Desarrollo de una herramienta de vigilancia tecnológica para identificar oportunidades de financiamiento de proyectos de investigación

Objetivo General

Desarrollar una herramienta automatizada que realice vigilancia tecnológica en la web, identificando convocatorias nacionales e internacionales de financiamiento para proyectos de investigación, y que genere informes explicativos sobre dichas oportunidades.

Objetivos Específicos

- Diseñar un sistema que explore fuentes oficiales y confiables (portales de agencias de financiación, universidades, organizaciones multilaterales, etc.).
- Implementar algoritmos de web scraping y/o APIs para recolección automatizada de datos.
- Clasificar las oportunidades encontradas por áreas de investigación, requisitos, fechas y tipo de financiamiento.
- Generar reportes periódicos (semanales o mensuales) con resúmenes ejecutivos de las nuevas convocatorias.
- Permitir búsquedas manuales y suscripciones personalizadas a temas de interés.

Entregables

- Módulo funcional de recolección de datos desde fuentes web (scraper/API).
- Base de datos estructurada de convocatorias vigentes e históricas.
- Interfaz web o de escritorio para consultas y visualización de oportunidades.
- Generador de informes explicativos en formatos PDF/Word.
- Sistema de alertas por correo o notificaciones automáticas.
- Documentación técnica y manual de usuario.
- Informe final del desarrollo del sistema.

Requerimientos

Funcionales

- A partir de un proyecto, identificar oportunidades de financiación para dicho proyecto.
- Realizar un enriquecimiento periódico de las oportunidades encontradas para los proyectos.

- Realizar reportes periódicos de las investigaciones realizadas y los resultados obtenidos.
- Generar servicio de notificación sobre los resultados de las investigaciones periódicas.
- CRUD y autenticación de usuarios.

No Funcionales

- Investigación manual de oportunidades.

Sistema

- Implementación de MicroServicios.
- Manejo asíncrono de tareas (Investigaciones de proyectos se realizarán en segundo plano).

Arquitectura y Flujo de Tecnologías

La arquitectura se basa en un ecosistema de **MicroServicios orquestados por Docker**, lo que permite que cada componente del sistema opere de forma independiente, sea escalable y fácil de mantener. A continuación, se detalla la tecnología asociada a cada servicio y cómo interactúan entre sí.

Stack Tecnológico por Servicio

Componente	Tecnologías Clave	Responsabilidad
FrontEnd	**Angular, TypeScript**	Proporciona la interfaz de usuario (UI) para la interacción. Gestiona la visualización de datos y las solicitudes del cliente.
API (Backend)	**Python, Django**	Expone los endpoints para la autenticación, gestión de usuarios y proyectos (CRUD). Orquesta el inicio de tareas asíncronas.
Base de Datos	**PostgreSQL**	Almacena de forma persistente y estructurada todos los datos: usuarios, proyectos, convocatorias y resultados.
Cola de Tareas	**Redis**	Actúa como un *message broker*. Recibe las tareas pesadas de la API y las encola para que los Workers las procesen.
Worker (Procesamiento)	**Python, Celery, LangChain**	Ejecuta las tareas de larga duración en segundo plano (búsqueda, scraping, análisis de IA). Es el motor de la herramienta.
Orquestación	**Docker, Docker-Compose**	Gestiona el ciclo de vida de todos los servicios, define las redes, volúmenes y variables de entorno para el despliegue.

Flujo de Trabajo e Interacción entre Servicios

El funcionamiento del sistema sigue un flujo lógico y asíncrono para garantizar una experiencia de usuario fluida y un procesamiento eficiente.

1. **Inicio de la Solicitud**: El usuario interactúa con la interfaz de **Angular**, por ejemplo, registrando un nuevo proyecto de investigación y solicitando la búsqueda de financiamiento.

2. **Gestión de la API**: El FrontEnd envía una petición a la **API de Django**. El backend valida la solicitud, autentica al usuario y guarda la información del proyecto en la base de datos **PostgreSQL**.

3. **Encolado de Tareas**: En lugar de ejecutar la búsqueda directamente (lo que bloquearía el sistema), la API crea una tarea específica (ej: "buscar_oportunidades") y la envía a **Redis**. Redis la almacena en una cola y la API responde inmediatamente al usuario, informándole que el proceso ha comenzado.

4. **Procesamiento en Segundo Plano**: El **Worker de Celery**, que está constantemente escuchando la cola de Redis, detecta la nueva tarea y la toma para su procesamiento.

5. **Investigación y Enriquecimiento**: El Worker utiliza **LangChain** para orquestar una serie de acciones:

    - Consulta a APIs de búsqueda como **Brave** o **Tavily** y fuentes **RSS** para recopilar información de la web.
    - Envía los datos recolectados a la **API de Gemini** para su análisis, resumen, extracción de entidades clave (fechas, requisitos) y clasificación.
    - **LangSmith** se utiliza para monitorear y depurar las cadenas de llamadas a los modelos de lenguaje, asegurando su fiabilidad.

6. **Almacenamiento de Resultados**: Una vez procesada la información, el Worker estructura los resultados (oportunidades de financiamiento, resúmenes, etc.) y los guarda en la base de datos **PostgreSQL**, asociándolos con el proyecto y usuario correspondientes.

7. **Notificación y Visualización**: Al finalizar la tarea, el sistema puede generar una notificación automática (vía email o en la app). El usuario ya puede acceder a su panel en **Angular** para visualizar los resultados, los cuales son servidos por la **API de Django** consultando la información previamente almacenada en la base de datos.

Servicios Externos

Para realizar la vigilancia tecnológica y el enriquecimiento de datos, el sistema se apoya en un conjunto de APIs y tecnologías externas que son fundamentales para su operación.

- **Gemini API**: Es el núcleo de la inteligencia artificial del sistema. Este modelo de lenguaje avanzado de Google se utiliza para procesar el texto no estructurado de las convocatorias encontradas. Sus tareas principales incluyen: resumir los objetivos de la convocatoria, extraer datos clave como fechas límite, montos de financiamiento y requisitos de elegibilidad, y clasificar las oportunidades por área de investigación.

- **Brave API / Tavily API**: Son los motores de búsqueda que alimentan al sistema. Se utilizan para realizar consultas web automatizadas y descubrir nuevas convocatorias en portales que no ofrecen feeds RSS. Tavily está especialmente optimizado para ser utilizado por agentes de IA, proporcionando resultados de búsqueda concisos y relevantes que facilitan el posterior análisis por parte de Gemini.

- **Tecnologías RSS**: El sistema se suscribe a feeds RSS (Really Simple Syndication) de fuentes confiables y conocidas, como agencias de financiamiento gubernamentales, fundaciones y universidades. Este es un método altamente eficiente y estructurado para recibir notificaciones automáticas sobre nuevas publicaciones y convocatorias directamente de la fuente.

- **LangSmith**: Es una plataforma de observabilidad y depuración para aplicaciones de modelos de lenguaje. Se integra con LangChain para ofrecer una trazabilidad completa de las operaciones del Worker. Permite a los desarrolladores monitorear cada paso del proceso de investigación (búsquedas, llamadas a la API de Gemini, procesamiento de datos), facilitando la identificación de errores, la optimización de los *prompts* y la mejora continua de la fiabilidad del sistema.

Modelo de Datos

La persistencia de los datos se gestiona a través de una base de datos relacional PostgreSQL, diseñada para reflejar lógicamente el flujo de trabajo de la aplicación. Este modelo garantiza que toda la información, desde los proyectos de los usuarios hasta las oportunidades identificadas, esté organizada, relacionada y sea fácilmente accesible.

Diagrama Entidad-Relación (ERD)

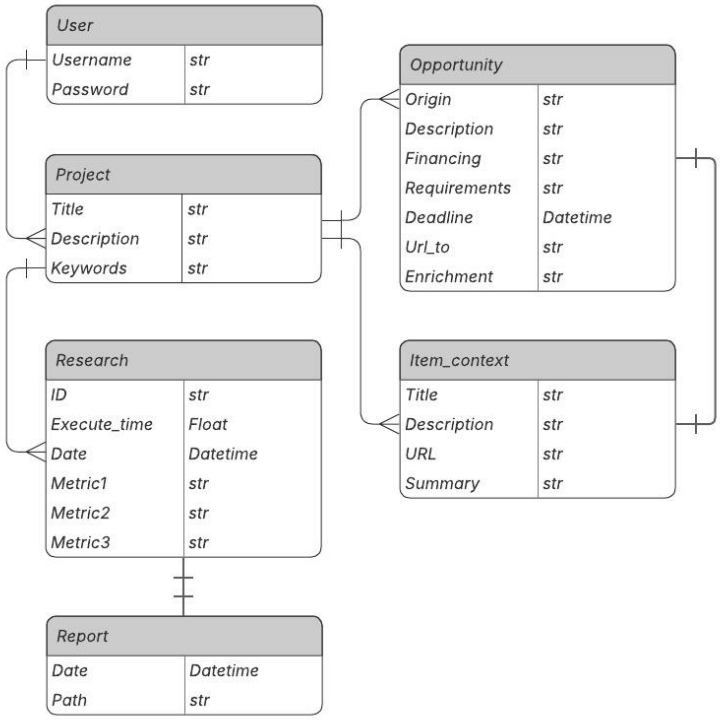

Diagrama de la Base de Datos

Descripción de las Entidades

- **User**: Almacena las credenciales de los usuarios para la autenticación y gestión de acceso al sistema.
- **Project**: Representa el objeto de vigilancia definido por el usuario, conteniendo el título, la descripción y las palabras clave que sirven como entrada para los agentes de IA.
- **Research**: Registra cada ejecución del flujo de investigación para un proyecto. Funciona como un historial, guardando métricas clave como la fecha y el tiempo de ejecución.

- **Item_context**: Contiene los resultados normalizados de las búsquedas web. Cada registro es una fuente de información (un enlace con su título y descripción) que ha sido considerada relevante para ser analizada en profundidad.

- **Opportunity**: Almacena los datos estructurados y extraídos por el agente de IA. Cada registro es una oportunidad de financiación concreta, con detalles como su origen, descripción, tipo de financiamiento y fechas límite.

- **Report**: Guarda una referencia a los informes generados al finalizar una investigación, incluyendo la fecha de creación y la ruta de acceso al archivo.

Relaciones Clave

- Un **`User`** puede tener múltiples **`Projects`**.
- Un **`Project`** puede tener múltiples **`Research`** (ejecuciones) a lo largo del tiempo.
- Cada **`Research`** produce un **`Report`** final.
- Tanto los **`Item_context`** como las **`Opportunities`** están vinculados a un **`Project`** para mantener todo el contexto de la investigación organizado.