UNIVERSIDAD DE GRANADA

TRABAJO FIN DE GRADO

INGENIERÍA INFORMÁTICA

Visualizando relaciones entre sitios ocultos en Darknets

Autor

Juan Miguel Hernández Gómez

Tutor

Roberto Magán Carrión

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

—

Granada, Septiembre de 2023

Visualizando relaciones entre sitios ocultos en Darknets

Juan Miguel Hernández Gómez

Palabras clave: Darknets, Visualización, API, Dashboard, Conectividad.

Resumen

Este proyecto se enfoca en desarrollar un sistema integral para visualizar y analizar las relaciones entre sitios ocultos en Darknet, como Tor e I2P. Consta de una API REST en Python y un dashboard interactivo en Angular, permitiendo a los usuarios explorar y comprender en profundidad la estructura de conectividad en estas redes anónimas. Destacando la implementación de un sistema de caché para mejorar el rendimiento, el proyecto ofrece una herramienta valiosa para explorar estas redes misteriosas.

Visualizing Relationships Among Hidden Sites in Darknets

Juan Miguel Hernández Gómez

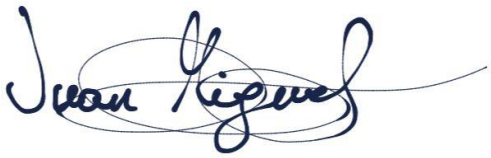
Keywords: Darknets, Visualization, API, Dashboard, Connectivity.

Abstract

This project aims to develop a comprehensive system for visualizing and analyzing relationships among hidden sites in Darknet, such as Tor and I2P. It consists of a Python-based REST API and an interactive Angular dashboard, enabling users to explore and gain in-depth insights into the connectivity structure within these anonymous networks. With a focus on implementing a caching system for enhanced performance, the project provides a valuable tool for exploring these enigmatic networks.

Yo, **Juan Miguel Hernández Gómez**, alumno de la titulación INGENIERÍA INFORMÁTICA de la **Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación de la Universidad de Granada**, con DNI 45315448-C, autorizo la ubicación de la siguiente copia de mi Trabajo Fin de Grado en la biblioteca del centro para que pueda ser consultada por las personas que lo deseen.

Fdo: **Juan Miguel Hernández Gómez**



Granada a 7 de Septiembre de 2023.

D. **Roberto Magán Carrión**, Profesor del Área de Ingeniería Telemática del departamento de Teoría de la Señal Telemática y Comunicaciones de la Universidad de Granada.

Informa:

Que el presente trabajo, titulado *Visualizando relaciones entre sitios ocultos en Darknet*, ha sido realizado bajo su supervisión por **Juan Miguel Hernández Gómez**, y autorizo la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada a 7 de Septiembre de 2023.

El tutor:

Roberto Magán Carrión

Agradecimientos

En este momento de culminación de mi carrera universitaria, quiero expresar mi más profundo agradecimiento a las personas que han sido fundamentales en este recorrido. Sus apoyos incondicionales y confianza en mí han sido el motor que me ha impulsado a superar desafíos y alcanzar metas. Sin su respaldo, este logro no sería posible.

En primer lugar, quiero agradecer a mis padres y a mi hermana por estar a mi lado en cada etapa de mi carrera. En los momentos de mayor estrés y dificultad, su apoyo emocional y aliento inquebrantable me han brindado la fortaleza necesaria para superar obstáculos y seguir adelante. Su confianza en mis capacidades y su dedicación para asegurarse de que tuviera todas las oportunidades para desarrollarme han sido un regalo invaluable.

También quiero expresar mi gratitud al profesor que me ha acompañado durante el desarrollo de mi Trabajo de Fin de Grado. Su disposición para responder a mis correos electrónicos y resolver mis dudas ha sido un factor crucial en el éxito de este proyecto. Su guía experta y su apoyo constante me han proporcionado la dirección y la confianza para llevar adelante este trabajo de investigación de manera efectiva.

Cada uno de ustedes ha contribuido de manera significativa a mi crecimiento académico y personal. Este logro no solo es mío, sino también de ustedes, porque han sido parte integral de mi viaje. Gracias por creer en mí, por brindarme su apoyo inquebrantable y por permitirme perseguir mi pasión. Siempre llevaré sus enseñanzas y su apoyo en mi camino hacia el futuro.

Con gratitud sincera,

Juan Miguel Hernández Gómez

Índice general

Índice de figuras

Índice de tablas

Acrónimos

API *Application Programming Interface* (Interfaz de Programación de Aplicaciones, en español).

REST *Representational State Transfer* (Transferencia de Estado Representacional, en español).
SOAP

XML *Extensible Markup Language* (Lenguaje de Marcado Extensible, en español)

JSON *JavaScript Object Notation* (Notación de Objetos de JavaScript, en español)

CORS *Cross-Origin Resource Sharing* (Compartir Recursos entre Orígenes Cruzados, en español)

CRUD *Create Read Update Delete* (Crear Leer Actualizar y Eliminar, en español)

CLI *Command Line Interface* (Interfaz de Línea de Comando, en español)

NLTK *Natural Language Toolkit* (herramientas de lenguaje natural, en español)

1. Introducción

En el vasto y enigmático mundo de la Deep Web, donde las fronteras digitales se desdibujan y la información fluye en la oscuridad virtual, se encuentra un desafío que ha capturado la atención de investigadores y entusiastas de la tecnología. Pero ¿dónde se ubica exactamente la Deep Web en el panorama de Internet? Para contextualizar, es importante diferenciar entre tres capas distintas [3]:

- ***Surface Web*:** Esta es la capa visible de Internet que la mayoría de las personas utiliza a diario. Aquí se encuentran los sitios web que son indexados por los motores de búsqueda convencionales, como Google y Bing. Puedes acceder a esta parte de Internet simplemente navegando por la web y realizando búsquedas en motores de búsqueda.
- ***Deep Web*:** La *Deep Web*, a menudo malinterpretada, se refiere a la parte de Internet que no está indexada por motores de búsqueda convencionales. Esto incluye bases de datos privadas, cuentas de correo electrónico, sistemas de gestión de bases de datos y otros recursos que no son de acceso público. Acceder a la *Deep Web* generalmente requiere credenciales o permisos específicos.
- ***Darknets*:** Las *Darknets*, por otro lado, son redes privadas y encriptadas dentro de la *Deep Web*. Son conocidas por su anonimato y su uso para actividades diversas, incluyendo la comunicación segura y el intercambio de información sensible. Tor e I2P son ejemplos de *Darknets* populares.

A medida que la exploración de la *Deep Web* ha cobrado impulso en los últimos años, el enfoque predominante ha estado dirigido hacia la comprensión del contenido y la información de los sitios ocultos. Sin embargo, mientras los académicos y los expertos han desentrañado los secretos que yacen en las *Darknets*, una cuestión esencial ha permanecido en las sombras: la intrincada interconexión que subyace entre estos sitios en la oscuridad.

En este contexto, surge la necesidad imperante de examinar y analizar cómo estos sitios en la *Darknet* se relacionan entre sí. Este estudio toma como punto de partida la herramienta C4Darknet, desarrollada por destacados colegas en la Universidad de Granada, que ha permitido incursionar en el vasto y misterioso territorio de la *Deep Web* de una manera previamente inexplorada. C4Darknet, una herramienta que

desempeñará un papel fundamental en este proyecto brinda la capacidad de llevar a cabo el proceso de "*crawling*" en las *Darknets*.

El "*crawling*" es el proceso mediante el cual los motores de búsqueda, a través de programas automatizados conocidos como "*spiders*" o "*crawlers*", rastrean la web en busca de nuevas páginas y actualizan la información de las ya existentes en sus índices. Estos "*spiders*" recorren enlaces de página en página, recopilando datos que luego se almacenan en bases de datos para su posterior indexación y disponibilidad en los resultados de búsqueda. Los "*spiders*" simulan el comportamiento humano al navegar por la web, pero a una escala y velocidad automatizada.

Este proyecto tiene como objetivo diseñar y desarrollar un *dashboard* que permita visualizar en tiempo real el intrincado grafo de conectividad entre sitios en las *Darknets*, utilizando la valiosa información recopilada por C4Darknet durante su proceso de "*crawling*". El *dashboard* presentará una variedad de representaciones visuales, como gráficas sectoriales, histogramas, gráficos de dispersión, tablas y grafos. Las gráficas serán dinámicas y estáticas, lo que permitirá a los usuarios finales interactuar con ellas y ajustar parámetros clave para obtener perspectivas personalizadas. Las tablas, por su parte, ofrecerán la posibilidad de organizar y visualizar datos de manera conveniente, mientras que los grafos proporcionarán una representación visual de las conexiones entre sitios, incluyendo subconjuntos específicos como los grafos de "*incoming*" y "*outgoing*".

En síntesis, este proyecto se sumerge en el fascinante mundo de la *Deep Web*, utilizando la herramienta C4Darknet y su proceso de "*crawling*" como base de datos fundamental. Al crear un *dashboard* visualmente informativo y funcional, buscamos arrojar luz sobre la intrincada interconexión de las *Darknets* y brindar a los investigadores y analistas una herramienta poderosa para comprender y navegar este entorno digital en constante evolución.

1.1. Objetivos

Objetivos principales:

- **Crear una herramienta de monitorización y gestión del proceso de *crawling***.
- **Desarrollo de una API REST en Python con Flask**: Un componente fundamental es la creación de una API REST utilizando el *framework* Flask en Python. Esta API permitirá interactuar con la base de datos generada por el proceso de "*crawling*" realizado por la herramienta C4Darknet. La API

proporcionará acceso a los datos de conectividad y otros detalles relevantes para su posterior uso en el *dashboard*.

- **Creación de un *dashboard* en Angular**: Se propone el diseño y desarrollo de un *dashboard* interactivo utilizando el framework Angular. Este *dashboard* actuará como una interfaz de usuario atractiva y funcional que permitirá a los usuarios explorar visualmente los datos de conectividad entre sitios en las *Darknets*. El *dashboard* presentará información enriquecida a través de gráficos, tablas y visualizaciones de datos.

Objetivos secundarios:

- **Interfaz intuitiva y usable**: Un objetivo adicional es asegurar que el *dashboard* sea fácil de usar y que ofrezca una experiencia de usuario intuitiva. La interfaz deberá ser amigable para los usuarios finales, independientemente de su nivel de experiencia técnica.
- **Interfaz interactiva**: Se pretende dotar al *dashboard* con características interactivas que permitan a los usuarios ajustar parámetros, filtrar datos y explorar diferentes aspectos de la conectividad entre sitios de *Darknets*. La interacción en tiempo real aumentará la utilidad y versatilidad del *dashboard*.
- **Visualizaciones variadas**: Además de ofrecer diferentes tipos de gráficos y visualizaciones, el *dashboard* aspira a proporcionar *insights* perspicaces a través de representaciones visuales de los datos. Esto ayudará a los usuarios a comprender mejor las relaciones entre sitios y a identificar patrones emergentes.

En resumen, este Trabajo de Fin de Grado tiene como objetivo principal *"*Crear una herramienta de monitorización y gestión del proceso de *crawling"*, con un enfoque en la creación de un *dashboard* interactivo que utilice una API REST en Python con Flask para acceder a datos generados por el proceso de "*crawling*" en *Darknets*. Los objetivos secundarios incluyen la usabilidad, interactividad y seguridad del *dashboard*, así como la documentación completa para garantizar la comprensión y el uso continuo del proyecto.

2. Planificación

En esta sección, abordaremos la planificación detallada de este proyecto, que incluye la definición de requisitos funcionales y no funcionales, así como las tareas necesarias para su ejecución.

2.1. Requisitos

2.1.1. Requisitos Funcionales

1. **Obtención de datos**: La API debe ser capaz de obtener datos de la base de datos de manera eficiente mediante consultas específicas.
2. **Generación de respuestas**: La API debe generar respuestas en formato JSON para cada solicitud, siguiendo un patrón de estructura predefinida.
3. **Actualización de caché**: La API debe implementar un sistema de caché que se actualice automáticamente cada 10 minutos.
4. **Visualización de gráficas**: La API debe proporcionar métodos para generar y enviar gráficas estáticas y dinámicas que representen los datos de manera visual.
5. **Interacción con el *dashboard***: La API debe ser compatible con el *dashboard* web, permitiendo que los datos se presenten de manera interactiva y coherente.
6. **Exportación de datos**: Los usuarios deben tener la capacidad de exportar datos visualizados en formato CSV.
7. **Notificaciones y alertas**: El *dashboard* puede proporcionar notificaciones y alertas visuales para informar a los usuarios sobre eventos relevantes.
8. **Personalización de configuración**: Los usuarios deben poder personalizar aspectos del *dashboard*, como temas visuales, preferencias de visualización y otros ajustes.
9. **Actualización manual de datos**: El *dashboard* debe tener una función para permitir a los usuarios forzar la actualización de los datos desde la API.
10. **Selección de vistas**: Los usuarios deben poder navegar y seleccionar diferentes vistas del *dashboard* para explorar diferentes aspectos de los datos.
11. **Visualización de datos**: El *dashboard* debe mostrar visualmente los datos generados por la API en forma de gráficas, tablas y otros elementos interactivos.

12. **Barra de progreso para visualización de progreso**: El *dashboard* debe incluir un 12. barra de progreso animado que se activa durante las consultas a la API y otras operaciones que puedan llevar tiempo. Esta funcionalidad tiene como objetivo proporcionar una representación visual del progreso de la operación en curso, permitiendo al usuario saber cuándo se están realizando tareas en segundo plano y cuándo se han completado.

2.1.2. Requisitos No Funcionales:

1. **Rendimiento**: La API debe ser capaz de manejar múltiples solicitudes concurrentes sin experimentar degradación en el rendimiento.
2. **Tiempo de respuesta**: El tiempo de respuesta de la API debe ser bajo, asegurando una experiencia de usuario ágil y sin demoras notables.
3. **Escalabilidad**: La arquitectura de la API debe ser escalable, permitiendo la incorporación de nuevos métodos y características sin afectar la funcionalidad existente.
4. **Documentación**: La documentación de la API debe ser exhaustiva y clara, proporcionando instrucciones detalladas para su uso y comprensión.
5. **Compatibilidad**: El *dashboard* debe ser compatible con una variedad de navegadores web modernos, sistemas operativos y dispositivos, incluidos dispositivos móviles y tabletas.
6. **Diseño responsivo**: La interfaz del usuario debe adaptarse de manera fluida a diferentes tamaños de pantalla y resoluciones, garantizando una experiencia consistente en diferentes dispositivos.

2.1.3. Requisitos de Documentación:

1. **Manual de usuario**: Debe existir un manual de usuario que explique cómo utilizar la API y el *dashboard*, incluyendo ejemplos de solicitudes y respuestas.
2. **Documentación técnica**: Se debe proporcionar documentación técnica que describa la arquitectura de la API, las tecnologías utilizadas y los detalles de implementación.
3. **Ejemplos de uso**: La documentación debe incluir ejemplos de casos de uso comunes y cómo realizar solicitudes para obtener los resultados deseados.

4. **Diagramas y gráficos**: Se pueden incluir diagramas de flujo, diagramas de entidad-relación y otros gráficos que ayuden a comprender la estructura y funcionalidad de la API.
5. **Consideraciones de desarrollo**: La documentación puede abordar decisiones de diseño, problemas encontrados durante el desarrollo y soluciones aplicadas.
6. **Instalación y configuración**: Si es necesario, se debe proporcionar información sobre cómo instalar, configurar y desplegar la API y el *dashboard* en diferentes entornos.

2.2. Tareas

En el desarrollo y ejecución del proyecto, se han identificado diversas tareas que contribuyen a la consecución de los objetivos planteados. Cada tarea se relaciona directamente con uno o varios objetivos específicos, lo que garantiza la alineación entre el trabajo realizado y los resultados esperados. A continuación, se detallan las tareas principales que se llevarán a cabo en las diferentes etapas del proyecto:

2.2.1. Planteamiento del proyecto

- Definir el alcance y los objetivos del proyecto en consulta con el tutor.
- Establecer un plan de trabajo y un cronograma tentativo para las etapas clave del proyecto.

2.2.2. Estudio del estado del arte

- Investigar herramientas existentes relacionadas con la conectividad en *Darknets*.
- Revisar literatura técnica y académica para comprender las metodologías y enfoques utilizados en proyectos similares.

2.2.3. Evaluación de la propuesta

- Analizar los objetivos y requisitos del proyecto para asegurarse de que sean coherentes y alcanzables.
- Evaluar la viabilidad técnica y la factibilidad de implementación de la API y el *dashboard*.

2.2.4. Reparto de tareas

- Asignar estimaciones temporales para cada objetivo y tarea identificados.

- Establecer un sistema de seguimiento y comunicación para garantizar el progreso y la colaboración efectiva.

2.2.5. Diseño e implementación

- Diseñar la arquitectura general de la API y el *dashboard*, definiendo los componentes y la interacción entre ellos.
- Implementar la API REST utilizando el *framework* Flask, estableciendo rutas y funcionalidades para acceder a los datos de conectividad.

2.2.6. Investigación

- Realizar investigaciones adicionales sobre *Darknets* y su conectividad para comprender las características específicas y los desafíos involucrados.

2.2.7. Búsqueda de software necesario

- Identificar y seleccionar las herramientas, bibliotecas y tecnologías necesarias para el desarrollo de la API y el *dashboard*.

2.2.8. Aprendizaje

- Familiarizarse con las tecnologías seleccionadas, como Flask y Angular, a través de recursos de aprendizaje en línea y documentación oficial.

2.2.9. Desarrollo inicial

- Crear los componentes iniciales de la API y el *front-end*, estableciendo la base para la funcionalidad futura.

2.2.10. Desarrollo del *front-end*

- Implementar las funcionalidades de interacción en el *front-end* y conectarlas con la API.

2.2.11. Desarrollo de la API y desarrollo

- Conectarse y obtener los datos de la base de datos.
- Implementar las funciones que procesaran los datos.

2.2.12. Testeo y adaptación

- Realizar pruebas exhaustivas en la API y el *dashboard* para identificar errores, problemas de rendimiento y posibles mejoras.
- Realizar ajustes y adaptaciones basados en los resultados de las pruebas y la retroalimentación del tutor.

2.2.13. Documentación

- Crear documentación detallada de la API REST y el *dashboard*, incluyendo guías de instalación, uso y descripciones de funcionalidades.
- Generar documentación técnica para facilitar el mantenimiento futuro y la posible expansión del proyecto.

Cada una de estas tareas se relaciona directamente con los objetivos planteados para el proyecto. El plan de trabajo se ha estructurado de manera que permita un enfoque ordenado y eficiente para lograr los resultados deseados. La asignación de responsabilidades, el seguimiento del progreso y la adaptación continua son elementos clave en el proceso de cumplir con los objetivos del proyecto.

Las diferentes tareas que hemos llevado a cabo las mostramos en el diagrama de Gantt donde hemos realizado una planificación por semanas (ver Figura 2.2.1).

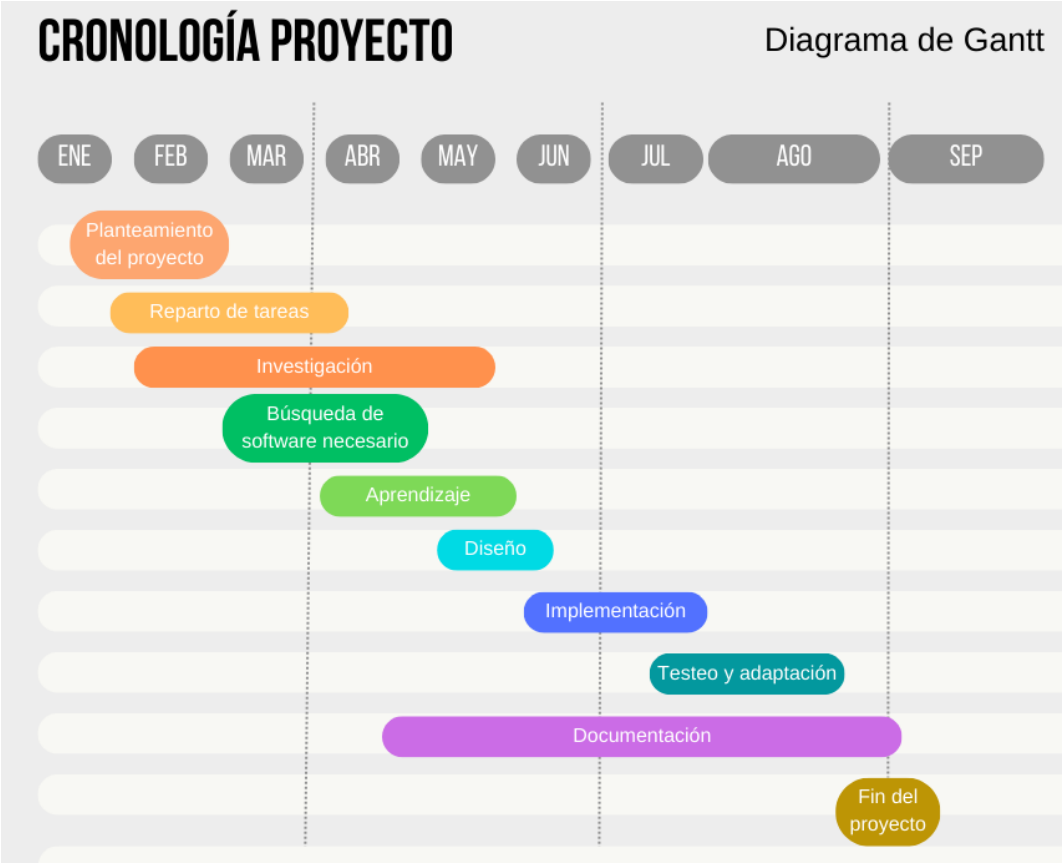


Figura 2.2.1: Diagrama de Gantt del proyecto

Para controlar que estas tareas se cumplen en los plazos establecidos, se han llevado a cabo diferentes reuniones con el tutor en las cuales se exponía el trabajo realizado y se determinaban las siguientes tareas a llevar a cabo.

2.3. Recursos humanos y técnicos

Los recursos que he usado para llevar a cabo el proyecto son:

- **Hardware**:

  - Ordenador de sobremesa utilizado también como servidor para la API y el *dashboard* con los siguientes componentes:
    - EVGA GeForce RTX 3060 XC GAMING 12GB GDDR6
    - Nfortec Sagitta RGB 750W 80 Plus Gold Full Modular
    - WD Blue SN550 SSD 1TB NVMe M.2 PCIe Gen 3
    - "Cooler Master MasterLiquid ML240L V2 RGB Kit de Refrigeración Líquida" (“Cooler Master MasterLiquid ML240L V2 RGB Kit de ... - PcComponentes”)
    - Corsair Vengeance LPX DDR4 3200 PC4-25600 32GB 4X8GB CL16 Negro
    - Asus PRIME Z690-P D4
    - Intel Core i5-12600K 3.7 GHz

- **Software**:

  - Sistema operativo Ubuntu 22.04 LTS. Será la distribución Linux principal con la que vamos a trabajar.

Presupuesto:

El equipo de desarrollo del producto software está compuesto por el alumno. El coste de un desarrollador será estimado a 22 euros por hora de trabajo. También se contará con la supervisión del tutor con el que se realizarán revisiones de 30 a una hora, por lo que estimaremos una media de 45 minutos por reunión cada dos semanas. Estimamos el coste de la supervisión a 65 euros la hora.

Tarea	Coste del desarrollador (22€/h)	Coste del tutor (65€/h)	Total
Análisis	330€	390€	720€
Aprendizaje	2.200€	292,5€	2.492,5€
Desarrollo API	880€	97,5€	977,5€
Desarrollo *Dashboard*	704€	97,5€	801,5€
Memoria	330€	90€	420€
Total	4.444€	967,5€	5.411,5€

Tabla 2.4. Coste del desarrollo

Para simplificar la presentación de los costos, se han agrupado las diferentes tareas de acuerdo con el diagrama de Gantt. A continuación, se detallan los grupos y sus componentes:

- **Análisis**: Incluye las tareas de Planteamiento del proyecto, Reparto de tareas, Investigación y Búsqueda de software necesario. El costo total estimado para esta fase es de 720€.
- **Aprendizaje**: Corresponde a la tarea de Aprendizaje según el diagrama de Gantt. El costo estimado es de 2.492,5€.
- **Desarrollo de la API**: Incluye las tareas de Diseño y Desarrollo inicial, Testeo y Adaptación, y Desarrollo de la API. El costo total estimado es de 977,5€.
- **Desarrollo del *dashboard***: Incluye las tareas de Diseño y Desarrollo inicial, Testeo y Adaptación, y Desarrollo del *Front-end*. El costo total estimado es de 801,5€.

- **Memoria**: Corresponde a la tarea de Documentación. El costo estimado es de 420€.

Si se añade el precio de adquisición del hardware utilizado durante el proyecto, que fue de aproximadamente 1.500€, el costo total del proyecto ascendería a **6.911,5€**. Este costo incluye tanto los gastos relacionados con el desarrollo y la supervisión, como el valor económico del hardware utilizado para llevar a cabo el proyecto en su totalidad.

Nótese que el coste por parte del trabajo del tutor en el desarrollo de la API y del *dashboard* coincide porque prácticamente se desarrollaron en paralelo. El equipo de desarrollo del producto de software se compone únicamente del alumno, quien ha invertido significativamente en su formación y habilidades a través de una serie de cursos de duración prolongada. Estos cursos han sido ofrecidos por destacadas compañías y se pueden verificar mediante sus correspondientes identificadores. La justificación del costo en aprendizaje se basa en la calidad y profundidad de la educación obtenida. A continuación, se presentan los cursos completados por el desarrollador:

- **Introducción al desarrollo web I**

  Google Actívate

  ID de la credencial: UTX 9FM VY3

- **Curso de Introducción al Desarrollo Web: HTML y CSS (2/2) Curso de Introducción al Desarrollo Web: HTML y CSS (2/2)**

  Google Actívate

  Expedición: abr. 2023

  ID de la credencial: GXK EJL QRP

- **Angular esencial**

  LinkedIn

  Fecha de finalización: 8 de marzo de 2023

  Duración: 4 h 14 min

  ID de la credencial: Aaq3mt-PcLLBvUcWHB1IW5ck3mrj

- **Fundamentos del desarrollo web: *Full Stack o Front-end***

  LinkedIn

  Expedición: feb. 2023

ID de la credencial: AdRgKrgzhD4NutOkp5z2mulNk27T

- **Desarrollo *full-stack* práctico: Configuración profesional de proyectos**

  LinkedIn

  Fecha de finalización: 5 de marzo de 2023

  Duración: 3 h 49 min

  ID de la credencial: AXNhqZmZOPvGv8aqRkcHpzHrXJIs

Estos cursos no solo reflejan el compromiso del desarrollador con la mejora continua de sus habilidades, sino que también son un testimonio de su capacidad para abordar desafíos complejos en el desarrollo de software. A medida que el desarrollador aplica los conocimientos adquiridos en estos cursos al proyecto, se espera que se traduzca en un producto de mayor calidad y funcionalidad.

Por lo tanto, el costo en aprendizaje refleja no solo el tiempo dedicado a la adquisición de habilidades, sino también la confianza en la capacidad del desarrollador para contribuir de manera significativa al proyecto y lograr sus objetivos.

3. Herramientas, técnicas y tecnologías empleadas en procesos de crawling web

En esta sección del trabajo, se realiza un análisis exhaustivo tanto de la tecnología subyacente en el proyecto como del problema que se aborda. Se estructura en varias secciones para proporcionar una visión completa del panorama actual y las contribuciones propuestas.

3.1. Herramientas para crawling web

Durante la investigación, se identificaron diversas herramientas que permiten el análisis y la visualización de las relaciones entre sitios en la *Darknet*. A continuación, se presentan algunas de las herramientas más destacadas en este campo:

- **OnionScan**: Esta herramienta se enfoca en el escaneo y análisis de servicios ocultos en la red Tor. Proporciona información detallada sobre la configuración de los sitios, su nivel de seguridad y otros detalles relevantes. Una característica notable de OnionScan es su capacidad para visualizar la red Tor, mostrando las conexiones entre los distintos sitios y nodos. [4]
- **TorFlow**: Proyecto que ofrece visualizaciones en tiempo real de la red Tor. Mediante gráficos informativos, TorFlow muestra datos sobre los nodos de entrada y salida, así como las interacciones entre los nodos y los servicios ocultos. Esta herramienta proporciona una visión dinámica de la red Tor. [5]
- **Ahmia**: Si bien Ahmia funciona principalmente como un motor de búsqueda para sitios ocultos en la red Tor, también ofrece una función de visualización. Esta característica muestra las relaciones entre los diferentes sitios, permitiendo una comprensión más profunda de la estructura de la *Darknet*. [6]
- **Darknet Mapper**: Diseñada para mapear y visualizar la estructura de la red Tor, esta herramienta ofrece información sobre nodos y servicios ocultos. A través de sus características, los usuarios pueden obtener una representación gráfica de las conexiones en la *Darknet*. [7]

C4Darknet: Un Enfoque Innovador

C4Darknet elaborada por colegas de la universidad, se destaca en este panorama de herramientas al abordar un enfoque distinto. A diferencia de las herramientas mencionadas anteriormente, que están orientadas principalmente a la red Tor, C4Darknet originalmente se concibió para ser utilizada en la red anónima I2P [8]. Sin

embargo, esta herramienta carecía de una interfaz gráfica interactiva y atractiva para sus usuarios, lo que limitaba su utilidad en la visualización de resultados.

Este proyecto tiene como objetivo principal llenar ese vacío al proporcionar a C4Darknet una interfaz gráfica basada en la web en forma de un *dashboard*. Al hacerlo, se permitirá a los usuarios explorar y analizar las relaciones entre sitios en la *Darknet* de manera más intuitiva y efectiva. La creación de esta interfaz mejorará significativamente la accesibilidad y la utilidad de C4Darknet, al brindar a los usuarios una experiencia visualmente atractiva que simplifica la comprensión de los resultados generados por la herramienta.

Para explicar el funcionamiento de esta herramienta, la Figura 3.1 extraída del estudio de Abellán Galera, Alberto [8] muestra el diseño del sistema, incluyendo actores, elementos y relaciones. Estos elementos o módulos son cruciales para llevar a cabo las tareas y funciones del procedimiento. A continuación, se describen brevemente:

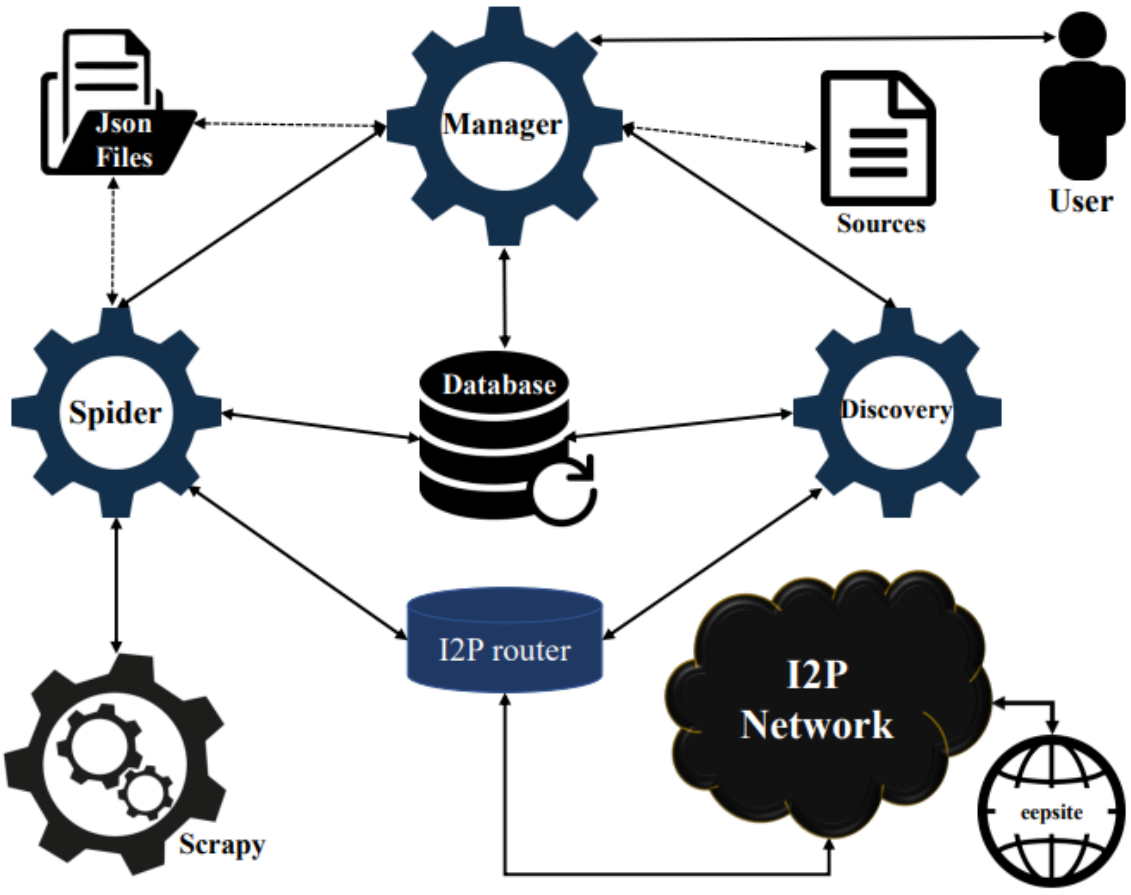


Figura 3.1. Diseño del sistema: actores, elementos y relaciones.

- **Archivos JSON**: Estos documentos contienen la información obtenida de los *eepsites*, un eepsite es un website en I2P, procesados durante el proceso de

crawling. Una vez que se completa el *crawling* de un *eepsite*, se genera un archivo JSON con el nombre del *eepsite* en el directorio "terminados". Durante el *crawling*, este archivo se encuentra en el directorio "en curso" y se elimina después de procesar su contenido y almacenarlo en la base de datos.

- **Fuentes**: La herramienta se alimenta de direcciones que corresponden a los nombres de *eepsites* en I2P, obtenidos de diversas fuentes.
- **Supervisor**: Este componente actúa como el núcleo de la herramienta y coordina todas las funciones. Supervisa los *eepsites* activos, pendientes, visitados y descartados. También inicia módulos que verifican la disponibilidad de los *eepsites*, inicia el proceso de *crawling* para cada *eepsite* y gestiona los archivos que contienen las semillas. Además, genera un registro de todas las operaciones y su estado.
- **Descubridor**: Este módulo verifica si un *eepsite* está activo, es decir, en línea y operativo, antes de que el módulo de extracción realice el *crawling*. Clasifica los sitios en activos (estado PENDIENTE) y no activos (estado DESCARTADO).
- **Extractor**: Es responsable de realizar el *crawling* de un *eepsite*, explorando todas sus páginas hasta un cierto nivel de profundidad para extraer información relevante.
- **Marco de trabajo *Scrapy***: El módulo de extracción se basa en el marco de trabajo *Scrapy*, especializado en el raspado y recorrido de sitios web en Python, que simplifica tareas como la configuración de proxy, la omisión de las instrucciones del archivo robots.txt y el control de parámetros de recorrido.
- **Base de datos**: Para administrar la gran cantidad de información recopilada durante el *crawling*, así como para el registro y seguimiento de operaciones, se utiliza una base de datos organizada.
- **Red I2P**: Esta red es el objetivo principal, donde se encuentran los *eepsites* de los que se extrae información. Por lo tanto, se requiere una conexión con esta red.
- **Enrutador I2P**: Esta instancia I2P permite la conexión de los módulos de descubrimiento y extracción con la Red I2P. Puede configurarse en varios modos, como el modo *Floodfill*, que también es útil para obtener más fuentes o direcciones semilla.

En el contexto de los grafos y sus relaciones en este proyecto, es importante entender la distinción entre dos tipos de enlaces: los "*incoming*" (entrantes) y los "*outgoing*" (salientes).

- **Enlaces entrantes (*incoming*)**: Estos son todos los enlaces que provienen de otras páginas web y apuntan hacia un *eepsite* específico. En otras palabras, son los enlaces que llevan a los usuarios desde otros sitios web hasta el *eepsite* en cuestión. Estos enlaces pueden considerarse como "referencias" o "vínculos de entrada" hacia el *eepsite* y pueden proporcionar información sobre quién está vinculado a ese *eepsite* desde otros lugares de la web.
- **Enlaces salientes (*outgoing*)**: Por otro lado, los enlaces salientes son todos los enlaces desde un *eepsite* hacia otras páginas web. Estos enlaces representan los destinos a los que el *eepsite* dirige a sus usuarios. En otras palabras, son los vínculos que llevan a los usuarios desde el *eepsite* hacia otros lugares en la web. Estos enlaces salientes pueden revelar a dónde se dirigen los usuarios desde el *eepsite* y qué recursos externos están relacionados con él.

En resumen, al diferenciar entre enlaces entrantes (*incoming*) y enlaces salientes (*outgoing*), podemos comprender mejor las relaciones y conexiones que existen entre un *eepsite* y el resto de la web. Los enlaces entrantes nos dicen quién se vincula con el *eepsite*, mientras que los enlaces salientes nos indican hacia dónde se dirigen los usuarios desde ese *eepsite*. Esta distinción es fundamental para analizar y comprender la conectividad de la Deep Web de manera más detallada.

3.2. Tecnología Subyacente

Para el desarrollo de este proyecto, se han utilizado diversas tecnologías que desempeñan roles clave en su implementación y colaboración. Python Flask ha sido el motor detrás de la API REST que permite acceder y gestionar los datos recopilados durante el proceso de "*crawling*". Angular v16 se ha empleado para construir el *dashboard* interactivo, aprovechando su robusta estructura de componentes y capacidades de renderizado dinámico. La combinación de HTML y SCSS ha permitido la creación de interfaces de usuario atractivas y responsivas.

La elección de Visual Studio Code como entorno de desarrollo integrado (IDE) ha sido determinante debido a su flexibilidad y soporte para diversas tecnologías. La gestión de dependencias de Python se ha llevado a cabo utilizando herramientas de entorno virtual, asegurando la coexistencia de diferentes versiones de paquetes y librerías. Para el

almacenamiento de datos, se optó por MySQL y se utilizó PHPMyAdmin como interfaz de gestión y visualización de la base de datos.

GitHub se ha convertido en una herramienta fundamental para el desarrollo de proyectos de software colaborativo, aunque en este caso, el proyecto ha sido abordado en solitario. La plataforma ha permitido el seguimiento preciso de los cambios realizados en el código a lo largo del tiempo, la gestión de versiones y la documentación detallada de las actualizaciones.

Aunque el proyecto ha sido desarrollado por un individuo, el uso de GitHub ha proporcionado una metodología estructurada para mantener el código organizado y mantener un historial claro de las decisiones y modificaciones realizadas. Se ha enfocado en la adopción de "*commit* convencionales", una práctica que implica proporcionar mensajes claros y descriptivos en cada confirmación. Esto mejora la trazabilidad y comprensión de los cambios realizados, facilitando la colaboración futura y la comprensión del desarrollo.

La elección de utilizar GitHub y aplicar prácticas de colaboración incluso en proyectos individuales demuestra un compromiso con las mejores prácticas de desarrollo y una mentalidad orientada a la calidad y la transparencia.

La inclusión estratégica de Google Icons y Angular Material ha tenido un impacto significativo en la estética y funcionalidad del *dashboard*. Estas herramientas han sido seleccionadas con cuidado para realzar la experiencia del usuario al interactuar con la interfaz.

Al adoptar Google Icons, se ha dotado al *dashboard* de un conjunto versátil de iconos que no solo son visualmente atractivos, sino que también comunican de manera efectiva la función de cada elemento. Esto contribuye a una comprensión intuitiva de la navegación y las opciones disponibles, mejorando la usabilidad y la eficiencia.

Por otro lado, la integración de Angular Material ha permitido aprovechar patrones de diseño bien establecidos y componentes prediseñados que siguen las mejores prácticas de diseño de interfaces. Los elementos como botones, tarjetas y barras de navegación han sido implementados de manera coherente y elegante, creando un aspecto uniforme en toda la aplicación. Además, Angular Material también aporta funcionalidades interactivas y animaciones sutiles que enriquecen la experiencia visual y de uso.

Esta combinación de Google Icons y Angular Material no solo ha embellecido la interfaz, sino que también ha impactado positivamente en la sensación general de calidad y profesionalismo. Los usuarios se benefician de una apariencia moderna y atractiva que se traduce en una interacción más agradable y satisfactoria con el *dashboard*.

En cuanto al sistema operativo, Linux Ubuntu 22.04 ha sido la plataforma elegida para la implementación del proyecto, aprovechando su robustez y flexibilidad en entornos de desarrollo.

3.3. Problema Abordado

El problema que se aborda en este proyecto es la carencia de una visión integral y visual de la conectividad entre sitios en las *Darknets*. A pesar de que muchos trabajos anteriores han investigado y analizado contenido e información en estas redes ocultas, pocos se han centrado en cómo los sitios en las *Darknets* se relacionan entre sí.

3.4. Novedad y Contribución del Proyecto

La contribución principal de este proyecto radica en la creación de un *dashboard* interactivo que proporciona una visión en tiempo real del grafo completo de conectividad entre sitios en las *Darknets*. Esta visualización detallada, alimentada por una API REST en Python Flask y desarrollada en Angular v16, presenta información a través de gráficos, tablas y visualizaciones diversas. Además, se ha implementado la posibilidad de interactuar con los datos y personalizar los resultados en función de las preferencias del usuario.

Un aspecto destacado es la recomendación de utilizar la herramienta Gephi para visualizar el grafo completo, lo que amplía la utilidad del proyecto al facilitar la importación de archivos CSV generados desde el *dashboard* y permitir análisis más avanzados de la conectividad entre sitios, la recomendación de esta herramienta se debe a que debido al gran número de nodos y aristas del grafo el navegador no es capaz de renderizar tantos elementos simultáneamente.

3.5. Alternativas Consideradas

3.5.1. Para la API REST en Python

Django: Un *framework* ampliamente utilizado para aplicaciones web en Python. Ofrece una gran cantidad de funcionalidades, pero puede ser más complejo y pesado para proyectos más pequeños y específicos.

3.5.2 Para el *dashboard* en Angular

React: Un popular *framework* de JavaScript para construir interfaces de usuario. Ofrece una amplia comunidad y rendimiento, pero la elección final se inclinó hacia Angular

debido a su capacidad de crear aplicaciones más grandes y la integración con TypeScript.

3.5.3. Para el almacenamiento de datos

PostgreSQL: Una base de datos relacional con características avanzadas. La elección de MySQL se basó en la familiaridad y su idoneidad para proyectos de tamaño medio.

3.6. Ventajas de las elecciones

Python Flask: La elección de Flask para la API REST se justifica por su simplicidad y ligereza. En comparación con alternativas más complejas como Django, Flask se adapta mejor a proyectos más específicos y pequeños, proporcionando un enfoque más simple y directo para construir una API funcional.

Angular v16: La selección de Angular v16 como base para el *dashboard* se basa en su robusta estructura de componentes y su capacidad para crear aplicaciones web escalables. Aunque React es popular, Angular ofrece un enfoque más completo para aplicaciones de mayor envergadura y se integra eficientemente con TypeScript, además del dominio previo del *framework*.

MySQL y PHPMyAdmin: La preferencia por MySQL se debe a su amplia adopción y a su eficacia en la gestión de datos relacionales. La elección de PHPMyAdmin como interfaz de gestión facilita la administración de la base de datos de manera visual y accesible.

Linux Ubuntu 22.04: La elección de Linux Ubuntu como sistema operativo se basa en su confiabilidad y flexibilidad para el desarrollo. Ubuntu es conocido por su sólido rendimiento y amplia comunidad de soporte.

En resumen, las elecciones tecnológicas en este proyecto se derivan de una evaluación rigurosa de alternativas similares disponibles en el mercado. Cada elección se ha justificado en función de su adecuación al alcance del proyecto y las ventajas específicas que ofrecen. Python Flask y Angular v16 brindan un equilibrio entre simplicidad y funcionalidad, MySQL y PHPMyAdmin simplifican la gestión de datos, y Linux Ubuntu 22.04 proporciona un entorno estable y amigable para el desarrollo. Estas elecciones permiten abordar eficazmente el problema planteado y desarrollar una solución robusta y efectiva.

4. Diseño e implementación de la propuesta

En este capítulo, se exponen los resultados del proceso de desarrollo, centrándose en la presentación de la interfaz web desarrollada y proporcionando una guía detallada sobre su utilización. Además, se describen en profundidad las funcionalidades de las llamadas a la API Rest, las cuales desempeñan un papel fundamental en esta implementación.

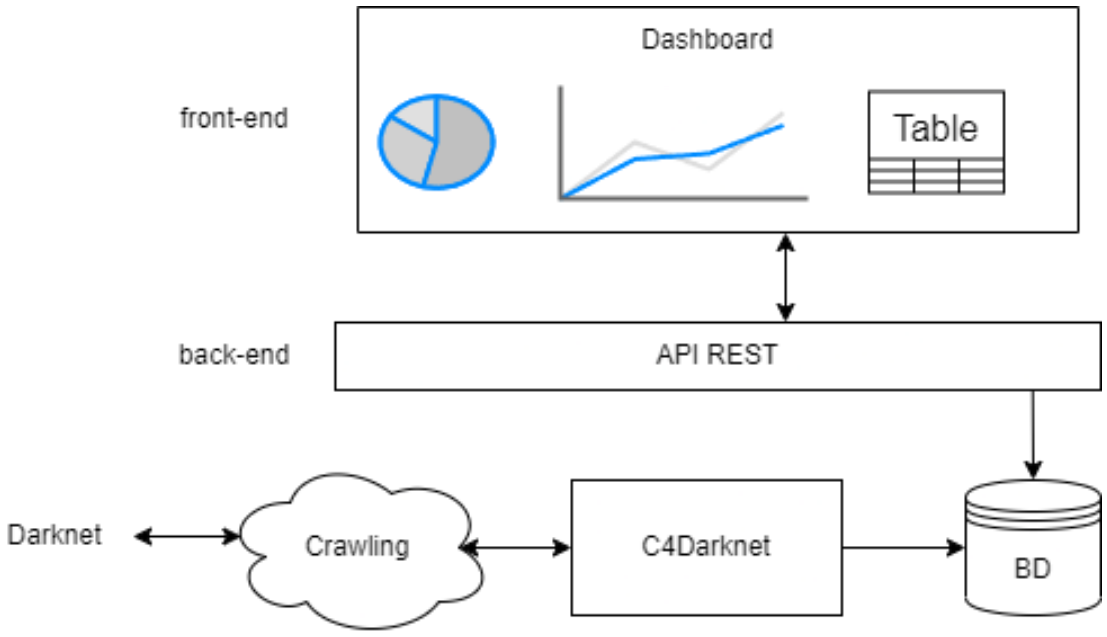


Figura 4. Esquema de relaciones entre el software a desarrollar en este proyecto.

En la Figura 4 se presentan todas las relaciones entre los software que desempeñarán un papel fundamental en este proyecto. El esquema es una imagen extraída de una de las primeras reuniones con el tutor, donde se esquematiza la interacción entre las diferentes herramientas y componentes que se utilizarán para llevar a cabo este proyecto.

4.1. Interfaz web

La elección del *framework* Angular ha sido estratégica, aprovechando su capacidad para desarrollar aplicaciones web dinámicas y altamente responsivas. El diseño de la interfaz se enfoca en lograr una disposición equilibrada de elementos visuales, como gráficos, tablas y controles de interacción, para ofrecer una experiencia intuitiva y efectiva al usuario. A continuación, describiremos en detalle cada uno de los componentes que hemos desarrollado.

Dada la diversidad de dispositivos y pantallas con diferentes resoluciones que se utilizan para navegar por internet, hemos priorizado la creación de una interfaz web responsive. Para lograrlo, hemos implementado estrategias como el uso de "*display flex*" y "*display grid*". Esto asegura que la experiencia del usuario no se vea comprometida sin importar el dispositivo que utilice. Nuestro objetivo es permitir a los usuarios acceder al *dashboard* desde su dispositivo preferido, manteniendo la funcionalidad y la estética de la interfaz.

A continuación, presentamos una captura de pantalla que muestra la distribución de las carpetas y los componentes que hemos desarrollado en nuestra aplicación:

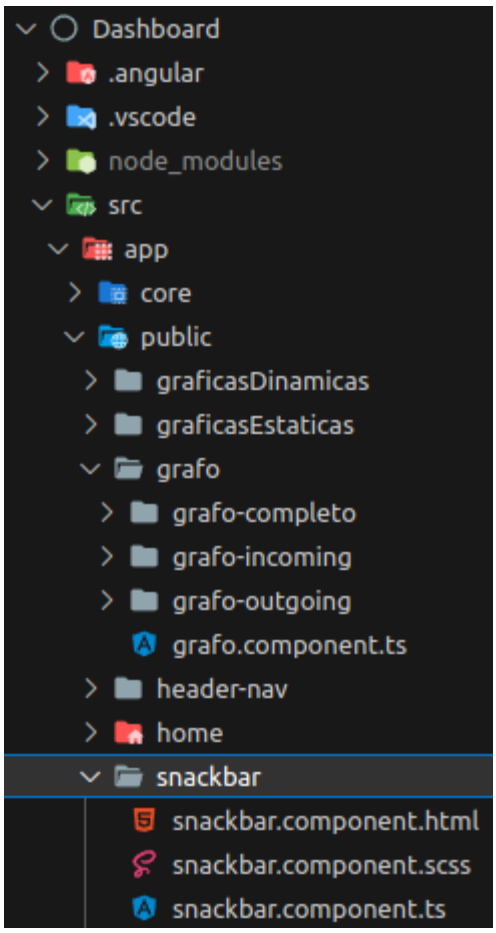


Figura 4.0.2: Estructura de Carpetas

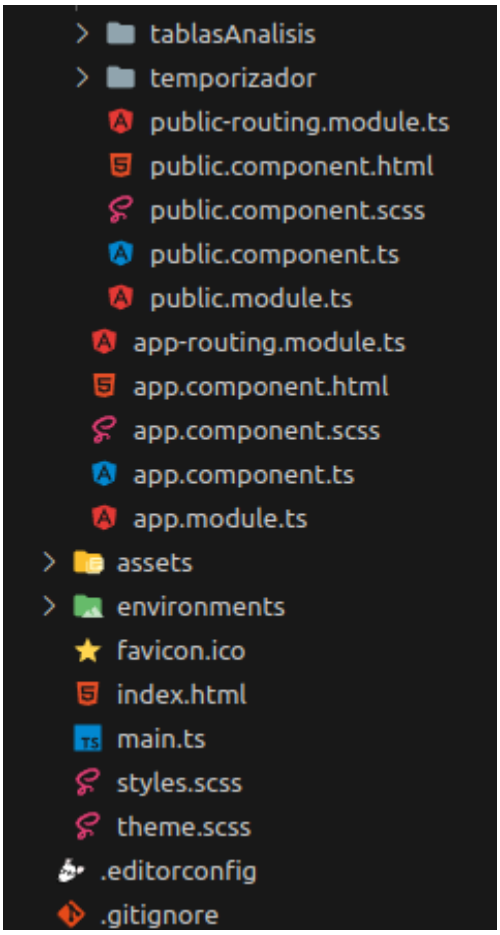


Figura 4.0.3: Estructura de Carpetas (1)

En concreto cuenta con un total de 7 módulos, 14 componentes, 4 inyectables, 2 clases, 3 interfaces y 7 rutas accesibles por el usuario.

Esta estructura de carpetas refleja nuestra aproximación modular al desarrollo, donde hemos creado un componente separado para cada elemento importante en el *dashboard*. Dentro de la carpeta de cada componente, se encuentran los archivos

TypeScript (`.ts`), HTML (`.html`) y de hojas de estilo (`.scss`) asociados. Esta organización en carpetas y archivos individuales por componente nos ha permitido mantener un código limpio y organizado, así como facilitar la escalabilidad y el mantenimiento del proyecto.

Cada componente encapsula su propia lógica y aspecto visual, lo que mejora la reusabilidad y modularidad de nuestro código. En la Figura 4.0.4. y 4.0.5. mostramos el diagrama de paquetes del proyecto Angular, generado con el paquete *compodoc*.

En la figura 4.0.10. podemos ver su diagrama de clases.

Figura 4.0.4. y Figura 4.0.5. Diagrama de paquetes front-end

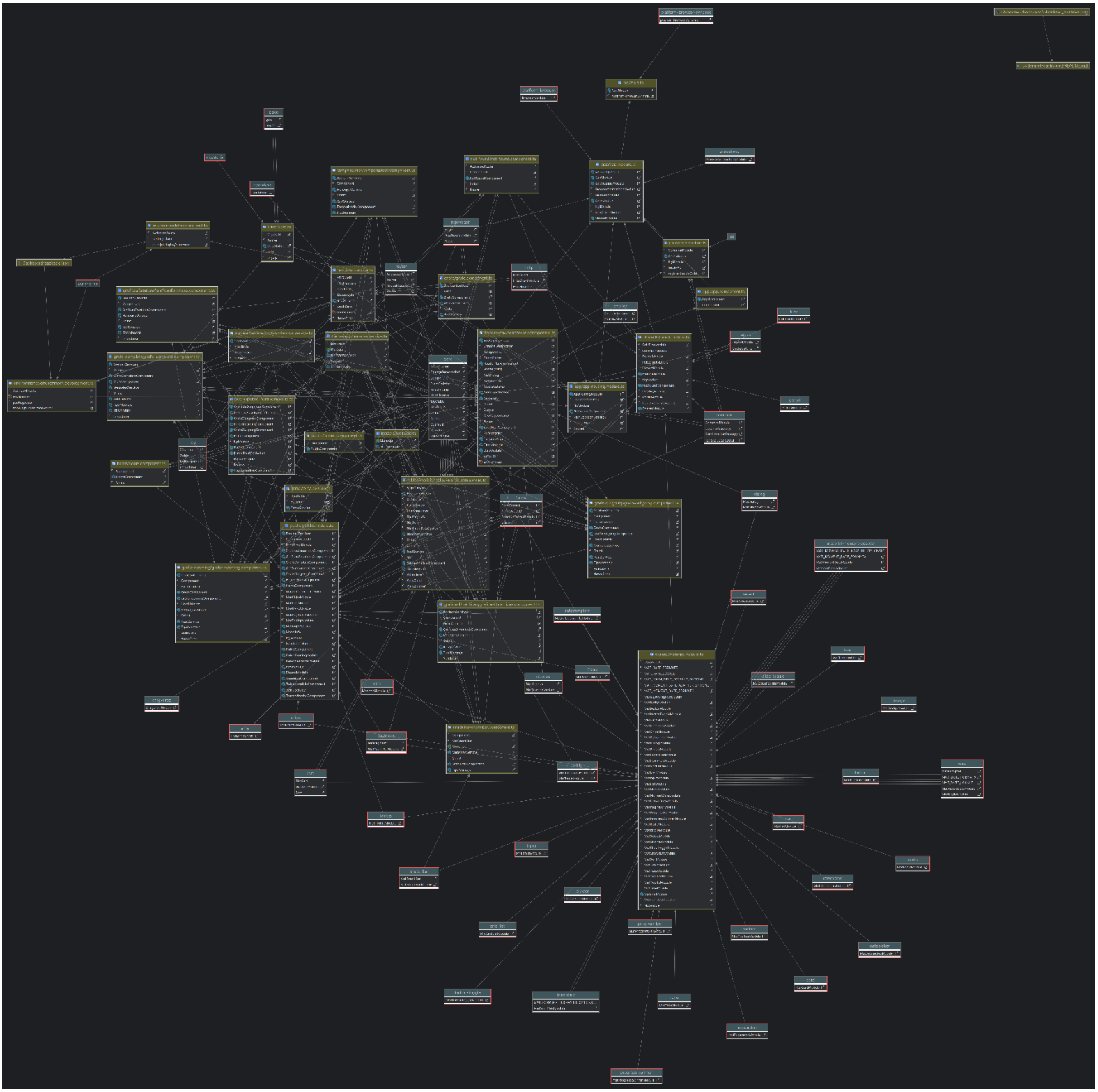


Figura 4.0.10. Diagrama de clases fron-end

Antes de comenzar el desarrollo de la interfaz web, se realizaron bocetos preliminares para definir la estructura de las diversas vistas del *dashboard*. Estos mockups se presentan en las Figuras 4.0.6, 4.0.7, 4.0.8 y 4.0.9. Estos bocetos proporcionaron una

visión visual inicial de cómo se organizarían y presentarían los datos en el *dashboard*, lo que sirvió como punto de partida para el diseño y desarrollo de la interfaz final.

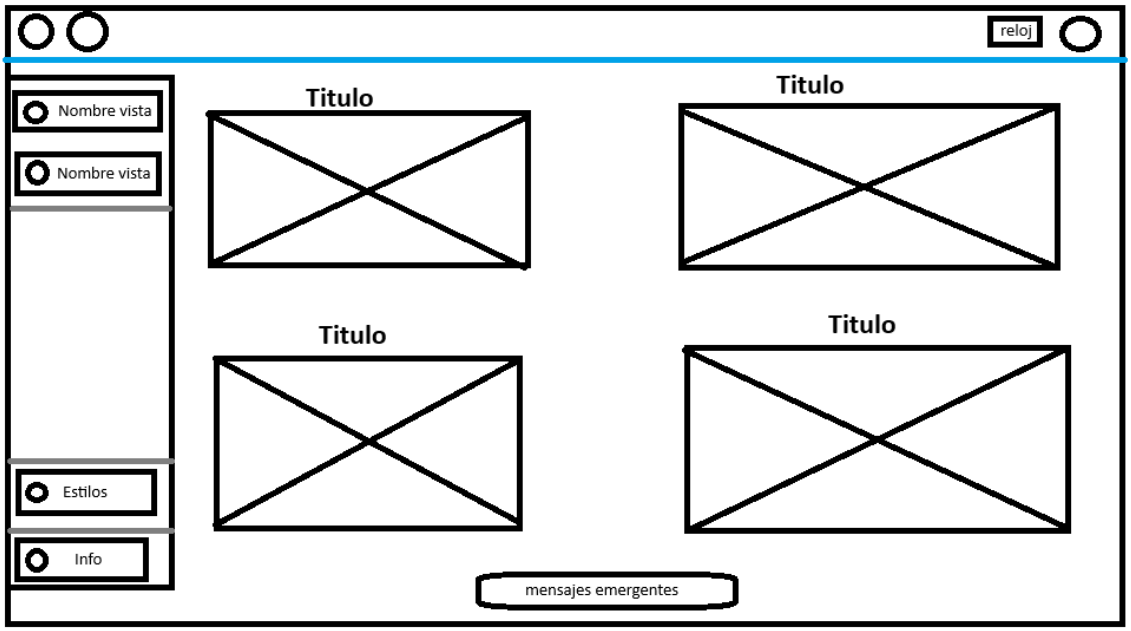


Figura 4.0.6. Mockup gráficas estáticas

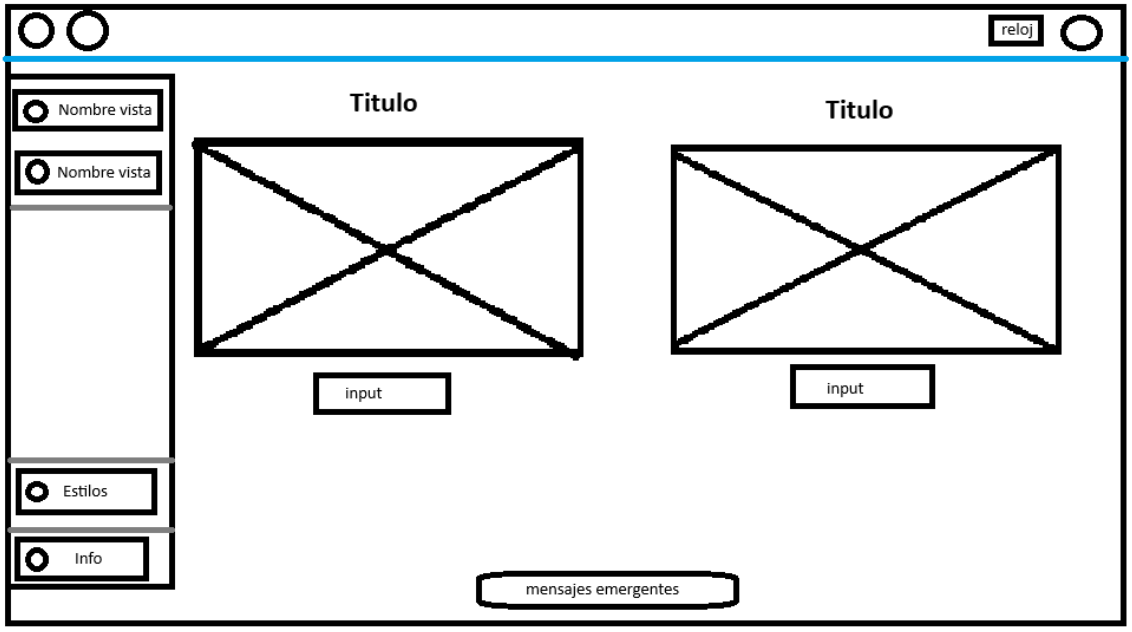


Figura 4.0.7. Mockup gráficas dinámicas

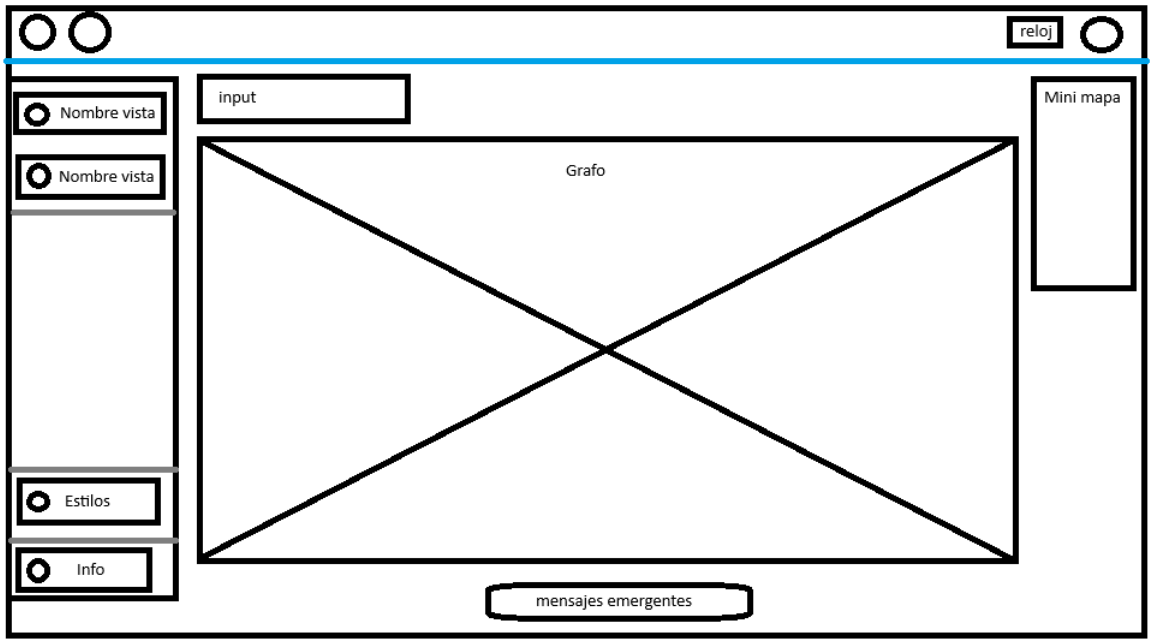


Figura 4.0.8. Mockup grafos

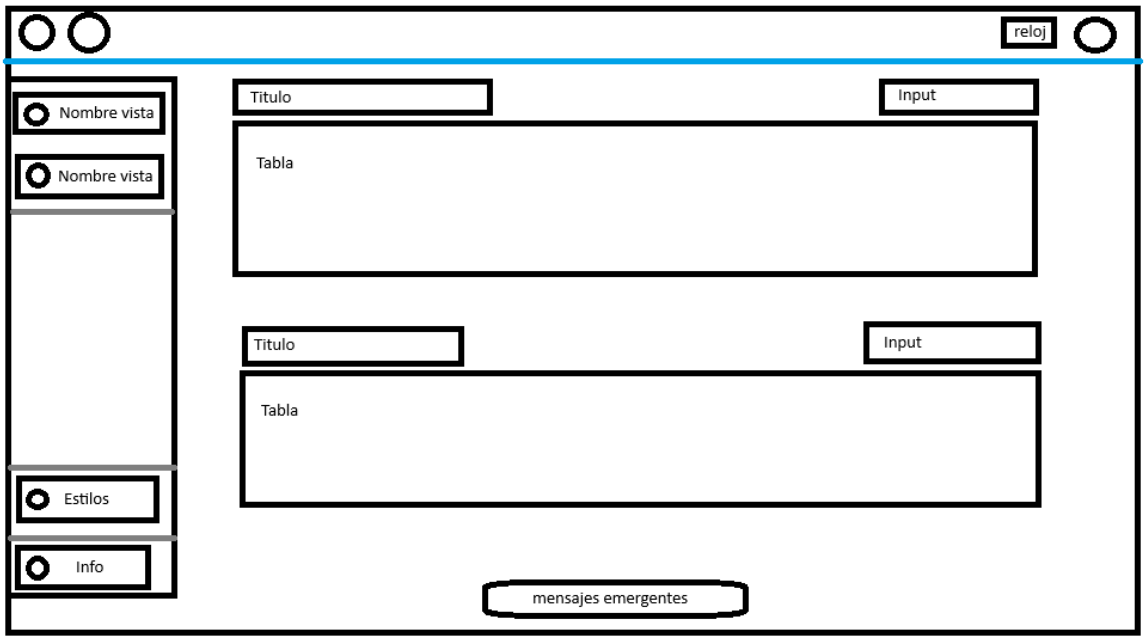


Figura 4.0.9. Mockup tablas

En resumen, nuestra interfaz web se caracteriza por su capacidad de respuesta, diseño equilibrado y enfoque en la experiencia del usuario. Buscamos proporcionar una plataforma accesible y amigable que permita a los usuarios interactuar con el *dashboard* de manera eficiente y cómoda, sin importar el dispositivo que utilicen para acceder a él.

4.1.1. Encabezado

Figura 4.1.1: Encabezado de la Interfaz web

El encabezado de nuestra página web presenta una estructura sencilla y funcional que facilita la navegación y proporciona información relevante al usuario. En el extremo izquierdo del encabezado, se encuentran dos botones que desempeñan funciones esenciales. El primer botón despliega el "*sidenav*", una herramienta que agiliza la navegación entre las diversas vistas disponibles en nuestro panel de control. En secciones posteriores, exploraremos en detalle las numerosas funciones ofrecidas por el "*sidenav*". El segundo botón redirige al usuario a la página de inicio del panel de control.

En la parte derecha del encabezado, se muestra un temporizador que indica el tiempo restante para la próxima actualización de la caché. Junto a este, se encuentra un botón con una descripción emergente (*mat tooltip*) que proporciona información explicativa sobre su funcionalidad. Este botón tiene la función de forzar el refresco de la caché, permitiendo al usuario actualizar manualmente los datos almacenados en la caché en cualquier momento.

Justo debajo de estos elementos, una línea coloreada con el tono de énfasis seleccionado por el usuario actúa como un componente distintivo. Esta línea cumple un doble propósito: separa visualmente el encabezado del resto del sitio web y, al mismo tiempo, funciona como una barra de progreso. Durante las consultas a la API, esta barra se anima, ofreciendo una representación visual del proceso en curso. Cuando una consulta está en proceso y la barra avanza, el usuario recibe una señal visual clara de que su solicitud está siendo atendida, mejorando la experiencia al brindar información visualmente informativa y atractiva.

4.1.2. *Sidenav*

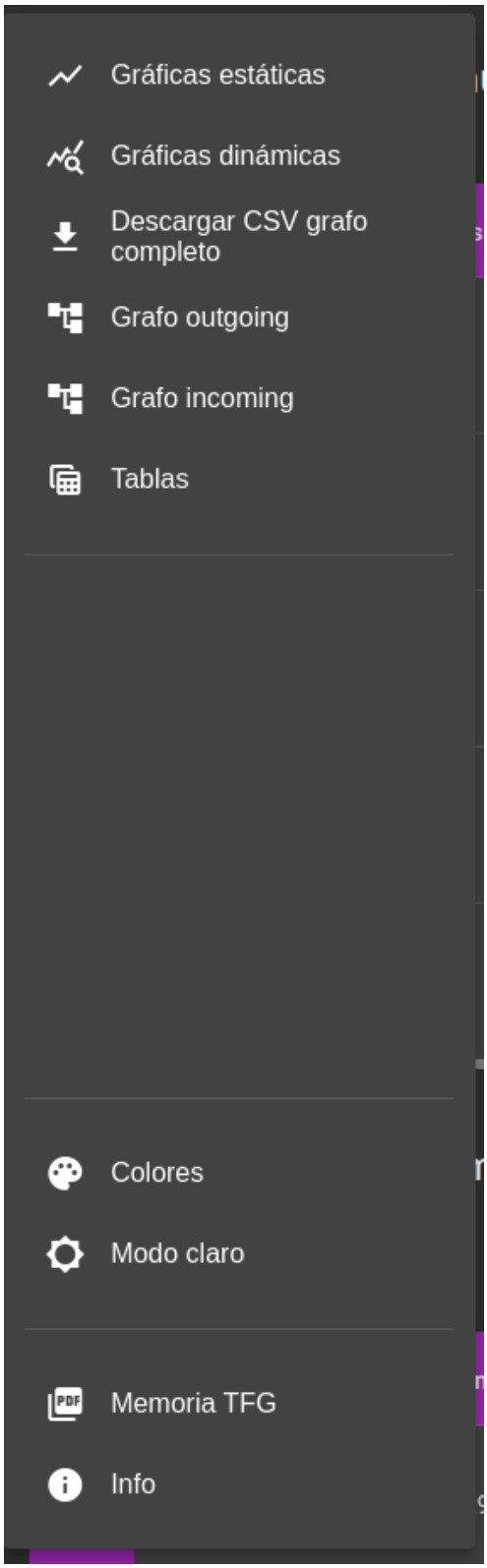


Figura 4.1.2: Sidenav de la Interfaz web

Este componente desempeña un papel fundamental al proporcionarnos una manera intuitiva de navegar por las diversas vistas y recursos disponibles en nuestro *dashboard*. El diseño del *sidenav* se organiza en secciones claramente definidas, separadas por líneas sutiles. En la primera sección, se presentan las distintas vistas, agrupadas según su funcionalidad. En los puntos siguientes, exploraremos detalladamente cómo aprovechar estas vistas y cómo interactuar con ellas de manera efectiva.

En la segunda sección del *sidenav*, se brinda la opción de alternar entre los modos de visualización claro u oscuro, según las preferencias del usuario. Además, se otorga la libertad de seleccionar un color de énfasis personalizado, lo que permite una experiencia de usuario aún más personalizada y agradable.

Por último, la tercera sección del *sidenav* provee acceso a información relevante relacionada con el *dashboard*, enriqueciendo la experiencia del usuario al ofrecer un acceso conveniente a detalles y recursos adicionales.

4.1.3. *Snackbar*

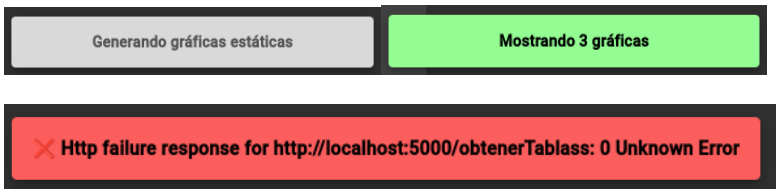


Figura 4.1.3: Snackbar

La incorporación del componente *snackbar* en nuestro *dashboard* desempeña un papel esencial al proporcionar una forma efectiva de transmitir mensajes emergentes al usuario. Estos mensajes temporales emergen en la parte inferior de la pantalla y están diseñados para desaparecer después de un período predefinido. La finalidad de estos mensajes es mantener al usuario informado sobre el estado y las acciones del *dashboard*, contribuyendo así a una interfaz gráfica amigable, intuitiva y de fácil comprensión.

El *snackbar* tiene la capacidad de mostrar mensajes en cuatro categorías diferentes: error, correcto, informativo y advertencia. Según la categoría del mensaje, el diseño del *snackbar* adoptará un color específico que permitirá al usuario identificar de manera rápida y visual la naturaleza del mensaje que se está mostrando.

En resumen, el componente *snackbar* mejora significativamente la experiencia del usuario al proporcionar notificaciones contextuales que mantienen al usuario siempre al tanto de lo que está ocurriendo en el *dashboard*. La elección de colores distintivos para las categorías de mensajes garantiza que la interpretación visual sea rápida y eficiente.

4.1.4. Gráficas estáticas

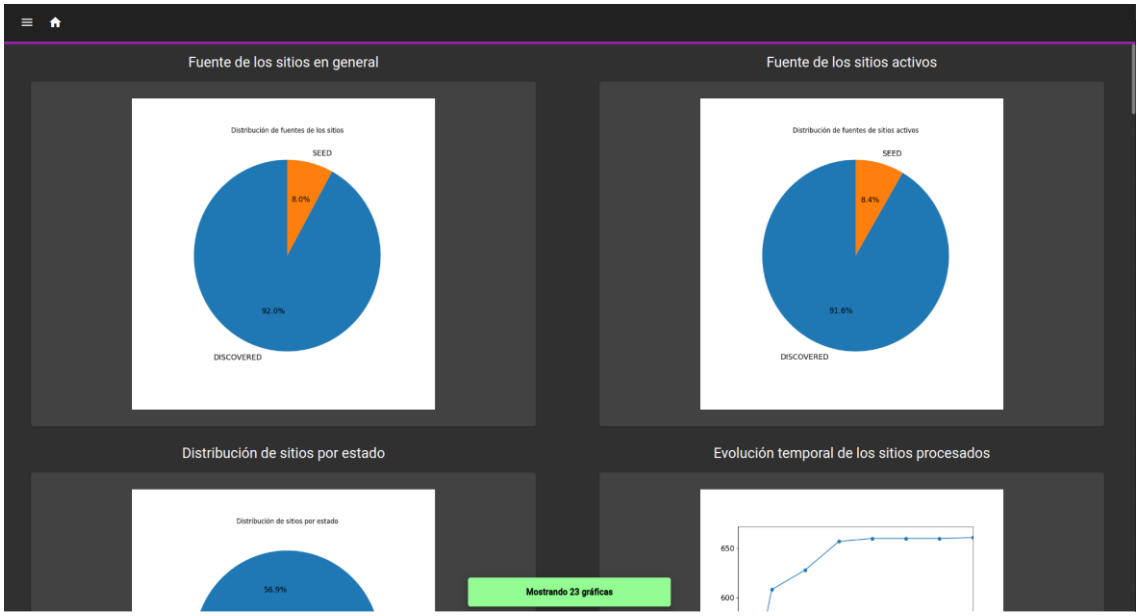


Figura 4.1.4: Vista de las gráficas estáticas

La vista de gráficas estáticas despliega información crucial acerca del proceso de "*crawleo*". Esta vista tiene la particularidad de no requerir una interacción activa por parte del usuario. Una vez que se accede a esta sección, la llamada a la API se ejecuta automáticamente. Como resultado, se obtienen las imágenes que representan las gráficas ya procesadas. Angular, por su parte, contribuye a mejorar la apariencia visual de estas gráficas, brindando una experiencia más agradable al usuario.

A diferencia de otras vistas, en esta no se espera una participación directa del usuario. En lugar de eso, el usuario es recibido con los resultados procesados de forma inmediata. Esta simplificación en la interacción permite al usuario obtener información relevante sin la necesidad de intervención manual, enriqueciendo así la experiencia general del usuario en el *dashboard*.

Las gráficas generadas son:

- Fuente de los sitios en general: muestra el porcentaje de sitios almacenados en la base de datos en función de cómo se accedió a dicho sitio.
- Fuente de los sitios activos: muestra el porcentaje en función a como se accedieron a los sitios catalogados como activos.
- Distribución de sitios por estado.
- Evolución temporal de los sitios procesados.
- Evolución temporal de los sitios *crawleados.*
- Número de sitios con *crawling* finalizado tras el primer día VS Número de sitios con *crawling* finalizado tras el último día.

- Histograma de los sites *crawleados* tras el primer día en función de los *outgoing* sites, se salta por ahora.
- Tiempo que tarda en *crawlear* los sites.
- relación entre duración y número de páginas.
- relación entre duración y los intentos de *discovering*.
- Análisis del número de palabras en los sitios.
- Análisis del número de scripts en los sitios.
- Análisis del número de imágenes en los sitios.
- Análisis de la conectividad *outgoing*.
- Análisis de enlaces *outoging* sin contar los que tienen 0.
- Análisis de la relación entre el número de páginas y *outgoing*.
- Análisis de los nodos *incoming*.
- Análisis de la cantidad de sitios *incoming* sin contar los que tienen 0.
- Análisis de sitios aislados y su conectividad.
- Análisis número de páginas.

4.1.5. Gráficas dinámicas

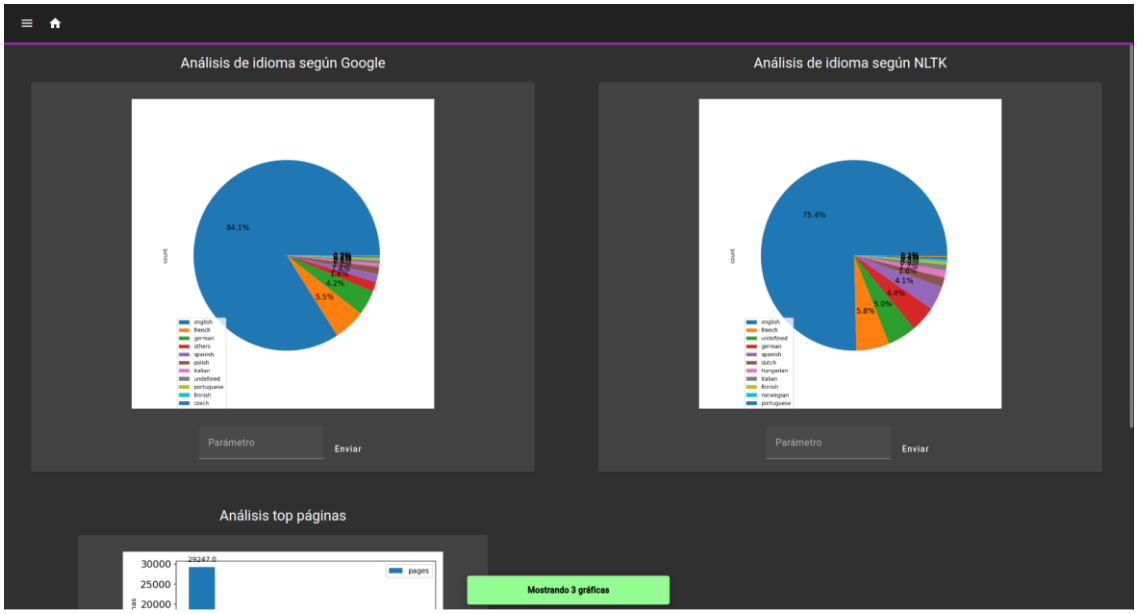


Figura 4.1.5: Vista de las gráficas dinámicas

En contraste con las gráficas estáticas, la vista de gráficas dinámicas ofrece la posibilidad de interacción por parte del usuario. Como se puede apreciar en la imagen, bajo cada gráfica se encuentra un campo de entrada (input), que permite al usuario introducir un valor. Este valor tiene el propósito de modificar la visualización de la gráfica en función del parámetro proporcionado. Por ejemplo, en el caso de las gráficas de

análisis de idioma, el usuario puede establecer el límite de páginas necesario para agruparlas en la categoría de "otros".

Esta funcionalidad ofrece un nivel adicional de personalización y adaptabilidad al usuario, ya que puede ajustar los parámetros según su interés o necesidades específicas. La capacidad de interactuar con las gráficas y modificarlas de manera instantánea permite una exploración más profunda y una comprensión más completa de los datos presentados en el *dashboard*.

Las gráficas generadas son:

- Análisis de idioma (según Google)
- Análisis de idioma (según *NLTK*)
- TOP n sitios con más paginas

4.1.6. Descargar CSV grafo completo

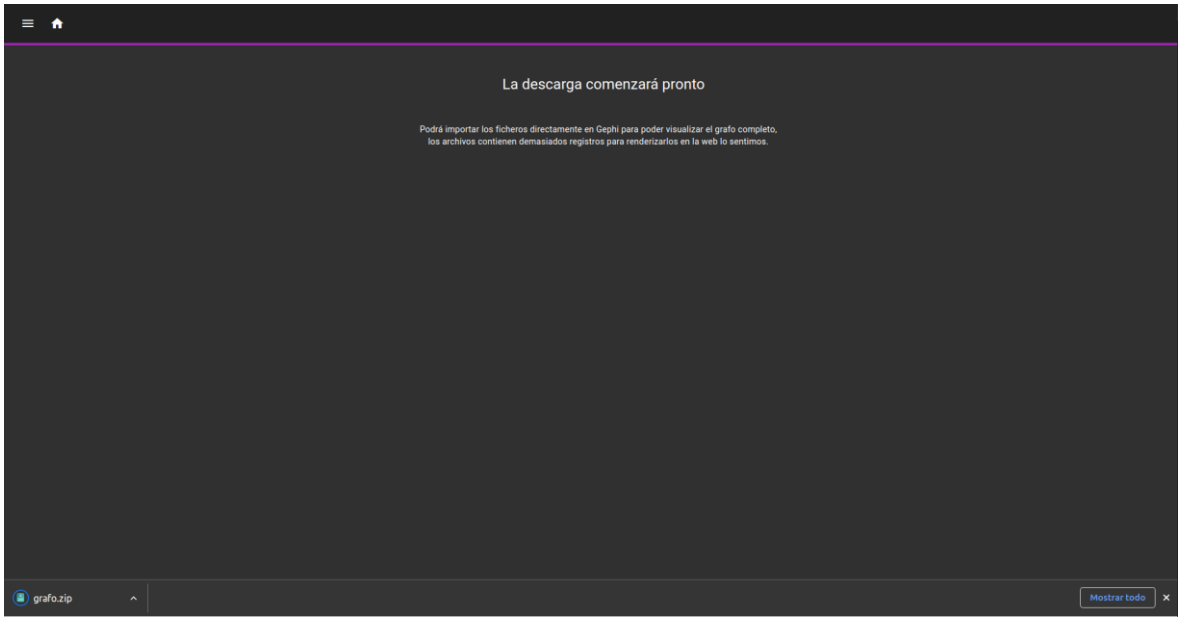


Figura 4.1.6: Vista descargar csv grafo completo

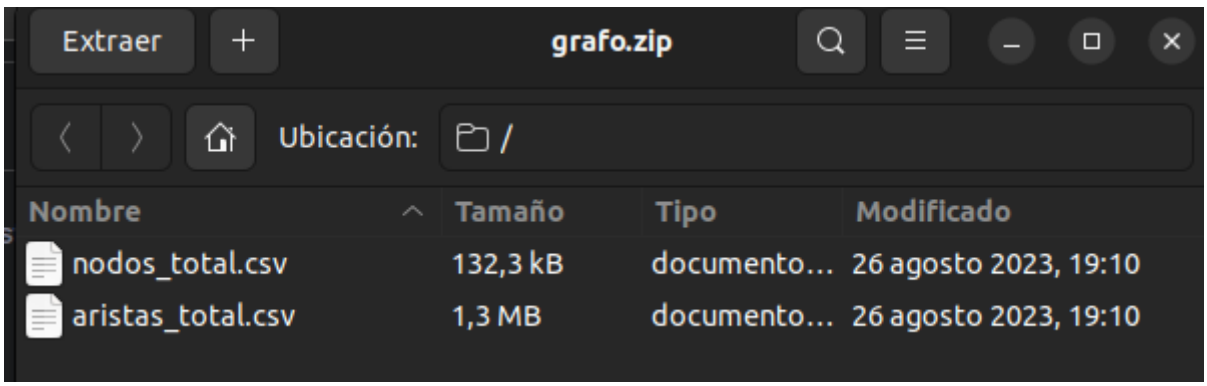


Figura 4.1.7: Contenido del archivo descargado en “descargar csv grafo completo”

Al acceder a esta vista, se activa automáticamente la descarga de un archivo comprimido en formato ZIP. Dentro de este archivo, se encuentran dos archivos con extensión CSV. Estos archivos están preparados para ser importados directamente en herramientas como Gephi, permitiendo así la visualización completa de un grafo que representa el proceso de "*crawleo*".

Originalmente, se consideró la opción de visualizar el grafo completo dentro del propio *dashboard*. Sin embargo, dado que el número de nodos y aristas podría ser excesivamente grande, esto podría ocasionar problemas al renderizar el grafo en el navegador, debido a las limitaciones de recursos. Como alternativa, se optó por la descarga de los archivos CSV, lo que brinda a los usuarios la capacidad de explorar el grafo en Gephi o en otras herramientas de visualización, donde se pueden gestionar de manera más efectiva los recursos necesarios para representar grafos complejos.

Vale la pena destacar que las otras vistas de grafos en el *dashboard* permiten la visualización de grafos con un número ajustable de nodos, utilizando parámetros para determinar cuántos nodos desean ser visualizados con los máximos enlaces de salida o entrada. Esta elección se hizo para proporcionar una experiencia óptima al usuario, manteniendo un equilibrio entre la exploración completa del grafo y la capacidad de representación en el navegador.

4.1.7. Grafo *outgoing*

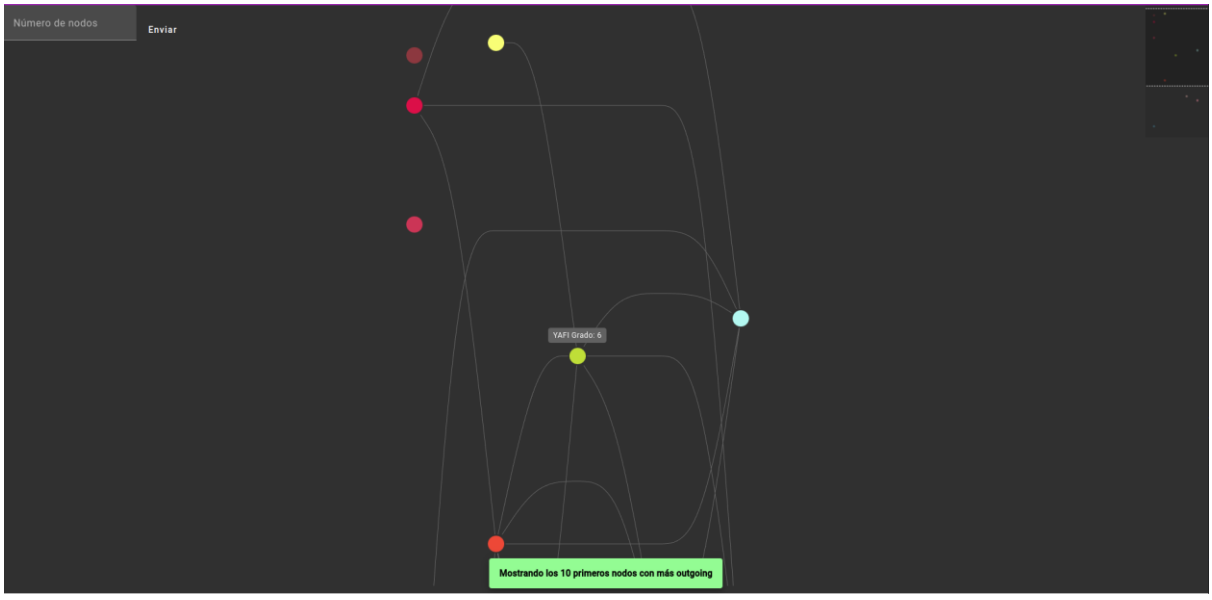


Figura 4.1.8: Vista grafo outgoing

La vista de Grafo *Outgoing* presenta una representación gráfica de los nodos con el mayor número de enlaces salientes (*outgoing*). Aquí, los usuarios tienen la capacidad de definir el número de nodos que desean visualizar a través de un campo de entrada

ubicado en la parte superior izquierda del *dashboard*. Esta funcionalidad les permite personalizar la visualización según sus necesidades.

En esta vista, la navegación por el grafo es flexible y libre. Los usuarios pueden moverse con total libertad por el grafo para explorar los diferentes nodos y conexiones. Para mejorar la experiencia de navegación, se ha incorporado un mini mapa en la esquina superior derecha. Este mini mapa proporciona una vista en miniatura del grafo completo y resalta la ubicación actual del usuario dentro del grafo.

Además, se ha implementado la capacidad de arrastrar y soltar nodos para reorganizarlos según la preferencia del usuario dentro del *viewport*. Al posicionar el cursor sobre un nodo, un *tooltip* informativo muestra el nombre abreviado del sitio y su grado en el grafo, brindando información útil y contextual mientras se navega por la visualización.

Esta vista de Grafo *Outgoing* brinda una manera interactiva y personalizada de explorar las conexiones salientes entre los sitios, permitiendo a los usuarios profundizar en la estructura y las relaciones de la red.

4.1.8. Grafo *incoming*

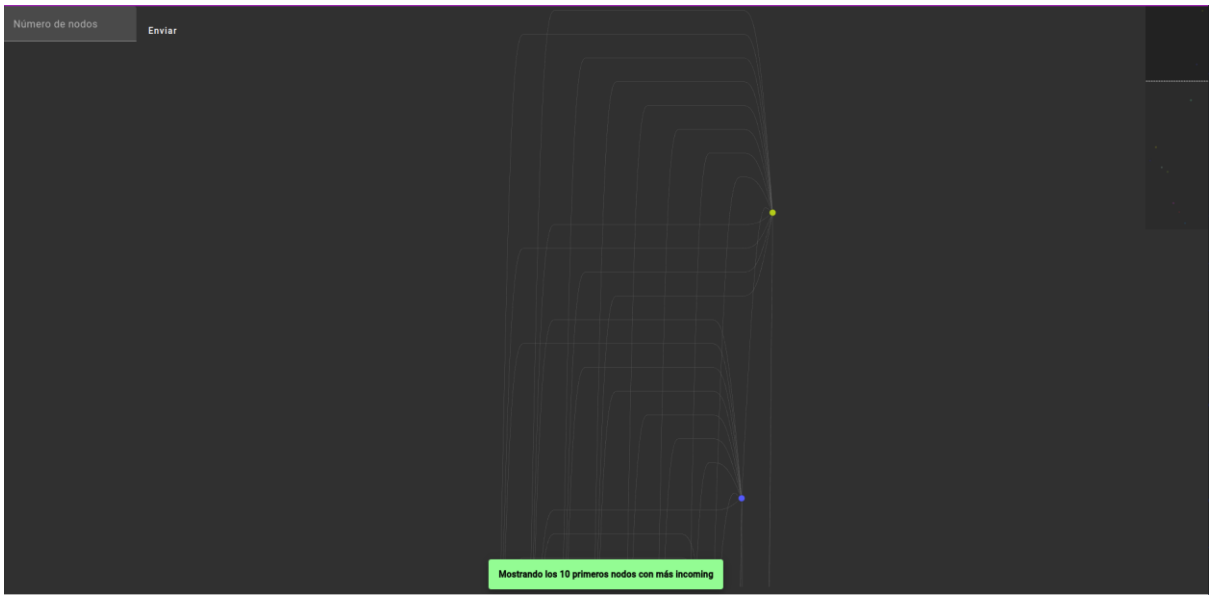


Figura 4.1.9: Vista grafo incoming

La vista de Grafo *Incoming* presenta una estructura idéntica a la de Grafo *Outgoing*, pero está diseñada específicamente para mostrar los nodos con el mayor número de enlaces entrantes (*incoming*). Al igual que en la vista anterior, los usuarios tienen la capacidad de especificar el número de nodos que desean visualizar a través de un campo de entrada ubicado en la parte superior izquierda del *dashboard*.

La navegación por el grafo sigue siendo altamente interactiva y permite a los usuarios explorar los nodos y conexiones entrantes en profundidad. El mini mapa en la esquina superior derecha continúa proporcionando una vista en miniatura del grafo completo y muestra la ubicación actual del usuario en el contexto de la visualización general.

La funcionalidad de arrastrar y soltar nodos para reorganizar su posición dentro del *viewport* se mantiene en esta vista. Al igual que en la vista de Grafo *Outgoing*, los *tooltips* informativos proporcionan el nombre abreviado del sitio y su grado en el grafo al posicionar el cursor sobre un nodo.

En resumen, la vista de Grafo *Incoming* ofrece una experiencia de usuario similar a la de Grafo *Outgoing*, pero se enfoca en destacar los enlaces entrantes más relevantes. Esta estructura coherente en el diseño permite a los usuarios cambiar entre las vistas con facilidad mientras exploran la interconexión de los sitios en el *dashboard*.

4.1.9. Tablas

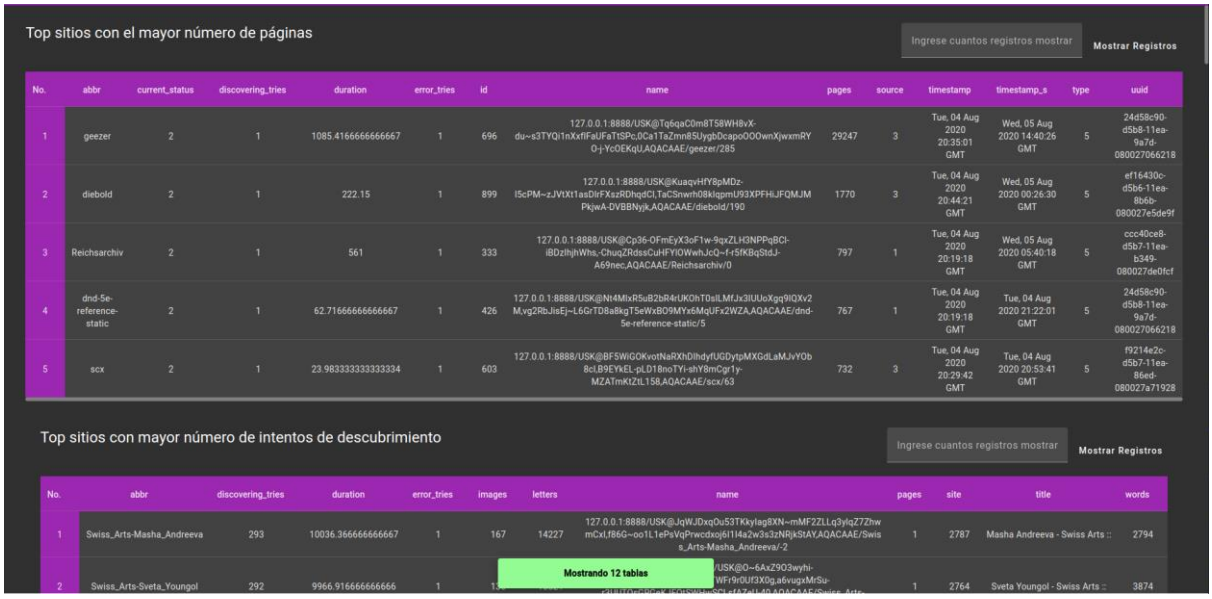


Figura 4.1.10: Vista tablas

La vista de Tablas presenta información valiosa sobre el proceso de "*crawleo*" en un formato tabular. Cada tabla contiene datos detallados que pueden ser altamente relevantes para el usuario. La singularidad de esta vista reside en la capacidad de brindar al usuario una visión más exhaustiva y detallada sobre cada sitio, ya que las tablas pueden incluir una amplia variedad de columnas y campos relacionados con cada sitio.

Cada tabla en esta vista cuenta con un campo de entrada (input) que permite al usuario definir la cantidad de registros que desea visualizar en cada tabla. Esto otorga al usuario

un control total sobre la cantidad de información que desea explorar en un momento dado.

A diferencia de las otras vistas, en la vista de Tablas, se puede acceder a una mayor cantidad de información sobre cada sitio, ya que las tablas pueden mostrar muchas más columnas y campos relacionados con cada respuesta recibida de la API. Estas tablas se generan de manera dinámica, ajustando automáticamente las columnas según los diferentes atributos presentes en las respuestas obtenidas de la API.

En resumen, la vista de Tablas proporciona una representación organizada y detallada de la información relacionada con el proceso de "*crawleo*". Esta vista se adapta a las necesidades del usuario, permitiendo una exploración profunda y personalizada de los datos relevantes.

Tablas generadas:

- Top sitios con el mayor número de páginas
- Top sitios con mayor número de intentos de descubrimiento
- Análisis del mayor número de palabras en los sitios *crawleados*
- Análisis de los sitios con el mayor número de imágenes
- Mayor número de conexiones *outgoing* en los sitios
- Mayor número de conexiones *incoming*
- Análisis de los sitios con menor número de conexiones *incoming*
- Análisis de los sitios con menor número de conexiones *outgoing*
- Top sitios con más conexiones *outgoing*
- Top sitios con más conexiones *incoming*
- Top menos sitios con conexiones *incoming*
- Top menos sitios con conexiones *outgoing*

4.1.10. Modal Info

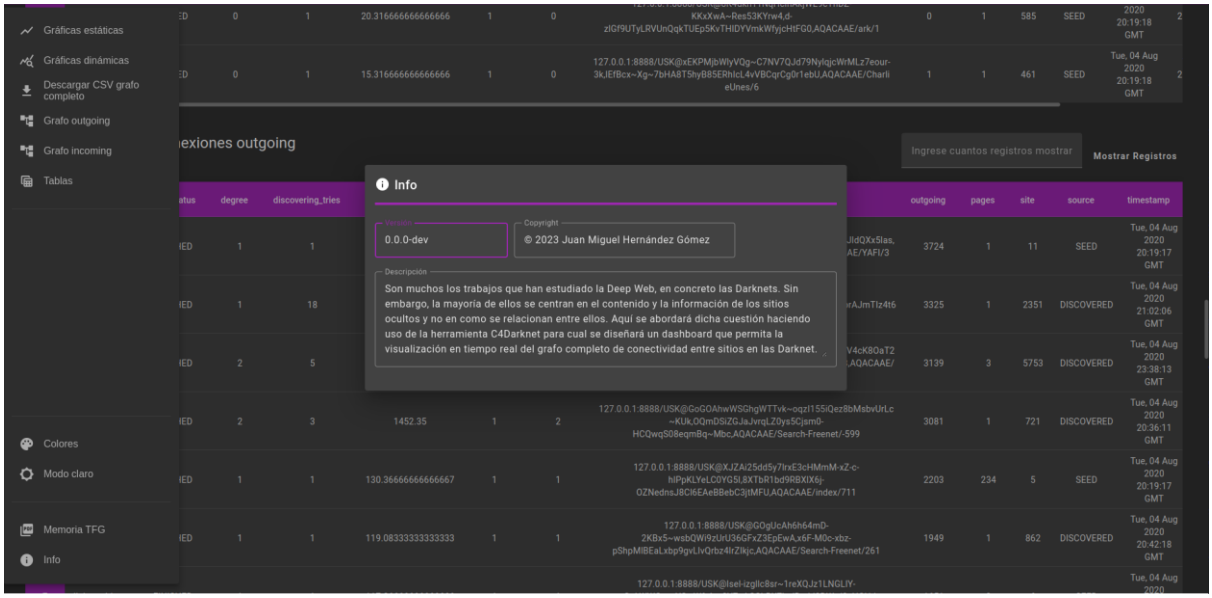


Figura 4.1.11: Modal de info

En la sección final del *sidenav*, se encuentra el apartado "Info". Al hacer clic en él, se despliega un modal de información que proporciona detalles relevantes sobre el *dashboard* y su desarrollo. Este modal ofrece información esencial para el usuario interesado en conocer más sobre la creación y el propósito del *dashboard*.

Dentro del modal, se incluye información sobre el desarrollador del *dashboard*, la versión actual en la que se encuentra y un resumen conciso de las características y objetivos del *dashboard*. Este resumen brinda al usuario una visión general de lo que el *dashboard* puede ofrecer y cuáles son sus principales metas.

En definitiva, el modal de información enriquece la experiencia del usuario al proporcionar contexto y detalles sobre la creación y el propósito del *dashboard*. Es una fuente útil para los usuarios que desean obtener una visión más completa y detallada de la herramienta que están utilizando.

4.1.11. Página no encontrada 404

Figura 4.1.12 Página Not-Found 404

La Figura 4.1.12 muestra la página de error "*Not-Found* 404". Esta vista se mostrará cuando el usuario intente acceder a una URL que no corresponda a ningún recurso implementado en nuestro *dashboard*.

En esta página de error, hemos optado por un diseño original que incluye dos botones. El primer botón permite al usuario volver a la página anterior, lo que puede ser útil si han llegado a esta página de error por accidente o si desean regresar a su navegación anterior. El segundo botón redirige al usuario a la página de inicio "home" de nuestro *dashboard*, proporcionando una opción fácil para continuar explorando el contenido y las funcionalidades del *dashboard*.

Esta página de error 404 asegura que los usuarios reciban un mensaje claro y una opción de recuperación cuando intenten acceder a un recurso que no existe en nuestro *dashboard*, mejorando así la experiencia del usuario al proporcionar un camino claro para continuar navegando.

4.2. API Rest

En el panorama actual de la implementación de APIs, los servicios de tipo REST han emergido como el estándar dominante. Aunque existen alternativas como SOAP, su popularidad ha disminuido considerablemente. Esto se debe en gran parte a que el formato JSON ha superado a XML debido a su amplia integración en los lenguajes de programación y *frameworks* modernos. Optar por JSON proporciona una serie de

ventajas, como versatilidad, simplicidad en la implementación y facilidad tanto en la comprensión como en el uso.

La elección de una API REST se basa en su capacidad para ofrecer una arquitectura flexible y altamente escalable. La estructura ligera y orientada a recursos de REST se adapta de manera natural a las necesidades de interacción entre sistemas en la actualidad. Al utilizar JSON como formato de intercambio de datos, se logra una mayor agilidad en la comunicación, lo que se traduce en una experiencia más fluida tanto para los desarrolladores como para los usuarios finales.

La simplicidad inherente en la implementación de una API REST, junto con la familiaridad de JSON en la comunidad de desarrollo, ha consolidado la posición de REST como el enfoque preferido para la creación de interfaces de programación. En última instancia, elegir una API REST nos brinda una herramienta poderosa y eficiente para facilitar la comunicación y la interoperabilidad entre sistemas, impulsando el éxito y la efectividad de proyectos como este.

El término REST corresponde a las siglas de *Representational State Transfer*, es un conjunto de patrones de diseño que buscan la usabilidad, simpleza escalabilidad y extensibilidad.

Si bien las especificaciones de una API REST no imponen el uso de un lenguaje de descripción específico, es notable que con frecuencia se asocia con el formato JSON. Esta combinación se ha vuelto común debido a varias razones que contribuyen a la eficiencia y compatibilidad en el desarrollo de aplicaciones.

JSON, o *JavaScript Object Notation*, se ha convertido en un formato de elección para el intercambio de datos en la comunidad de desarrollo. Su estructura basada en pares clave-valor es intuitiva y fácil de entender tanto para humanos como para sistemas. Además, JSON es altamente legible y presenta una sintaxis concisa, lo que simplifica el proceso de análisis y manipulación de datos.

La combinación de JSON con API REST se ajusta perfectamente a la naturaleza de la arquitectura de servicios web *RESTful*. La versatilidad y la flexibilidad de JSON permiten que los datos se transmitan de manera eficiente entre el cliente y el servidor, sin importar el tipo de aplicación o la plataforma utilizada.

Si bien JSON es el socio más común para las APIs REST, las especificaciones de REST también permiten otras opciones de formato de datos, como XML. Sin embargo, JSON ha ganado prominencia debido a su simplicidad, su facilidad de uso en una amplia gama de lenguajes de programación y su adopción generalizada en la industria.

En resumen, aunque no existe una obligación estricta de usar un lenguaje de descripción específico en las especificaciones de una API REST, la combinación con JSON ha demostrado ser una elección práctica y beneficiosa que contribuye a la fluidez en el desarrollo de aplicaciones y la comunicación entre sistemas.

4.2.1. Diseño

La API que hemos desarrollado está diseñada para facilitar la interacción del usuario a través de métodos HTTP GET. Esto significa que el proceso de solicitud de información es simple y accesible para el usuario, ya que solo requiere ingresar una URL para realizar consultas. Los parámetros necesarios serán transmitidos a través de la URL, lo que simplifica el proceso y permite una interacción fluida.

En la mayoría de nuestros métodos, hemos implementado la estrategia de enviar los parámetros a través de la URL. Esto es particularmente efectivo cuando el usuario solo necesita enviar un valor numérico u otra información simple. Este enfoque agiliza aún más la experiencia del usuario al reducir la complejidad del proceso de solicitud.

Como mencionamos previamente, el propósito central de este proyecto es desarrollar un *dashboard* complementario para la herramienta C4Darknet. Para lograrlo, hemos utilizado una copia de una base de datos con datos reales, la cual nos ha servido como fuente de información para la elaboración de nuestro proyecto. A continuación, presentamos el diagrama de entidad-relación resultante, el cual es un reflejo visual de la estructura de la base de datos creada a partir del respaldo proporcionado por nuestros compañeros de C4Darknet.

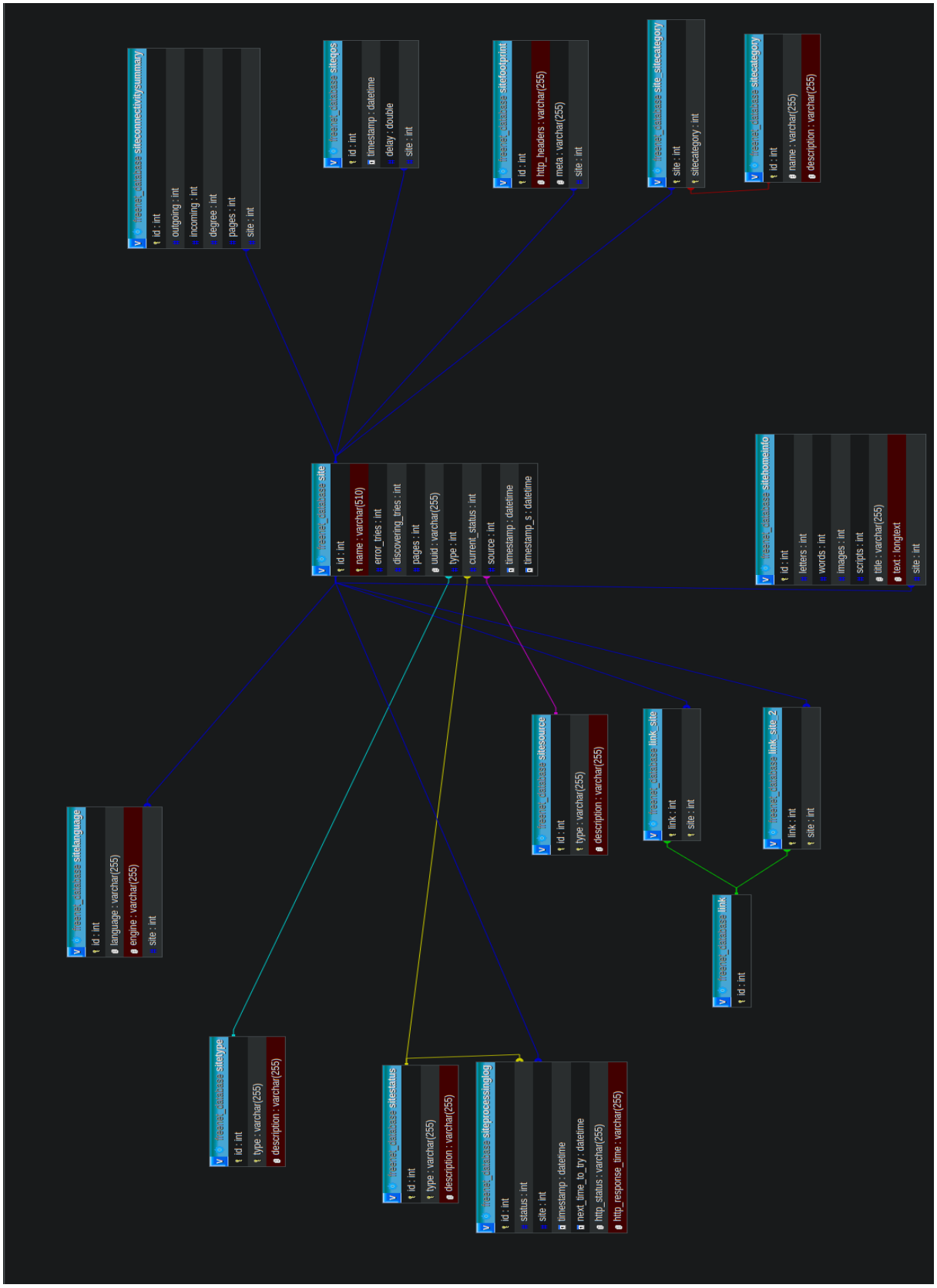


Figura 4.2.1: Diagrama entidad-relación de la base de datos

La API desempeña un papel fundamental al facilitar la comunicación con la base de datos. Para mejorar el rendimiento y la eficiencia, se ha implementado un sistema de caché que se actualiza periódicamente, específicamente cada 10 minutos, según una variable global. Esta elección se basa en la consideración de que las consultas a la base de datos pueden ser intensivas en recursos. Además, el proceso de *crawleo*, que introduce nuevos registros en la base de datos, puede ser prolongado. Dadas estas consideraciones, la adopción de un sistema de caché se considera la solución más apropiada para garantizar una experiencia óptima para el usuario.

Es esencial destacar que, aunque el intervalo de actualización de la caché se ha establecido en 10 minutos, se reconoce la posibilidad de ajustarlo de manera más precisa para lograr un equilibrio óptimo entre el rendimiento y la frescura de los datos. Para abordar este aspecto, se ha implementado un botón en la interfaz de usuario (*front-end*) que permite forzar la actualización de la caché cuando sea necesario. Esto brinda a los usuarios un mayor control sobre la frescura de los datos y garantiza que siempre tengan acceso a la información más actualizada.

Una de las ventajas destacadas de esta implementación es que, si la caché de la base de datos se ha cargado previamente y, por alguna razón, deja de estar disponible o no es accesible a través de la API, el *dashboard* seguirá funcionando y mostrando datos utilizando la información recuperada en la última actualización de la caché. Esto asegura una experiencia continua para los usuarios incluso en situaciones adversas.

Vale la pena señalar que, durante la concepción de la implementación, se consideró la posibilidad de utilizar hebras (*threads*) para la generación de gráficos y visualizaciones con la biblioteca Matplotlib. Sin embargo, se identificó que Matplotlib no es *"thread-safe"*, lo que significa que su uso simultáneo en múltiples hebras puede ocasionar problemas de sincronización y comportamientos inesperados. Por tanto, se optó por un enfoque más seguro y estable al generar las representaciones visuales en una secuencia ordenada, lo que garantiza la integridad y coherencia de los resultados sin comprometer la estabilidad del sistema.

Adicionalmente, con el fin de facilitar el desarrollo y evitar problemas de restricciones de políticas de mismo origen (CORS) durante la fase de desarrollo, se configuró CORS (*Cross-Origin Resource Sharing*) utilizando la biblioteca Flask-CORS. Esto permitió establecer un conjunto específico de orígenes permitidos para las solicitudes, en este caso, se autorizó el acceso desde "http://localhost:4200", que corresponde al servidor de desarrollo en el puerto 4200. Esta configuración se implementó para garantizar que

el desarrollo y las pruebas pudieran llevarse a cabo sin restricciones innecesarias relacionadas con CORS.

Hemos decidido manejar las imágenes mediante su codificación en base64 y posterior inclusión en formato JSON como parte de la respuesta. Esta elección nos permite ofrecer una manera eficiente de transmitir imágenes junto con otros datos, manteniendo la coherencia en la estructura de las respuestas.

En síntesis, la API desempeña un rol esencial al facilitar la comunicación con la base de datos. La implementación de un sistema de caché con actualización regular busca optimizar el rendimiento, mientras que el diseño coherente y estructurado de las respuestas JSON asegura una experiencia fluida y comprensible para los usuarios y desarrolladores por igual.

4.2.2. Principales llamadas

En las secciones siguientes, se detallan las llamadas esenciales a la API que impulsan la generación del contenido visual en cada vista del *dashboard*. Estas llamadas se ejecutan automáticamente al cargar la vista correspondiente, y la API inicia llamadas internas a diversas funciones para generar los distintos elementos que conformarán la vista general de cada sección dentro de nuestro *dashboard* (cabe resaltar que estas funciones también son accesibles desde el exterior mediante las llamadas adecuadas). Cada función se encarga de generar elementos individuales que, posteriormente, se combinan en un objeto JSON. Este conjunto de elementos está destinado para su visualización en el *dashboard*.

Especialmente, las tablas y las gráficas dinámicas tienen un comportamiento interactivo. Si el usuario introduce valores en los campos de entrada proporcionados, el *dashboard* realiza llamadas precisas a métodos específicos de la API. Esto se hace con el propósito de solicitar nueva información para su representación visual. Es fundamental mencionar que, después de esta llamada, solo se actualiza el elemento vinculado a la entrada del usuario en la vista. Los demás elementos mantienen su estado previo a la llamada a la API, asegurando una experiencia uniforme y sin interrupciones para el usuario.

- **Obtener Gráficas Estáticas**

GET {host}/obtenerGraficasEstaticas

Content-Type*: {application/json}*

Esta ruta permite obtener todas las gráficas estáticas a partir de datos previamente recopilados y procesados. Proporciona visualizaciones significativas y relevantes para el análisis de los sitios y su comportamiento en la plataforma.

Parámetros de la Llamada:

N/A (No requiere parámetros en la llamada)

Respuestas:

Código 200 OK: La API ha procesado la petición correctamente y devuelve una lista de datos en formato JSON que representan las gráficas estáticas generadas. Cada elemento en la lista contiene información detallada sobre una gráfica específica.

Código *500 Internal Server Error*: La API no pudo completar la petición debido a un error interno en el servidor.

- ### Obtener Gráficas Dinámicas

GET {host}/obtenerGraficasDinamicas

***Content-Type**: {application/json}*

Esta ruta permite obtener todas las gráficas dinámicas basadas en datos procesados y actualizados en tiempo real. Estas visualizaciones ofrecen una perspectiva en constante evolución sobre el análisis de los sitios y su comportamiento en la plataforma. Para el array de gráficas dinámicas se generan utilizando como parámetros valores por defectos, que luego el usuario podrá modificar desde el *dashboard*.

Parámetros de la Llamada:

N/A (No requiere parámetros en la llamada)

Respuestas:

Código 200 OK: La API ha procesado la petición correctamente y devuelve un objeto en formato JSON que contiene las gráficas dinámicas generadas. El objeto contiene información detallada sobre cada gráfica específica.

Código *500 Internal Server Error*: La API no pudo completar la petición debido a un error interno en el servidor.

- ### Obtener Tablas

GET {host}/obtenerTablas

***Content-Type**: {application/json}*

Esta ruta permite obtener todas las tablas con información relevante basada en datos previamente procesados y recopilados. Estas tablas proporcionan un resumen estructurado y significativo para el análisis de los sitios y su comportamiento en la plataforma.

Parámetros de la Llamada:

N/A (No requiere parámetros en la llamada)

Respuestas:

Código 200 OK: La API ha procesado la petición correctamente y devuelve un objeto en formato JSON que contiene las tablas generadas. El objeto contiene información detallada sobre cada tabla específica.

Código *500 Internal Server Error*: La API no pudo completar la petición debido a un error interno en el servidor.

- ### Obtener archivos CSV comprimidos en ZIP

GET {host}/getCompressedCSVs

***Content-Type**: {text/html; charset=utf-8}*

Esta ruta permite obtener archivos CSV comprimidos en formato ZIP y codificados en base64. Los archivos CSV contienen información procesada y recopilada, listos para su importación y análisis en herramientas como Gephi. Los contenidos de los archivos CSV se generan a partir de *DataFrames* previamente creados, que se han filtrado y estructurado según las necesidades. A continuación, se detalla el proceso de generación y entrega de los archivos CSV comprimidos en formato JSON.

Este método realiza varias tareas importantes:

1. Genera los contenidos de los archivos CSV a partir de los *DataFrames* correspondientes.
2. Crea un objeto *ZipFile* en memoria y agrega los contenidos de los archivos CSV.
3. Codifica el archivo ZIP resultante en formato base64 para su fácil transferencia.
4. Devuelve una respuesta JSON que contiene el archivo ZIP comprimido en base64, listo para su descarga y uso.

Parámetros de la Llamada:

N/A (No requiere parámetros en la llamada)

Respuestas:

Código 200 OK: La API ha procesado la petición correctamente y devuelve una respuesta JSON que contiene el archivo ZIP comprimido en formato base64. El archivo ZIP contiene dos archivos CSV: "aristas_total.csv" que representa los enlaces y conexiones entre nodos, y "nodos_total.csv" que representa los nodos con su respectiva información.

Código *500 Internal Server Error*: La API no pudo completar la petición debido a un error interno en el servidor.

- **Generar grafo top n nodos con más conexiones entrantes (Grafo Dinámico)**

GET {host}/generarArchivosJSONGrafoTopIncoming/<int:param>

***Content-Type**: {application/json}*

Esta ruta permite generar objetos JSON que representan los nodos y las aristas de un grafo. La generación se basa en los sitios con el mayor número de conexiones entrantes, y el número de sitios seleccionados es determinado por el parámetro opcional "param" (10 por defecto). Los objetos JSON resultantes contienen información sobre los nodos y las relaciones entre ellos, permitiendo su visualización y análisis.

Este método desempeña las siguientes funciones clave:

1. Selecciona los sitios con el mayor número de conexiones entrantes según el parámetro opcional "param".
2. Identifica las relaciones entre los sitios seleccionados y crea un *DataFrame* con las aristas correspondientes.
3. Calcula los grados de los nodos y agrega esta información al JSON de nodos.
4. Genera y devuelve el objeto JSON para nodos y aristas.

Parámetros de la Llamada:

param (opcional, tipo: int): Número de sitios con mayor número de conexiones entrantes a incluir en el grafo (valor por defecto: 10).

Respuestas:

Código 200 OK: La API ha procesado la petición correctamente y devuelve una respuesta JSON que contiene los nodos y aristas.

Código *500 Internal Server Error*: La API no pudo completar la petición debido a un error interno en el servidor.

- **Generar grafo top n nodos con más conexiones salientes (Grafo Dinámico)**

GET {host}/generarArchivosJSONGrafoTopOutgoing/<int:param>

***Content-Type**: {application/json}*

Esta ruta permite generar objetos JSON que representan los nodos y las aristas de un grafo. La generación se basa en los sitios con el mayor número de conexiones salientes, y el número de sitios seleccionados es determinado por el parámetro opcional "param" (10 por defecto). Los objetos JSON resultantes contienen información sobre los nodos y las relaciones entre ellos, permitiendo su visualización y análisis.

Este método desempeña las siguientes funciones clave:

1. Selecciona los sitios con el mayor número de conexiones entrantes según el parámetro opcional "param".
2. Identifica las relaciones entre los sitios seleccionados y crea un *DataFrame* con las aristas correspondientes.
3. Calcula los grados de los nodos y agrega esta información al JSON de nodos.
4. Genera y devuelve el objeto JSON para nodos y aristas.

Parámetros de la Llamada:

param (opcional, tipo: int): Número de sitios con mayor número de conexiones entrantes a incluir en el grafo (valor por defecto: 10).

Respuestas:

Código 200 OK: La API ha procesado la petición correctamente y devuelve una respuesta JSON que contiene los nodos y aristas.

Código *500 Internal Server Error*: La API no pudo completar la petición debido a un error interno en el servidor.

4.2.2. Principales funciones auxiliares

Estas funciones no son accesibles desde el exterior de la API, pero son realmente útiles para la lógica de nuestra API.

- **actualizarValoresBD**

Esta función se encarga de actualizar los valores de la base de datos y generar *DataFrames* a partir de las tablas de la base de datos. Luego, realiza una serie de transformaciones y combinaciones de datos para obtener información relevante para el

funcionamiento del *dashboard*. A continuación, se describen las principales tareas realizadas por esta función:

1. Selecciona datos relevantes de las tablas de la base de datos y los almacena en *DataFrames* individuales.
2. Agrega el dato de la duración en minutos a partir de las fechas de inicio y fin de cada registro en la tabla df_site.
3. Genera abreviaturas para los sitios de Freenet en la columna abbr del *DataFrame* df_site.
4. Agrega información de estado y fuente del sitio al *DataFrame* df_site_status.
5. Combina la información del sitio con la información de conectividad en el *DataFrame* df_site_conn.
6. Combina la información de conectividad de nodos en el *DataFrame* df_links.
7. Combina la información del sitio con la información de home en el *DataFrame* df_site_home.
8. Combina la información del sitio y home con la información de lenguaje en el *DataFrame* df_site_home_lan.
9. Actualiza la variable global ultima_actualizacion con la fecha y hora actuales.
10. Imprime un mensaje en la consola indicando que la caché ha sido actualizada.

Esta función es esencial para mantener los datos actualizados y preparar la información necesaria para la generación del contenido visual en el *dashboard*.

- ***shouldRefresh***

Esta función se encarga de verificar si es necesario actualizar los valores almacenados en la caché. Utiliza la variable global ultima_actualizacion para determinar cuándo se realizó la última actualización y compara con el intervalo de tiempo definido en tiempo_siguiente_actualizacion_cache.

Si la última actualización es *None* (es decir, aún no se ha realizado ninguna actualización) o si ha pasado más tiempo del intervalo definido, la función invoca la función actualizarValoresBD() para actualizar los valores en la caché. De esta manera, se asegura que los datos utilizados para generar el contenido visual en el *dashboard* estén siempre actualizados y reflejen la información más reciente de la base de datos.

- **obtenerImagenBase64**

Esta función toma un gráfico de Matplotlib (plt) y opcionalmente una figura (fig). Luego, genera la representación en formato PNG de ese gráfico y la convierte en una cadena base64 para que pueda ser incrustada en el contenido HTML. La función devuelve la representación base64 de la imagen.

Si se proporciona una figura (fig), se cierra para liberar recursos. El uso de io.BytesIO() permite crear un búfer en memoria para almacenar la imagen PNG generada antes de convertirla en formato base64. Esto es útil cuando se necesita enviar la imagen como parte de una respuesta de la API o para mostrarla en una interfaz web.

5. Evaluación y resultados

En esta sección, realizamos una evaluación exhaustiva de los resultados generados por la API y el *dashboard* desarrollados. Para validar la precisión y coherencia de los datos presentados en el *dashboard*, se llevaron a cabo comparaciones con las visualizaciones y tablas generadas por nuestros compañeros de C4Darknet. A continuación, presentamos los métodos y resultados de esta evaluación.

5.1. Validación de resultados

La validación de los resultados se llevó a cabo mediante la comparación de las gráficas y tablas generadas por el *dashboard* con las presentadas en los notebooks de Jupyter utilizados por nuestros compañeros de C4Darknet. Se seleccionaron muestras representativas de datos para las cuales se generaron visualizaciones en ambos sistemas. Cabe mencionar que, si bien se intentó garantizar la coherencia entre los datos, no podemos afirmar con certeza que las bases de datos subyacentes fueran idénticas, lo que podría haber ocasionado ligeras variaciones en algunos resultados.

5.1.1. Comparación de gráficas

Se examinaron gráficas clave generadas en ambos sistemas. Por ejemplo, la Figura 5.1.1.1. en nuestro *dashboard*, se comparó con la Figura 5.1.1.2. obtenida de uno de los notebooks de C4Darknet. En la mayoría de los casos, las gráficas coincidieron de manera notable, con similitudes en patrones y tendencias. Sin embargo, se observaron diferencias marginales en algunas cifras porcentuales, generalmente inferiores al 1%.

Figura 5.1.1.1. Gráficas generadas en el dashboard

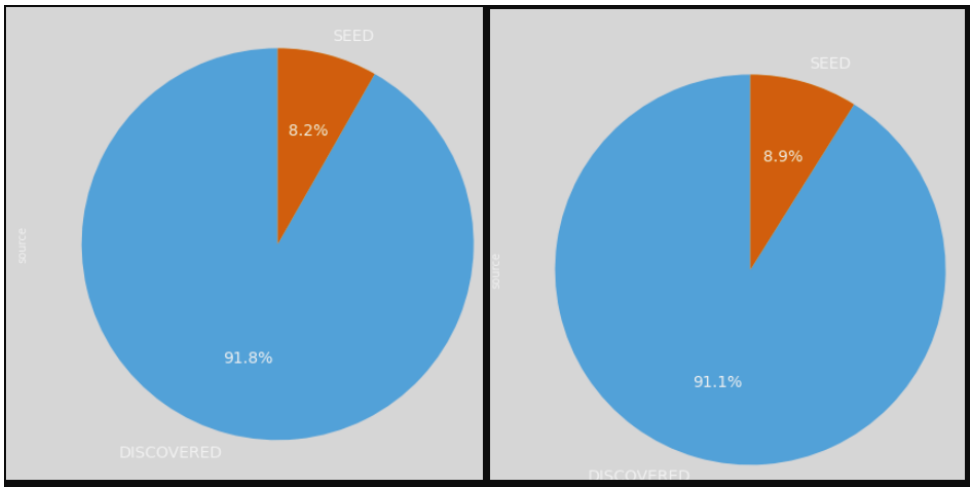


Figura 5.1.1.2. Gráficas extraídas de los notebooks de los desarrolladores de C4Darknet

5.1.2. Validación de tablas

Se seleccionaron tablas relevantes que mostraban datos específicos, como el número de conexiones entre sitios de *Darknets* en un período determinado. Estas tablas se compararon con los resultados proporcionados por C4Darknet. En general, los valores numéricos eran consistentes, aunque se identificaron discrepancias mínimas en ciertos totales, nuevamente con diferencias inferiores al 1%. En las Figuras 5.1.2.1. y 5.1.2.2. podemos apreciar como hemos obtenido los mismos resultados en las tablas mostradas por el *dashboard* y por el notebook.

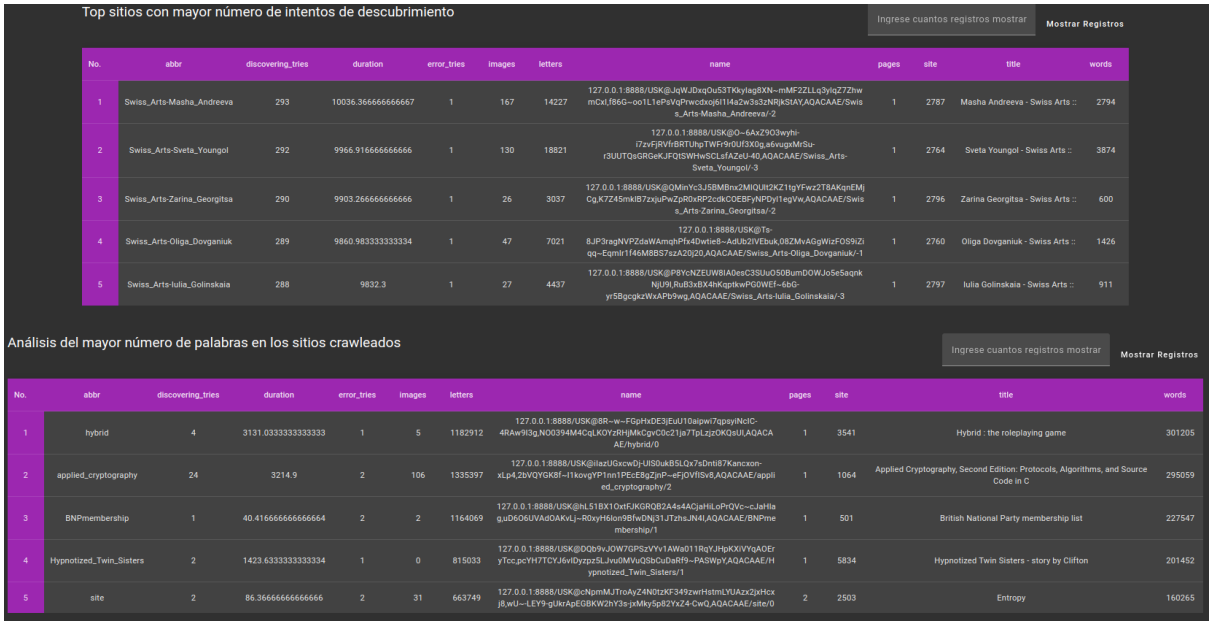
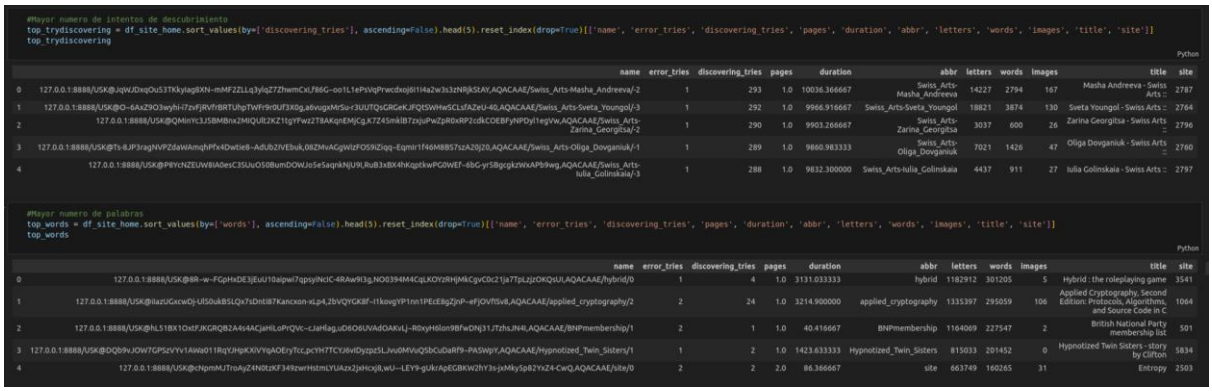


Figura 5.1.2.1. Tablas generadas en el dashboard



```
#Mayor numero de intentos de descubrimiento
top_trydiscovering = df_site_home.sort_values(by=['discovering_tries'], ascending=False).head(5).reset_index(drop=True)[['name', 'error_tries', 'discovering_tries', 'pages', 'duration', 'abbr', 'letters', 'words', 'images', 'title', 'site']]
top_trydiscovering
```

Python

	name	error_tries	discovering_tries	pages	duration	abbr	letters	words	images	title	site
0	127.0.0.1:8888/USK@JqWJDxqOu53TKkyIag8XN~mMF2ZLLq3ylqZ7ZhwmCxI,f86G~oo1L1ePsVqPrwcdxojiI1l4a2w3s3zNRjkStAY,AQACAAE/Swiss_Arts-Masha_Andreeva/-2		1	293	1.0	10036.366667	Swiss_Arts-Masha_Andreeva	14227	2794	167	Masha Andreeva - Swiss Arts ::	2787
1	127.0.0.1:8888/USK@O~6AxZ9O3wyhi-I7zvFjRVfrBRTUhpTWFr9r0Uf3X0g,a6vugxMrSu-r3UUTQsGRGeKJFQtSWHwSCLsfAZeU-40,AQACAAE/Swiss_Arts-Sveta_Youngol/-3		1	292	1.0	9966.916667	Swiss_Arts-Sveta_Youngol	18821	3874	130	Sveta Youngol - Swiss Arts ::	2764
2	127.0.0.1:8888/USK@QMinYc3J5BMBnx2MIQUlI2KZ1tgYFwz2T8AKqnEMjCg,K7Z45mklB7zxjuPwZpR0xRP2cdkCOEBFyNPDyl1egVw,AQACAAE/Swiss_Arts-Zarina_Georgitsa/-2		1	290	1.0	9903.266667	Swiss_Arts-Zarina_Georgitsa	3037	600	26	Zarina Georgitsa - Swiss Arts ::	2796
3	127.0.0.1:8888/USK@Ts-8JP3ragNVPZdaWAmqhPfx4Dwtie8~AdUbZiVEbuk,08ZMvACgWlzFOS9iZiqq~Eqmir1f46M8B57szA20j20,AQACAAE/Swiss_Arts-Oliga_Dovganiuk/-1		1	289	1.0	9860.983333	Swiss_Arts-Oliga_Dovganiuk	7021	1426	47	Oliga Dovganiuk - Swiss Arts ::	2760
4	127.0.0.1:8888/USK@PBYchZEUW8IAOesC35UuO50BumDOWJo5e5aqrikNjU9I,RuB3xBX4hKqptkwPG0WEf~6bG-yr5BgcgkzWxAPb9wg,AQACAAE/Swiss_Arts-Iulia_Golinskaia/-3		1	288	1.0	9832.300000	Swiss_Arts-Iulia_Golinskaia	4437	911	27	Iulia Golinskaia - Swiss Arts ::	2797

```
#Mayor numero de palabras
top_words = df_site_home.sort_values(by=['words'], ascending=False).head(5).reset_index(drop=True)[['name', 'error_tries', 'discovering_tries', 'pages', 'duration', 'abbr', 'letters', 'words', 'images', 'title', 'site']]
top_words
```

Python

	name	error_tries	discovering_tries	pages	duration	abbr	letters	words	images	title	site
0	127.0.0.1:8888/USK@8R-w-FGpHxDE3jEuU10aipwi7qpsyiNcIC-4RAw9I3g,NO6394M4CqLKOYzRHjMkCgvC0c21ja7TpLzjzOKQtUI,AQACAAE/hybrid/0	1	4	1.0	3131.033333	hybrid	1182912	301205	5	Hybrid : the roleplaying game	3541
1	127.0.0.1:8888/USK@ilazUGxcwDj-UlS0ukBSLQx7sDntiB7Kancxon-xLp4,2bVQYGK8f~i1kovgYP1nn1PEcE8gZjnP~eFjOVfSv8,AQACAAE/applied_cryptography/2	2	24	1.0	3214.900000	applied_cryptography	1335397	295059	106	Applied Cryptography, Second Edition: Protocols, Algorithms, and Source Code in C	1064
2	127.0.0.1:8888/USK@hL51BX1OxtFJKGRQB2A4s4ACjaHiLoPrQVc~cJaHag,uD6O6UVAdOAKvLj~R0xyH6lon9BfwDNj31,JTzhsJN4I,AQACAAE/BNPmembership/1	2	1	1.0	40.416667	BNPmembership	1164069	227547	2	British National Party membership list	501
3	127.0.0.1:8888/USK@DQb9vJOW7GPSzVYv1AWa011RqYJHpKXiVYgAOEryTcc,pcYH7TCYJ6vIDyzpz5LJvu0MVuQSbCuDaRf9~PASWpY,AQACAAE/Hypnotized_Twin_Sisters/1	1	2	1.0	1423.633333	Hypnotized_Twin_Sisters	815033	201452	0	Hypnotized Twin Sisters - story by Clifton	5834
4	127.0.0.1:8888/USK@cNpmMJTroAyZ4N0tzKF349zwrHstmLYUAzx2jxHcxj8,wU~LEY9-gUkrApEGBKW2hY3s-jxMky5p8ZYxZ4-CwQ,AQACAAE/site/0	2	2	2.0	86.366667	site	663749	160265	31	Entropy	2503

Figura 5.1.2.2. Tablas extraídas de los notebooks de los desarrolladores de C4Darknet

5.2. Limitaciones y consideraciones

Es importante resaltar que, aunque se realizaron esfuerzos para garantizar la coherencia de los resultados, existen algunas limitaciones en esta validación:

- Diferencias en bases de datos: No podemos asegurar que las bases de datos utilizadas para generar las visualizaciones en C4Darknet sean idénticas a la base de datos respaldada en este proyecto, lo que podría haber causado las leves variaciones observadas.
- Métodos de cálculo: Las diferencias marginales podrían deberse a variaciones en los métodos de cálculo utilizados en ambos sistemas para generar las visualizaciones y tablas.

6. Conclusiones y líneas de trabajo futuro

6.1. Contribuciones

El presente proyecto ha realizado diversas contribuciones significativas en el ámbito del análisis y visualización de la conectividad en *Darknet*, así como en la mejora de la herramienta C4Darknet. A continuación, se detallan las principales contribuciones logradas a través de este proyecto:

1. **Desarrollo de una interfaz de usuario atractiva, funcional e interactiva**: Una de las contribuciones más notables de este proyecto es la creación de un *dashboard* interactivo y visualmente atractivo utilizando el *framework* Angular. Esta interfaz proporciona a los usuarios una manera intuitiva y eficaz de explorar y comprender los datos de conectividad en *Darknet* generados por la herramienta C4Darknet. La inclusión de gráficos, tablas y visualizaciones enriquece la experiencia del usuario, facilitando la identificación de patrones y relaciones entre los sitios.
2. **Generación de una API REST eficiente**: La implementación de una API REST en Python utilizando Flask permite a los usuarios acceder de manera sencilla y eficiente a los datos de conectividad recopilados por el proceso de "*crawling*" de la herramienta C4Darknet. La API actúa como un puente de comunicación entre la base de datos y el *dashboard*, proporcionando una capa de abstracción que simplifica la interacción y el acceso a la información.
3. **Optimización del rendimiento con un sistema de caché**: La introducción de un sistema de caché periódica contribuye significativamente a la optimización del rendimiento y la eficiencia en la recuperación de datos. Al actualizar la caché cada 10 minutos, se reduce la carga en la base de datos y se mejora la velocidad de respuesta, brindando a los usuarios una experiencia más fluida y rápida al interactuar con el *dashboard*.
4. **Incorporación de visualizaciones perspicaces**: La inclusión de diversas visualizaciones y gráficos en el *dashboard* permite a los usuarios obtener una comprensión más profunda de la estructura de conectividad en *Darknet*. Esta contribución no solo facilita la identificación de patrones y relaciones entre los sitios, sino que también proporciona *insights* valiosos para el análisis y la toma de decisiones informadas.

5. **Enriquecimiento de la herramienta C4Darknet**: El proyecto ha enriquecido la funcionalidad de la herramienta C4Darknet al dotarla de una interfaz web interactiva. Esta adición convierte a la herramienta en una solución más completa y accesible para los usuarios, permitiéndoles no solo recopilar y analizar datos de conectividad, sino también visualizarlos de manera eficaz y atractiva.

En síntesis, las contribuciones de este proyecto abarcan desde la creación de una interfaz gráfica interactiva y la optimización del acceso a datos mediante una API REST, hasta la mejora del rendimiento y la funcionalidad de la herramienta C4Darknet. Estas contribuciones en conjunto aportan un valor significativo al campo de análisis y visualización de la conectividad en *Darknet*, facilitando la comprensión y exploración de un entorno enigmático y complejo.

6.2 Grado de consecución de los objetivos planteados

La evaluación del cumplimiento de los objetivos planteados en este proyecto es esencial para determinar en qué medida se ha logrado el propósito previsto y para identificar posibles áreas de mejora. A continuación, analizamos cada uno de los objetivos principales y secundarios establecidos al comienzo del proyecto y evaluamos su realización.

6.2.1. Objetivos principales

Análisis de la conectividad en *Darknets*

Resultado: Se ha llevado a cabo un análisis riguroso y sistemático de la estructura de conectividad entre sitios en las *Darknets*. Mediante la generación de gráficas y visualizaciones, se ha proporcionado una representación clara de cómo los sitios ocultos se interrelacionan.

Desarrollo de una API REST en Python con Flask

Resultado: Se ha desarrollado una API REST utilizando el *framework* Flask en Python. Esta API permite interactuar con la base de datos generada por el proceso de "*crawling*" realizado por la herramienta C4Darknet. La API proporciona acceso a los datos de conectividad y otros detalles relevantes para su posterior uso en el *dashboard*.

Creación de un *dashboard* en Angular

Resultado: Se ha diseñado y desarrollado un *dashboard* interactivo utilizando el *framework* Angular. Este *dashboard* actúa como una interfaz de usuario atractiva y funcional que permite a los usuarios explorar visualmente los datos de conectividad entre sitios en las *Darknets*. El *dashboard* presenta información enriquecida a través de gráficos, tablas y visualizaciones de datos.

6.2.2. Objetivos secundarios

Resultado: El *dashboard* ha sido diseñado con una interfaz intuitiva y fácil de usar. Se han implementado elementos de navegación y diseño que garantizan una experiencia amigable para los usuarios finales, independientemente de su nivel de experiencia técnica.

Implementación de funcionalidades de interacción

Resultado: El *dashboard* está equipado con características interactivas que permiten a los usuarios ajustar parámetros, filtrar datos y explorar diferentes aspectos de la conectividad entre sitios de *Darknets*. La interacción en tiempo real aumenta la utilidad y versatilidad del *dashboard*.

Visualizaciones Variadas y Perspicaces

Resultado: El *dashboard* ofrece una variedad de gráficos y visualizaciones que ayudan a los usuarios a comprender mejor las relaciones entre sitios y a identificar patrones emergentes. Las representaciones visuales de los datos proporcionan *insights* perspicaces.

Documentación Completa

Resultado: Se ha generado documentación detallada para la API REST y el *dashboard*. Esto facilita el uso continuo y la posible expansión del proyecto en el futuro, brindando instrucciones claras sobre la configuración, implementación y funcionamiento del sistema.

6.2.3. Evaluación General del Cumplimiento de Objetivos

En general, se puede afirmar que los objetivos planteados en este proyecto han sido cumplidos exitosamente. Se ha logrado desarrollar una herramienta completa que analiza la conectividad en *Darknets*, proporciona acceso a los datos a través de una API y presenta estos datos de manera interactiva en un *dashboard*. Además, se han

alcanzado los objetivos secundarios relacionados con la usabilidad, la interacción y la documentación. El proyecto ha logrado ofrecer una solución valiosa y funcional que contribuye al conocimiento en el ámbito de las *Darknets* y su análisis.

6.3. Retos y trabajo futuro

Durante el desarrollo del proyecto, se han identificado varios retos y áreas de mejora que podrían abordarse en futuras iteraciones. Estos retos y trabajos futuros tienen el potencial de enriquecer la funcionalidad, la usabilidad y la efectividad general del *dashboard*. A continuación, se enumeran algunos de los retos y posibles trabajos futuros a considerar:

Personalización de la recarga de caché

Implementar una funcionalidad que permita a los usuarios ajustar el tiempo por defecto de recarga de caché. Esto permitiría adaptar la frecuencia de actualización de acuerdo con las preferencias y necesidades del usuario.

CRUD de semillas para el *crawler*

Desarrollar un sistema CRUD (Crear, Leer, Actualizar, Eliminar) que permita a los usuarios agregar y gestionar nuevas semillas para el proceso de "*crawling*". Esto facilitaría la expansión de la base de datos y el análisis de nuevos sitios.

Configuración de parámetros del *crawler*

Incorporar una sección en el *dashboard* que permita a los usuarios configurar los parámetros del "*crawler*", como la profundidad de búsqueda y la frecuencia de recopilación de datos. Esto proporcionaría un mayor control sobre el proceso de recolección de datos.

Funcionalidad de favoritos

Agregar un sistema que permita a los usuarios marcar ciertas gráficas y visualizaciones como favoritas. Estas gráficas podrían almacenarse en una sección de "favoritos", lo que permitiría a los usuarios acceder rápidamente a las visualizaciones que más utilizan.

Mejoras en la interfaz de usuario

Continuar optimizando la interfaz de usuario para garantizar una experiencia fluida y agradable. Esto podría incluir mejoras en el diseño, la disposición de elementos y la claridad de las instrucciones.

Integración de Análisis Avanzados

Investigar y desarrollar análisis más avanzados, como la detección de anomalías y patrones de comportamiento atípicos entre los sitios en las *Darknets*.

Mejoras en la eficiencia y rendimiento

Realizar optimizaciones para mejorar la velocidad de carga de las visualizaciones y la respuesta de la API, asegurando una experiencia rápida y eficiente para el usuario.

Implementación de Autenticación y Seguridad

Introducir capas de autenticación y seguridad para garantizar que solo los usuarios autorizados puedan acceder al *dashboard* y a la API.

Internacionalización

Implementar soporte para múltiples idiomas en el *dashboard*, lo que permitiría a los usuarios de diferentes regiones interactuar con la plataforma en su idioma preferido.

Estos retos y trabajos futuros representan oportunidades para mejorar y expandir el proyecto en el futuro. A medida que el *dashboard* evoluciona y se adapta a las necesidades de los usuarios y los avances tecnológicos, abordar estos aspectos contribuirá a la creación de una herramienta aún más efectiva y valiosa para la exploración de la conectividad en las *Darknets*.

Programación de tareas

Implementar un programador de tareas para permitir la ejecución programada de tareas de *crawling*. Esto es especialmente útil para sistemas distribuidos, donde se pueden planificar tareas de *crawling* en momentos específicos para evitar la sobrecarga de recursos.

Políticas de exclusión e inclusión

Permitir a los administradores de sistemas definir políticas de exclusión e inclusión para el crawling. Esto les daría control sobre qué páginas o recursos rastrear y cuáles omitir. También podría incluir la capacidad de definir patrones de URL para la inclusión o exclusión.

Anexo A

Guía de puesta en marcha

Para comenzar con la puesta en marcha de la API y el *dashboard* de C4Darknet, es necesario seguir los siguientes pasos:

1. Descarga del repositorio

Descarga el repositorio de GitHub donde se encuentra la API y el *dashboard*. Utiliza el comando:

```
git clone https://github.com/juanmihdz/c4darknet.git
```

2. Entorno virtual para la API

Crea un entorno virtual para la API utilizando Python. Ejecuta el siguiente comando en la terminal:

```
virtualenv -p python3 env
```

3. Activación del entorno virtual

Activa el entorno virtual que acabas de crear. Esto ayudará a mantener las dependencias separadas del sistema global. Usa:

```
source ./env/bin/activate
```

4. Instalación de requisitos para la API

Una vez activado el entorno virtual, instala los requisitos necesarios que se encuentran en el archivo `./API/requirements.txt`. Ejecuta:

```
pip install -r requirements.txt
```

5. Ejecución de la API

Ahora puedes ejecutar el archivo `app.py` para lanzar la API. Utiliza el siguiente comando:

```
python ./src/app.py
```

Por defecto, la API estará disponible en `localhost:5000`.

6. Configuración del *front-end*

Para la parte del *front-end*, asegúrate de tener Node.js y npm (*Node Package Manager*) instalados. Si no los tienes, puedes descargarlos desde su sitio oficial.

7. Instalación de Angular CLI

Instala Angular CLI (*Command Line Interface*) utilizando npm. Ejecuta el siguiente comando:

```
npm install -g @angular/cli
```

8. Instalación de dependencias del proyecto Angular

Desde el directorio raíz del proyecto Angular, ejecuta el siguiente comando para instalar todas las dependencias necesarias:

```
npm install
```

9. Ejecución de la aplicación *front-end*

Finalmente, para correr la aplicación del *front-end*, utiliza el siguiente comando:

```
npm start
```

Por defecto, la aplicación se abrirá en `localhost:4200`.

Crear un entorno virtual es recomendable para evitar conflictos entre las dependencias de diferentes proyectos. Al activar el entorno virtual, todas las bibliotecas y paquetes se instalan y utilizan dentro de este entorno específico, lo que ayuda a mantener limpio y organizado el sistema global de Python.

Siguiendo estos pasos, estarás listo para utilizar la API y el *dashboard* de C4Darknet en tu entorno local.

Bibliografía

[1] M. Leo, F. Battisti, M. Carli, and A. Neri, “A federated architecture approach for Internet of Things security,” in *2014 Euro Med Telco Conference (EMTC)*, Noviembre 2014, pp. 1–5.

[2] L. Li and W. Chou, “Design and Describe REST API without Violating REST: A Petri Net Based Approach,” in 2011 IEEE International Conference on Web Services, Jul. 2011, pp. 508–515.

[3] González, G. (2016). Surface web, deep web y darknet: ¿en qué se diferencian? *Blogthinkbig.com*. https://blogthinkbig.com/surface-web-deep-web-darknet-se-diferencian

[4] *OnionScan: Investigating the dark web*. (s. f.). https://onionscan.org/

[5] José, J. S. (2021). TorFlow, el mapa del tráfico de la Red TOR. *Derecho de la Red*. https://derechodelared.com/torflow-tor/

[6] colaboradores de Wikipedia. (2023). Ahmia. *Wikipedia, la enciclopedia libre*. https://es.wikipedia.org/wiki/Ahmia

[7] Amitschendel. (s. f.). *GitHub - amitschendel/darknet-mapper: Darknet crawler to map links*. GitHub. https://github.com/amitschendel/darknet-mapper

[8] Abellán Galera, Alberto. (2020). Estudio y análisis de la conectividad en la Deep Web mediante el desarrollo de nuevas herramientas: c4i2p (Crawling for I2P) en I2P. Universidad de Granada.

[9] Magán-Carrión, R., Urda, D., Díaz-Cano, I., & Dorronsoro, B. (2020). Towards a reliable comparison and evaluation of network intrusion detection systems based on machine learning approaches. *Applied Sciences (Switzerland)*, *10*(5). https://doi.org/10.3390/app10051775

[10] Sanjuan Aguilera, Rafael. (2022). Automatización y mejoras en los procesos de ingeniería de características aplicada a la detección y clasificación de ataques a la seguridad: FaaC (Feature as a Counter) como caso de estudio. Universidad de Granada.

[11] Figueras Martín, Emilio. (2020). ANÁLISIS DE LA CONECTIVIDAD Y LOS CONTENIDOS OCULTOS EN REDES ANÓNIMAS O DARKNETS. Universidad de Cádiz.