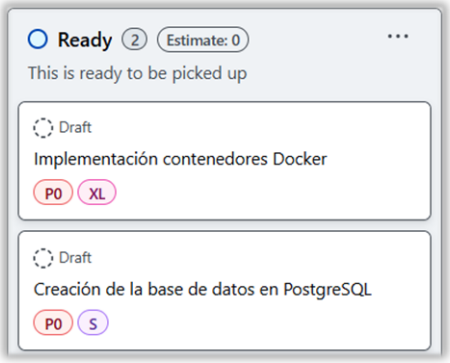
53 - Encabezado de la Aplicación

Pie de página: El pie de página incluye accesos rápidos a políticas, configuraciones y derechos, asegurando claridad y cumplimiento legal. Además, se incluye un diseño secundario del logo que será el utilizado a modo de icono de la app, reforzando de nuevo la identidad.



54 - Pie de la página de la aplicación

Desarrollo y puesta en marcha del proyecto



55 - Primeras tareas de implementación añadidas a "Ready"

El punto de partida para el desarrollo de Clinic Cloud, tal y como se definió en los objetivos (OBJ-01), consistió en la implementación del primer componente clave del proyecto: el contenedor de la base de datos.

En primer lugar, se instaló WSL2 (Windows Subsystem for Linux), ya que Docker requiere de este subsistema para ejecutar sus servicios de contenedores en sistemas operativos Windows. Una vez habilitado WSL2 y configurado correctamente, se procedió a instalar Docker Desktop.

Con estas herramientas instaladas, el entorno quedó listo para la creación y configuración del primer contenedor. Este contenedor aloja la base de datos del sistema, permitiendo la gestión de los datos desde las primeras etapas de desarrollo. Para ello, se creó un archivo docker-compose.yml, en el que se definieron los parámetros esenciales para desplegar la base de datos con PostgreSQL y su extensión pgvector, junto con un script init.sql encargado de inicializar las tablas necesarias para el proyecto.

```sql
CREATE EXTENSION IF NOT EXISTS vector;

CREATE TABLE categoria (
    id SERIAL PRIMARY KEY,
    nombre VARCHAR(255) UNIQUE NOT NULL
);

CREATE TABLE documento (
    id SERIAL PRIMARY KEY,
    titulo VARCHAR(500) NOT NULL,
    autor VARCHAR(255),
    fecha_publicacion DATE,
    contenido_vectorizado VECTOR(768), -- 768 dimensiones como el embedding de BART
    url_fuente TEXT NOT NULL,
    id_categoria INTEGER REFERENCES categoria(id)
);

CREATE TABLE resumen (
    id SERIAL PRIMARY KEY,
    id_documento INTEGER REFERENCES documento(id) ON DELETE CASCADE,
    texto_resumen TEXT NOT NULL
);
```

57 - Archivo init.sql

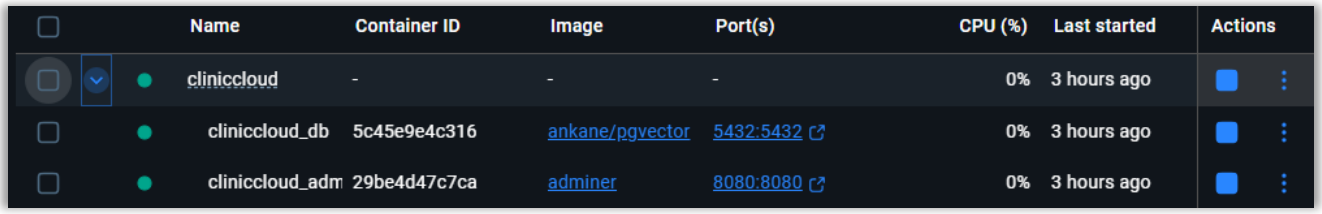
En el archivo YAML se configura un servicio llamado db, que actúa como la base de datos del proyecto. En lugar de utilizar una imagen estándar de PostgreSQL, se emplea ankane/pgvector, que incluye la extensión pgvector preinstalada. Este servicio gestiona la base de datos y garantiza la persistencia de la información mediante un volumen local, previniendo la pérdida de datos al reiniciar el contenedor. Además, se inicializa automáticamente con el script init.sql que configura las tablas directamente al crear el contenedor.

```yaml
services:
  db:
    image: ankane/pgvector
    container_name: cliniccloud_db
    restart: always
    environment:
      POSTGRES_DB: cliniccloud
      POSTGRES_USER: admin
      POSTGRES_PASSWORD: ruben
    ports:
      - "5432:5432"
    volumes:
      - pgdata:/var/lib/postgresql/data
      - ./db-init:/docker-entrypoint-initdb.d

    networks:
      - cliniccloud-net

  adminer:
    image: adminer
    container_name: cliniccloud_adminer
    restart: always
    ports:
      - "8080:8080"
    networks:
      - cliniccloud-net

volumes:
  pgdata:

networks:
  cliniccloud-net:
    driver: bridge
```

56 - Archivo docker-compose.yml de la BBDD

También se configura el adminer, un servicio que proporciona una interfaz web accesible a través del puerto que se le asigne (le fue asignado el puerto 8080), facilitando la gestión y visualización de la base de datos durante el desarrollo.

Ambos servicios están conectados mediante una red privada, cliniccloud-net, lo que asegura una comunicación eficiente y segura entre ellos. Con esta configuración se tiene un entorno para la base de datos, reproducible en cualquier equipo de forma sencilla y escalable.



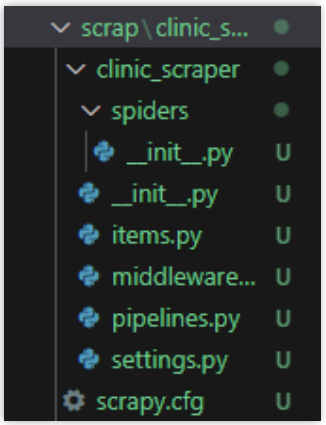
59 - Servicios "db" y "adminer" corriendo en Docker

58 - Captura de la BBDD en la Interfaz de Adminer

Implementación del Microservicio de Scraping

Una vez configurada la base de datos, el siguiente paso en el desarrollo de Clinic Cloud consistió en la implementación del microservicio de scraping (OBJ-02), encargado de la recopilación automatizada de los datos de fuentes públicas.

El primer paso fue instalar las dependencias necesarias para el desarrollo del scraper, principalmente el framework Scrapy que se había seleccionado en la fase de planificación. Una vez instalado, se utilizó el comando *startproject* para generar la estructura de directorios y archivos del scraper, siguiendo las buenas prácticas recomendadas por el framework.



60 - Estructura por defecto de un proyecto Scrapy

Implementación del Spider para PubMed

Para la fase inicial del proyecto, se desarrolló un spider específico para la extracción de datos de PubMed, una de las fuentes principales identificadas durante la planificación. Este componente implementa la lógica necesaria para interactuar con la API de PubMed (eutils), extraer los metadatos relevantes de los artículos y preparar estos datos para su almacenamiento.

```
def __init__(self, query, max_results, *args, **kwargs):
    super(PubmedSpider, self).__init__(*args, **kwargs)
    self.query = query
    self.max_results = int(max_results)
    self.base_url = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils'
```

61- Constructor del Spider de PubMed

El proceso de extracción se lleva a cabo en dos fases:

1. Búsqueda de artículos utilizando esearch.fcgi, filtrando los resultados según términos específicos definidos.
2. Obtención de detalles completos con efetch.fcgi, extrayendo información para su posterior almacenamiento en la base de datos.

```python
def start_requests(self):
    # Paso 1: Buscar IDs de artículos usando esearch
    esearch_url = f"{self.base_url}/esearch.fcgi"
    params = {
        'db': 'pubmed',
        'term': self.query,
        'retmode': 'json',
        'retmax': str(self.max_results)
    }

    yield scrapy.FormRequest(
        url=esearch_url,
        formdata=params,
        callback=self.parse_search,
        errback=self.handle_error
    )
```

63 - Método start_requests() del spider

```python
def parse_search(self, response):
    try:
        # Extraer los ids de los resultados de la query
        data = json.loads(response.body)
        id_list = data.get('esearchresult', {}).get('idlist', [])

        if not id_list:
            self.logger.warning(f"No se han encontrado artículos para la query: {self.query}")
            return

        # 2: Obtener detalles de los artículos usando efetch
        ids = ','.join(id_list)
        efetch_url = f"{self.base_url}/efetch.fcgi"
        params = {
            'db': 'pubmed',
            'id': ids,
            'retmode': 'xml'
        }

        yield scrapy.FormRequest(
            url=efetch_url,
            formdata=params,
            callback=self.parse_articles,
            errback=self.handle_error
        )
    except Exception as e:
        self.logger.error(f"Error parseando los resultados de la búsqueda: {e}")
        raise CloseSpider(f"Parse error: {e}")
```

62 - Método parse_search() del spider

Durante el segundo paso, es necesario el procesamiento de los datos para que concuerden con la estructura de la base de datos. Para ello, se implementó el método *parse_articles*, que:

1. Extrae el título del artículo.
2. Obtiene la lista de autores.
3. Extrae y formatea la fecha.
4. Construye la URL del artículo con el id.
5. Obtener el resumen del artículo.

Por último, se ejecuta el yield, enviando ese diccionario con los datos procesados al pipeline, que recibe cada elemento generado por el yield y lo procesa.

Cabe destacar que, para el MVP (Minimum Viable Product), el desarrollo se enfocó exclusivamente en el spider de PubMed. Otras fuentes de datos serán integradas en futuras iteraciones, una vez que la arquitectura y la estructura del sistema sean lo suficientemente sólidas para gestionar múltiples fuentes de información.

```python
def parse_articles(self, response):
    """ Método para extraer los datos de los XML de los artículos"""
    articles = response.xpath('//PubmedArticle')

    for article in articles:
        try:
            # Extraer titulo
            title = article.xpath('.//ArticleTitle/text()').get('')

            # Extraer autores
            authors = []
            author_list = article.xpath('.//Author')
            for author in author_list:
                last_name = author.xpath('./LastName/text()').get('')
                first_name = author.xpath('./ForeName/text()').get('')
                if last_name and first_name:
                    authors.append(f"{last_name} {first_name}")

            # Extraer fecha de publicación
            year = article.xpath('.//PubDate/Year/text()').get('')
            month = article.xpath('.//PubDate/Month/text()').get('01')
            day = article.xpath('.//PubDate/Day/text()').get('01')

            # formatearla para la base de datos
            pub_date = None
            if year:
                try:
                    pub_date = f"{year}-{month.zfill(2)}-{day.zfill(2)}"
                except:
                    pub_date = None

            # Extraemos la URL
            pmid = article.xpath('.//PMID/text()').get('')
            url = f"https://pubmed.ncbi.nlm.nih.gov/{pmid}/" if pmid else ""

            # y el resumen del artuculo:
            abstract_parts = article.xpath('.//AbstractText/text()').getall()
            abstract = ' '.join(abstract_parts)

            yield {
                'titulo': title,
                'autor': ', '.join(authors) if authors else '',
                'fecha_publicacion': pub_date,
                'url_fuente': url,
                'abstract': abstract,
                'origen': 'PubMed',
                'categoria': 'Sin categoría',
                'termino_busqueda': self.query
            }
        except Exception as e:
            self.logger.error(f"Error parseando el articulo: {e}")
```

64 - Método parse_articles() del spider

Pipeline para Persistencia de Datos

Una vez extraídos y procesados los datos en el spider mediante *parse_articles()*, es fundamental garantizar su correcta persistencia en la base de datos PostgreSQL. Para ello, se implementó un pipeline de datos, cuya función es recibir cada elemento generado por el yield, procesarlo y almacenarlo en las tablas correspondientes.

El archivo settings.py del scraper, determina el pipeline encargado de gestionar la persistencia. En la versión actual, cuando el spider extrae información y la estructura en un diccionario, Scrapy lo redirige a la clase PostgreSQLPipeline, encargada de procesar y almacenar los datos en la base de datos.

```
ITEM_PIPELINES = {
    #'clinic_scraper.pipelines.PrintPipeline': 300,
    'clinic_scraper.pipelines.PostgreSQLPipeline': 300,
}
```

65 - Variable (en settings.py) que define qué pipeline está activo.

El primer paso consiste en establecer una conexión con PostgreSQL al iniciar el spider. Durante esta fase, se verifica la existencia de las tablas esenciales y se asegura que la categoría "Sin categoría" esté disponible para cualquier documento sin clasificación específica.

```
def open_spider(self, spider):
    try:
        spider.logger.info(f"Conectando a la base de datos: {self.pg_host}:{self.pg_port}/{self.pg_db}")
        self.connection = psycopg2.connect(
            host=self.pg_host,
            port=self.pg_port,
            dbname=self.pg_db,
            user=self.pg_user,
            password=self.pg_password
        )
        self.cursor = self.connection.cursor()

        # Verificar que la tabla categoría existe
        self.cursor.execute("SELECT EXISTS (SELECT FROM information_schema.tables WHERE table_name = 'categoria')")
        tabla_existe = self.cursor.fetchone()[0]
        if not tabla_existe:
            spider.logger.error("La tabla 'categoria' no existe en la base de datos")
            raise Exception("Tabla 'categoria' no encontrada")

        # Comprobar la estructura de la tabla documento
        self.cursor.execute("SELECT column_name, data_type FROM information_schema.columns WHERE table_name = 'documento'")
        columnas = {col[0]: col[1] for col in self.cursor.fetchall()}
        spider.logger.info(f"Estructura de la tabla 'documento': {columnas}")

        # Asegurarse de que existe la categoría "Sin categoría"
        self.cursor.execute(
            "INSERT INTO categoria (nombre) VALUES (%s) ON CONFLICT (nombre) DO NOTHING",
            ("Sin categoría",)
        )
        self.connection.commit()
```

66 - Método open_spider() del pipeline. (Recortado)

Una vez conectado, el sistema determina la categoría del documento. Si el término de búsqueda corresponde a una categoría registrada, se vincula el artículo a ella. En caso contrario, se crea una nueva categoría y se almacena junto al documento.

Seguidamente, se genera un embedding utilizando el modelo SentenceTransformer. Este proceso transforma el contenido textual, combinando título y resumen, en una representación numérica. Si el modelo no está disponible, se genera un vector aleatorio como alternativa.

Con el documento estructurado, el pipeline inserta la información en la base de datos. Se registran el título, los autores, la fecha de publicación y el vector de características en la tabla documento, mientras que el resumen del artículo se guarda en la tabla resumen, asegurando que ambos elementos queden correctamente asociados.

```python
def process_item(self, item, spider):
    try:
        spider.logger.info(f"Procesando item: {item.get('titulo', '')[:50]}...")

        # Generar el embedding del texto
        text_to_embed = f"{item.get('titulo', '')} {item.get('abstract', '')}"
        vector_768 = self._generate_embedding(text_to_embed, spider)
        spider.logger.info("Embedding generado correctamente")

        # Obtener el ID de la categoría
        categoria_id = self._get_categoria_id(item.get('termino_busqueda', ''), spider)
        spider.logger.info(f"Categoría ID obtenida: {categoria_id}")
```

```python
def _generate_embedding(self, text, spider):
    """Genera el embedding para el texto dado"""
    if self.model is not None:
        try:
            # Usar el modelo para generar el embedding
            embedding = self.model.encode(text)
            # Convertir a vector de 768 dimensiones
            return self._convert_embedding_to_pgvector(embedding)
        except Exception as e:
            spider.logger.error(f"Error al generar embedding: {e}")

    # Fallback: generar vector aleatorio
    spider.logger.warning("Usando vector aleatorio como fallback")
    return self._generate_random_vector()
```

67 - Método de generación de embeddings del pipeline

```python
# Insertar el documento con su embedding
spider.logger.info("Intentando insertar documento en la base de datos
self.cursor.execute(
    """
    INSERT INTO documento
    (titulo, autor, fecha_publicacion, contenido_vectorizado, url_fue
    VALUES (%s, %s, %s, %s, %s, %s)
    RETURNING id
    """,
    (
        item.get('titulo', ''),
        item.get('autor', ''),
        fecha_pub,
        vector_768,
        item.get('url_fuente', ''),
        categoria_id
    )
)
documento_id = self.cursor.fetchone()[0]
spider.logger.info(f"Documento insertado con ID: {documento_id}")
```

```python
self.connection.commit()
spider.logger.info(f"Transacción completada exitosamente para '{item.get('titulo')[:50]}...'")
```

68 - Método process_item() del pipeline. (Recortado)

Finalmente, tras confirmar la transacción, se cierra la conexión (close_spider), liberando recursos y garantizando la estabilidad del sistema. Este flujo asegura que la información extraída por el spider se almacene de manera eficiente, lista para ser utilizada en futuros análisis o consultas.

```python
def close_spider(self, spider):
    if self.cursor:
        self.cursor.close()
    if self.connection:
        self.connection.close()
    spider.logger.info("Conexión a la base de datos cerrada")
```

69 - Método del pipeline que cierra la conexión con la BBDD.

Script Principal y Programación de Tareas

El microservicio incorpora un script de control (main.py), que coordina la ejecución del scraper y permite:

- Ejecución automática del scraper al iniciar el contenedor.
- Programación de tareas con ejecuciones periódicas, actualmente configurado para 24 horas.

Contenerización del Microservicio

Siguiendo la arquitectura de microservicios establecida en la planificación, se preparó un Dockerfile específico para el scraper, así como un archivo de requisitos para instalar las herramientas necesarias en el entorno contenerizado:

```
scrapy==2.10.0
psycopg2-binary==2.9.7
sqlalchemy==2.0.21
python-dotenv==1.0.0
schedule==1.2.0
sentence-transformers==2.2.2
huggingface_hub==0.14.1
transformers==4.26.0
torch>=1.10.0
twisted
numpy>=1.20.0
```

70 - Archivo requeriments.txt

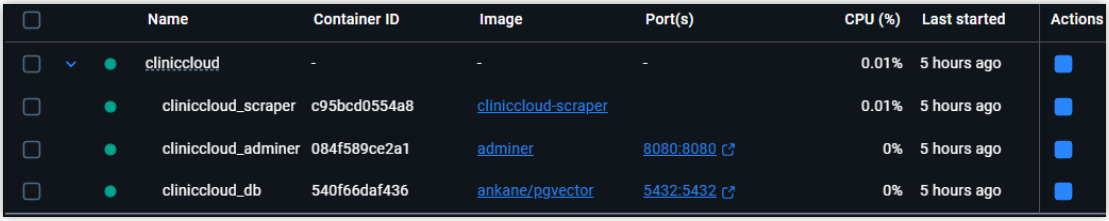
```
FROM python:3.10-slim

WORKDIR /app

# Instalar dependencias de sistema necesarias
RUN apt-get update && apt-get install -y \
    build-essential \
    libpq-dev \
    && apt-get clean \
    && rm -rf /var/lib/apt/lists/*

# Copiar requirements.txt primero para aprovechar la caché de Docker
COPY requirements.txt .

# Instalar dependencias de Python
RUN pip install --no-cache-dir -r requirements.txt

# Copiar el resto del código fuente
COPY . .

# Variables de entorno
ENV PYTHONPATH=/app
ENV PYTHONUNBUFFERED=1
```

71 - Dockerfile del Scraper

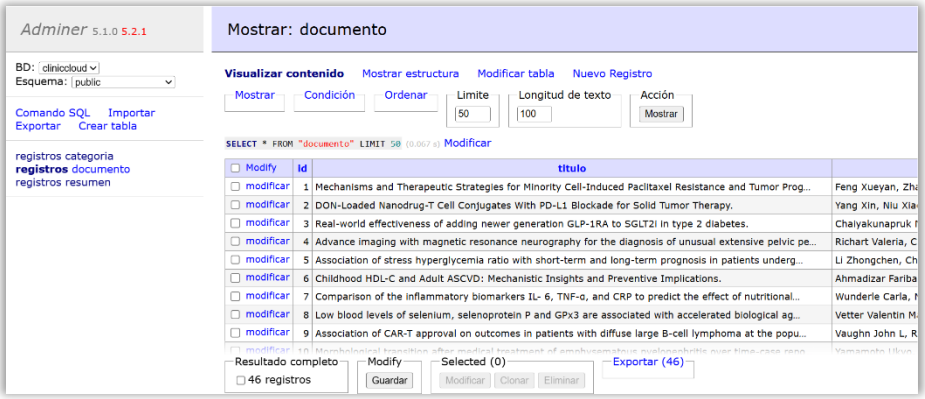
```
▷ Run Service
scraper:
  build:
    context: ./scraper
    dockerfile: Dockerfile
  container_name: cliniccloud_scraper
  restart: always
  depends_on:
    - db
  volumes:
    - ./scraper:/app
  environment:
    - PYTHONPATH=/app
  networks:
    - cliniccloud-net
```

72 - Actualización del docker-compose.yml

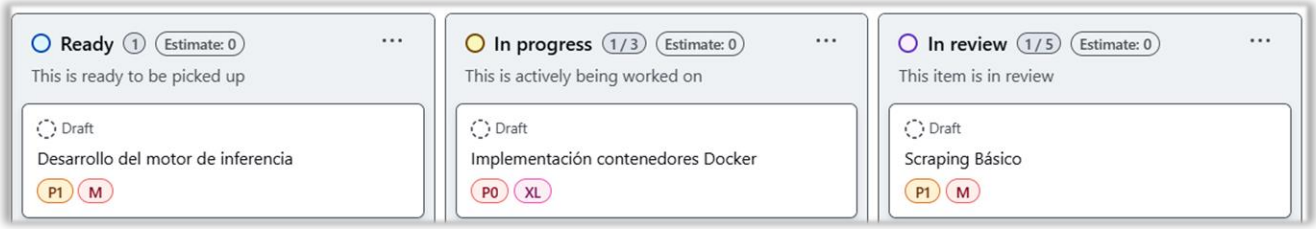
Posteriormente, se actualizó el archivo docker-compose.yml para incluir este nuevo servicio junto a los componentes existentes. Dentro de esta configuración, el scraper se comunica con la base de datos mediante la red privada cliniccloud-net, manteniendo en todo momento un entorno estable, escalable y reproducible en cualquier máquina.



74 - Servicio de scraping corriendo en Docker



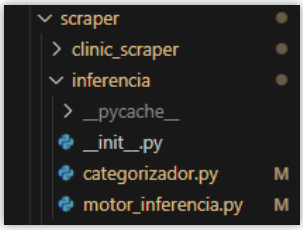
73 - Base de datos tras las primeras inserciones del scraper.



75 - Actualización del Backlog del Proyecto (3)

Implementación del motor de inferencia

Una vez consolidado el proceso de extracción y almacenamiento de documentos mediante el microservicio de scraping, el siguiente paso en el desarrollo consistió en la implementación del motor de inferencia (OBJ-03). Este componente tiene como objetivo condensar el contenido de cada artículo científico en un resumen breve y claro, optimizado para su presentación en la interfaz de búsqueda y navegación.



76 – Estructura del scraper actualizada

Para mantener la modularidad y favorecer futuras mejoras, el motor de inferencia se encapsuló en un módulo independiente, ubicado en *inferencia/motor_inferencia.py*. La función principal, *generar_miniresumen()*, recibe un texto y devuelve la versión condensada. Para lograr resúmenes precisos se integró el modelo preentrenado BART (facebook/bart-large-cnn) mediante la librería Transformers de Hugging Face.

```python
def generar_miniresumen(texto: str, max_length: int = 150, min_length: int = 50) -> str:
    """Genera un miniresumen condensado del texto."""
    if not texto or not texto.strip():
        return ""

    try:
        # Obtener singleton del gestor de modelos
        model_manager = ModelManager()

        # Preprocesamiento del texto
        texto_limpio = _preprocesar_texto(texto)

        # Generar resumen
        if model_manager.summarizer:
            resumen = model_manager.summarizer(
                texto_limpio,
                max_length=max_length,
                min_length=min_length,
                do_sample=False
            )
            resumen_texto = resumen[0]['summary_text']

            # Aplicar postprocesamiento
            resumen_final = _postprocesar_texto(resumen_texto)
            return resumen_final
        else:
            logger.warning("Modelo de resumen no disponible, devolviendo fragmento del texto original")
            return texto[:min_length]
    except Exception as e:
        logger.error(f"Error generando resumen: {e}")
        # Fallback: devolver las primeras frases
        return texto[:min_length] if texto else ""
```

77 - Método de generación de resúmenes en motor_inferencia.py

Con el objetivo de optimizar recursos y reducir la complejidad en esta fase inicial, el motor de inferencia se incorporó directamente en el mismo contenedor y pipeline del microservicio de scraping. Así, durante el procesamiento de cada ítem en el pipeline PostgreSQLPipeline, una vez extraída y formateada la información del artículo, se invoca la función de inferencia para generar de forma automática un resumen condensado antes de proceder a su almacenamiento definitivo en la base de datos.

Características y funcionalidades del Motor de Inferencia

Se ha implementado el patrón “Singleton” para cargar el modelo solo una vez, lo que evita redundancias y optimiza el consumo de memoria durante la ejecución. Además, al iniciarse, el sistema detecta automáticamente si se dispone de una GPU y, en función de ello, ajusta el procesamiento para mejorar el rendimiento.

Se realiza una limpieza del contenido de entrada (preprocesamiento) y se garantiza un formato uniforme en los resúmenes (postprocesamiento), lo que incluye el manejo adecuado de caracteres especiales y la implementación de mecanismos para gestionar errores o fallos en el modelo.

```python
def _postprocesar_texto(texto: str) -> str:
    """Mejora la calidad del resumen generado"""
    if not texto:
        return ""

    # que termine con un punto
    if texto and texto[-1] not in ['.', '!', '?']:
        texto += '.'

    # primera letra en mayús
    if texto:
        texto = texto[0].upper() + texto[1:]

    return texto
```

```python
def _preprocesar_texto(texto: str) -> str:
    """Preprocesa el texto para mejorar la calidad del resumen"""
    if not texto:
        return ""

    # borrar espacios en blanco (mas de 1) y caracteres problematicos
    texto = re.sub(r'\s+', ' ', texto)
    texto = re.sub(r'[^\w\s.,;:?!-]', '', texto)

    return texto.strip()
```

78 - Métodos de procesamiento de texto del motor de inferencia

Además de generar resúmenes, el motor incorpora métodos para la extracción de palabras clave y la clasificación del contenido, dotando de una mayor funcionalidad al análisis de cada documento.

Paralelamente al motor de inferencia, se ha desarrollado un sistema de inferencia para la categorización automática de los documentos médicos basado en su contenido. Este sistema se implementa en el módulo categorizador.py y combina dos enfoques:

1. Modelo basado en Machine Learning: Se utiliza un transformer biomédico (RoBERTa) para analizar y clasificar el contenido de manera automática.

2. Método basado en reglas: Sirve como respaldo si no se ha podido cargar el modelo, analizando la frecuencia de términos específicos y utilizando una lista de términos clave con especialidades como Oncología, Cardiología o Neurología para compararlos.

```python
def categorizar_texto(self, titulo: str, abstract: str) -> List[Tuple[str, float]]:
    """ Categoriza un texto médico basado en su título y resumen"""
    # título y resumen combinados, dando más peso al título
    texto_completo = f"{titulo} {titulo} {abstract}"
    texto_completo = texto_completo.lower()

    if self.use_model:
        return self._categorizar_con_modelo(texto_completo)    1 machine learning
    else:
        return self._categorizar_con_reglas(texto_completo)    2 reglas propias
```

79 - Método principal del categorizador

Las funcion, *obtener_mejor_categoria(),* determina la categoría más adecuada para cada documento. Además, la integración de este módulo en el pipeline PostgreSQLPipeline permite que, junto con el resumen generado, se asigne de forma automática la categoría correspondiente. Se crean las categorías principales al iniciar el proceso, y en caso de no identificarse una especialidad específica, se asigna por defecto la categoría "Medicina General".

```
# Obtener el ID de la categoría mediante inferencia
categoria_id = self._get_categoria_id(titulo, abstract, spider)
spider.logger.info(f"Categoría ID obtenida: {categoria_id}")
```

80 - Actualización del método process_item() del pipeline del scraper

```
def _get_categoria_id(self, titulo, abstract, spider):
    """ Obtiene o crea una categoría apropiada basada en el contenido del documento """
    try:
        # Usar el motor de inferencia para determinar la categoría
        categoria_nombre = obtener_mejor_categoria(titulo, abstract)
        spider.logger.info(f"Categoría inferida: {categoria_nombre}")

        # Intentar obtener el ID de la categoría
        self.cursor.execute("SELECT id FROM categoria WHERE nombre = %s", (categoria_nombre,))
        resultado = self.cursor.fetchone()

        if resultado:
            return resultado[0]
        # Si no existe la categoría (poco probable dado que pre-creamos las principales), la creamos
        self.cursor.execute(
            "INSERT INTO categoria (nombre) VALUES (%s) RETURNING id",
            (categoria_nombre,)
        )
        self.connection.commit()
        nuevo_id = self.cursor.fetchone()[0]
        spider.logger.info(f"Creada nueva categoría '{categoria_nombre}' con ID {nuevo_id}")
        return nuevo_id
    except Exception as e:
        spider.logger.error(f"Error al obtener/crear categoría: {e}")
        return self.categoria_default_id  # Fallback a categoría por defecto
```

81 - Nuevo método del pipeline para obtener la categoría

```
def obtener_mejor_categoria(titulo: str, abstract: str) -> str:
    """Obtiene la mejor categoría para un documento médico"""
    try:
        categorizer = MedicalCategorizer()
        categorias = categorizer.categorizar_texto(titulo, abstract)

        if categorias:
            return categorias[0][0] # la categoría con mayor puntuación
        else:
            return "Medicina General"
    except Exception as e:
        logger.error(f"Error al categorizar documento: {e}")
        return "Medicina General"
```

82 - Método del categorizador (inferencia) que devuelve la categoría del documento

Definición de procedimientos de control y evaluación de la ejecución de proyectos

Pruebas y Depuración del Scraper

Una vez implementado el código base del scraper, se procedió a realizar pruebas para verificar su correcto funcionamiento antes de integrar el sistema completo. Este proceso reveló diversos desafíos que fueron solucionados de manera sistemática.

Preparación del Entorno de Pruebas

Para garantizar un entorno controlado, se decidió probar el scraper de manera aislada antes de integrarlo con el resto del sistema:

python -m venv myenv

myenv\Scripts\activate

pip install scrapy psycopg2-binary

Se creó un entorno virtual dedicado para el scraper, asegurando que las dependencias estuvieran correctamente instaladas y aisladas.

Ejecución de Pruebas Iniciales

La primera ejecución del spider reveló un error de conexión a la base de datos:

psycopg2.OperationalError: could not translate host name "db" to address: Host desconocido.

Este error era esperado ya que la configuración estaba preparada para el entorno Docker donde "db" sería el nombre del servicio de la base de datos, pero no es accesible desde el entorno local.

Adaptación para Pruebas Locales

Para superar esta limitación durante la fase de pruebas, se realizaron las siguientes modificaciones:

Creación de un pipeline alternativo en pipelines.py: PrintPipeline.

Esta adaptación permitió ejecutar pruebas sin dependencia de la base de datos, facilitando la verificación del correcto funcionamiento del spider.

Gestión de Restricciones de Acceso

Una segunda ejecución reveló un problema relacionado con el cumplimiento de robots.txt:

La API de eutils.ncbi.nlm.nih.gov, aunque pública y diseñada para acceso programático, tiene ciertas restricciones en su archivo robots.txt. Para solucionar este problema:

Modificación de settings.py:

Se desactivó el cumplimiento de robots.txt para permitir el acceso a la API

ROBOTSTXT_OBEY = False

Mantenimiento de buenas prácticas:

Se mantuvo un delay entre peticiones para respetar los límites de la API con DOWNLOAD_DELAY = 3

Resultados de las Pruebas

Tras estas modificaciones, el scraper fue capaz de:

- Conectarse exitosamente a la API de PubMed
- Realizar búsquedas basadas en términos especificados
- Extraer metadatos relevantes como títulos, autores y resumenes
- Presentar esta información de manera estructurada

Estas pruebas sentaron las bases para la siguiente fase: la integración del scraper con la base de datos PostgreSQL y su despliegue en contenedores Docker junto con el resto de los microservicios del sistema.

Referencias y bibliografía

Cristian Palau. (2024). Obtenido de https://cristianpalau.com/tipos-de-licencias-open-source-codigo-abierto/

elDiario. (2025). Obtenido de https://www.eldiario.es/sociedad/csic-universidades-quedan-publicar-articulos-revistas-cientificas-prestigiosas_1_12084431.html

EmergenResearch. (2023). Obtenido de https://www.emergenresearch.com/es/industry-report/mercado-de-salud-digital

gestionandote.org. (2021). Obtenido de https://www.gestionandote.org/subvenciones-para-ideas-de-emprendimiento-social-shuttleworth/

Goteo Wiki. (2024). Obtenido de https://es.wikipedia.org/wiki/Goteo

iapp. (2024). Obtenido de https://iapp.org/news/a/the-state-of-web-scraping-in-the-eu

metricson. (2024). Obtenido de https://metricson.com/legalidad-del-web-scraping-y-otras-formas-de-recabar-datos-de-internet-bajo-el-rgpd

PubMed Wiki. (2024). Obtenido de https://es.wikipedia.org/wiki/PubMed

SciSpace Wiki. (2025). Obtenido de https://es.wikipedia.org/wiki/SciSpace

Semantic Scholar Wiki. (2023). Obtenido de https://es.wikipedia.org/wiki/Semantic_Scholar

The Lancet. (2021). Obtenido de https://www.thelancet.com/journals/landig/article/PIIS2589-7500(21)00238-7/fulltext

Anexos

<Cualquier añadido se incluirá en este apartado.>

Por ejemplo, a continuación, se incluyen unas pautas a tener en cuenta a la hora de elaborar la documentación del proyecto.

Guía de estilo

Título del proyecto

Elegir un nombre llamativo y relacionado con la temática que va a tratar.

Figuras y tablas

Cualquier figura, tabla... incluida en el documento deberá tener un título a pie de página.

Incluir tablas, gráfico, mapas conceptuales...que ayuden a leer y comprender el documento.

Índices

Además del índice de contenidos, ya incluido en la plantilla, se añadirá a continuación el índice de figuras, si fuera necesario.

Redacción

Se evitarán las mayúsculas, salvo en los títulos y poco más.

No se emplearán formas personales (instalamos, seleccionamos...) en su lugar se utilizarán formas impersonales (instalar, se instalará, seleccionar, se selecciona,...).

Se evitará la voz pasiva (casi siempre traducción literal del inglés). En vez de: es desarrollado para cumplir... mejor: se desarrolla para cumplir...

Se evitarán los párrafos largos.

Se utilizarán las viñetas para facilitar la lectura del documento.

Formato

El documento se generará en formato pdf.

Entrega

Todo el material del módulo Proyecto (documentos, ficheros fuentes, herramientas...) se entregará en formato electrónico, en una carpeta comprimida:

CICLO-CURSO-TITULO DEL PROYECTO