eman ta zabal zazu

Universidad del País Vasco
Euskal Herriko Unibertsitatea

INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Ingeniería del Software

Recolección y gestión de memorias de TFGs de Facultades de Informática

Ander Fernandez Carrero

Dirección
Juanan Pereira Varela

30 de octubre de 2023

Resumen

Castellano

Este trabajo consiste en la expansión y mejora de un proyecto existente, *Reposearch*. *Reposearch* es un buscador de memorias de informática en España. Se han añadido nuevos repositorios a indexar y se ha implementado un análisis de disponibilidad de las memorias. Se ha desarrollado un servidor *frontend* y una *API* para mejorar la experiencia del usuario, incluyendo nuevos filtros de búsqueda. Además, se ha integrado un sistema de análisis semántico para mejorar la precisión de las búsquedas. Estas mejoras ofrecen a los usuarios una plataforma más completa y accesible para explorar trabajos de fin de grado en el campo de la informática.

Euskara

Lan honen helburua *Reposearch*, kode proiektua hedatu eta hobetzea du. *Reposearch* Espainiako Informatika memoriak bilatzailea da. Biltegi berriak gehitu dira indizeari eta memoriaren erabilgarritasunaren analisia ezarri da. Erabiltzailearen esperientzia hobetzeko *frontend* zerbitzaria eta *API* garatu dira, bilaketarako iragazki berriak sartuz. Gainera, bilaketa-zehaztasuna hobetzeko, analisi semantiko sistema bat integratu da. Hobekuntza hauek plataforma osatuagoa eta eskuragarriagoa eskaintzen diete erabiltzaileei, informatika gradu amaierako lanak aztertzeko.

English

This work involves the expansion and improvement of an existing project, *Reposearch*. *Reposearch* is a search engine for computer science theses in Spain. New repositories have been added for indexing, and an analysis of thesis availability has been implemented. A *frontend* server and an *API* have

been developed to enhance user experience, including new search filters. Additionally, a semantic analysis system has been integrated to improve search precision. These enhancements provide users with a more comprehensive and accessible platform to explore undergraduate theses in the field of computer science.

Resumen ejecutivo

En esta sección, se presenta un resumen ejecutivo que destaca las principales funcionalidades del proyecto Reposearch **antes** de comenzar el proyecto descrito en esta memoria, y las funcionalidades añadidas **durante** el desarrollo de este TFG.

Funcionalidades pre-existentes	***Funcionalidades desarrolladas***
Indexación de 19 universidades	Indexación de 33 universidades
	Actualización de los sets[1]
	Refactorización de la indexación
	Manejo de excepciones en la indexación
	Estructura del código más limpia
Buscador Web	Buscador Web Renovada
	Secciones nuevas para los filtros
	Búsqueda semántica agregada
	Concatenación de filtros
API	API renovada
	Nuevas rutas
	Análisis de disponibilidad de las memorias
	Descarga de memorias
	Generación de embeddings
	Generación de Excel con resumen del análisis
	Remodelación de la BD
	Nuevos sistemas de backup

Tabla 1: Resumen ejecutivo

[1]En el protocolo OAI-PMH, un "set" es un grupo de registros relacionados en un repositorio digital, organizados por criterios específicos para facilitar la búsqueda y recuperación de información.

Prefacio

El objetivo primordial de este proyecto es mejorar en diversos ámbitos un proyecto anterior, ya existente y utilizada por numerosas personas a lo largo de la península: Reposearch. En el proceso, se ha aplicado el conocimiento adquirido durante el grado en un contexto que abarca diversas áreas de la informática. Se ha investigado a fondo múltiples tecnologías modernas para su ejecución. Se han introducido nuevas funcionalidades y se ha optimizado la experiencia de usuario.

El proyecto "Reposearch" es esencialmente un motor de búsqueda para trabajos de fin de grado en informática. Centraliza y almacena metadatos de diversas memorias provenientes de múltiples repositorios en una única base de datos. Los usuarios pueden acceder fácilmente a esta información a través de una interfaz gráfica intuitiva en el navegador. Aunque el proyecto emplea varios lenguajes de programación para sus distintas funciones, Python destaca como el principal lenguaje utilizado.

Agradecimientos

Quiero expresar mi sincero agradecimiento a las personas que han sido fundamentales en la realización de este Trabajo de Fin de Grado.

A mi tutor, el Profesor Juanan Pereira, le agradezco por su orientación y apoyo constante. A mi familia, amigos y en especial a Jun, les estoy agradecido por su paciencia y compañía durante todo este proceso. Gracias por estar a mi lado.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

En España, los alumnos de último curso de grado universitario deben desarrollar un Trabajo de Fin de Grado(*TFG*) para completar sus estudios. Aunque no todas las universidades lo hacen, una gran parte opta por almacenar los trabajos de fin de grado en sus respectivos repositorios institucionales. Los repositorios institucionales almacenan y ofrecen acceso abierto a una amplia gama de producción científica y académica, como tesis, artículos y ponencias, entre otros tipos de trabajos. En España, los repositorios institucionales están experimentando un auge significativo, actualmente contando con un total de 76 universidades que disponen de ello.

Los alumnos que quieran consultar las memorias ya realizadas por otros alumnos deberían acceder a dichos repositorios para su búsqueda. Ya que cada repositorio sólo incluye los trabajos realizados en dicha universidad, habría que buscar en todos los repositorios para una búsqueda más específica. Esto genera una dificultad tanto de tiempo como de ineficiencia, generando una experiencia de uso pésima para el usuario.

La *Red de Bibliotecas Universitarias Españolas (REBIUN)* es una organización que agrupa todas las bibliotecas universitarias y científicas de España. Todos los repositorios que forman REBIUN implementan el protocolo *OAI-PMH*(Open Archives Initiative Protocol for Metadata Harvesting)(McCown et al. [2006]). El protocolo OAI-PMH sirve para recopilar descripciones de metadatos de registros dentro de un archivo, en este caso dentro de un repositorio institucional. Haciendo uso de este protocolo nace el proyecto Reposearch, obteniendo la información de los TFGs de informática de varios repositorios y mostrándolos al usuario en un buscador de página web de manera centralizada.

1.1. Contexto

El proyecto Reposearch(Pereira [2021]) fue ideado originalmente por el profesor Juanan Pereira, docente en la Facultad de Informática de Donostia/San Sebastián (UPV-EHU). Juanan fue quien incentivó al autor de este Trabajo Fin de Grado a continuar con el desarrollo del proyecto.

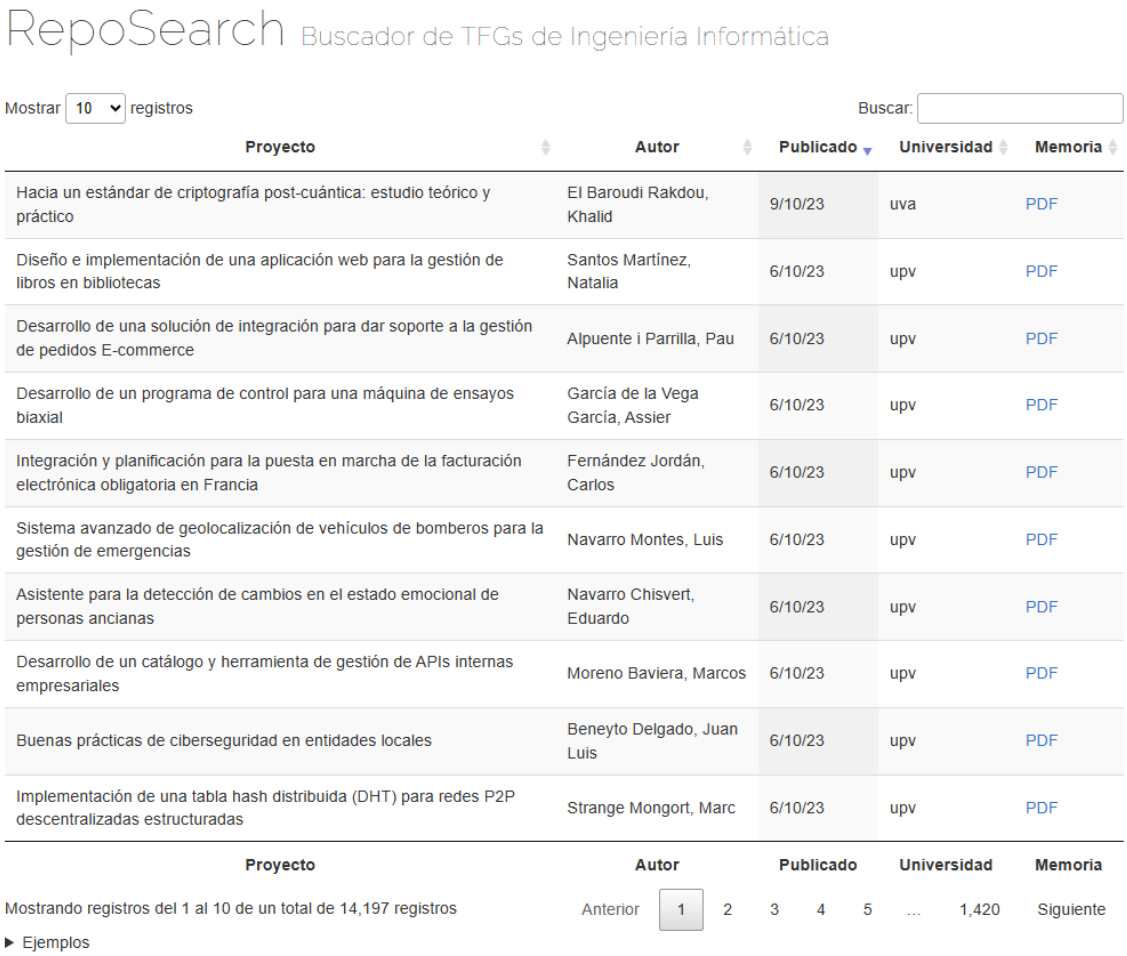
Uno de los propósitos es añadir valor a un proyecto ya existente, brindando la oportunidad de explorar nuevas tecnologías más allá del entorno académico, como el protocolo *OAI-PMH* y tecnologías en alza en la gestión de datos (como bases de datos vectoriales). Esto permite expandir las habilidades más allá del ámbito del software.

Al ser un proyecto pequeño con un gran potencial de crecimiento, Reposearch es un buen proyecto candidato para un TFG, teniendo que lidiar con multitud de diferentes problemas y diseñar una solución acorde, proporcionando una estructura sólida en la que ir implementando y desarrolando nuevas funcionalidades.

Por último, tener al creador original como mentor hace que la experiencia del proceso sea única y enriquecedora, proporcionando una guía invaluable y una conexión personal con el proyecto que va más allá de lo académico.

1.2. Reposearch

Reposearch es un buscador de Trabajos de Fin de Grado de Ingeniería Informática(Reposeach [2021]). Alguna de sus características son las siguientes:



RepoSearch Buscador de TFGs de Ingeniería Informática

Mostrar 10 registros Buscar:

Proyecto	Autor	Publicado	Universidad	Memoria
Hacia un estándar de criptografía post-cuántica: estudio teórico y práctico	El Baroudi Rakdou, Khalid	9/10/23	uva	PDF
Diseño e implementación de una aplicación web para la gestión de libros en bibliotecas	Santos Martínez, Natalia	6/10/23	upv	PDF
Desarrollo de una solución de integración para dar soporte a la gestión de pedidos E-commerce	Alpuente i Parrilla, Pau	6/10/23	upv	PDF
Desarrollo de un programa de control para una máquina de ensayos biaxial	García de la Vega García, Assier	6/10/23	upv	PDF
Integración y planificación para la puesta en marcha de la facturación electrónica obligatoria en Francia	Fernández Jordán, Carlos	6/10/23	upv	PDF
Sistema avanzado de geolocalización de vehículos de bomberos para la gestión de emergencias	Navarro Montes, Luis	6/10/23	upv	PDF
Asistente para la detección de cambios en el estado emocional de personas ancianas	Navarro Chisvert, Eduardo	6/10/23	upv	PDF
Desarrollo de un catálogo y herramienta de gestión de APIs internas empresariales	Moreno Baviera, Marcos	6/10/23	upv	PDF
Buenas prácticas de ciberseguridad en entidades locales	Beneyto Delgado, Juan Luis	6/10/23	upv	PDF
Implementación de una tabla hash distribuida (DHT) para redes P2P descentralizadas estructuradas	Strange Mongort, Marc	6/10/23	upv	PDF
Proyecto	Autor	Publicado	Universidad	Memoria

Mostrando registros del 1 al 10 de un total de 14,197 registros

Anterior 1 2 3 4 5 … 1,420 Siguiente

▶ Ejemplos

Figura 1.1: Página Web de Reposearch

- **Multi-Repositorio**: El buscador de Reposearch indexa un total de 19 universidades.

Universidad	***Acrónimo***
Universidad Politécnica de Madrid	UPM
Universidad del País Vasco - EHU	UPVEHU
Universidad de Las Palmas de Gran Canaria	ULPGC
Universidad Autónoma de Madrid	UAM
Universitat Pompeu Fabra	UPF
Universidad de Castilla-La Mancha	UCLM
Universidad de La Laguna	ULL
Universidad de Málaga	UMA
Universidad de Sevilla	US
Universitat Autònoma de Barcelona	UAB
Universitat de Barcelona	UB
Universitat Politècnica de Catalunya	UPC
Universidad de Granada	UGR
Universidad de Murcia	UM
Universidad Pública de Navarra	UPNA
Universidad de Valladolid	UVA
Universidad de Zaragoza	UNIZAR
Universitat Politècnica de València	UPV
Universidade de Santiago de Compostela	USC

Tabla 1.1: Universidades indexadas por Reposearch

- **Filtros y ordenación**: El buscador provee de herramientas y ejemplos para poder buscar de manera más específica en una interfaz muy sencilla de utilizar.

1.2.1. Situación del proyecto

Reposearch está activo[1] desde 2021, desarrollado por Juanan Pereira, docente en la Facultad de Informatica de Donostia/San Sebastián. Actualmente contiene un total de 14,209 trabajos de fin de grado indexados, listos para su búsqueda.

Desde su fecha de lanzamiento, el proyecto no ha experimentado modificaciones evidentes, lo que hace difícil realizar comparaciones directas con versiones anteriores.

Este TFG tiene como objetivo desarrollar una nueva versión con funcionalidades ampliadas y una interfaz web más moderna y funcional.

1.2.2. Arquitectura

Reposearch requiere de varios servicios para su correcto funcionamiento. Uno es la interfaz web, donde se muestran los resultados de manera gráfica. El segundo es la API que envía los datos paginados a la interfaz web. Por último los servicios de *scrapping*, encargados de buscar nuevos TFGs en los repositorios y almacenarlos en la *BD* de manera automatizada.

La interfaz web de Reposearch tiene una arquitectura cliente-servidor. El cliente(usuario) envia una petición al dominio(servidor), y el servidor le responde. Para hacer la petición se utiliza un navegador, y el servidor responde con una respuesta HTML.

La página web se ejecuta sobre un servidor web que es el encargado de gestionar las solicitudes y respuestas *HTTP*. El protocolo HTTP es el estándar utilizado para la transmisión de información en la World Wide Web.

La lógica de negocio, en este caso la parte del servidor que se encarga de obtener los datos de la BD e incrustarlos en el html es mediante el lenguaje de programación PHP, y el documento que tiene el contenido estilizado es HTML5 con CSS3.

La API del proyecto, implementada en scripts PHP, tiene la función de obtener (hacer *fetch*) de la BD de MySQL devolviendo la información paginada. La *paginación* es un mecanismo utilizado para dividir grandes conjuntos

[1]Enlace al buscador: https://reposearch.ikasten.io/v1/data_sources/server_side.html.

de datos en partes más pequeñas y manejables, llamadas "páginas".

El código del proyecto Reposearch hace uso de varias dependencias que simplifican el desarrollo de las funcionalidades, evitando la necesidad de implementar herramientas complejas, como la manipulación de la base de datos, desde cero. Las principales son las siguientes:

- **Sickle**[2]: biblioteca de Python diseñada específicamente para interactuar con servicios que siguen el protocolo OAI-PMH.
- **SqlAlchemy**[3]: es una biblioteca de mapeo objeto-relacional (ORM) en Python. Permite interactuar con la BD utilizando objetos en lugar de consultas SQL directas.
- **BeautifulSoup**[4]: biblioteca de Python que se utiliza para extraer datos de archivos HTML y XML.

[2]OAI-PMH for humans (https://github.com/mloesch/sickle)
[3]The Python SQL Toolkit and Object Relational Mapper (https://www.sqlalchemy.org/)
[4]Scrapping de HTML y XML (https://www.crummy.com/software/BeautifulSoup/)

1.3. Objetivos

Trás una formación en diferentes areas y competencias, con este proyecto se intenta cumplir los siguientes objetivos interpersonales enfocado a un proyecto de software:

1.3.1. Poner en practica las habilidades y conocimientos adquiridos

Como estudiante de ingeniería se busca una resolución acorde a las necesidades que requiere el proyecto.

Para ello se hará uso de todo los conocimientos adquiridos, que involucra desde la arquitectura y la lógica de programación hasta las buenas prácticas del software.

Como ingeniero también será necesario documentar todo el proceso, explicando con todo detalle todas las modificaciones y nuevas caracterísiticas que comprenderá el nuevo software. El documento deberá estar escrito correctamente, evitando ambigüedades.

Se busca también establecer una estructura base de código a partir de la cual los futuros alumnos de ingeniería informática puedan dar continuidad al proyecto, corrigiendo, documentando y ampliando aún más sus funcionalidades.

1.3.2. Mejorar la experiencia del usuario

La página web actual de Reposearch presenta cierta complejidad al filtrar resultados por parámetros distintos al título.

Se busca potenciar la experiencia del usuario en Reposearch, implementando mejoras significativas en la plataforma. Estas mejoras están diseñadas para hacer que la búsqueda y la exploración de las memorias de Trabajos de Fin de Grado (TFG) sean más intuitivas, eficientes y atractivas para los usuarios. A continuación, se describe alguna de dichas mejoras:

- **Interfaz de usuario intuitiva:** Se ha rediseñado la interfaz de usuario para que sea más fácil de usar y accesible para todos los usuarios. Tiene

un estilo minimalista, que busca la sencillez.

- **Filtros Avanzados:** Se han agregado nuevos filtros de búsqueda que permiten a los usuarios refinar sus consultas de manera más específica. Estos filtros ayudan a los usuarios a encontrar memorias basadas en criterios como fecha, autor y universidad entre otros.

1.3.3. Aprendizaje de nuevas Tecnologías

En el transcurso de este proyecto, uno de los objetivos fundamentales ha sido el aprendizaje continuo y uso de nuevas tecnologías. Para los estudiantes de ingeniería informática, es crucial estar al tanto de las últimas tendencias y herramientas tecnológicas para desarrollar habilidades relevantes y mantenerse competitivos en el campo.

Se ha profundizado en varios lenguajes de programación impartidos en el grado, y se han aplicado *frameworks* modernos para el desarrollo del proyecto. Los conocimientos adquiridos están destinados a ser aplicados en futuros trabajos con el fin de desarrollar software más seguro y avanzado.

2. Metodología

En esta sección, se presenta la metodología empleada para llevar a cabo la investigación y el desarrollo del presente Trabajo de Fin de Grado (TFG). La metodología adoptada se basó en un enfoque híbrido[1], combinando elementos del modelo Waterfall y el prototipado iterativo, lo que permitió una evolución progresiva y adaptativa de las soluciones propuestas.

En las etapas iniciales del proyecto, se siguió el enfoque Waterfall para establecer una definición clara y detallada de los requisitos del TFG. Esta fase proporcionó una estructura sólida, ofreciendo una comprensión precisa de los objetivos y restricciones del proyecto antes de avanzar a la fase de implementación.

Con los requisitos definidos, se procedió al análisis y diseño del proyecto. Durante la implementación del diseño, se crearon prototipos que se utilizaron para realizar pruebas y detectar posibles errores o requisitos no previstos en la fase de análisis. Posteriormente, se realizó un nuevo análisis y diseño con la retroalimentación proporcionada por el tutor del TFG. Esta guía ayudó a corregir los errores identificados y mejorar el código del proyecto de manera efectiva y eficiente.

[1]La metodología híbrida puede incluir combinaciones de elementos del Waterfall y Agile, como por ejemplo, "AgileFall" y "Scrumban", entre otras.(Belling [2020])

2.1. Herramientas utilizadas

En esta sección se nombrarán las herramientas, librerías y servicios principales que se han utilizado para el desarrollo del trabajo de fin de grado. Las más destacable son las siguientes:

- **Python**: Python es un lenguaje de programación de alto nivel de ámbito general. Se ha utilizado como el lenguaje principal para el desarrollo del proyecto, tanto en scripts de scrapping como en la API de la aplicación.
- **Javascript**: Javascript es un lenguaje de programación ampliamente utilizado para el desarrollo web. Se ha empleado en el desarrollo del servicio frontend, específicamente dentro del framework ReactJS.
- **NodeJS**: Node.js es un entorno de ejecución de JavaScript del lado del servidor. En este proyecto, se utiliza como la base de ReactJS, generando un servidor web, gestionar las solicitudes del cliente y proporcionando la lógica del servidor para la aplicación.
- **LaTeX**: Es un procesador de texto para la creación de documentos científicos y técnicos, especialmente en tesis, informes y artículos académicos.
- **Overleaf**: Es un servicio en línea que permite la edición de documentos basado en Latex. Se ha utilizado para el desarrollo del proyecto.
- **Visual Studio Code**: Es un editor de código fuente desarrollado por Microsoft. Se ha utilizado como el entorno de desarrollo integrado (*IDE*) principal para escribir y depurar el código del proyecto.
- **Git**: Git es un sistema de control de versiones distribuido que se utiliza para el seguimiento de cambios en el código fuente durante el desarrollo del software. Se ha empleado para gestionar las versiones del código.
- **GitHub**: GitHub es una plataforma de alojamiento de Git que proporciona funcionalidades adicionales, como control de versiones en la nube, seguimiento de problemas y colaboración en equipo. Se ha utilizado para alojar el repositorio del proyecto, permitiendo el trabajo colaborativo y el seguimiento del progreso del código.

- **ReactJS**: ReactJS es una biblioteca de Javascript utilizada para construir interfaces de usuario interactivas. Se ha empleado en el frontend del proyecto para crear una experiencia de usuario dinámica y altamente interactiva, permitiendo a los usuarios interactuar fácilmente con la aplicación web.
- **Uvicorn**: Uvicorn es un servidor Asynchronous Server Gateway Interface(ASGI) que se utiliza para servir aplicaciones web Python basadas en ASGI. En este proyecto, Uvicorn se ha utilizado para ejecutar la API, permitiendo la ejecución asincrónica y eficiente del código Python.
- **FastAPI**: FastAPI es un framework web moderno para construir APIs. Se ha utilizado en el *backend* del proyecto para desarrollar una API rápida y segura, facilitando la comunicación entre el frontend y la base de datos.
- **MySQL Workbench**: MySQL Workbench es una herramienta de diseño y administración de bases de datos para MySQL. En este proyecto, se ha utilizado para realizar pruebas y comprobar en tiempo real los cambios ejecutados por los scripts de Python en la base de datos.
- **ChatGPT**: ChatGPT es un modelo de lenguaje desarrollado por OpenAI. En este proyecto, ChatGPT se ha utilizado para realizar búsquedas de información rápida y precisas, buscar errores en el código y corregir errores ortográficos y gramaticales.
- **Penpot**: Penpot es una herramienta de diseño de código abierto que se utiliza para el diseños de interfaces de usuario. La interfaz web ha sido diseñada con esta herramienta.
- **PlantUML**: Es una herramienta de código abierto que permite crear diagramas UML de manera sencilla y textual.

2.2. Alcance

En esta sección se va a definir el alcance del proyecto. Para facilitar la planificación se han descompuesto las tareas en un diagrama Estructura de Desglose de Trabajo (EDT).

Se han definido siete bloques principales, divididos por agrupaciones de tareas de un mismo ámbito:

1. **Planificación y Gestión**: Este bloque incluye las tareas para organizar y gestionar las tareas para el desarrollo del proyecto.

2. **Estudio**: Este bloque agrupa todas las herramientas y lenguajes que requiren de formación para el desarrollo del proyecto.

3. **Captura de Requisitos**: Conjunto de actividades para comprender y definir los requisitos del proyecto.

4. **Análisis**: En este bloque se realiza una evaluación profunda del proyecto desde una perspectiva de software. Durante esta fase, se identifican y comprenden los posibles problemas y necesidades del sistema.

5. **Diseño**: Agrupación de tareas para el modelado y definición de la estructura del software.

6. **Implementación**: Este bloque agrupa las tareas para el desarrollo del código software siguiendo el diseño establecido.

7. **Documentación memoria**: A este bloque pertenecen las tareas de redacción de la memoria del proyecto.

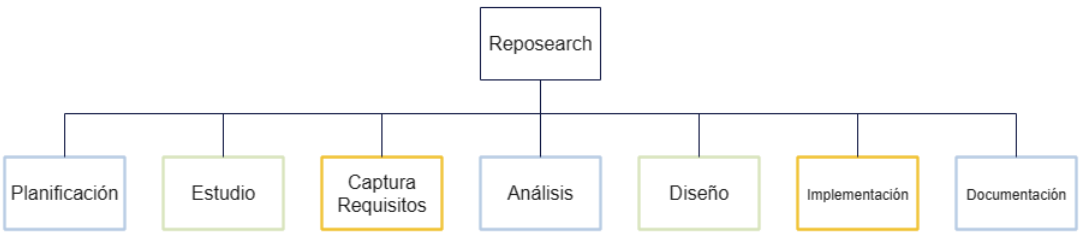


Figura 2.1: Diagrama EDT general

2.2.1. Planificacion y Gestión

En este bloque se incluyen las tareas para organizar y gestionar las tareas para el desarrollo del proyecto.

Figura 2.2: Diagrama EDT del bloque de Planificación y Gestión

Planificación de tareas
Paquete de trabajo: Planificación y Gestión. **Duración estimada**: 15 horas.
Descripción: Identificar y definir las tareas a realizar en el proyecto, junto con su estimación de duración.
Salidas/Entregables: Documento con tareas a realizar
Recursos necesarios: -

Tabla 2.1: Planificación de tareas

Reuniones
Paquete de trabajo: Planificación y Gestión. **Duración estimada**: 10 horas.
Descripción: Reuniones con el tutor del proyecto con el objetivo de obtener recomendaciones, discutir los problemas encontrados y establecer puntos de control para el desarrollo del proyecto.
Salidas/Entregables: Anotaciones sobre el tema tratado.
Recursos necesarios: -

Tabla 2.2: Reuniones

2.2.2. Estudio

En este bloque se agrupan todas las herramientas y lenguajes que requiren de formación para el desarrollo del proyecto.

Figura 2.3: Diagrama EDT del bloque de Estudio

Proyecto Reposearch
Paquete de trabajo: Estudio. **Duración estimada**: 10 horas.
Descripción: Estudio del proyecto original Reposearch para comprender su estructura y funcionamiento.
Salidas/Entregables: -
Recursos necesarios: Acceso al proyecto original.

Tabla 2.3: Proyecto Reposearch

NodeJS + Express
Paquete de trabajo: Estudio. **Duración estimada**: 25 horas.
Descripción: Formación del entorno de ejecución de Node.js y el framework de aplicaciones web Express, incluyendo sus características, funcionalidades y mejores prácticas de desarrollo.
Salidas/Entregables: -
Recursos necesarios: Documentación de NodeJS y Express.

Tabla 2.4: NodeJS + Express

ReactJS
Paquete de trabajo: Estudio. **Duración estimada**: 25 horas.
Descripción: Formación del framework web ReactJS, incluyendo características, funcionalidades y mejores prácticas de desarrollo.
Salidas/Entregables: -
Recursos necesarios: Documentación de ReactJS.

Tabla 2.5: ReactJS

Lucene
Paquete de trabajo: Estudio. **Duración estimada**: 15 horas.
Descripción: Formación sobre el framework Apache Lucene, incluyendo características, funcionalidades y mejores prácticas de desarrollo.
Salidas/Entregables: -
Recursos necesarios: Documentación de Apache Lucene.

Tabla 2.6: Lucene

2.2.3. Captura de Requisitos

En este conjunto de actividades se encuentran las tareas para comprender y definir los requisitos del proyecto.

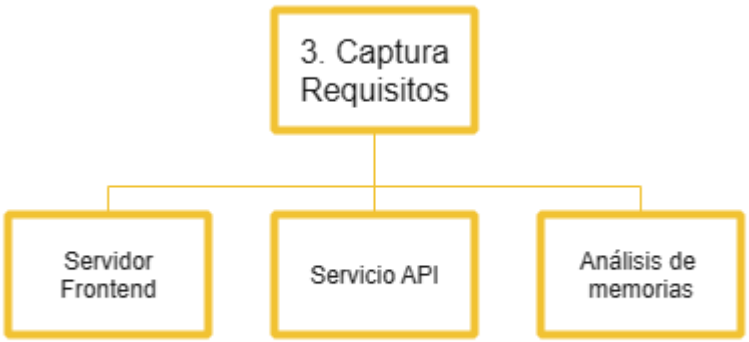


Figura 2.4: Diagrama EDT del bloque de Captura de Requisitos

Servidor Frontend
Paquete de trabajo: Captura de Requisitos. **Duración estimada**: 5 horas.
Descripción: Se definirán las funciones que deberá cumplir el buscador web.
Salidas/Entregables: Documento formal con diagrama de casos de uso.
Recursos necesarios: PlantUML.

Tabla 2.7: Servidor Frontend

Servicio API
Paquete de trabajo: Captura de Requisitos. **Duración estimada**: 10 horas.
Descripción: Se definirán los requisitos funcionales y no funcionales del servicio.
Salidas/Entregables: Documento formal con diagrama de casos de uso.
Recursos necesarios: PlantUML.

Tabla 2.8: Servicio API

Análisis de memorias
Paquete de trabajo: Captura de Requisitos. **Duración estimada**: 15 horas.
Descripción: Se definirán las funciones que implica realizar un análisis de memorias junto con sus limitaciones.
Salidas/Entregables: Documento formal con diagrama de casos de uso.
Recursos necesarios: PlantUML.

Tabla 2.9: Análisis de memorias

2.2.4. Análisis

En este bloque se realiza una evaluación profunda del proyecto desde una perspectiva de software. Durante esta fase, se identifican y comprenden los posibles problemas y necesidades del sistema.

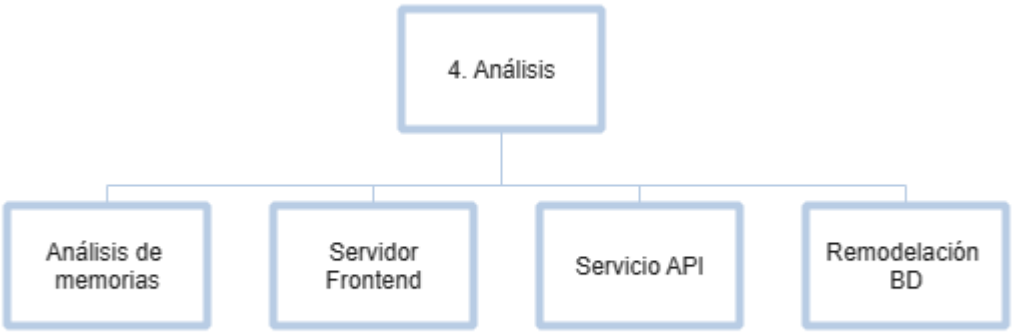


Figura 2.5: Diagrama EDT del bloque de Análisis

Análisis de memorias
Paquete de trabajo: Análisis. **Duración estimada**: 20 horas.
Descripción: Establecimiento de criterios y normativas para categorizar las memorias y definición de requisitos específicos del software a considerar durante el diseño.
Salidas/Entregables: Borrador.
Recursos necesarios: Diagramas definidos en la captura de requisitos.

Tabla 2.10: Análisis de memorias

Servidor Frontend
Paquete de trabajo: Análisis. **Duración estimada**: 5 horas.
Descripción: Establecimiento de estados para gestionar los filtros y comunicar a los usuarios su estado actual.
Salidas/Entregables: Lista de estados
Recursos necesarios: Diagramas definidos en la captura de requisitos.

Tabla 2.11: Implementación

Servicio API
Paquete de trabajo: Análisis. **Duración estimada**: 10 horas.
Descripción: Definición de parámetros de entrada y salida y tipos de excepciones.
Salidas/Entregables: Borrador.
Recursos necesarios: Diagramas definidos en la captura de requisitos.

Tabla 2.12: Servicio API

Remodelación BD
Paquete de trabajo: Análisis. **Duración estimada**: 5 horas.
Descripción: Se identificarán los cambios que requiere la BD para almacenar todo lo necesario por el proyecto.
Salidas/Entregables: Lista de cambios.
Recursos necesarios: -

Tabla 2.13: Remodelación BD

2.2.5. Diseño

En esta agrupación de tareas se realizan las tareas para el modelado y definición de la estructura del software. También se explicará el funcionamiento del diseño con diagramas de secuencia.

Figura 2.6: Diagrama EDT del bloque de Diseño

Análisis de memorias
Paquete de trabajo: Diseño. **Duración estimada**: 25 horas.
Descripción: Diseño del proceso para analizar memorias y clasificarlas en la categoría apropiada definida en el análisis.
Salidas/Entregables: Documento formal con diagramas de secuencia.
Recursos necesarios: PlantUML y Diagramas definidos en la sección de análisis.

Tabla 2.14: Análisis de memorias

Servidor Frontend
Paquete de trabajo: Diseño. **Duración estimada**: 10 horas.
Descripción: Diseño del mecanismo para gestionar los estados y sus transiciones en el buscador web.
Salidas/Entregables: Documento formal con diagramas de secuencia.
Recursos necesarios: PlantUML y Diagramas definidos en la sección de análisis.

Tabla 2.15: Servidor Frontend

Servicio API
Paquete de trabajo: Diseño. **Duración estimada**: 15 horas.
Descripción: Diseño detallado de las operaciones y funcionalidades que la API proporcionará.
Salidas/Entregables: Documento formal con diagramas de secuencia.
Recursos necesarios: PlantUML y Diagramas definidos en la sección de análisis.

Tabla 2.16: Servicio API

Lucene
Paquete de trabajo: Diseño. **Duración estimada**: 15 horas.
Descripción: Definición detallada del algoritmo y las técnicas de búsqueda que se utilizarán para analizar el contenido de las memorias.
Salidas/Entregables: Documento formal con diagramas de secuencia.
Recursos necesarios: PlantUML y Diagramas definidos en la sección de análisis.

Tabla 2.17: Lucene

Remodelación BD
Paquete de trabajo: Diseño. **Duración estimada**: 5 horas.
Descripción: Diseño de las nuevas tablas y columnas que contendrá la BD.
Salidas/Entregables: UML.
Recursos necesarios: Diagramas utilizado en el análisis.

Tabla 2.18: Remodelación BD

2.2.6. Implementación

En este bloque se encuentran las tareas para el desarrollo del código del software siguiendo el diseño establecido.

Figura 2.7: Diagrama EDT del bloque de Implementación

Servidor Frontend
Paquete de trabajo: Implementación. **Duración estimada**: 15 horas.
Descripción: Desarrollo del código en ReactJS siguiendo el diseño establecido.
Salidas/Entregables: Código relacionado al servidor Frontend.
Recursos necesarios: ReactJS y Diagramas definidos en la sección de análisis y diseño.

Tabla 2.19: Servidor Frontend

Servicio API
Paquete de trabajo: Implementación. **Duración estimada**: 10 horas.
Descripción: Desarrollo del código en NodeJS y Express siguiendo el diseño establecido.
Salidas/Entregables: Código relacionado al servicio API.
Recursos necesarios: NodeJS, Express y Diagramas definidos en la sección de análisis y diseño.

Tabla 2.20: Servicio API

Análisis de memorias
Paquete de trabajo: Implementación. **Duración estimada**: 25 horas.
Descripción: Desarrollo del código para implementar el análisis de las memorias.
Salidas/Entregables: Código relativo al desarrollo del análisis de las memorias.
Recursos necesarios: Python y Diagramas utilizado en el análisis y diseño.

Tabla 2.21: Análisis de memorias

Lucene
Paquete de trabajo: Implementación. **Duración estimada**: 15 horas.
Descripción: Desarrollo del código para implementar un buscador de contenido.
Salidas/Entregables: Código relativo al buscador de contenido.
Recursos necesarios: Lucene y Diagramas utilizado en el análisis y diseño.

Tabla 2.22: Lucene

Remodelación BD
Paquete de trabajo: Implementación. **Duración estimada**: 3 horas.
Descripción: Implementación de modificaciones en la base de datos de acuerdo con el diseño previamente establecido, así como el desarrollo de los archivos necesarios con el esquema de la base de datos para los servicios que lo usan.
Salidas/Entregables: Esquema de la nueva BD.
Recursos necesarios: MySQL y Diagramas utilizado en el análisis y diseño.

Tabla 2.23: Remodelación BD

2.2.7. Documentación

En este bloque pertenecen las tareas de la redacción de la memoria del proyecto.

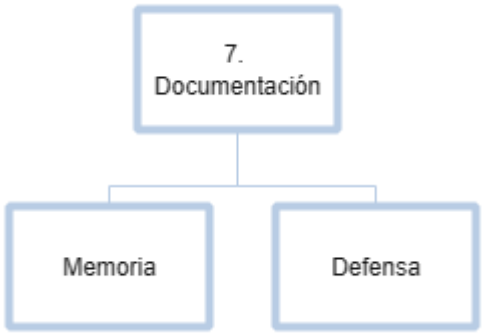


Figura 2.8: Diagrama EDT del bloque de Documentación

Memoria
Paquete de trabajo: Documentación. **Duración estimada**: 120 horas.
Descripción: Creación de un documento PDF que detalla la organización, planificación y tareas realizadas durante el proyecto.
Salidas/Entregables: Memoria TFG.
Recursos necesarios: Overleaf, Diagramas generados y todo el código desarrollado.

Tabla 2.24: Memoria

Defensa
Paquete de trabajo: Documentación. **Duración estimada**: 10 horas.
Descripción: Elaboración y preparación de una presentación visual y oral que resume el TFG.
Salidas/Entregables: Presentación en formato PPT.
Recursos necesarios: Diagramas generados para el desarrollo de la memoria.

Tabla 2.25: Defensa

2.2.8. Resumen en tabla

Tarea	Nombre de la Tarea	Estimación
1.1	Planificación de tareas	15
1.2	Reuniones	10
2.1	Estudio Proyecto Reposearch	10
2.2	Estudio NodeJS + Express	25
2.3	Estudio ReactJS	25
2.4	Estudio Lucene	15
3.1	Captura de requisitos Frontend	5
3.2	Captura de requisitos API	10
3.3	Captura de requisitos Análisis de memorias	15
4.1	Análisis del Análisis de memorias	20
4.2	Análisis Frontend	5
4.3	Análisis API	10
4.4	Análisis BD	5
5.1	Diseño Análisis de memorias	25
5.2	Diseño Frontend	10
5.3	Diseño API	15
5.4	Diseño Lucene	15
5.5	Diseño BD	5
6.1	Implementación Frontend	15
6.2	Implementación API	10
6.3	Implementación Análisis de memorias	25
6.4	Implementación Lucene	15
6.5	Implementación BD	3
7.1	Memoria	120
7.2	Defensa	10
Estimación Total		438

Figura 2.9: Tabla de tareas y estimaciones

2.3. Cambios en el Alcance del proyecto

En esta sección se explicarán los dos cambios significativos que han ocurrido en el alcance inicial del proyecto.

El primer cambio significativo ha sido la estimación inicial de la fecha de finalización del proyecto. En el inicio del proyecto, la estimación de las tareas era difícil debido a la falta de conocimiento, y se planeó inicialmente terminar a principios de febrero de 2023. Sin embargo, esta estimación inicial resultó ser incorrecta, lo que llevó a un cronograma más ajustado de lo previsto. El proyecto comenzó apenas dos meses antes del segundo cuatrimestre académico. Debido a estos factores y las exigencias de las cinco asignaturas, se detuvo temporalmente el desarrollo del TFG, ya que estas asignaturas se volvieron prioritarias.

Además, otro cambio significativo se produjo en la elección tecnológica. Durante las etapas de estudio, diseño e implementación de Lucene, que abarcó aproximadamente el 70-80 % de las expectativas iniciales, se evidenció una complejidad de uso considerable para los usuarios finales. Tras reconocer estas dificultades, se tomó la decisión de explorar alternativas más adecuadas. Después de una cuidadosa evaluación, se optó por el uso de una base de datos vectorial, una solución que se percibió como más eficiente y amigable tanto para el desarrollo como para los usuarios. Esta transición implicó ajustes en los paquetes de trabajo existentes, incluida la migración de la API de Node.js y Express a FastAPI, para alinear el sistema con la nueva elección tecnológica y garantizar una experiencia mejorada para todos los usuarios.

A continuación se van a mostrar las modificaciones y nuevos paquetes de trabajo:

2.3.1. Servicio API

El paquete de trabajo "Servicio API" abarca todas las tareas relacionadas con el desarrollo del servicio API. Este paquete está interconectado con los paquetes de trabajo del EDT: Estudio, Captura de Requisitos, Análisis, Diseño e Implementación.

La Tabla 2.4 hace referencia al estudio de NodeJS y Express, será modificada para el estudio de FastAPI. La Tabla 2.8 tiene como objetivo capturar los requisitos; aunque se realizarán modificaciones, la gran mayoría del con-

tenido permanecerá igual. En la Tabla 2.12 se realiza el análisis de la API, por lo que será necesario rehacer el contenido. En la tabla 2.16 se realiza el diseño de la API, será necesario rehacer el contenido. Finalmente en la Tabla 2.20 se realiza la implementación de la API, será necesario implementar el nuevo diseño. El resultado es el siguiente:

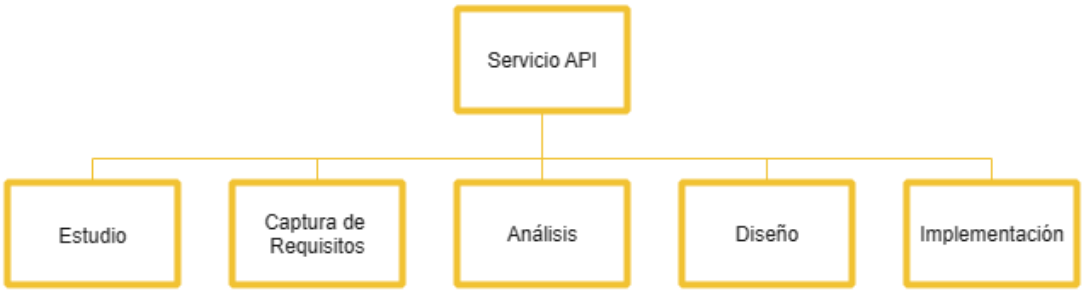


Figura 2.10: Paquete de Trabajo Servicio API

Estudio
Paquete de trabajo: Servicio API. **Duración estimada**: 25 horas.
Descripción: Formación del framework de aplicaciones web FastAPI, incluyendo sus características, funcionalidades y mejores prácticas de desarrollo.
Salidas/Entregables: -
Recursos necesarios: Documentación de FastAPI.

Tabla 2.26: Estudio

Captura de Requisitos
Paquete de trabajo: Servicio API. **Duración estimada**: 10 horas.
Descripción: Se definirán los requisitos funcionales y no funcionales del servicio.
Salidas/Entregables: Documento formal con diagrama de casos de uso.
Recursos necesarios: PlantUML.

Tabla 2.27: Captura de Requisitos

Análisis
Paquete de trabajo: Servicio API. **Duración estimada**: 10 horas.
Descripción: Definición de parámetros de entrada y salida y tipos de excepciones.
Salidas/Entregables: Borrador.
Recursos necesarios: Diagramas definidos en la captura de requisitos.

Tabla 2.28: Análisis

Diseño
Paquete de trabajo: Servicio API. **Duración estimada**: 15 horas.
Descripción: Diseño detallado de las operaciones y funcionalidades que la API proporcionará usando FastAPI.
Salidas/Entregables: Documento formal con diagramas de secuencia.
Recursos necesarios: PlantUML y Diagramas definidos en la sección de análisis.

Tabla 2.29: Diseño

Implementación
Paquete de trabajo: Servicio API. **Duración estimada**: 10 horas.
Descripción: Desarrollo del código en FastAPI siguiendo el diseño establecido.
Salidas/Entregables: Código relacionado al servicio API.
Recursos necesarios: FastAPI y Diagramas definidos en la sección de análisis y diseño.

Tabla 2.30: Implementación

2.3.2. Lucene

Todas las tareas que involucran Lucene serán eliminadas.

2.3.3. Base de datos Vectorial

El paquete de trabajo "Base de datos Vectorial" abarca todas las tareas relacionadas con el desarrollo de un buscador semántico. Este paquete está interconectado con los paquetes de trabajo del EDT: Estudio, Captura de Requisitos, Análisis, Diseño e Implementación.

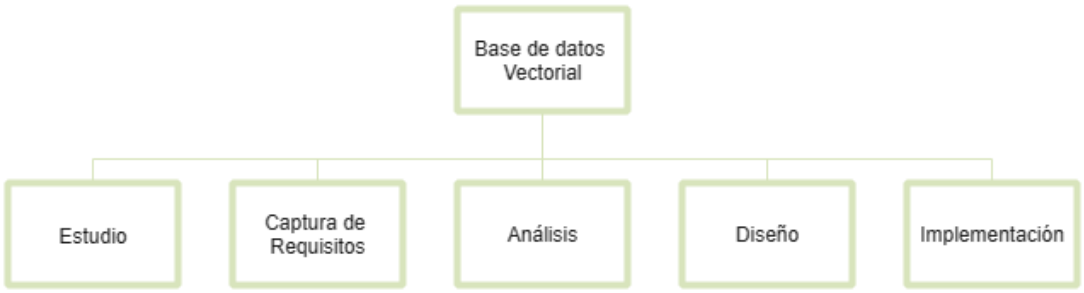


Figura 2.11: Paquete de Trabajo Base de datos Vectorial

Estudio
Paquete de trabajo: Base de datos Vectorial. **Duración estimada**: 15 horas.
Descripción: Estudio y formación en bases de datos vectoriales, enfocándose en la tecnología Milvus para comprender su estructura y funcionalidad.
Salidas/Entregables: -
Recursos necesarios: Documentación de Milvus.

Tabla 2.31: Estudio

Captura de Requisitos
Paquete de trabajo: Base de datos Vectorial. **Duración estimada**: 5 horas.
Descripción: Se defininirán los requisitos funcionales y no funcionales del servicio.
Salidas/Entregables: Documento formal con diagrama de casos de uso.
Recursos necesarios: PlantUML.

Tabla 2.32: Captura de Requisitos

Análisis
Paquete de trabajo: Base de datos Vectorial. **Duración estimada**: 5 horas.
Descripción: Definición del contenido y los metadatos de los TFGs que se desean almacenar.
Salidas/Entregables: Borrador.
Recursos necesarios: Diagramas definidos en la captura de requisitos.

Tabla 2.33: Análisis

Diseño
Paquete de trabajo: Base de datos Vectorial. **Duración estimada**: 15 horas.
Descripción: Diseño detallado del proceso para la inserción de contenido y búsqueda semántica.
Salidas/Entregables: Documento formal con diagrama de secuencia.
Recursos necesarios: PlantUML y Diagramas definidos en la sección de análisis.

Tabla 2.34: Diseño

Implementación
Paquete de trabajo: Base de datos Vectorial. **Duración estimada**: 15 horas.
Descripción: Desarrollo del código en Python siguiendo el diseño establecido.
Salidas/Entregables: Código realacionado con la Base de datos Vectorial.
Recursos necesarios: Python y Diagramas definidos en la sección de análisis y diseño.

Tabla 2.35: Implementación

2.3.4. Resumen en tabla

La tabla con todas las tareas nuevas y eliminando las antiguas quedaría de la siguiente manera:

Tarea	Nombre de la Tarea	Estimación
1.1	Planificación de tareas	15
1.2	Reuniones	10
2.1	Estudio Proyecto Reposearch	10
2.2	Estudio FastAPI	25
2.3	Estudio ReactJS	25
2.4	Estudio Base de datos Vectorial	15
3.1	Captura de requisitos Frontend	5
3.2	Captura de requisitos API	10
3.3	Captura de requisitos Análisis de memorias	15
3.4	Captura de requisitos Base de datos vectorial	5
4.1	Análisis del Análisis de memorias	20
4.2	Análisis Frontend	5
4.3	Análisis API	10
4.4	Análisis BD	5
4.5	Análisis Base de datos Vectorial	5
5.1	Diseño Análisis de memorias	25
5.2	Diseño Frontend	10
5.3	Diseño API	15
5.4	Diseño Base de datos Vectorial	15
5.5	Diseño BD	5
6.1	Implementación Frontend	15
6.2	Implementación API	10
6.3	Implementación Análisis de memorias	25
6.4	Implementación Base de datos Vectorial	15
6.5	Implementación BD	3
7.1	Memoria	120
7.2	Defensa	10
Estimación Total		448

Figura 2.12: Tabla de tareas y estimaciones actualizada

2.4. Cronograma

El proyecto fue adjudicado el día 2 de noviembre de 2022 y tiene fecha de finalización el 29 de octubre de 2023.

Todas las tareas a realizar se compaginarán con un trabajo, por lo que no se podrá emplear un tiempo específico todos los días. Algunas tareas, como las reuniones, redacción y planificación del proyecto, se podrán llevar a cabo en paralelo. Sin embargo, otras tareas solo podrán realizarse después de que una tarea previa haya sido finalizada.

Se ha realizado un diagrama Gantt para representar visualmente las tareas a realizar en la franja de tiempo determinado, mostrando todos los paquetes de trabajo del EDT. En la Figura 2.13 se muestran las tareas desde Octubre de 2022 hasta Febrero de 2023, incluyendo las tareas iniciales propuestas, mientras que en la Figura 2.14 se presentan las tareas desde Junio hasta Octubre con los nuevos paquetes de trabajo actualizados.

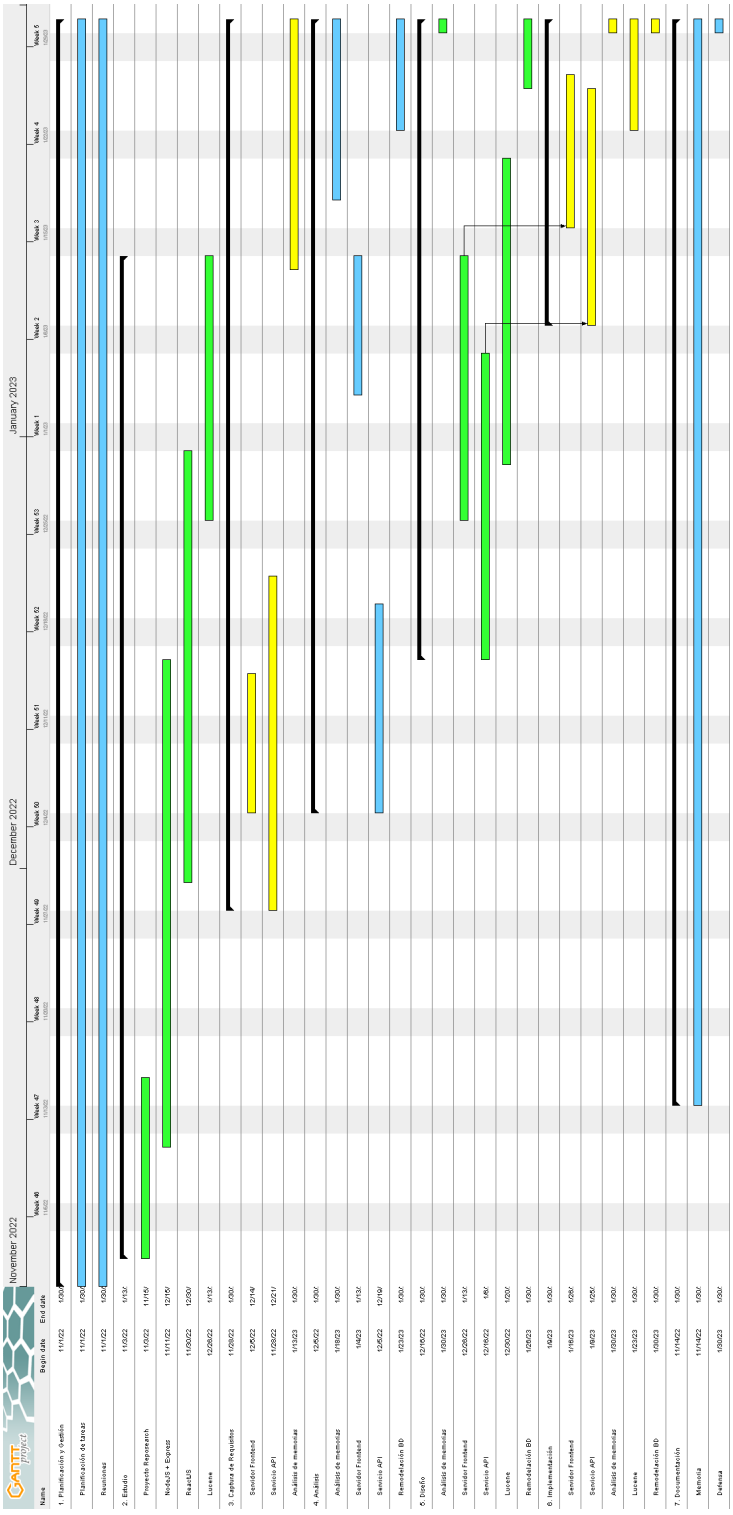


Figura 2.13: Planificación de las tareas desde Noviembre hasta Febrero

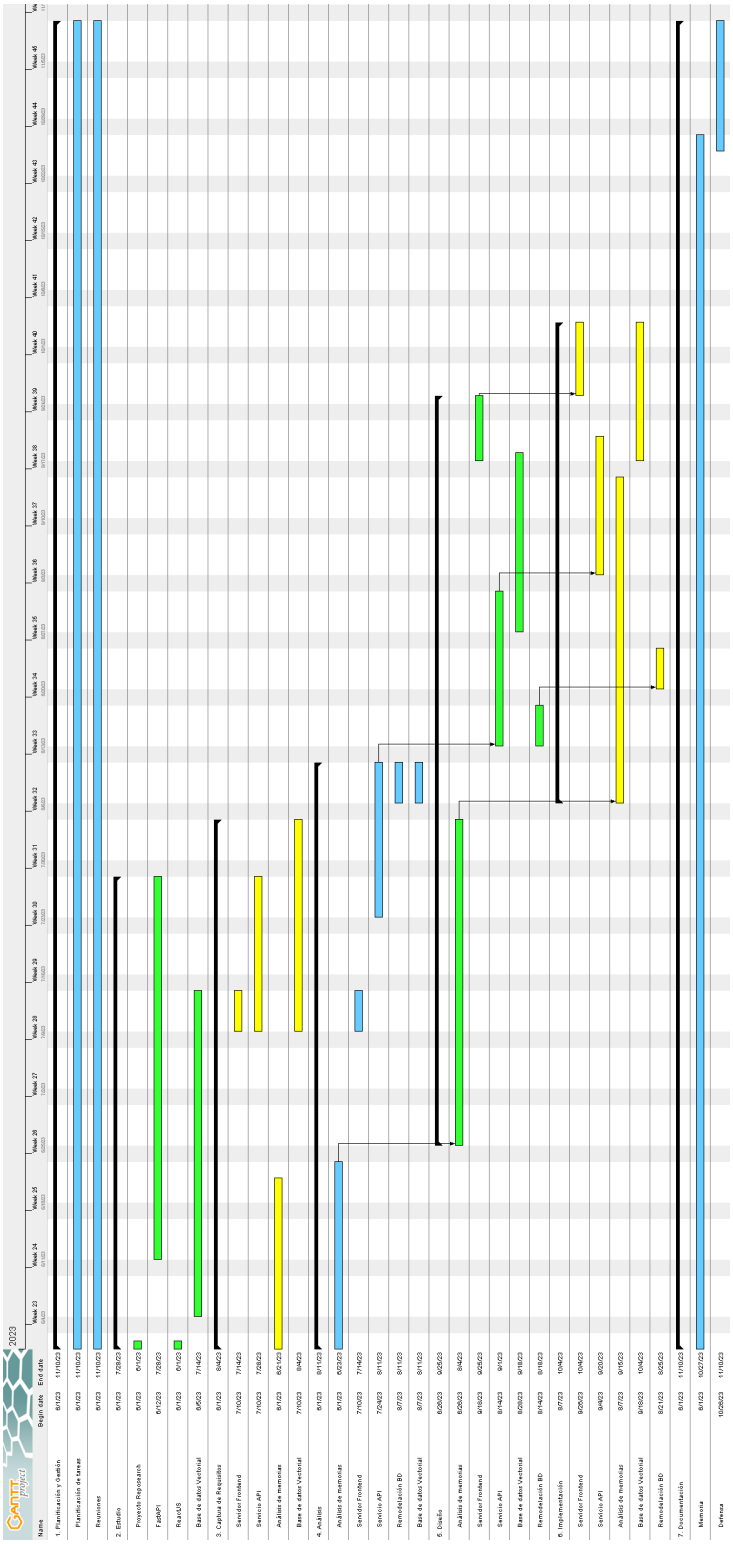


Figura 2.14: Planificación de las tareas desde Junio hasta Octubre

2.5. Gestión de riesgos

La gestión de riesgos en cualquier proyecto es esencial para identificar, evaluar y mitigar posibles obstáculos que podrían surgir durante el desarrollo(ISO [2018]).

En esta sección, se han identificado los posibles riesgos que podrían surgir durante la realización del proyecto, junto con las medidas planificadas para mitigar sus posibles consecuencias. A continuación, se detallan los riesgos identificados y las estrategias para abordarlos:

2.5.1. Enfermedad o lesión

Imposibilidad de continuar trabajando debido a enfermedad o lesión.

Prevención

- Mantener un estilo de vida saludable.
- Realizar descansos.
- Mantener una postura correcta.

Plan de contingencia

- Valoración por un médico de la lesión o enfermedad.
- Seguir las instrucciones del médico.

Probabilidad

- Lesión leve(3-5 días de baja) - probabilidad estimada 20 %
- Lesión grave(2 semanas o más) - probabilidad estimada 1 %

Impacto

- Lesión leve: 5 días x 3 horas/día = 15 horas. Impacto medio.
- Lesión grave: 14 días x 3horas/día = 42 horas. Impacto alto.

Riesgo

- Lesión leve: 20 % x 15 horas = 3. Riesgo bajo.
- Lesión grave: 1 % x 42 horas = 0,42 horas. Riesgo muy bajo.

2.5.2. Planificación incorrecta

Un posible riesgo es establecer un alcance demasiado ambicioso, estimación del tiempo de tareas imprecisa o modificación de los requisitos durante el desarrollo del proyecto.

Prevención

- Establecer márgenes de tiempo realistas. Evitar apretujar demasiadas responsabilidades en un período corto, permitiendo suficiente tiempo para completar las tareas de manera adecuada y sin prisas excesivas.
- Análisis minucioso para estimar el tiempo de las tareas de manera más precisa.
- Establecer metas alcanzables y realistas.

Plan de contingencia

- Limitar o reducir el alcance.
- Rehacer el alcance.
- Aumentar el tiempo de realización de las tareas.

Probabilidad

- Alcance ligeramente incorrecto - probabilidad estimada 70 %
- Alcance excesivamente incorrecto - probabilidad estimada 5 %

Impacto

- Alcance ligeramente incorrecto: 5 días x 3 horas/día = 15 horas. Impacto medio.
- Alcance excesivamente incorrecto: 30 días x 3horas/día = 90 horas. Impacto alto.

Riesgo

- Alcance ligeramente incorrecto: 70 % x 15 horas = 10,5. Riesgo alto.
- Alcance excesivamente incorrecto: 5 % x 90 horas = 4,5. Riesgo medio.

2.5.3. Incapacidad temporal

Incapacidad temporal debido a la necesidad de atención prioritaria del cuatrimestre universitario.

Prevención

- Definir un cronograma de los tiempos de ausencia.
- Fomentar la práctica del repaso regular del proyecto dedicando al menos una hora semanal.

Plan de contingencia

- En las épocas más lejanas de los exámenes, intentar proseguir con el proyecto.
- Llevar las materias al día para no tener que alargar el cuatrimestre en recuperaciones.

Probabilidad

- Paralizar el proyecto por cuatrimestre - probabilidad estimada 70 %

Impacto

- Paralizar el proyecto por cuatrimestre: (5 días x 4 semanas x 4 meses) x 3 horas/día = 240 horas. Impacto crítico.

Riesgo

- Paralizar el proyecto por cuatrimestre: 70 % x 240 horas = 168. Riesgo crítico.

3. Captura de requisitos

La captura de requisitos es el proceso de recopilación, análisis, definición y documentación de las necesidades y expectativas de los usuarios o clientes para un sistema o producto de software. Implica identificar y comprender lo que el sistema debe hacer y cómo debe comportarse desde la perspectiva de los usuarios y otras partes interesadas.

En este apartado se van a recoger todos los requisitos que requiere el proyecto. Se va a dividir en 3 paquetes grandes, divididos por funcionalidades, que son:

- **API**: En este apartado se recoge todo lo relacionado con el desarrollo de un servicio API, con la misión de enviar la información de la BD al servicio Frontend.
- **Frontend**: Este apartado recoge las característicias con las que el servicio mostrará la información al usuario.
- **Scripts**: Este es el paquete que más cambios va a requerir, y hace referencia a las funciones de búsqueda de trabajos de fin de grado de los repositorios, más las nuevas funcionalidades que incluirá.

3.1. Requisitos Funcionales

Los requisitos funcionales son especificaciones detalladas de las funciones y características que debe tener un software. Estos requisitos describen lo que el sistema debe hacer y están enfocados en las acciones que el sistema debe llevar a cabo en respuesta a ciertas entradas.

3.1.1. API

La API es la encargada de enviar la información en un formato *JSON* a los usuarios. Los usuarios recibirán cuatro tipos de respuesta, según las circunstancias:

- **Código de estado 200**: Cuando no ha ocurrido ningún error, responderá con la información solicitada.
- **Código de estado 404**: Cuando acceden a una ruta no existente, o al solicitar una página fuera de rango. El servidor responde con un mensaje acorde según la incidencia.
- **Codigo de estado 500**: Cuando el servidor contiene un error interno.
- **Codigo de estado 503**: El estado indicará que hay algún problema con alguno de los servicios, en este caso pueden ser dos, error con la base de datos relacional o error con la base de datos vectorial.

La información de respuesta proviene de dos bases de datos, según la petición del usuario. Una de ellas proviene de una base de datos relacional, donde se podrá filtrar exclusivamente por metadata, es decir, en busca de TFGs según título, autor, etc...

El otro tipo de información proviene de una *base de datos vectorial*, donde se busca semánticamente. En una búsqueda semántica, se intenta entender el significado detrás de las palabras que el usuario ingresa y proporciona resultados que están relacionados no solo directamente con las palabras clave, sino también con su significado conceptual. Devolverá resultados acorde a ellos.

Para obtener detalles más precisos, el usuario debe ingresar términos específicos como entrada. Ya que la API puede tratar con información sensible dentro de las BD, es necesario un mecanismo de seguridad, como prevención

ante intrusos. Para ello se require la validación de todos los terminos introducidos por el usuario.

En este proyecto, la API trata con muchos registros, por lo que será necesario devolver una parte de la información, para no abrumar al usuario con demasiada información, y para prevenir tiempos de respuesta prolongados(Bande [2019]).

Las respuestas enviadas deberán ser acorde a la petición del usuario, por lo que es crucial cumplir con todos los parámetros solicitados por el usuario.

Por último, para el proyecto actual, solo se implementarán las operaciones de lectura de las operaciones CRUD (Crear, Leer, Actualizar, Eliminar). Todos los terminos que requiere el usuario serán enviada por parámetro en la URL.

En este proyecto, no se solicitarán credenciales para la utilización de la API. Sin embargo, en la configuración del servidor existirá una lista blanca (*whitelist*) con los dispositivos de origen que podrán realizar solicitudes.

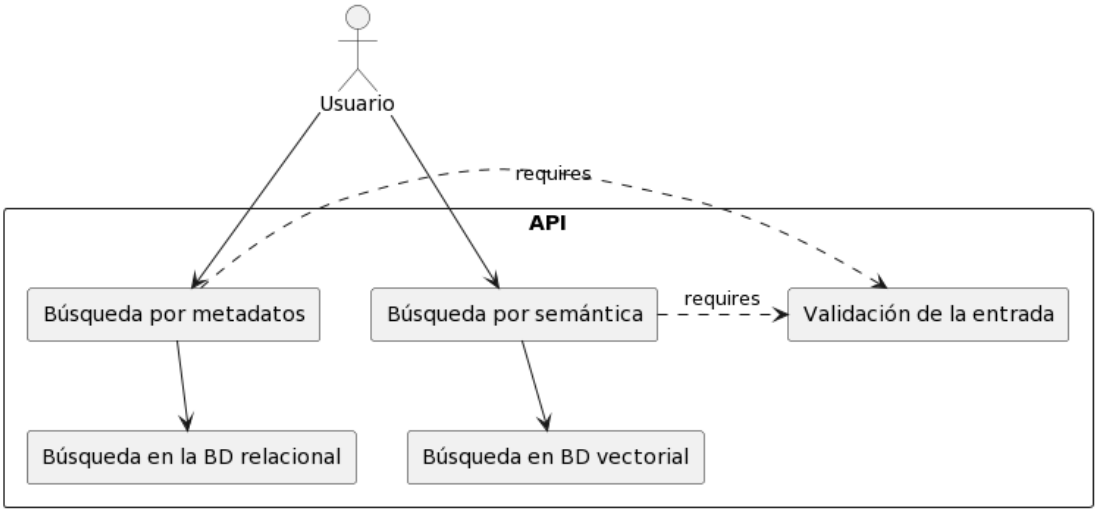


Figura 3.1: Caso de uso Usuario a API

3.1.2. Frontend

El servidor Frontend es el encargado de mostrar los trabajos de fin de grado, de manera estilizada. La propia interfaz debe brindar las herramien-

tas para hacer búsquedas más concretas.

La página actual de Reposearch tiene varios filtros, como puede ser por nombre, fecha, etc... Pero su uso es complicado cuando no se saben los valores concretos que hay que enviar por parámetro. La nueva versión de Reposearch contemplará todos los filtros disponibles, y desarrollará una sección específica para cada una de ellas. En esta nueva versión, será posible superponer múltiples filtros, lo que significa que se podrán buscar memorias por título y universidad específica, por ejemplo.

En la versión actual de Reposearch, solo se puede buscar por metadatos. En la nueva versión, se incluirá una búsqueda semántica. Para ello, habrá una separación entre ambos filtros: metadatos y semántica. Se creará una nueva sección específica para la búsqueda semántica.

Siempre que los usuarios tengan la capacidad de introducir valores y enviarlos al servidor backend, es una buena práctica realizar una validación del input. Para ello, cada campo específico que se envíe deberá tener ciertas restricciones; por ejemplo, si se trata de una fecha, debe seguir un formato específico y no deben permitirse letras, entre otras restricciones.

En la sección anterior se mencionó que el servidor API devolverá los datos paginados para evitar abrumar al usuario con una gran cantidad de trabajos de fin de grado. Además de la paginación, el servidor API también proporciona información adicional, como el total de memorias encontradas y el número total de páginas disponibles para la búsqueda. Por lo tanto, en el frontend se implementará una función de paginación. Debajo de la tabla de memorias, se indicará el número total de páginas disponibles para la navegación, y se proporcionará una interfaz gráfica para permitir a los usuarios moverse entre las páginas.

No se puede asumir que el servidor API funcione correctamente o tenga tiempos de respuesta muy rápidos, por lo que es necesario proporcionar un indicador para informar si se están buscando nuevas memorias o si el servidor está inactivo. Se mostrará un mensaje adecuado en cada caso.

El proyecto ha tenido muchos requisitos y cumplir con todos ellos superaba las 300 horas de desarrollo. Por ello, se han establecido restricciones y solo se ha desarrollado la versión para ordenador, sin ser responsive. Se ha optado por un estilo de tipo claro, y la versión oscura ha sido descartada.

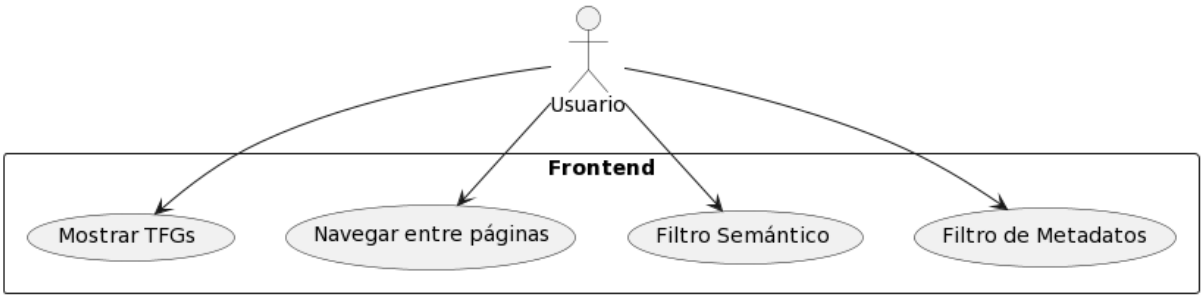


Figura 3.2: Caso de uso Usuario a Frontend

3.1.3. Scripts

Este paquete incluye las funciones de búsqueda de trabajos de fin de grado de los repositorios. Sobre este paquete base se va a desarrollar nuevas funcionalidades.

3.1.3.1. Refactorización

Al analizar el código de la versión inicial de Reposearch, lo primero que se pudo observar es la repetición de código en varios archivos. Se hace necesario refactorizar el código, extrayendo las partes comunes en archivos individuales, siguiendo los principios *SOLID*.

3.1.3.2. Análisis de disponibilidad de memorias de TFGs

Se pretende realizar un análisis de los trabajos de fin de grado, evaluando su disponibilidad real (algunos repositorios indican que disponen de la memoria, pero al intentar descargarla, indican que no está disponible, bien por deseo del autor o bien por otras razones). Para ello, será necesario verificar cada memoria manualmente y registrar su estado en la base de datos. Se desarrollará un nuevo script que realizará esta verificación y clasificará las memorias de manera adecuada. Será esencial definir una clasificación para todos los posibles estados en los que podría encontrarse una memoria.

En el proceso de verificación, la mayoría de los trabajos están disponibles en formato PDF, aunque no todos (algunas memorias están en formato ZIP, otras en formato RAR, etc.). Se necesita ampliar el rango de búsqueda, aunque no se podrá aceptar cualquier extensión de archivo.

También es común encontrar varios archivos junto con la memoria, como anexos y portadas. Es necesario identificar la memoria entre todos los archivos, en caso de que esté presente.

Para llevar a cabo un análisis preciso, será fundamental presentar la información de manera ordenada y concisa. Para este propósito, se desarrollará un nuevo script con el objetivo de obtener datos de la base de datos y generar un documento resumido y bien documentado con esa información.

3.1.3.3. Búsquedas semánticas

En este proyecto, se desea realizar búsquedas semánticas utilizando una base de datos vectorial donde se almacenan los *embeddings*. Los embeddings son representaciones numéricas de datos complejos, como el contenido de los PDF de las memorias, que han sido convertidos en vectores multidimensionales. Estos vectores capturan las relaciones y similitudes semánticas entre los datos originales(Espejel [2022]).

Para almacenar los embeddings, primero será necesario obtener los trabajos originales y guardarlos en el dispositivo donde se esté ejecutando el proyecto. Se requerirá un nuevo script con el objetivo de acceder a la URL de cada memoria y buscar la memoria entre todos los archivos para luego descargarla.

Una vez se tengan las memorias, es necesario generar los embeddings y luego insertarlos en la base de datos vectorial. Para generar los embeddings, se requiere un modelo de lenguaje que transforme el texto en vectores multidimensionales. Este mismo modelo de lenguaje se utilizará para realizar búsquedas dentro de la base de datos vectorial.

Todos estos scripts están diseñados para ejecutarse de manera conjunta y diaria. Por lo tanto, se ha optado por crear un único script que agrupe todas las funcionalidades en uno solo. Este script mostrará la sección que está ejecutando y, en caso de ocurrir algún error, indicará la naturaleza del error producido.

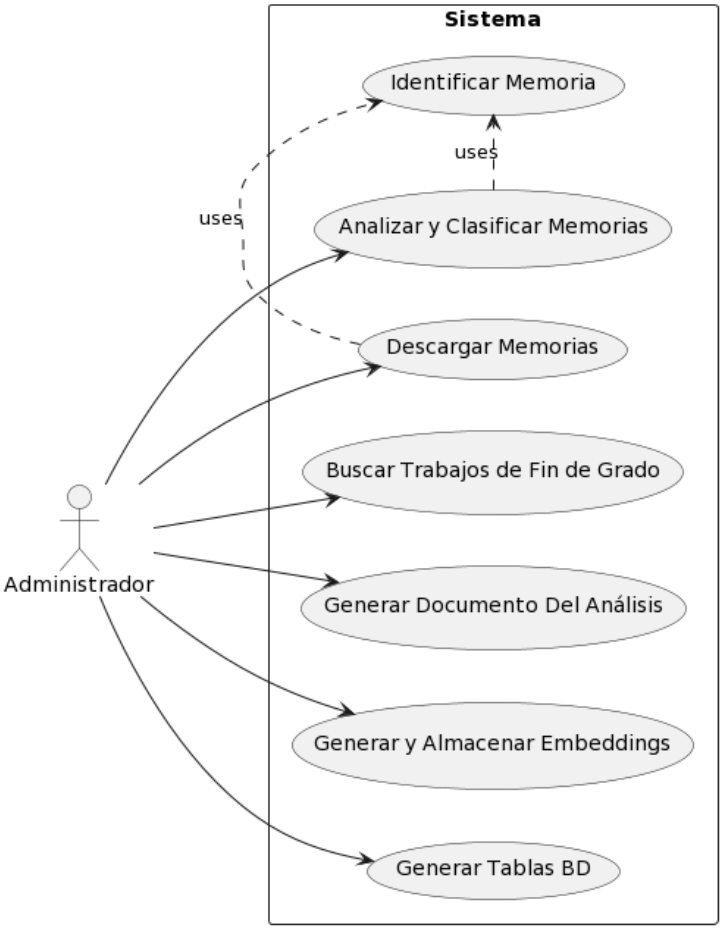


Figura 3.3: Caso de uso Usuario a Scripts

3.2. Requisitos No Funcionales

Los requisitos no funcionales son criterios o restricciones que especifican cómo debe ser el sistema, en lugar de lo que debe hacer. A diferencia de los requisitos funcionales, que se centran en las funciones específicas y comportamientos del sistema, los requisitos no funcionales se centran en las cualidades del sistema, como su rendimiento, seguridad, usabilidad y fiabilidad.

3.2.1. Seguridad en el API

En este proyecto, la API es actualmente un servicio de escala pequeña. Sin embargo, como se ha mencionado anteriormente, tiene el potencial de escalar a gran escala en el futuro.

Para ello, la API debe estar estructurada de manera adecuada para que en el futuro sea fácil agregar nuevas rutas y funcionalidades sin complicaciones.

En este proyecto, los datos que existen en la BD son no confidenciales, por lo que no será necesario un sistema de autenticación, como podrían ser credenciales. Pero como se hacen búsquedas en la BD con entradas del usuario, será necesario hacer validación de todo lo recibido, para evitar ataques de inyeccion SQL.

Por último, la API debe garantizar tiempos de respuesta reducidos para manejar cargas de trabajo intensivas. Se implementarán estrategias como la paginación y el procesamiento asíncrono para mejorar la eficiencia y la capacidad de respuesta del sistema.

3.2.2. Buen rendimiento en el Frontend

Respecto a los requisitos de la interfaz web, se definen los siguientes criterios:

La interfaz web debe ser sencilla y fácil de usar, priorizando la usabilidad y la accesibilidad para los usuarios. Los filtros deben estar organizadas de forma intuitiva, permitiendo a los usuarios realizar búsquedas sin confusiones.

Los tiempos de carga de la interfaz web deben ser mínimos. Esto implica optimizar el código, reducir las solicitudes HTTP y hacer uso mínimo de recursos multimedia para garantizar una carga rápida de la página.

3.2.3. Escalabilidad de lo Scripts

Los scripts de Python desempeñan un papel fundamental en el proyecto al interactuar con la base de datos para realizar operaciones de búsqueda, inserción y actualización.

Algunas funcionalidades pueden experimentar demoras debido a factores externos, como las solicitudes a enlaces de trabajos de fin de grado. En tales situaciones, es necesario actualizar la base de datos durante el proceso para reflejar los cambios en tiempo real.

En otros casos, se podría optimizar el proceso mediante inserciones y actualizaciones a granel(*bulk updates*), mejorando la eficiencia y minimizando el tiempo de ejecución de las operaciones.

Los scripts deben ser escalables y capaces de manejar un volumen creciente de datos y operaciones a medida que la aplicación se expande. Es esencial garantizar que los scripts sean eficientes y puedan adaptarse sin problemas a la creciente carga de datos, manteniendo un rendimiento óptimo incluso con un aumento en el número de registros.

Por último, se deberán tomar medidas de precaución en caso de que el servicio de la base de datos caiga. Los scripts deben estar diseñados para manejar situaciones de indisponibilidad del servicio de manera robusta, implementando mecanismos de manejo de errores para garantizar la integridad de la información.

4. Análisis y diseño

En esta sección, se analizarán cada una de las características o funcionalidades nuevas que se desean implementar. Se llevará a cabo un proceso de diseño detallado para resolver estas funcionalidades. Este diseño implicará la creación de diagramas de secuencia para mejorar la comprensión del sistema en su conjunto.

Para el análisis del proyecto, se volverá a separar en 3 secciones, API, Frontend y Scripts. Antes de comenzar con el análisis específico de cada grupo de trabajo, se va explicar como trabajará la aplicación completa en conjunto.

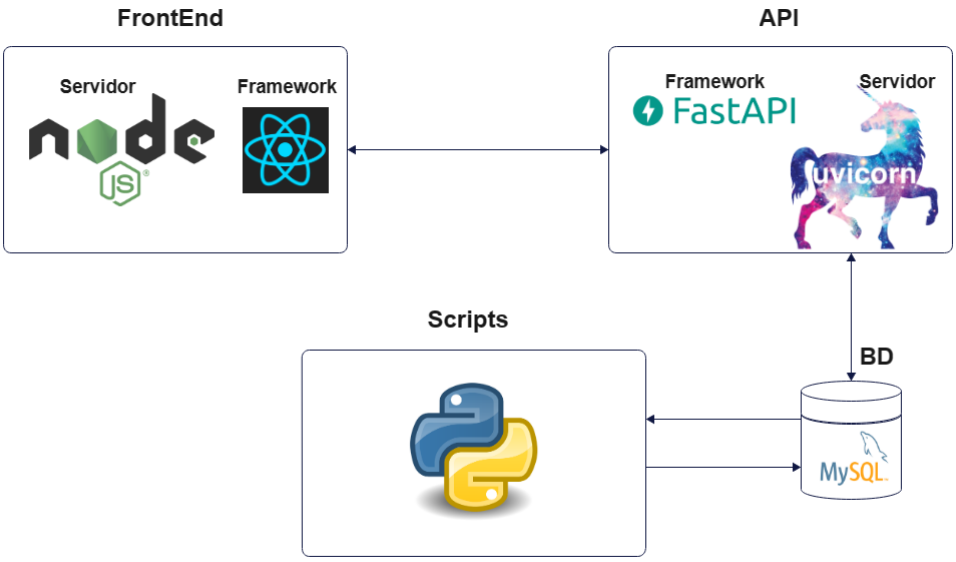


Figura 4.1: Arquitectura del proyecto

En la Figura 4.1 se aprecian los 3 grupos de trabajo funcionando en conjunto. El servicio Frontend será el encargado de mostrar la información de los TFGs al usuario. La información la solicitará al servidor API, y la API de-

volverá la información obtenida de la BD de MySQL. Los scripts de Python son los encargados de poblar la BD con nuevos TFGs, y también actualizará los valores de ciertas tablas como se verá más adelante.

El servicio Frontend y la API se ejecutan en servidores separados, y los scripts pueden ejecutarse manualmente o programarse para ejecutarse automáticamente a diario. Ahora que se tiene un entendimiento completo de cómo operan todas las partes en conjunto, a continuación se detallará cada sección de manera más exhaustiva.

4.1. API

Se va a desarrollar un servidor API con la cual enviar información de los TFGs al servicio Frontend, obteniendo la información desde una base de datos relacional y una base de datos vectorial. Se desarrollará la API desde cero, por lo que será necesario justificar las herramientas que se han utilizado y explicar de manera exhaustiva su funcionamiento, incluyendo restricciones y tipos de respuesta. En esta sección, se detallará el funcionamiento que tendrá la API y se definirán los datos de entrada y salida con los que operará.

Antes de adentrarnos en el análisis detallado del desarrollo, se explicará el funcionamiento de manera genérica.

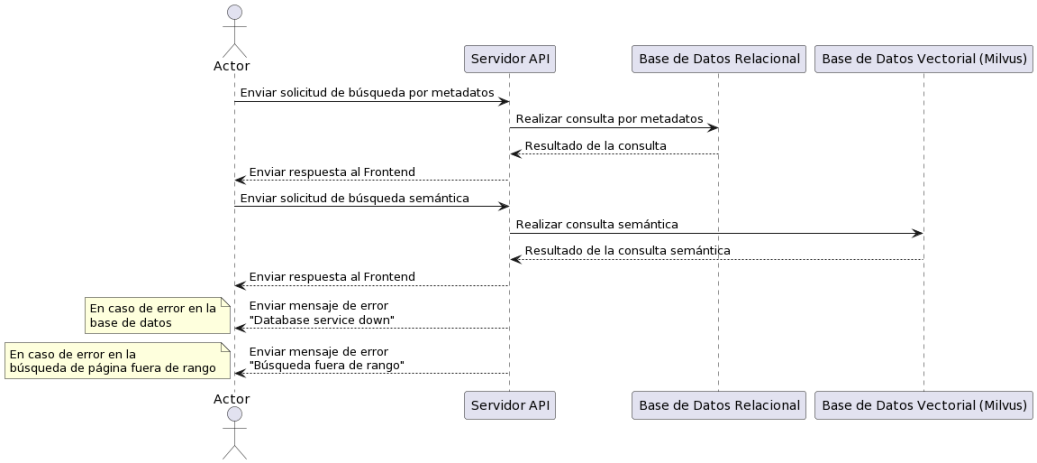


Figura 4.2: Diagrama de secuencia del funcionamiento de la API

En la Figura 4.2 se muestra el funcionamiento que se quiere obtener. Se pueden apreciar los dos tipos de búsquedas que se pueden realizar a la API,

cada una de estas obtienendo la información de distintas bases de datos. También se indican los posibles mensajes en casos de errores. Ahora se comenzará con el análisis.

En este proyecto, se desarrollará el servicio de API utilizando el lenguaje de programación Python. La elección de Python se basa en la necesidad de integrar una base de datos vectorial específica, en este caso, *Milvus*. Python ofrece bibliotecas y herramientas adecuadas para interactuar eficientemente con Milvus y realizar querys para las operaciones semánticas.

Otra de las ventajas de utilizar Python es la capacidad de reutilizar código de la parte de los scripts, ya que se emplea SQLAlchemy para las operaciones de búsqueda en la base de datos. Esto permite una integración más fluida y eficiente, optimizando el desarrollo y mantenimiento del proyecto.

Con el lenguaje de programación ya escogido, el siguiente paso será escoger un framework con el cual desarrollar la API. Los más conocidos para esta tarea son Django, Flask y FastAPI. Se ha escogido FastAPI por las siguientes razones:

- **Rendimiento**: FastAPI está diseñado para ser extremadamente rápido. Según la documentación oficial, capaz de competir contra NodeJS y Go.
- **Tipado estático y documentación automática**: FastAPI utiliza el tipado de Python para definir los tipos de entrada y salida de las API. Esto no solo ayuda a atrapar errores en tiempo de compilación, sino que también permite que FastAPI genere automáticamente documentación interactiva para la API basada en estos tipos de datos. Esto facilita la comprensión de la API y su uso.
- **Soporte para aplicaciones asincrónicas**: Es compatible con código asíncrono, lo que significa que puede manejar solicitudes asincrónicas de manera eficiente. Será útil para gestionar una gran cantidad de conexiones simultáneas.

Se ha seguido la documentación oficial, y FastAPI correrá sobre Uvicorn. Uvicorn es un servidor ASGI (*Asynchronous Server Gateway Interface*) para Python. Actúa como un servidor de desarrollo y producción para aplicaciones web basadas en ASGI.

Ya se tiene todo el marco de desarrollo escogido. Ahora es necesario decidir qué información se quiere devolver y qué tipos de parámetros se van a recibir. Lo primero será dividirlo por búsqueda, en este caso son dos, por metadatos o por semántica. Se desarrollará una ruta distinta para cada tipo de búsqueda.

4.1.0.1. Búsqueda por metadatos

Se empezará por la búsqueda de metadatos. Para la versión nueva de la API, se ha decidido devolver los mismos campos de la tabla de proyectos de la BD, como la versión original.

Figura 4.3: Información sobre los proyectos devuelta por la API

Se devolverán varias memorias, por lo que esta información se agrupará en una lista. Como se mencionó anteriormente, se proporcionará la información de manera paginada. A continuación, se muestra en la Figura 4.4 el esquema de los datos de respuesta de cada trabajo de fin de grado, y la respuesta paginada con toda la información.

```python
class ProjectResponseItem(BaseModel):
    id: int
    name : str
    author : str
    date : date
    university: str
    link : str

class PaginatedProjectResponse(BaseModel):
    totalMemories: int
    totalPages: int
    memories: List[ProjectResponseItem]
```

Figura 4.4: Respuesta de la API

Con el esquema de la respuesta de la API definido, es el momento de determinar los valores de entrada para realizar la búsqueda en la base de

datos. Se va a generar un filtro de búsqueda por cada columna de la tabla de respuesta, como se muestra en la Figura 4.3. Además, se realizarán pequeñas variaciones, como agregar tipos de operaciones para las fechas, dividir el autor en nombre y apellido, y especificar el tipo de orden.

```python
class ProjectRequest(BaseModel):
    name : Optional[str] = None
    author_name : Optional[str] = None
    author_last_name: Optional[str] = None
    dt : Optional[date] = None
    date_operator: Optional[DateOperatorEnum] = None
    universities: Optional[List[str]] = None
    order_by_column: Optional[OrderByColumnEnum] = None
    order_type: Optional[OrderTypeEnum] = None
```

Figura 4.5: Entrada de la API

En la Figura 4.5, se pueden observar varias enumeraciones. Esto se debe a que, para asegurar el correcto funcionamiento de la búsqueda, queremos restringir los datos de entrada a valores específicos. También se aprecia que falta la página a buscar, esta estará definida en la ruta.

```python
class DateOperatorEnum(str, Enum):
    greater_than_equal = ">="
    equal = "=="
    less_than_equal = "<="
    not_equal = "!="
```

Figura 4.6: Enumeración del tipo de operación sobre la fecha

Los problemas específicos que podemos encontrar en este tipo de búsquedas incluyen intentar buscar en una página superior a las existentes, lo cual se reduce a medida que se aplican filtros más específicos. Es necesario indicar el error, mostrando un mensaje acorde.

```python
class PageOutOfRange(NotFound):
    DETAIL = "Requested memory page out of range"
```

Figura 4.7: Error búsqueda página fuera de rango

La excepción mostrada en la figura 4.7 es una excepción específica para la búsqueda por metadatos, que hereda de la excepción genérica "Not Found".

```python
class DetailedHTTPException(HTTPException):
    STATUS_CODE = status.HTTP_500_INTERNAL_SERVER_ERROR
    DETAIL = "Server error"

    def __init__(self, **kwargs: dict[str, Any]) -> None:
        super().__init__(status_code=self.STATUS_CODE, detail=self.DETAIL, **kwargs)

class NotFound(DetailedHTTPException):
    STATUS_CODE = status.HTTP_404_NOT_FOUND
```

Figura 4.8: Error genérico

Una vez definidos los datos de entrada y salida, junto con las posibles excepciones, se mostrará el procedimiento que hará la API para la búsqueda por metadatos, como se muestra en la Figura 4.9

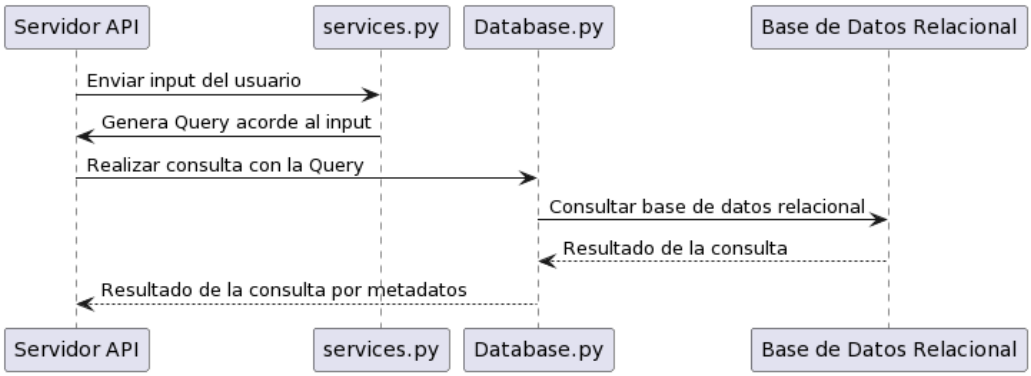


Figura 4.9: Diagrama de secuencia de la API a la búsqueda por metadatos

Para la búsqueda por metadatos, el servidor enviará todos los datos recibidos por el usuario a 'services', que será el encargado de identificar los valores recibidos y generar una consulta SQL acorde. Luego, esta consulta será devuelta a la API, que la ejecutará en la base de datos con la ayuda del archivo 'Database'. En este punto, se dispone de toda la información necesaria para enviarla al usuario.

4.1.0.2. Búsqueda Semántica

En esta subsección se analizará la búsqueda semántica. La respuesta de la API deberá ser la misma que se muestra en la figura 4.4. Sin embargo, la entrada del usuario variará. Para este tipo de búsquedas, el usuario proporcionará un único campo de tipo texto. Con la entrada del usuario, se

tranformará en un embeddings. Luego, se buscará en la base de datos vectorial los valores más similares para proporcionar la respuesta adecuada.

En la búsqueda vectorial, devolverá los identificadores de las memorias, y con ellas se realizará una búsqueda en la base de datos relacional para obtener toda la información necesaria para responder al usuario.

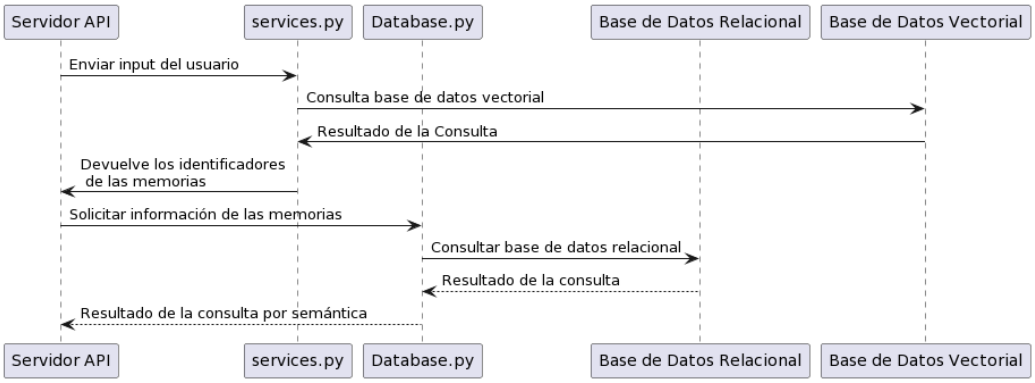


Figura 4.10: Diagrama de secuencia de la API a la búsqueda por semántica

Para las búsquedas semánticas, no se ha definido ninguna excepción específica. Ambos tipos de búsqueda comparten una excepción común, que es cuando el servicio de la base de datos no está disponible. En este caso, la API deberá devolver información adecuada al usuario.

```
class DatabaseDown(DetailedHTTPException):
    STATUS_CODE = status.HTTP_503_SERVICE_UNAVAILABLE
    DETAIL = "Database service down"
```

Figura 4.11: Error servicio BD caído

4.2. Frontend

Se desarrollará un servidor web que presentará los TFGs de manera estilizada al usuario. En esta sección se indicará el lenguaje de programación junto con las herramientas utilizadas para el diseño y se explicará el funcionamiento que desempeñará.

Para el desarrollo de la parte del Frontend, se hará con el lenguaje de programación JavaScript. JS es lenguaje más utilizado para el desarrollo web.

Se hará uso de la biblioteca ReactJS para el desarrollo de la interfaz web.

Para el desarrollo con React, es recomendable dividir la interfaz en múltiples componentes. Esto hace el desarrollo más sencillo, ya que permite crear funciones más simples y menos complicadas. Dividir la interfaz en componentes más pequeños y reutilizables facilita la gestión del estado, la lógica y la presentación, lo que mejora la mantenibilidad del código(Joshi [2021]).

Para este proyecto, donde no habrá muchos componentes, no será necesario utilizar bibliotecas de gestión del estado de los componentes. Habrá un componente padre encargado de compartir su estado con sus componentes hijos, y toda la lógica estará en el padre.

Se han definido los principales estados que deberá almacenar el componente padre:

- **Datos**: Este estado almacenará todos los datos que reciba por parte de la API. También será la información que se mostrará en la interfaz.
- **Página actual**: Se necesita hacer un seguimiento de la página en la que se encuentra.
- **Carga de datos**: Este estado indicará si se están buscando datos nuevos en la API.
- **Error de servidor API**: Este estado indicará si el servidor está funcional.
- **Orden**: En este estado se indica el campo por el cual se estará ordenando las memorias y el sentido del orden, ya sea ascendente o descendente. Por ejemplo, ordenar por título en orden descendente.
- **Estado de los filtros**: Es necesario almacenar los datos que está introduciendo el usuario para enviarlos a la API.
- **Tipo de búsqueda**: Este estado indicará el tipo de búsqueda que se estará haciendo.

Es necesario crear funciones para manejar los cambios en los estados. La funcionalidad de cada función dependerá del estado que se esté alterando.

Cuando los valores introducidos en los filtros por el usuario cambien, se actualizarán con los nuevos valores introducidos en lugar de conservar los

valores antiguos.

Cuando cambie el tipo de búsqueda, cambiará los filtros que podrá introducir el usuario.

Cuando cambie el orden de búsqueda, el servidor deberá enviar la nueva información a la API, y actualizará el estado de datos.

Siempre que se haga una búsqueda, el estado de carga será activado, indicando que se están buscando datos. Cuando se termine de realizar la búsqueda, el estado cambiará a desactivado.

Si al solicitar datos a la API, esta nos devuelve un estado de error, actualizaremos el estado a verdad para indicar que ha habido un error. Además, se mostrará un mensaje en la interfaz indicando que el servidor de la API tiene problemas.

Cuando se cambie de página, se solicitarán los datos a la API correspondientes a la nueva página actual.

Cuando se acceda a la página, se realizará una petición de datos a la API de forma automática.

En la Figura 4.12 se resume el procedimiento que tomará el Frontend teniendo en cuenta los puntos anteriores.

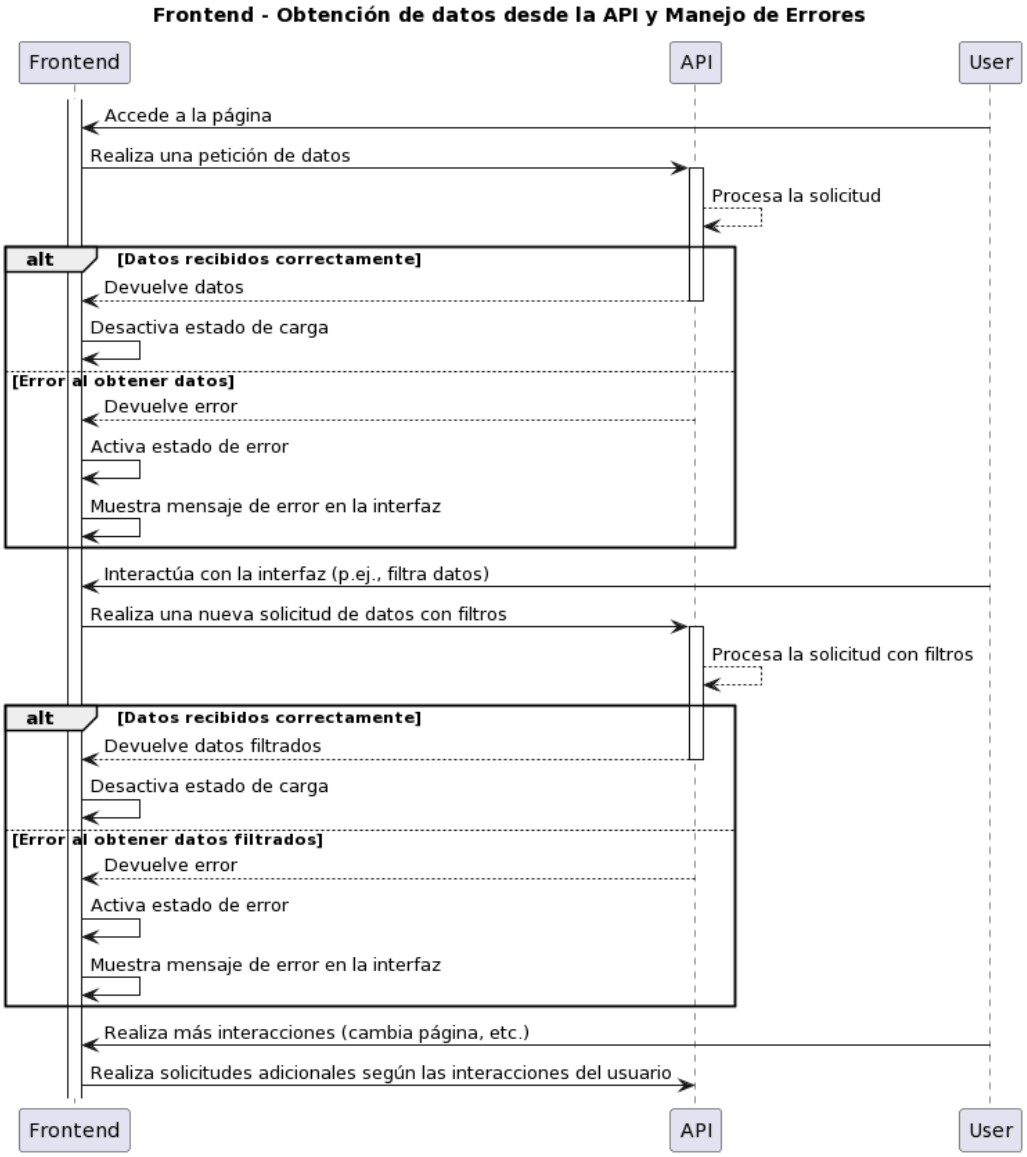


Figura 4.12: Diagrama de secuencia del funcionamiento del Frontend

4.3. Scripts

En este apartado, se reestructurarán los scripts encargados de buscar TFGs en los repositorios y almacenarlos en la base de datos. Sobre la nueva estructura, se implementarán nuevas funcionalidades y mejoras. El apartado se dividirá en dos subsecciones, cada una diseñará una funcionalidad nueva,

aunque esto puede requerir la creación de funcionalidades adicionales para su correcto funcionamiento.

En este proyecto, antes de empezar a analizar y diseñar nuevas funcionalidades, es necesario realizar una refactorización. Se require que no haya código duplicado, pudiendo reutilizar las mismas funciones en distintos scripts. Se iniciara generando una estructura más limpia, diviendo los scripts por funcionalidades.

Se definirá una carpeta común para el proyecto que incluirá funcionalidades a pequeña escala, como generar una sesión con la base de datos, textos constantes, fichero de configuración y un diccionario de diccionarios con todos los repositorios e información sobre estos. Este último punto será fundamental para la mayoría de las funcionalidades a desarrollar. Sobre este diccionario, se irán incluyendo nuevos repositorios que se quieran indexar e incluir las memorias de dicha universidad.

4.3.1. Análisis de disponibilidad

La primera funcionalidad mencionada previamente ha sido el análisis de disponibilidad de los trabajos de fin de grado. Antes de analizar y diseñar el script, es necesario definir la clasificación de las memorias. Las opciones posibles para los estados en los que se puede encontrar la memoria en la base de datos son las siguientes:

- **Sin verificar**: Aún no se ha verificado la memoria. Descripción en la BD 'Por verificar'
- **Disponible**: La memoria es libre y disponible. Descripción en la BD 'Licencia abierta y disponible'
- **No disponible**: La memoria no es accesible. Descripción en la BD 'Licencia abierta y no disponible'
- **Restricción solo para alumnos de la universidad**: La memoria está disponible pero tiene una restricción, solo es accesible por los miembros de la universidad. Descripción en la BD 'Licencia abierta con restricción, sólo para miembros de la universidad del autor'
- **Restricción por fecha**: La memoria estará disponible a partir de cierta fecha. Descripción en la BD 'Licencia abierta con restricción, sólo libres a partir de cierta fecha'

- **Restricción por demanda**: La memoria solo se podrá obtener cuando se solicite por correo a la universidad. Descripción en la BD 'Licencia abierta con restricción, sólo accesible bajo demanda'
- **Restricción solo para personal del repositorio**: La memoria solo está disponible para los administradores y personal de la universidad. Descripción en la BD 'Licencia abierta con restricción, sólo accesible para el personal del repositorio'
- **Restricción desconocida**: La memoria tiene alguna restricción, pero no se indica en la página de la universidad. Descripción en la BD 'Licencia abierta con restricción, por identificar'
- **Servidor caído**: Al momento de acceder a la memoria, el servidor estaba caído o se recibió un código de respuesta distinto de 200, por ejemplo 500 por mantenimiento. Descripción en la BD 'Servidor caido o respuesta con estado distinto de 200'

Se necesita almacenar esta información en la base de datos. En la base de datos actual, no existe ningún campo para almacenarla, por lo que se diseñará una nueva tabla para llevar un seguimiento de las memorias.

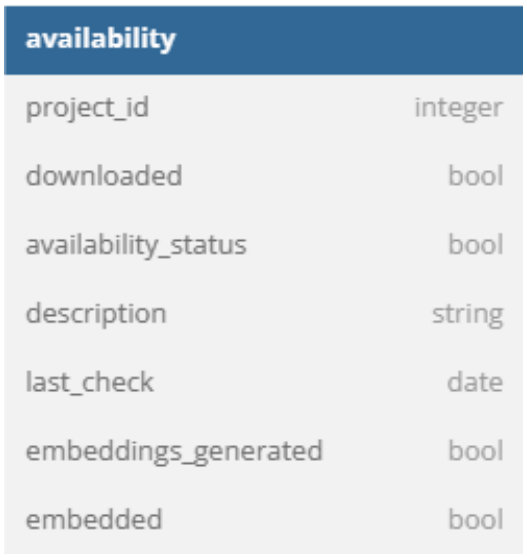


Figura 4.13: Tabla de disponibilidad

En la Figura 4.13 se puede apreciar un campo para el estado de disponibilidad y su descripción correspondiente. También existen un par de campos

adicionales para hacer seguimiento de las descargas de memorias y los embeddings. Más adelante se volverán a mencionar.

Se desea analizar el estado de muchas memorias, por lo que trabajar de forma síncrona no será la mejor opción. Para mejorar el tiempo de ejecución, será necesario trabajar de forma paralela. Para este proceso el diccionario de los repositorios será clave.

Se quieren hacer peticiones a varias memorias simultaneamente. Los servidores de las universidades pueden bloquear o rechazar ciertas IPs cuando se intentan realizar múltiples peticiones simultáneas para evitar caídas del sistema. Por ello hay que limitar el número de peticiones simultaneas por cada universidad.

Para el proceso, por cada universidad, es necesario obtener las memorias de dicha universidad que no han sido verificadas o que tienen restricción por fecha en la base de datos. Para cada universidad, se debe verificar las memorias, limitando el número de peticiones simultáneas. De esta manera, se podrían verificar un número X de memorias por universidad mientras se realizan peticiones a todas las universidades de manera simultánea.

Para añadir seguridad al proceso de verificación, se incluirá una función de reintento en las peticiones, en caso de que haya algún error por parte del servidor y rechace o corte la conexión.

En el proceso de verificación, el primer paso es buscar la memoria entre todos los archivos disponibles dentro de la universidad. Primero, es necesario verificar que el archivo tenga una extensión válida. Para este proyecto, se aceptarán ficheros en formato PDF y DOC. El siguiente paso es confirmar si el archivo es la memoria buscada. Para ello, se debe definir un filtro específico para cada universidad. En algunas universidades, el nombre del archivo puede ser suficiente, ya que incluye palabras clave como 'TFG' o 'PFG'. En otros casos, es posible buscar elementos que estén en el *DOM* para confirmar que se trata de la memoria correcta. La definición de estos filtros requerirá una revisión manual de cada repositorio.

El siguiente paso es verificar el estado de la memoria. Utilizando el enlace del elemento que definimos como memoria, se intentará acceder a ella mediante una nueva petición. Se requiere un estado válido, es decir, un código HTTP 200, y que el contenido pueda leerse sin errores. Si pasa todos los filtros hasta ahora sin errores, se considerará disponible.

Todo este proceso se realiza para verificar la disponibilidad de la memoria. En el caso de que no esté disponible, será necesario clasificarla correctamente en las categorías definidas previamente.

El proceso inicial es el mismo: buscar la memoria entre todos los archivos utilizando los filtros definidos previamente. Una vez seleccionado el elemento, es necesario establecer un nuevo filtro específico para cada universidad a fin de definir la restricción correspondiente. La mayoría de estos filtros se aplicarán buscando en el DOM, en busca de palabras clave que definan la restricción.

Para optimizar el proceso de actualización del estado de las memorias, en lugar de modificarlas a medida que se analizan, se anotarán los identificadores de cada memoria en una lista específica y se actualizarán los valores de manera masiva (bulk).

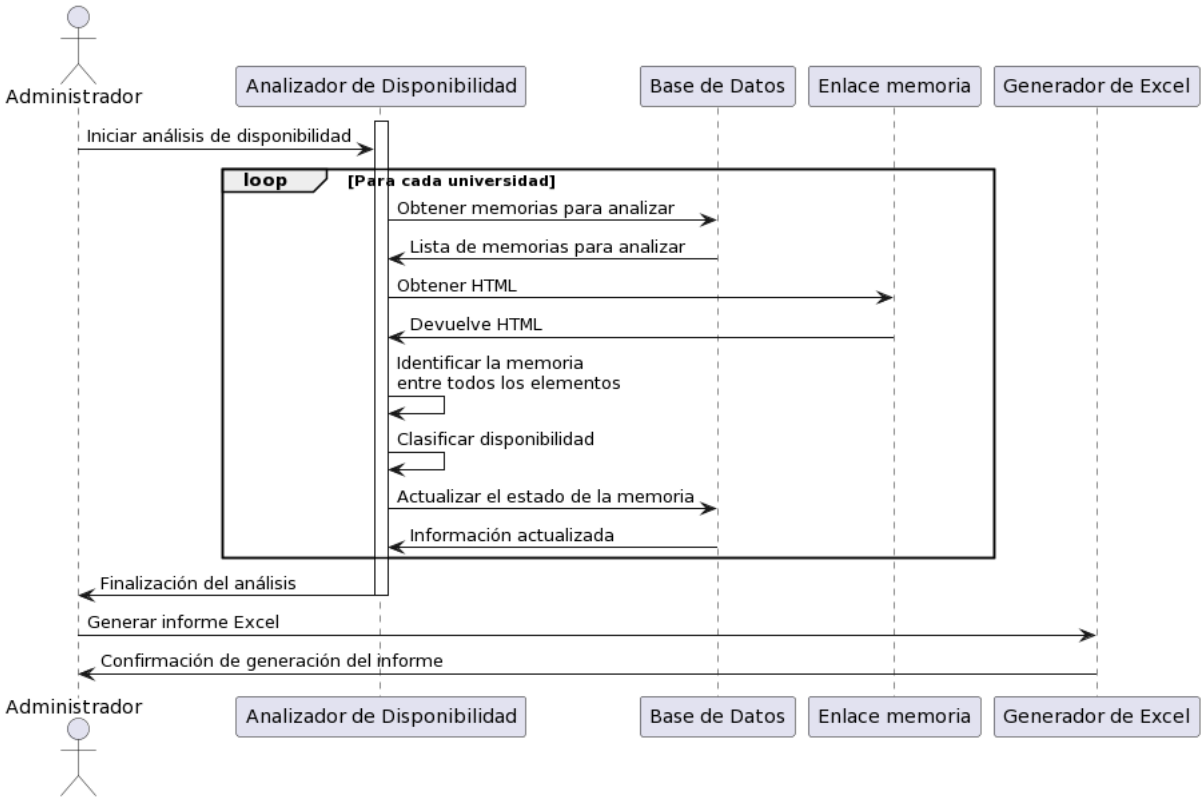


Figura 4.14: Diagrama de secuencia del análisis de disponibilidad

En la Figura 4.14 se resume el procedimiento de manera visual para facilitar la comprensión.

Para finalizar el análisis de la disponibilidad, se desea realizar un estudio detallado sobre este aspecto. El objetivo es presentar toda la información de manera resumida y organizada en un documento. Se ha decidido utilizar un archivo Excel para este propósito debido a sus funcionalidades integradas, como la capacidad de generar gráficos basados en datos agrupados. Otra ventaja significativa es la capacidad de manipular el archivo para crear nuevos gráficos, extrayendo únicamente la información deseada de la tabla principal.

Se diseñará un archivo Excel con un formato estilizado donde se insertará la información agrupada. A partir de esta información, se generará un gráfico explicativo. Los datos se organizarán por universidad y por tipo de restricción. Esto permitirá analizar, por universidad, cuántas memorias están disponibles, cuántas no están disponibles y cuántas tienen algún tipo de restricción específica, entre otros aspectos.

El proceso de inserción en Excel es sencillo. Para cada universidad, se obtendrá el total de memorias con una descripción específica y se asignará el valor obtenido de la base de datos. Estos datos se organizarán en el archivo Excel según las categorías definidas previamente, permitiendo así una visualización clara y estructurada de la información.

4.3.2. Búsqueda semántica

Falta la última funcionalidad por desarrollar, que es la generación de embeddings para la búsqueda semántica. El primer paso es obtener los trabajos originales para su transformación en embeddings.

Se va a desarrollar un script para buscar y descargar las memorias. La estructura que seguirá será similar a la del análisis de disponibilidad, pero variará ligeramente. El primer paso consiste en obtener, para cada universidad, las memorias que no estén descargadas y se consideren disponibles. Esta funcionalidad está pensada para ejecutarse después del análisis de disponibilidad. Acto seguido, se realizarán peticiones simultáneas y se verificará la memoria entre todos los archivos utilizando los mismos filtros desarrollados para el análisis de disponibilidad.

Una vez se obtenga la memoria entre todos los elementos, se procederá a descargarla y almacenarla en una carpeta específica, donde se almacenarán todas las memorias. El nombre de la memoria será identico al de su identificador. Se actualizará el estado en la BD de su descarga, verificando su existencia en el sistema de ficheros.

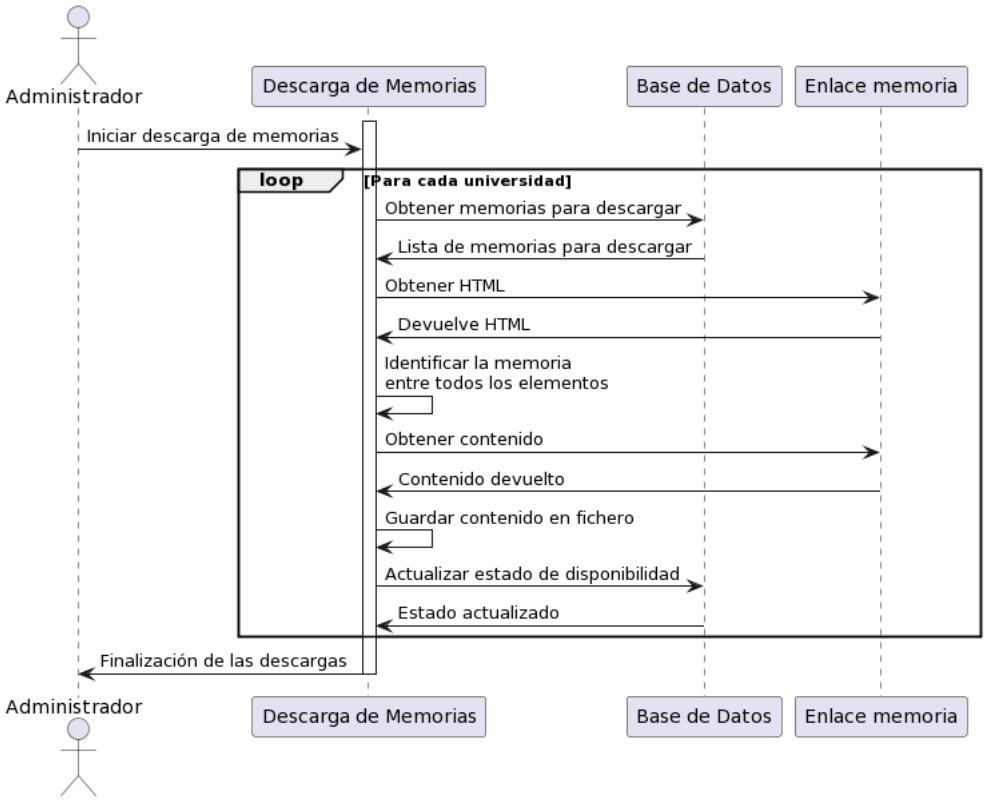


Figura 4.15: Diagrama de secuencia para la descarga de memorias

Con los trabajos ya almacenados en el sistema de archivos, se podrá dar inicio al desarrollo de la función que genere embeddings a partir de los archivos PDF.

Se seleccionarán los identificadores de los trabajos que han sido descargados y aún no se han generado los embeddings. En la Figura 4.13 se aprecia que en la base de datos las memorias tienen un estado para indicar si se han generado los embeddings, lo que no implica que estén insertados en la base de datos de Milvus.

Se iterará por la carpeta en la que estén todas las memorias descargadas, y solo se aceptarán las memorias que coincidan con el resultado devuelto por la base de datos. El proceso para generar embeddings es sencillo: se obtiene el texto del documento, se divide en trozos fijos (*chunk*), y se obtienen los datos que se necesitan. En este caso, se crearán dos listas: una con el contenido de la memoria y otra con los metadatos de cada chunk obtenido, como la página a la que se refiere, la memoria a la que pertenece y el número de chunk, entre

otros.

Con los textos obtenidos de los trabajos de fin de grado, se generarán los embeddings utilizando el modelo de lenguaje deseado, en este caso, el modelo bge-small-en [1]. Los embeddings resultantes se almacenarán en formato JSON como sistema de backup y se indicará en la base de datos que se han generado los embeddings para los respectivos trabajos. Posteriormente, se almacenará en Milvus tanto el contenido de la memoria como sus metadatos, y se marcará como incrustado (embedded).

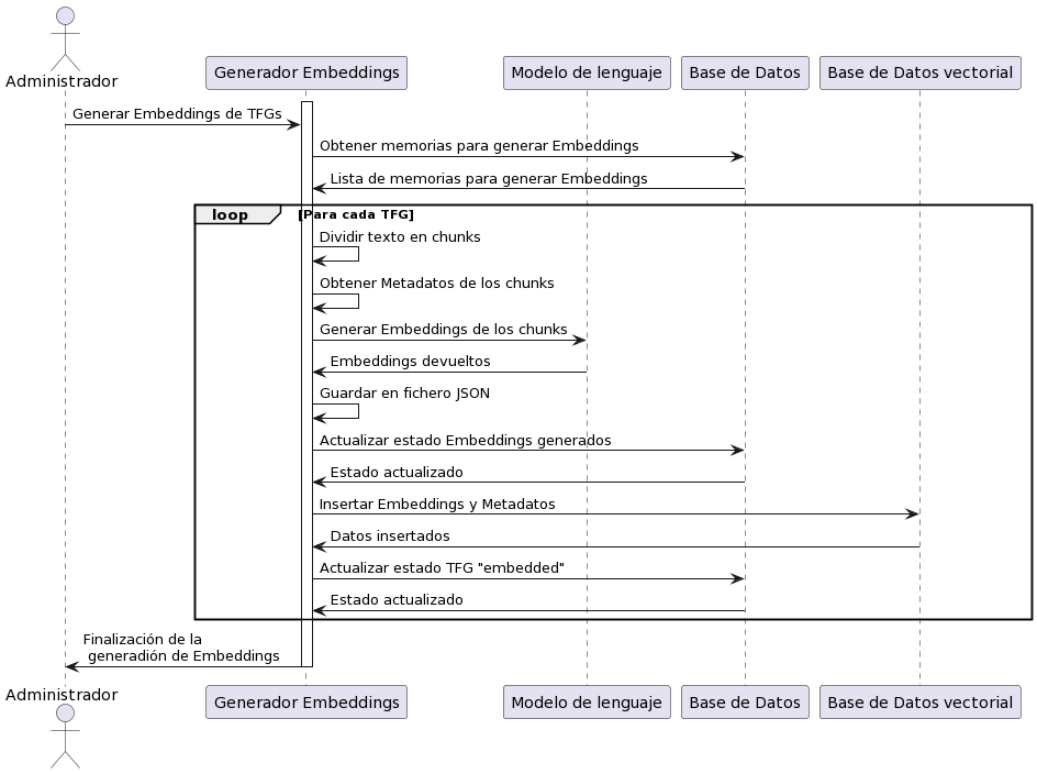


Figura 4.16: Diagrama de secuencia para generar Embeddings

[1] Enlace al modelo de lenguaje: https://huggingface.co/BAAI/bge-small-en-v1.5

5. Desarrollo

En esta sección, se detallarán los pasos seguidos para el desarrollo de todas las funcionalidades y mejoras, siguiendo el diseño establecido en la sección 4.

5.1. API

Para el desarrollo de la API, se han seguido las buenas prácticas descritas por el usuarios de GitHub Zhanymkanov(Yerassyl [2022]).

Se ha comenzado desarrollando la estructura del código. Como se muestra en la Figura 5.1, todo el código se encuentra dentro de la carpeta "src". A nivel superior, se encuentra el código para la configuración y puesta en marcha del servidor. Además, hay dos carpetas adicionales, cada una de las cuales contiene la lógica y las rutas para un tipo específico de búsqueda, es decir, búsqueda semántica y búsqueda por metadatos.

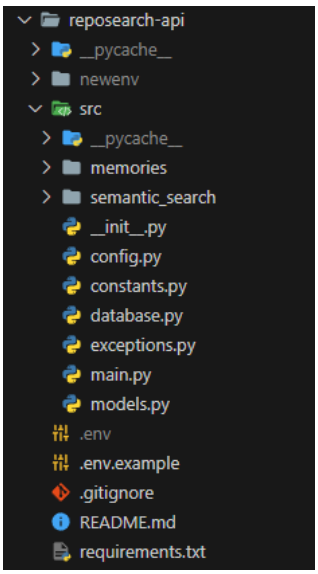


Figura 5.1: Estructura base de la API

Se iniciará por desarrollar la búsqueda por metadatos primero. Lo primero es definir las rutas que incluirá y los esquemas de entrada y salida definidos en las Figuras 4.5 y 4.4 en un fichero llamado "schemas.py".

Las rutas definen los parametros de entrada, el tipo de respuesta, y el código a ejecutar cuando son llamados.

```
1 @router.get("/", status_code=status.HTTP_200_OK, response_model=
    PaginatedProjectResponse)
2 async def fetch_memories(project_data: ProjectRequest, page:
    Optional[int] -> PaginatedProjectResponse:
3     query = await services.get_memomy_query(project_data)
4     return await paginate(query, page)
```

La lógica interna que se ejecuta dentro de la ruta se ha trasladado al archivo 'services.py' para hacer el código más legible. Siguiendo el diseño establecido, el método ejecutado en 'services.py' recibe todos los parámetros de la ruta y, basándose en ellos, genera la consulta para enviar a la base de datos utilizando SQLAlchemy. En la linea 5, devuelve toda la información al usuario paginada.

```python
class Paginator:
    def __init__(self, query: Select, page: int):
        self.query = query
        self.page = page
        self.per_page = 10
        self.limit = self.per_page
        self.offset = (page - 1) * self.per_page
        # computed later
        self.number_of_pages = 0


    async def get_response(self) -> dict:
        return {
            'totalMemories' : await self._get_total_count(),
            'totalPages': self.number_of_pages,
            'memories': [memory for memory in await  fetch_all(
                self.query.limit(self.limit).offset(self.offset))
                ]
        }

    def _get_number_of_pages(self, count: int) -> int:
        rest = count % self.per_page
        quotient = count // self.per_page
        return quotient if not rest else quotient + 1

    async def _get_total_count(self) -> int:
        count = await fetch_one(select(func.count()).select_from
            (self.query.subquery()))
        self.number_of_pages = self._get_number_of_pages(count)
        if self.page > self.number_of_pages and self.
            number_of_pages>0:
            raise PageOutOfRange()
        return count
```

La clase Paginator[1] está desarrollada en 'helper.py', y es el encargado de generar la respuesta al usuario, obteniendo todos los datos de la BD, y siguiendo el esquema de la Figura 4.4.

Para el desarrollo de la búsqueda semántica, la diferencia principal estará en los parámetros de entrada de la ruta, además de la lógica de la clase 'services.py'. La lógica del proceso para este tipo de búsquedas implica obtener los identificadores de las memorias que contienen o están relacionadas con la entrada del usuario. Luego, utilizando estos identificadores, se solicita la información a la base de datos relacional.

[1]Esta es la paginación por página, existen más tipos como la paginación por limit-offset o cursor(Tewouda [2023]).

```
34 def get_query_results(query):
35     hf = get_embedding_model()
36     vectorstore = get_vector_store(hf)
37     docs = vectorstore.similarity_search(query, k=10)
38     return [int(hit.metadata.get('metadata').get('memory')) for
           hit in docs]
```

En 'services.py' se obtiene el modelo de lenguaje, y se hace la petición a la base de datos de Milvus. De los resultados obtenidos se sacan los identificadores de las memorias, y con ellas en la clase Paginator se busca la información respectiva para responder al usuario.

Todas las búsquedas en la BD relacional se han hecho haciendo uso del fichero 'database.py' como se muestra en la Figura 5.1. En el siguiente código se muestra un método generico para solicitar la información a la base de datos.

```
39 async def fetch_all(select_query: Select ] Insert
       Update) -> list[dict[str, Any]]:
40     async with get_session() as session:
41         result: CursorResult = await session.execute(
               select_query)
42         return result.scalars().all()
```

En la Figura 5.2 se muestran los métodos para cada fichero que se han creado para el proceso del diseño.

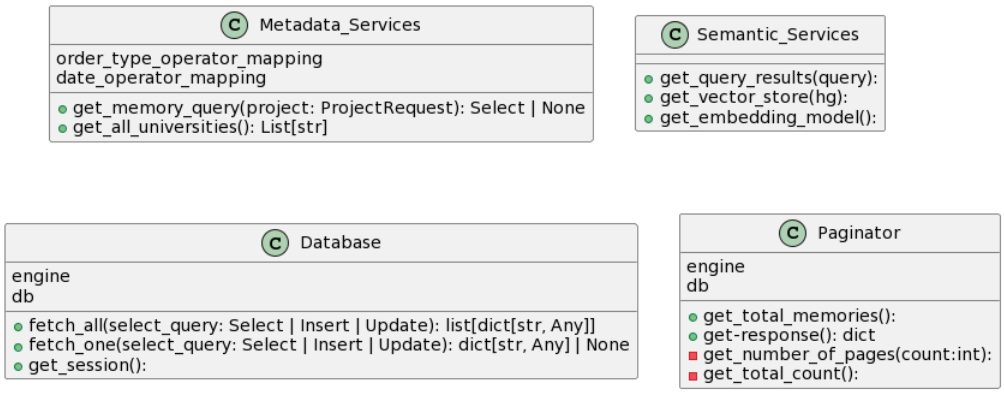


Figura 5.2: Diagrama de clases de la API

5.2. Frontend

Se va a generar el proyecto con la ayuda de NPM, encargado de generar una plantilla base a la cual se modificará el contenido.

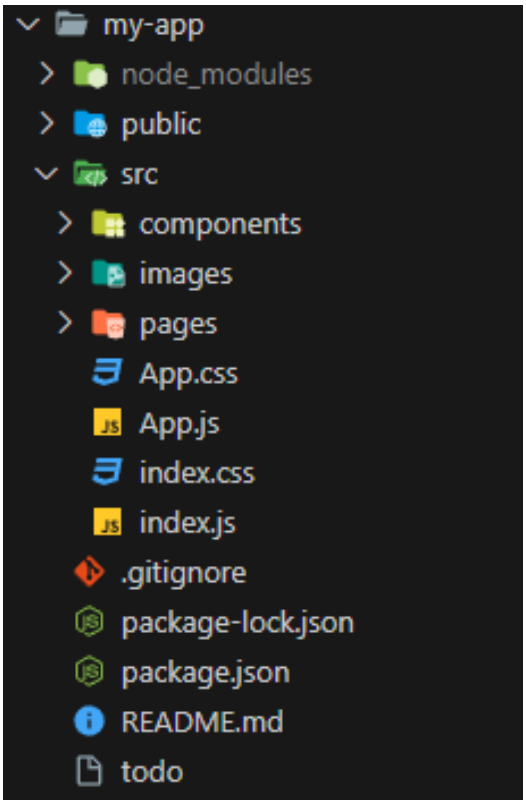


Figura 5.3: Estructura base del Frontend

En la Figura 5.3, se separa la parte 'pública', que incluye el favicon y el archivo HTML, de la parte lógica, colocándolos dentro de la carpeta 'src'. Todos los componentes se agruparán dentro de la carpeta 'components', y serán invocados desde la carpeta 'pages'. Para este proyecto, en 'pages' solo existirá un archivo que actuará como componente padre. En este archivo se desarrollará toda la lógica de estados y se enviarán dichos estados a los componentes hijos.

El buscador se ha dividido en dos componentes: el panel izquierdo, donde se muestran todos los filtros con los que el usuario podrá realizar la búsqueda, y el panel derecho, donde se muestran los trabajos.

En el panel izquierdo, se realiza un seguimiento de la entrada del usuario, y hay un botón de búsqueda para cuando el usuario decida usarlo. El botón realiza la siguiente función:

```
43 // function that fetches data from backend
44   async function getData() {
45     const url = getUrl()
46     setLoading(true)
47     fetch(url)
48     .then(function(response) {
49       if (!response.ok){
50         throw new Error("Failed with HTTP code " + response.
               status);
51       }
52       return response.json();
53   })
54     .then((json) => {
55       setServerError(false)
56       setData(json)
57       setLoading(false)
58
59     })
60     .catch(error => setServerError(true));
61   }
```

Siempre que se realicen búsquedas, se activa el estado de carga y se realiza la búsqueda. Si no ocurre ningún error o el código HTTP devuelto es 200, se desactiva el estado de error, se actualizan los datos de las memorias y se desactiva el estado de carga. En cambio si existe algún error, el estado del error se activa, y se mostrará el error en el panel derecho.

Siempre que se realicen búsquedas, cambios de página o alteraciones en el orden de los resultados, se llamará a la función getData() para realizar la búsqueda.

Un ejemplo de gestión de estados para los filtros de tipo texto es el siguiente.

```
62  // function to handle text input change
63   const onFilterChange = (event) =>{
64     const copiaState = {...filterState}
65     copiaState[event.target.name] = event.target.value
66     setFIlterState({...copiaState})
67   };
```

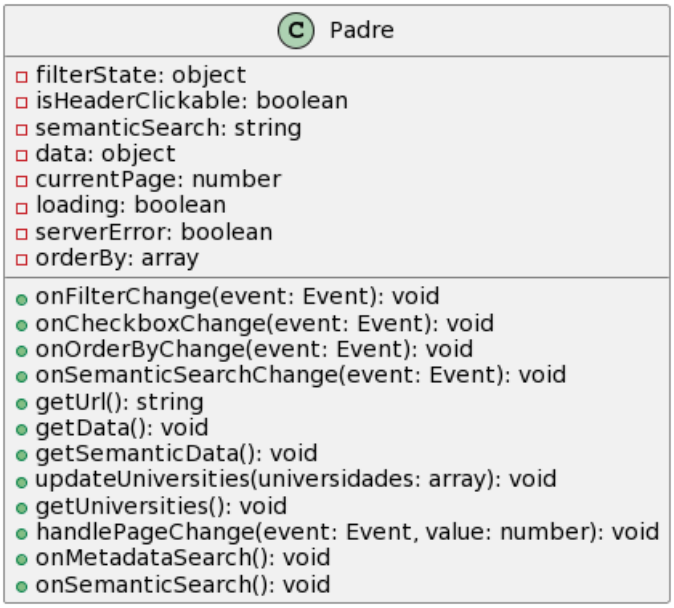


Figura 5.4: Diagrama de clases para el Frontend

En la Figura 5.4 se muestran todos los estados y las funciones necesarias para el correcto funcionamiento de la página web. Algunos de los métodos que aparecen son los que se pasan a los hijos para la gestión de los estados.

5.3. Scripts

La primera tarea antes de comenzar con el desarrollo de las nuevas funcionalidades, es reorganizar la estructura del proyecto, refactorizando el código existente y diviendo la lógica por funcionalidades.

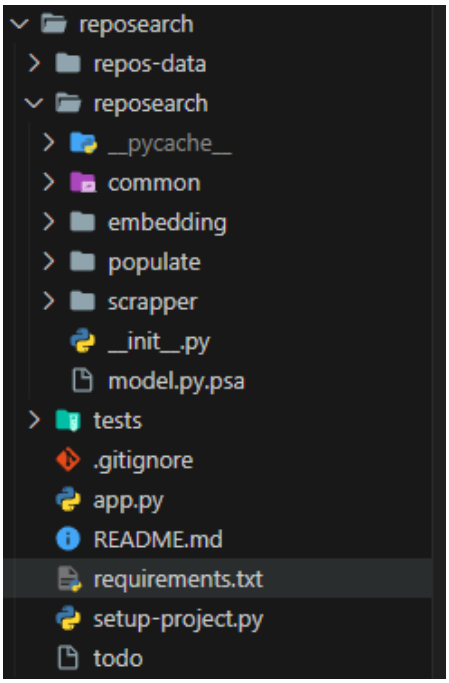


Figura 5.5: Estructura base de los Scripts

Se va a definir en la carpeta 'common' un diccionario con las universidades, que contendrá información relevante sobre cada una. Este diccionario será clave para el desarrollo de las funcionalidades, ya que contiene selectores de búsqueda de memorias con XPATH, así como para la búsqueda de memorias con el protocolo OAI-PMH.

```
68 repos = {
69     "upm": {
70         "id": 1,
71         "description": "upm",
72         "librarysoft": "ePrints",
73         "domain": "https://oa.upm.es/cgi/oai2",
74         "lastupdate": startingDate,
75         "sets": ["747970653D6F74686572"],
76         "requiredbasedomain": False,
77         'selector': '#abstract > div.ep_block.fulltext > table
               tr > td.text:not(:has(>strong)) >  em.requires_pdf ~
```

```
            a.ep_document_link',
78          'selector-constrain' : '#abstract > div.ep_block.
               fulltext > table tr > td.text > strong:has(~ em.
               requires_pdf) ~ a',
79          'single-file-filter': True,
80      },
81      "upvehu": {
82          "id": 2,
83          "description": "upvehu",
84          "librarysoft": "DSpace",
85          "domain": "https://addi.ehu.eus/oai/request",
86          "lastupdate": startingDate,
87          "sets": ["col_10810_2017", "com_10810_13137"],
88          "requiredbasedomain": True,
89          'selector': '#aspect_artifactbrowser_ItemViewer_div_item
               -view  a:has(>i.glyphicon.glyphicon-file)',
90          'selector-constrain' : None,
91          'single-file-filter': True,
92      },
93      ...
```

5.3.1. Análisis de disponibilidad

Se va a seguir el diseño hecho en la sección 4, y el proceso es:

Por universidad obtener las memorias que se van a verificar, filtrar la memoria entre todos los documentos, y clasificarlo en una categoría.

```
94  tasks = (check_university_memories(ri) for ri in repos.keys())
95  await asyncio.gather(*tasks)
```

Se comenzará iterando todas las universidades y ejecutando la función check-university-memories() utilizando programación asíncrona, permitiendo analizar las memorias de todas las universidades en paralelo.

El siguiente paso es por universidad, obtener las memorias a verificar, y analizarlas, limitando el número de peticiones simultaneas.

```
96  async def check_university_memories(ri):
97      memories = await fetch_database_university_memories(ri)
98      my_timeout = aiohttp.ClientTimeout(sock_connect=
            SOCKET_CONNECTION_TIME, sock_read=SOCKET_READ_TIME)
99      async with aiohttp.ClientSession(timeout=my_timeout) as
            csession:
100         semaphore=asyncio.Semaphore(MAX_CONCURRENT_TASKS)
101         print("Checking memories from:", ri)
```

```
102         tasks = (search_and_check(Memory_Info(memo.id, memo.link
                , ri,csession, semaphore, memo.description)) for memo
                 in memories)
103         await asyncio.gather(*tasks)
```

Para la clasificación de la memoria se ejecuta la siguiente función:

```
1  async def search_and_check(MemoClass):
2      has_element = await normal_selector(MemoClass)
3      if has_element: return
4      has_costrain = await constrain_selector(MemoClass)
5      if has_costrain: return
6      if MemoClass.description != NODISPONIBLE:
           memorias_no_disponibles.append(MemoClass.id)
```

El proceso es el siguiente: primero, se verifica si la memoria está disponible. Si lo está, se termina el proceso. Si no está disponible, se comprueba su restricción. Si no se encuentra ninguna restricción, se clasifica como 'no disponible'.

Antes de clasificar la memoria, es necesario filtrarla entre todos los documentos disponibles. Utilizando el diccionario de universidades, se obtiene el XPATH específico de la universidad. Se filtran los archivos de memoria seleccionados por el XPATH y se iteran para encontrar la memoria.

```
1  html = await fetch(MemoClass=MemoClass)
2  new_links = parse(html, ri)
3  for link in new_links:
4      url = link.get('href', "")
5      file_extension = obtain_file_extension(url.lower())
6      if file_extension in VALID_FILE_EXTENSIONS and (repos[ri]["
           valid-filter"](link) or (repos[ri]["single-file-filter"]
           and len(new_links)==1)):
7                       CODIGO específico
```

En la línea 6, se verifica su extensión y se aplica el filtro específico de universidad. El proceso para filtrar la memoria entre los documentos es idéntico tanto cuando no existen restricciones como cuando las hay. La variación en el proceso ocurre una vez que la memoria ha sido identificada. Cuando no hay restricción, lo único que se verifica es la respuesta 200 de HTTP y ningún error de lectura de la memoria. En el caso de que existan restricciones, se llama a la función específica de universidad para su clasificación, buscando palabras clave en el DOM. Vease el siguiente ejemplo de filtro para la universidad ULPGC.

```
8  async def Culpgc(element, MemoClass):
9      restriction = element.find_next_sibling('span').text.lower()
           .strip()
```

```
10      if restriction == 'inicia sesion para acceder':
11          if MemoClass.description != RESTRICCION_UNIVERSIDAD:
                memorias_restriccion_universidad.append(MemoClass.id)
12      else:
13          if MemoClass.description != RESTRICCION_DESCONOCIDA:
                memorias_restriccion_desconocida.append(MemoClass.id)
```

Como se aprecia en el ejemplo, se incluye el identificador de la memoria a una lista específica para realizar una actualización batch.

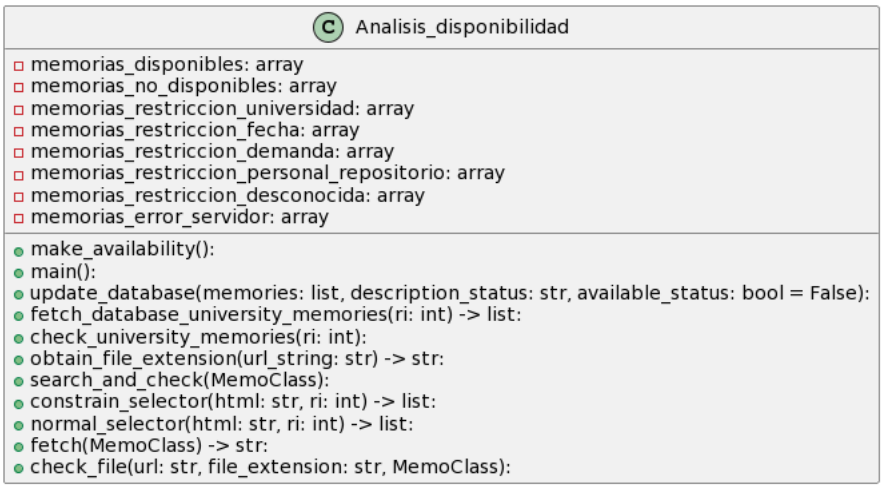


Figura 5.6: Diagrama de clases para el análisis de disponibilidad

En la Figura 5.6, los filtros para identificar las memorias no están representados entre los elementos, ya que estos filtros están contenidos en un archivo separado.

5.3.2. Búsqueda semántica

Para generar los embeddings para la búsqueda semántica, se requiere obtener los trabajos originales para su transformación. Se comenzará con la implementación de esta funcionalidad.

El procedimiento es muy similar al del análisis de disponibilidad. Por universidad se obtienen las memorias que se quieren descargar y se filtra la memoria entre todos los archivos utilizando los mismos filtros desarrollados para el análisis de disponibilidad. Cuando se obtiene la memoria, se prodece a descargar con el nombre del identificador en la BD.

La función de descarga tiene un decorador (@request-action) con el objetivo de reintentar el procedimiento en caso de que haya algún error. En caso de que falle por leer el contenido de la memoria, se indica en la BD que la memoria no está disponible, a pesar de que esté disponible en la página en la que se encontraba la memoria. En caso de que falle por un estado distinto de 200, se indicará en la BD como servidor caído.

La función decoradora es la siguiente:

```
14 def request_action(func):
15     @functools.wraps(func)
16     async def wrapper(*args, **kwargs):
17         result = None
18         for _ in range(NAX_REQUEST_RETRIES):
19             try:
20                 result = await func(*args, **kwargs)
21                 if result is not None:
22                     break
23             except Exception as e:
24                 print('Error while requesting (retrying):', e)
25                 # Wait a bit before retrying (you can adjust the
                       delay)
26                 await asyncio.sleep(2)
27         else:
28             clase = kwargs.get('MemoClass', None)
29             if clase is not None:
30                 await update_row_error(clase.id)
31                 error = 'error al hacer la request con ID: ' +
                       str(clase.id)
32                 print(error)
33                 log_error(error)
34         return result
35     return wrapper
```

Incluye un mecanismo básico para el registro de errorres.

Figura 5.7: Diagrama de clases para la descarga de memorias

Con las memorias ya descargadas, se podrá desarrollar la función que genera los embeddings a partir de los PDFs y los inserta en la base de datos de Milvus.

El proceso es bastante sencillo, se iterará por la carpeta en la que estén todas las memorias descargadas, y solo se generarán los embeddings de las memorias que estén incluidas en la lista con los identificadores de las memorias que no se han generado embeddings y estén descargadas.

Se procederá con los pasos de la generación de embeddings. Se obtendrá el contenido de la memoria, se dividirá en trozos fijos, y por cada trozo se obtendrá la metadata y el equivalente al texto en embedding. La función que divide el contenido en trozos y obtiene su metadata es la siguiente:

```
36 def get_file_text_and_metadata(file):
37     try:
38         name_without_extension = file.split('.')[0]
39         loader = PyPDFLoader(storage + file)
40         data = loader.load()
41         text_splitter = RecursiveCharacterTextSplitter(
               chunk_size=1000, chunk_overlap=0)
42         texts = text_splitter.split_documents(documents=data)
43         record_texts = [item.page_content for item in texts]
44         record_metadatas = [{
45             "chunk": j, "text": text.page_content, 'page': text.
                   metadata['page'], 'memory':name_without_extension
                   } for j, text in enumerate(texts)]
46         return record_texts,record_metadatas
47     except Exception as e:
48         print("cant decode")
49         print(e)
```

```
50          return [],[]
```

Devuelve dos listas, los textos troceados y los metadatos de los trozos. En caso de que ocurra algún error en el procedimiento, se devolverán dos listas vacias, indicando que no se ha podido trozear/generar su metadata.

El siguiente paso es generar los embeddings con la ayuda del modelo de lenguaje. Se le pasarán los textos y devolverá una lista con los embeddings.

```
51 def file_to_embedding(texts):
52     model_name = "BAAI/bge-small-en-v1.5"
53     model_kwargs = {'device': device}
54     encode_kwargs = {'normalize_embeddings': True}
55 
56     hf = HuggingFaceBgeEmbeddings(
57         model_name=model_name,
58         model_kwargs=model_kwargs,
59         encode_kwargs=encode_kwargs
60     )
61 
62     embedding = hf.embed_documents(texts)
63     return embedding
```

En el siguiente paso se comprobará que existen embeddings, y si existen actualizará la BD para indicar que se han generado los embeddings de dicha memoria, y se guardarán en formato JSON a medio de backup. El siguiente paso es introducir los embeddings + metadata + texto original en Milvus con la siguiente función:

```
64 def data_chunk_insert(texts, metadata, embeddings, collection):
65     data_batch = [[], [], []]
66     for i in range(len(texts)):
67         data_batch[0].append(texts[i])
68         data_batch[1].append(metadata[i])
69         data_batch[2].append(embeddings[i])
70         if len(data_batch[0]) % batch_size == 0:
71             embed_insert(collection, data_batch)
72             data_batch = [[], [], []]
73     # Embed and insert the remainder
74     if len(data_batch[0]) != 0:
75         embed_insert(collection, data_batch)
```

Se irán introduciendo los datos por partes, ya que Milvus tiene un límite de inserción, y se podría desbordar en caso de intentar introducir un trabajo de fin de grado de una vez. Una vez insertado, se actualiza en la BD como embedded(Figura 4.13).

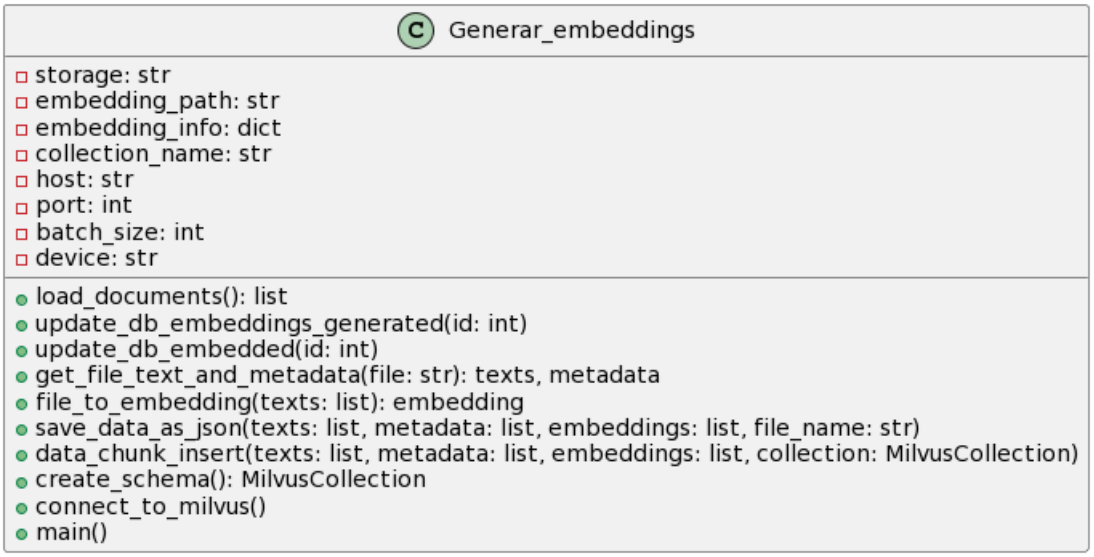


Figura 5.8: Diagrama de clases para la generación de embeddings

5.3.3. Script General

Está pensado en ejecutarse estas funcionalidades a diario, por lo que se ha desarrollado un último script, con el fin de automatizar el proceso. Este script llama a todas las funcionalidades descritas hasta ahora, en un orden específico, indicando en todo momento el proceso que está ejecutando. El orden es el siguiente: buscar memorias en los repositorios institucionales, analizar su disponibilidad posteriormente, descargar las memorias que estén disponibles y, finalmente, generar los embeddings e insertarlos en Milvus.

6. Resultados

Todo el código desarrollado se puede encontrar en GitHub.[1] Para probar el código, es necesario un almacenamiento inicial elevado (entorno a 40GB) ya que se almacenan los embeddings de los TFGs en formato JSON con el fin de ahorrar tiempo de transformación con el modelo de lenguaje.

En este apartado se mostrarán los resultados de ejecutar las funcionalidades desarrolladas.

6.1. API

En la sección de la API, se verificará que el resultado devuelto coincida con lo que se espera recibir. En la Figura 4.4 se indica que debe devolver tres variables en formato JSON: el total de memorias, el total de páginas y una lista de memorias.

Se va a realizar una solicitud a la API desde el navegador, y el resultado se muestra en la Figura 6.1. El resultado devuelto concuerda con el esquema especificado.

[1]Link al proyecto: https://github.com/AnderFernandez156/reposearch El tribunal puede solicitar permiso de acceso si lo requiere

Figura 6.1: Resultados devueltos por la API

Se pueden introducir todos los filtros(Figura 4.5) por parámetro, pero cuando se introducen valores erroneos para los parámetros, la API deberá devolver un mensaje acorde. En la Figura 6.2 se muestra el error cuando se envía un texto a un parámetro de tipo fecha.

Figura 6.2: Parámetros incorrectos

Figura 6.3: Error en el servicio de base de datos

En la Figura 6.3 se muestra la respuesta cuando el servicio de base de datos está caído.

6.2. Frontend

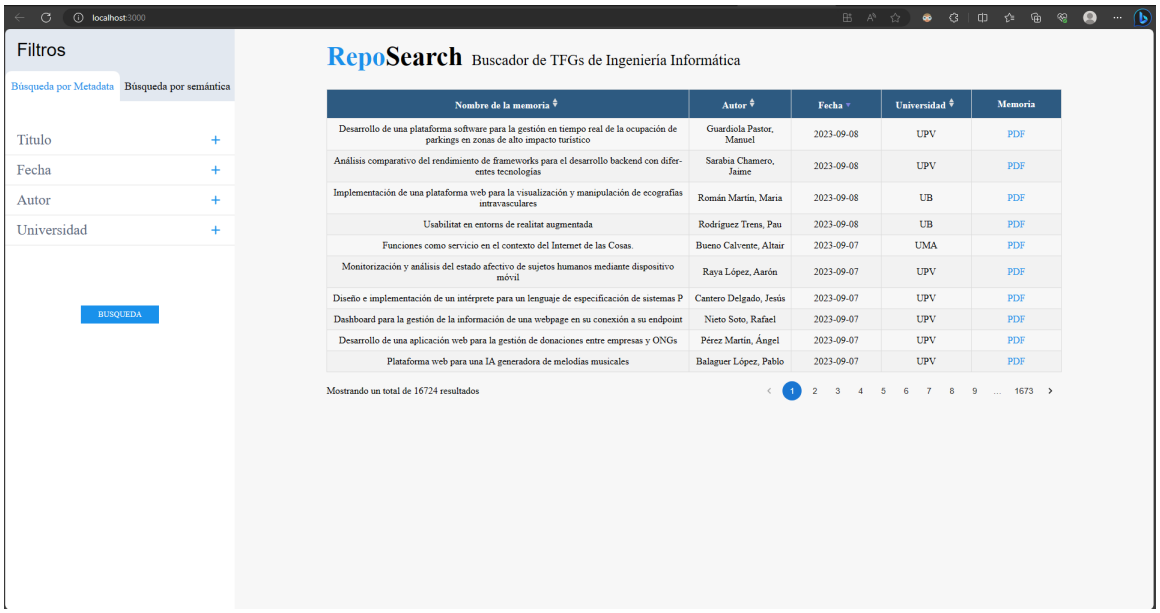


Figura 6.4: Nuevo buscador Reposearch

El resultado del servicio frontend es una nueva interfaz para el buscador, junto con los nuevos filtros. La Figura 6.4 muestra la versión actualizada de la interfaz, con todos los filtros en el panel izquierdo y los resultados de la API en el panel derecho.

En la sección derecha se muestran todas las memorias disponibles después de aplicar los filtros introducidos, junto con el número total de páginas y las memorias correspondientes a la página seleccionada. Pero no siempre la API funcionará perfectamente, por lo que en la Figura 6.5 se muestra de manera textual cuando el servidor ha tenido un fallo.

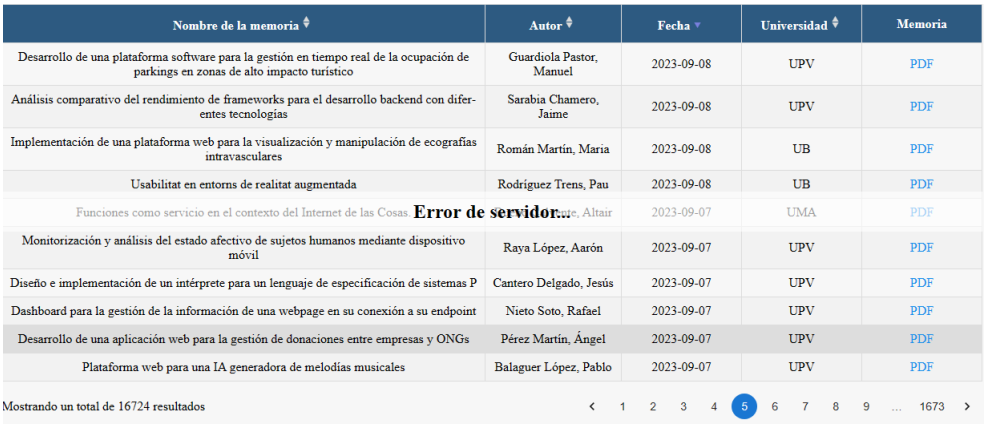


Figura 6.5: Error por parte de la API

Figura 6.6: Respuestas lentas de la API

En la Figura 6.6 se indica al usuario que se están buscando nuevas memorias en caso de que el tiempo de respuesta del servidor sea elevado.

Tras aplicar los filtros del usuario, a veces no existirán memorias que concuerden con la entrada del usuario, por lo que se indicará al usuario como en la Figura 6.7.

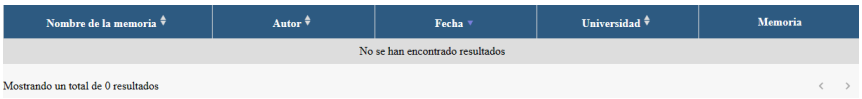


Figura 6.7: No se han encontrado memorias para el filtro

6.3. Scripts

Para mostrar los resultados de las funcionalidades desarrolladas de los scripts, se dividirá en dos subsecciones como en los apartados previos.

6.3.1. Análisis de disponibilidad

La clasificación de las memorias se guarda en una tabla de la base de datos relacional, como se muestra en la Figura 4.13. Para representar el estado de las memorias, se ha generado un script para almacenar el estado de las memorias de manera agrupada dentro de un archivo Excel. En la Figura 6.8 se muestra el estado de las memorias de las universidades de Valencia y del País Vasco, organizadas por categoría y el número total de memorias por categoría.

		Availability Status		Restricción solo para universitarios	Restricción por fecha	Restricción por demanda	Restricción para personal repositorio	Restricción por identificar		Memorias por repositorio
ID	Repo	Disponibles	No disponible						Por verificar	
1	UPM	2114	2	613	0	0	1	0	0	2730
2	UPVEHU	509	7	0	0	0	0	0	0	516

Figura 6.8: Clasificación visual de las memorias

Se pueden generar gráficas personalizadas con la tabla resultante, como en la Figura 6.9.

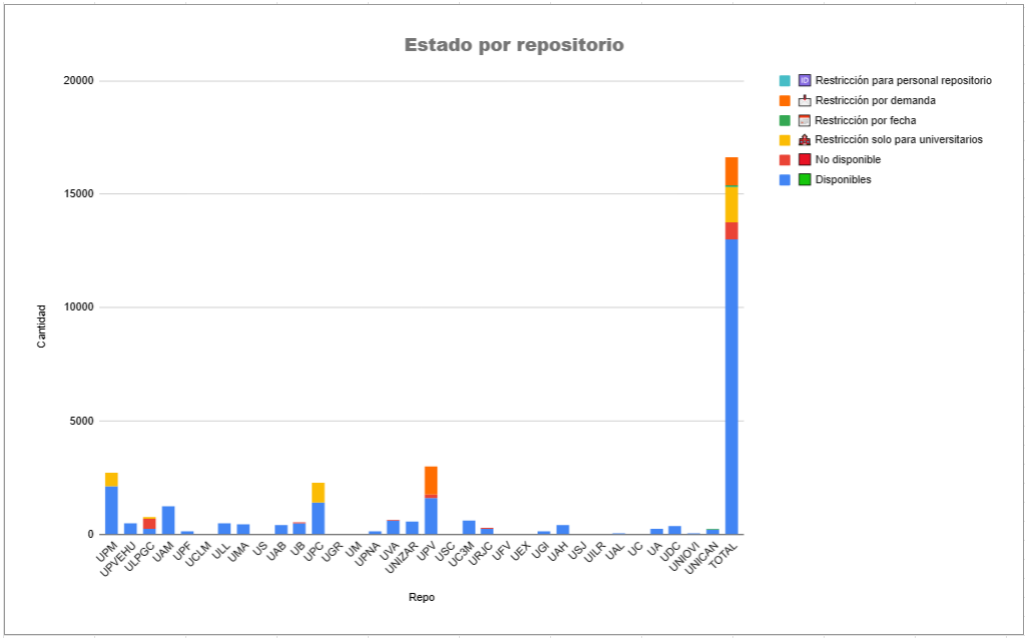


Figura 6.9: Gráfica generada de los datos

6.3.2. Búsqueda semántica

Para la búsqueda semántica, no es posible visualizar directamente la información de la base de datos vectorial de Milvus de la misma manera que se haría en una base de datos relacional. Sin embargo, es posible visualizar los embeddings. En los archivos JSON que se encuentran en el repositorio, se pueden identificar tres claves iniciales, que son:

- **text**: Contiene todo el contenido de los TFGs trozeado.
- **metadata**: Contiene los metadatos de cada trozo de texto, con el chunk, texto, página e identificador de memoria.
- **embeddings**: Contiene la representación de cada trozo de texto en formato vectorial.

Los embeddings se ven de la siguiente manera:

```
76 [-0.028844211250543594, 0.005296764429658651,
    0.017666678875684738, -0.003135725622996688,
    0.06543547660112381, -0.023996299132704735,
    0.019567426294088364, 0.04452540725469589,
    -0.012630555778741837, -0.04282118007540703,
    -0.03977464139461517, -0.05953918397426605,
    -0.03128354996442795, -0.00824088416993618,...]
```

El tamaño de los embeddings variará según el modelo de lenguaje utilizado. En el modelo utilizado en este proyecto, los embeddings se generan con una dimensión de 384.

7. Conclusiones

7.1. Diferencia entre estimación y tiempo real

Se va a realizar una comparación entre la planificación estimada y las horas realizadas.

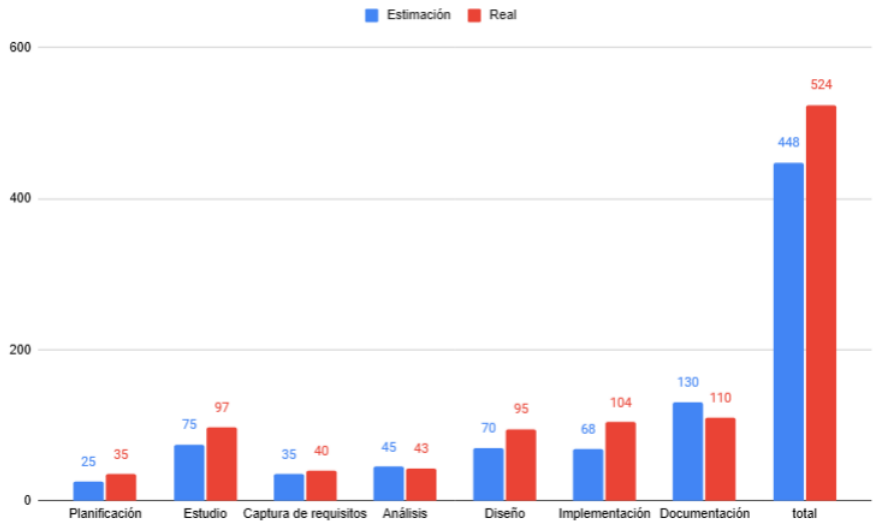


Figura 7.1: Tiempos estimados y realizados

En la Figura 7.1 se presentan los Paquetes de Trabajo del EDT, junto con sus horas estimadas y horas realizadas. Las horas estimadas están detalladas en la Tabla 2.12. Es importante mencionar que las horas realizadas incluyen las tareas que han sido descartadas, como es el caso de Lucene.

El proyecto en su totalidad ha requerido un total de 524 horas, en comparación con las 448 horas estimadas, lo que se traduce en una desviación del 17 % con respecto al plan original.

La diferencia significativa de horas radica en el replanteamiento de dos herramientas, así como en sus respectivos diseños e implementaciones.

7.2. Futuras líneas de trabajo

En este proyecto, se ha refactorizado un proyecto existente y se ha desarrollado una estructura inicial que servirá como base para la implementación de nuevas funcionalidades. Con la colaboración de Juanan Pereira y la documentación generada, se pretende proporcionar una base sólida para que los futuros estudiantes de Ingeniería Informática continúen con el desarrollo del proyecto.

Aunque este proyecto se encuentra en una fase inicial de desarrollo, está lleno de posibilidades para el futuro. A pesar de los avances realizados, aún queda mucho trabajo por hacer. Se requiere un esfuerzo continuo para mejorar y expandir sus funcionalidades, así como para abordar posibles desafíos y mejorar la experiencia del usuario. Con las contribuciones de alumnos comprometidos, este proyecto tiene el potencial de crecer significativamente y ofrecer un recurso valioso para los estudiantes de Ingeniería Informática en el futuro.

7.3. Licencia

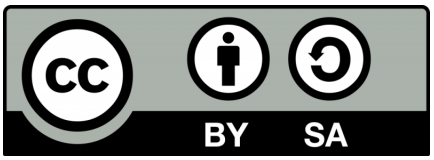


Figura 7.2: Licencia del proyecto

Este proyecto se encuentra bajo la licencia Creative Commons Attribution-ShareAlike 4.0 International License (CC BY-SA 4.0). Esto significa que eres libre de:

- **Compartir**: Copiar y redistribuir el material en cualquier medio o formato.
- **Adaptar**: Remixar, transformar y construir sobre el material para cualquier propósito, incluso comercialmente.

Bajo las siguientes condiciones:

- **Atribución**: Debes otorgar el crédito adecuado, proporcionar un enlace a la licencia e indicar si se han realizado cambios. Puedes hacerlo de

cualquier manera razonable, pero no de una manera que sugiera que el licenciante te respalda a ti o al uso que haces del material.

- **Compartir Igual**: Si remixas, transformas o construyes sobre el material, debes distribuir tus contribuciones bajo la misma licencia que el original.

7.4. Reflexión personal

Se finalizará el desarrollo de la memoria con la valoración personal del trabajo realizado.

Al inicio del proyecto, me enfrenté a la tarea de abarcar una amplia gama de ámbitos, desde el desarrollo del frontend hasta el backend, y utilizar herramientas que desconocía. Esta diversidad inicial de tareas me abrumaba, pero mi objetivo siempre fue claro: quería aprender y crecer en todos estos aspectos del desarrollo de software.

A lo largo del proceso, he adquirido conocimientos en áreas adicionales, como la captura de requisitos y la gestión de un proyecto mucho más extenso en comparación con lo que había realizado anteriormente. Esta experiencia me ha proporcionado una comprensión más profunda de los desafíos que implica coordinar diferentes aspectos del desarrollo de software.

La valoración final ha sido muy positiva, bastante satisfecho con el trabajo realizado. Sin duda seguiría ayudando con el desarrollo del proyecto.

Anexos

Anexo I: Para desarrolladores

Esta sección está dedicada a futuros alumnos que deseen continuar con el desarrollo del proyecto. Se detallarán algunas de las limitaciones o aspectos que aún requieren refinamiento, así como posibles mejoras que podrían ser implementadas en el futuro.

Debido a la gran cantidad de tareas que se planearon realizar e implementar, en un período estimado de 300 horas, dificultó la posibilidad de realizar pruebas exhaustivas en todas las funcionalidades del proyecto. Se han implementado ciertas medidas de precaución, como la realización de retrocesos (rollbacks) en la base de datos relacional en caso de errores al realizar una operación sobre esta. A pesar de estas precauciones, es necesario llevar a cabo pruebas más específicas para identificar y abordar posibles errores de manera efectiva.

Algunas posibles pruebas que se podrían implementar:

- **Prueba de servicio**: Realizar múltiples peticiones simultáneas sobre la API para evaluar los tiempos de respuesta y garantizar la estabilidad del servicio sin interrupciones.
- **Análisis de disponibilidad + descarga de memorias**: Para verificar que las memorias se están analizando y descargando correctamente, es esencial comprobar que los elementos de la página web no cambian de ubicación. También es necesario confirmar que el enlace del elemento de la memoria hace uso de una ruta relativa o completa, entre otras consideraciones.

Otra limitación de las funciones desarrolladas se encuentra en el análisis de disponibilidad y descarga de memorias. En este proyecto, al analizar y descargar las memorias, se omiten los archivos comprimidos. Inicialmente, se planeaba realizar las descargas de estos archivos, pero se descubrió que llevaría más tiempo del esperado y también contenía errores en el código producido. Por lo tanto, esta funcionalidad se dejó fuera del alcance del proyecto debido a estas dificultades técnicas y limitaciones de tiempo.

En el fichero 'constants.py' dentro de 'common' se definen las extensiones de ficheros que se quieren analizar y descargar. En los ficheros para el análisis y descarga de memorias, se requiere que la memoria se encuentre en una de estas extensiones.

```
VALID_FILE_EXTENSIONS = ['pdf', 'doc', 'docx']
COMPRESSED_FILE_EXTENSIONS = ['7z', 'zip', 'rar']
```

Figura A.1: Constantes para la descarga y análisis de memorias

El código relacionado con las memorias que se encuentran en la categoría de archivos comprimidos ha sido comentado para que el próximo desarrollador pueda continuar con ese trabajo. En el código comentado, se hace uso de 'helper-classes.py' dentro de la carpeta scrapper.

```
# THE COMMENTED CODE WAS INTENDED TO DOWNLOAD AND EXTRACT MEMORY FROM INSIDE COMPRESSED FILE, BUT THIS FEATURE WILL BE EXCLUDED FOR NOW
# elif file_extension in COMPRESSED_FILE_EXTENSIONS:
#     await download_file(url, file_extension, MemoClass=MemoClass)
#     handler = get_handler(file_extension, MemoClass.id)
#     if handler is None:
#         print('there is no compressed file')
#     else:
```

Figura A.2: Código comentado cuando la memoria es un fichero comprimido

En el análisis de disponibilidad y la descarga de memorias, se ha implementado una función llamada 'log-error' que tiene la tarea de registrar los posibles errores ocurridos durante la ejecución del programa. Esta función se utiliza principalmente en el decorador 'request-action', que actúa como un mecanismo para reintentar la petición a los servidores de las universidades en caso de que ocurriese un error y la conexión fuera abortada. Para mejorar el registro de errores, se podría considerar el uso de bibliotecas específicas desarrolladas para Python.

Ahora se van a exponer las contrapartes del mecanismo de reintento de peticiones.

Cuando se envían peticiones a los servidores de las universidades, se establece un límite de tiempo de espera para recibir una respuesta. Si no se recibe una respuesta dentro de este límite, la conexión se aborta y se intenta nuevamente utilizando el mecanismo de reintento. Esta estrategia es efectiva para manejar situaciones donde el servidor puede tener una carga momentánea, lo que podría causar tiempos de respuesta más largos. No obstante, surge un problema cuando el servidor experimenta problemas continuos y presenta

tiempos de respuesta extremadamente altos.

En casos donde se intenta realizar descargas o análisis de un gran número de memorias de una universidad con tiempos de carga muy altos, el proceso se alarga considerablemente debido a los múltiples intentos realizados para cada memoria. Esto puede generar demoras significativas en la operación global del sistema, lo que puede ser un área de mejora para futuras implementaciones.

Otra área en la que se podrían realizar mejoras está relacionada con la base de datos vectorial. Actualmente, la generación de embeddings se realiza únicamente para las memorias en formato PDF, que constituyen la gran mayoría de los TFGs. Sin embargo, existen excepciones, ya que algunos trabajos vienen en formato Doc, como se indica en la Figura A.1. Para mejorar la búsqueda semántica de manera más completa, sería beneficioso incluir estos archivos en el proceso de generación de embeddings. Esto ampliaría la cobertura y la eficacia del sistema para analizar una variedad más amplia de formatos de archivo y mejorar la precisión en las búsquedas semánticas.

Como estudiante de la rama de software, mis conocimientos sobre bases de datos vectoriales son bastante limitados. Por lo tanto, un estudiante de la rama de Inteligencia Artificial podría estar más capacitado para mejorar todos los procesos y algoritmos de generación de embeddings. Al contar con una comprensión más profunda de los conceptos y técnicas específicas en el campo de la IA, podrían implementar mejoras significativas en la generación de embeddings, optimizando así la eficacia del sistema en la búsqueda semántica de los trabajos de fin de grado.

Finalmente, algunas consideraciones adicionales del proyecto.

Los TFGs obtenidos de los repositorios se recuperan utilizando el protocolo OAI-PMH y la biblioteca Sickle de Python. Para adquirir las memorias, es necesario identificar conjuntos específicos(sets) para cada universidad, un proceso que debe realizarse manualmente. Sin embargo, existe la posibilidad de que en el futuro las universidades modifiquen la ubicación de sus TFGs, lo que implicaría cambios en sus conjuntos correspondientes. Para hacer frente a este desafío potencial, se debería establecer un sistema de seguimiento de estos conjuntos o desarrollar un script que realice el seguimiento de manera automática, asegurando así la continuidad del proceso de obtención de las memorias.

Uno de los problemas encontrados al usar Sickle es su dependencia del módulo 'requests' de Python para realizar las solicitudes. En el caso de dos universidades que se indexaban, tenían una versión de TLS obsoleta, considerada 'deprecated', lo que resultaba en que el módulo 'requests' no podía establecer una conexión debido a problemas de seguridad. Esta limitación impide obtener las memorias de estas universidades, resultando en la pérdida de una cantidad significativa de TFGs indexados.

Por último, como se ha mencionado respecto al seguimiento de los sets de las universidades, podría ser interesante realizar un seguimiento de los enlaces de las memorias. Estos enlaces podrían cambiar con el tiempo, y si no se actualizan correctamente, en el buscador los usuarios podrían ser redirigidos a enlaces erróneos.

Acrónimos

EDT

Estructura de Desglose de Trabajo. 12

REBIUN

Red de Bibliotecas Universitarias Españolas. 1

Glosario

API

La Interfaz de Programación de Aplicaciones proporciona procesos, funciones y métodos para ser utilizados por otro software.. 5, 39, 40

Asynchronous Server Gateway Interface

ASGI o Interfaz de puerta de enlace de servidor asincrónico es una especificación para la comunicación asíncrona entre servidores web y aplicaciones web.. 11, 51

backend

La parte de un sistema informático que se encarga de procesar datos y lógica de negocio.. 11, 42

backup

Copia de seguridad de datos o información utilizada para recuperarlos en caso de pérdida o daño.. 66, 80

base de datos vectorial

Una base de datos especializada en el almacenamiento y búsqueda de datos vectoriales. Son muy útiles para el procesamiento del lenguaje natural o el reconocimiento de imágenes.. 40, 44, 50

BD

Base de Datos, un sistema de almacenamiento de datos en formato de tablas.. 5, 39

chunk

Un bloque o fragmento de datos que se divide para su procesamiento o transferencia.. 65, 88

DOM

Modelo de Objetos del Documento, una representación jerárquica de una página web.. 61, 76

embeddings

Representaciones numéricas de objetos, a menudo utilizadas en procesamiento de lenguaje natural.. 44, 55, 64, 79, 83, 88

fetch

Función de JavaScript para realizar solicitudes a recursos de red.. 5

framework

Estructura o entorno que proporciona herramientas y bibliotecas para el desarrollo de software.. 8, 51

HTTP

Protocolo de Transferencia de Hipertexto, un protocolo de comunicación utilizado en la World Wide Web.. 5

IDE

Entorno de Desarrollo Integrado, una aplicación que facilita la escritura y el desarrollo de código.. 10

JSON

Notación de Objetos de JavaScript, un formato ligero de intercambio de datos.. 40

Milvus

Base de datos vectorial de código abierto que facilita el almacenamiento y la búsqueda eficientes de datos vectoriales.(Milvus [2021]). 51, 81

OAI-PMH

Protocolo de Acceso Abierto a Recursos de Metadatos, un protocolo para la recolección de metadatos de repositorios.. 1, 2, 6, 74

paginación

División de contenido en páginas para facilitar la navegación y la carga de datos.. 5, 42

scrapping

Técnica de extracción de datos de sitios web de manera automatizada.. 5, 10

SOLID

Acrónimo de cinco principios de diseño de software: SRP, OCP, LSP, ISP, DIP.(Erinç [2020]). 43

TFG

Trabajo de Fin de Grado, un proyecto académico final que se realiza al final de una carrera universitaria.. 1, 2, 9

whitelist

Lista de elementos o direcciones permitidas en un sistema o aplicación.. 41

XPATH

Lenguaje de consulta utilizado para navegar y seleccionar elementos en documentos XML.. 74, 76

Bibliografía

Ashwin Bande. Is it better to paginate on the server side or front end? https://stackoverflow.com/a/54493622, 2019.

Shawn Belling. *Succeeding with Agile Hybrids: Project Delivery Using Hybrid Methodologies*. Apress, 2020.

Yiğit Kemal Erinç. The SOLID principles of object-oriented programming explained in plain English. https://www.freecodecamp.org/news/solid-principles-explained-in-plain-english/, 2020.

Omar Espejel. Getting started with embeddings. https://huggingface.co/blog/getting-started-with-embeddings, 2022.

ISO. *Gestión del riesgo — Directrices*. International Organization for Standardization, 2018.

Akash Joshi. A better way to structure React projects. https://www.freecodecamp.org/news/a-better-way-to-structure-react-projects/, 2021.

F. McCown, M.L. Nelson, M. Zubair, and X. Liu. Search engine coverage of the oai-pmh corpus. *IEEE Internet Computing*, 10(2):66–73, 2006. doi: 10.1109/MIC.2006.41.

Milvus. Vector database built for scalable similarity search. https://milvus.io/, 2021.

Juanan Pereira. Reposearch, a centralized search engine for end-of-degree projects of the bachelor's degree in computer engineering. In *2021 International Symposium on Computers in Education (SIIE)*, pages 1–5, Sep. 2021. doi: 10.1109/SIIE53363.2021.9583638.

Reposeach. Buscador de TFGs de Ingeniería Informática. https://reposearch.ikasten.io/v1/data_sources/server_side.html, 2021.

Kevin Tewouda. FastAPI and pagination. https://lewoudar.medium.com/fastapi-and-pagination-d27ad52983a, 2023.

Zhanymkanov Yerassyl. FastAPI best practices. https://github.com/zhanymkanov/fastapi-best-practices, 2022.