UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Herramienta de visualización interactiva de tendencias en empresas

Trabajo Fin de Grado

Grado en Ciencia de Datos

AUTOR/A: Marco Cabanes, Rubén

Tutor/a: Doménech i de Soria, Josep

CURSO ACADÉMICO: 2022/2023

Resumen

La digitalización se ha vuelto omnipresente en diversos ámbitos de nuestras vidas, abarcando desde la educación hasta el entorno empresarial. Esta tendencia ha transformado nuestra era en una "era digital", adquiriendo una relevancia especial en el mundo de los negocios, donde prácticamente todas las empresas cuentan con presencia en línea a través de sus sitios web o de sus redes sociales. Esta digitalización conlleva inevitablemente una huella digital que se extiende en cada interacción en línea y que permite abarcar qué preocupaciones han tenido las empresas en distintos aspectos y cómo han evolucionado a través del tiempo. Gracias a esta huella digital, en el presente trabajo se ha llevado a cabo la creación de una base de datos utilizando la tecnología de ElasticSearch, partiendo de un corpus de textos empresariales. Esta base de datos almacena toda esta información y está diseñada específicamente para facilitar la búsqueda y recuperación de textos relevantes. Además, se ha desarrollado un sistema de visualización interactivo mediante la programación con Python y Django. Esta aplicación permite observar de manera dinámica la evolución del interés de las empresas en determinados temas a través de sus sitios web. Ambos procesos, tanto el almacén de datos como la aplicación de visualización, han sido desplegados sobre servidores de *Amazon Web Services* (AWS).

Palabras clave: ElasticSearch, Django, AWS, huella digital, visualización, economía digital, empresas

Resum

La digitalització s'ha tornat omnipresent en diversos àmbits de les nostres vides, abastant des de l'educació fins a l'entorn empresarial. Aquesta tendència ha transformat la nostra era en una "era digital", adquirint una rellevància especial en el món dels negocis, on pràcticament totes les empreses compten amb presència en línia a través dels seus llocs web o de les seues xarxes socials. Aquesta digitalització comporta inevitablement una empremta digital que s'estén en cada interacció en línia i que permet abastar que preocupacions han tingut les empreses en diferents aspectes i com han evolucionat a través del temps. Gràcies a aquesta empremta digital, en el present treball s'ha dut a terme la creació d'una base de dades utilitzant la tecnologia d'ElasticSearch, partint d'un corpus de textos empresarials. Aquesta base de dades emmagatzema tota aquesta informació i està dissenyada específicament per a facilitar la cerca i recuperació de textos rellevants. A més, s'ha desenvolupat un sistema de visualització interactiu mitjançant la programació amb Python i Django. Aquesta aplicació permet observar de manera dinàmica l'evolució de l'interés de les empreses en determinats temes a través dels seus llocs web. Tots dos processos, tant el magatzem de dades com l'aplicació de visualització han sigut desplegats sobre servidors d'*Amazon Web Services* (AWS).

Paraules clau: ElasticSearch, Django, AWS, empremta digital, visualització, economia digital, empreses

Abstract

Digitalization has become ubiquitous in various areas of our lives, from education to business. This trend has transformed our era into a "digital age", with relevance in the business world, where virtually all companies have an online presence through their websites or social media. This digitization inevitably brings with it a digital footprint that extends to every online interaction and allows us to see what concerns companies have had in different aspects and how they have evolved over time. Thanks to this digital footprint, this paper has carried out the creation of a database using ElasticSearch technology, starting from a corpus of business texts. This database stores all this information and is specifically designed to facilitate searching and retrieving relevant texts. In addition, an interactive visualization system has been developed using Python and Django programming. This application makes it possible to dynamically observe the evolution of companies' interest in specific topics through their websites. Both processes, the data warehouse and the visualization application, have been deployed on Amazon Web Services (AWS) servers.

Key words: ElasticSearch, Django, AWS, digital fingerprint, visualization, digital economy, companies

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

La presencia de las empresas en el mundo digital ha experimentado un crecimiento exponencial en los últimos años, convirtiéndose en un factor determinante para su éxito y supervivencia en el mercado. Las páginas web y redes sociales sirven como un escaparate para mostrar sus productos y servicios a un público cada vez más amplio y globalizado. Estos canales digitales permiten a las empresas llegar a clientes potenciales de manera directa y efectiva, rompiendo las barreras geográficas y ampliando su alcance de manera significativa. Toda esta información, a su vez, ofrece una gran fuente de conocimiento de la que se pueden extraer análisis interesantes, como el seguimiento de las tendencias de temas tratados o la identificación de patrones de comportamiento de los clientes y la comprensión de sus preferencias y necesidades.

Todo lo descrito anteriormente conlleva también el desafío de gestionar y aprovechar la ingente cantidad de datos generada, en muchas ocasiones, sin quererlo. Cada palabra, frase, imagen o metadato contenido en una página web y cada publicación en redes sociales deja una huella digital, lo que genera una valiosa fuente de información que permite comprender el comportamiento de los usuarios. La huella digital o *fingerprint* en internet hace referencia a la información y rastro que cada uno de los usuarios deja en su paso por la red [1]. La importancia de gestionar adecuadamente esta huella digital radica en la posibilidad de obtener conocimientos y ser capaz de almacenar esa gran cantidad de datos con el objetivo de extraer información relevante para la toma de decisiones estratégicas.

En este contexto se engloba el presente trabajo, donde se pretende optimizar el almacenamiento de estos datos, enfocándose sobre todo en las palabras y frases que se pueden encontrar en las páginas web de las empresas. Por ello, se pretende desarrollar una base de datos que sea capaz de almacenar toda esta información extraída, siendo necesario que esté orientada a la optimización de la búsqueda de texto.

La visualización de datos es un aspecto crucial en el análisis de la huella digital. La principal razón para almacenar y gestionar estos datos es poder extraer conclusiones significativas y datos de valor. Sin embargo, el simple almacenamiento de datos no es suficiente; se requiere una representación visual efectiva para comprender y comunicar la información de manera clara y accesible. Las visualizaciones permiten transformar los datos en gráficos, tablas y otros formatos visuales intuitivos. Estas representaciones visuales facilitan la identificación de patrones, tendencias y relaciones entre los datos, lo que implica que al presentar la información de forma visual, se puede obtener un valor oculto de los datos que sería difícil de percibir de otra forma. Además, las herramientas de visualización brindan una forma ágil de interactuar con los datos. Los usuarios pueden explorar y filtrar la información, realizar análisis y obtener respuestas a preguntas específicas.

En resumen, el almacenamiento de datos de la huella digital tiene como objetivo principal comprender el comportamiento de los individuos y las organizaciones que la generan. Las visualizaciones son herramientas esenciales en este proceso, ya que permiten acceder a los datos de forma sencilla y comprensible. A través de las visualizaciones, se puede obtener un mayor valor del conjunto de datos, identificar patrones relevantes y comunicar los resultados de manera efectiva.

1.1 Objetivos

Los objetivos de este Trabajo de Fin de Grado se pueden diferenciar claramente entre principal y secundarios, siendo los secundarios extensiones y mejoras del objetivo principal.

1.1.1. Objetivo principal

- **Crear una visualización interactiva de tendencias de los contenidos de las páginas web de las empresas**

1.1.2. Objetivos secundarios

- **Comprender los datos**. Una visualización de datos efectiva comienza por entender qué información se está presentando y los objetivos que se pretenden lograr con ella. Para conseguirlo, es necesario comprender los datos que se están utilizando y ser capaz de identificar la información relevante que se puede extraer de ellos
- **Conocer y comparar las tecnologías de base de datos disponibles**. Para ello, se realizará una comparativa exhaustiva de las características de cada tipo de base de datos no convencional y se evaluará su capacidad para soportar grandes cantidades de datos y consultas simultáneas. Además, la base de datos debe estar orientada al almacenamiento de texto
- **Diseñar y poner en marcha la base de datos**. Basándose en el conocimiento adquirido sobre los datos y las tecnologías de base de datos, se debe diseñar una base de datos que sea óptima para alcanzar el objetivo principal. Esto implica definir las tablas, los campos, las relaciones y los índices necesarios
- **Optimizar los tiempos de consulta**. La base de datos debe permitir reducir los tiempos de respuesta al mínimo posible
- **Diseñar y poner en marcha la herramienta de visualización**. Como último objetivo secundario, se propone diseñar una herramienta que permita crear las visualizaciones interactivas de tendencias de los contenidos de las páginas web de las empresas. Esta tarea implica la selección de los gráficos adecuados que describan de la mejor forma los datos disponibles

1.2 Metodología

En primer lugar, se ha llevado a cabo una revisión de la literatura existente relacionado con la huella digital y los datos obtenidos a partir de esta. En segundo lugar, se ha realizado una primera toma de contacto con los datos y una posterior limpieza de estos datos obtenidos a partir de un trabajo previo [2]. Esta limpieza de datos se ha realizado

mediante un script de Python que ha permitido en pocos pasos realizar la limpieza de texto.

Posteriormente, se ha llevado a cabo una revisión bibliográfica para obtener información sobre los distintos tipos bases de datos convencionales y no convencionales, herramientas y tecnologías utilizadas para el desarrollo de aplicaciones web. Tras esto, se ha realizado una comparativa de las características de diferentes bases de datos no convencionales para seleccionar la que mejor se adapte a las necesidades del proyecto, evaluándose su capacidad para soportar grandes cantidades de datos y consultas simultáneas, y sobre todo considerándose su orientación al almacenamiento de texto. Además, se ha procedido a diseñar y desarrollar la base de datos escogida, teniendo en cuenta las características específicas de la base de datos y los requisitos del proyecto. Este paso es uno de los puntos donde se ha centrado el trabajo debido a su extensión e importancia.

Por otro lado, se ha diseñado y desarrollado el *dashboard*, utilizando la tecnología Django para crear una aplicación web interactiva y de fácil uso. Se ha tenido en cuenta cual es el objetivo de visualización y como desarrollarlo de la forma más intuitiva para el usuario, incluyéndose otras funcionalidades interesantes. Además, se ha asegurado que el *dashboard* permita visualizar gráficos obtenidos a partir de los datos almacenados en la base de datos y que esté optimizado para su uso en diferentes dispositivos. Este es el otro punto importante donde se ha centrado el trabajo.

Se han evaluado los costes de despliegue teóricos de cada una de las posibilidades de base de datos, así como los costes reales de la base de datos final y de la aplicación desarrollada. Siempre teniendo en cuenta los costes energéticos y ambientales asociados con el uso de servidores y buscando minimizar el impacto medioambiental.

Por último, se ha creado una interfaz de usuario que permite insertar nuevos datos en la base de datos y que, instantáneamente, van a estar representados en la aplicación. Para ello, el usuario necesitará tener las claves de acceso y el dominio del servidor, así como el directorio de los nuevos datos.

El presente proyecto utiliza la metodología CRISP-DM [3] como enfoque para extraer valor a los datos. Cada tarea anteriormente descrita se engloba dentro de una fase de esta metodología:

- **Fase I. Entendimiento del negocio**. La fase inicial del proceso implica la exploración y análisis detallado de los datos disponibles, así como la comprensión de los objetivos del proyecto. Esta etapa se enfoca en convertir el conocimiento adquirido sobre los datos en una definición clara del problema y en un plan preliminar que servirá para alcanzar los objetivos establecidos. Aunque no se realiza un modelado de los datos, ya que han sido tratados previamente, esta fase sigue siendo esencial para comprender los objetivos del proyecto y determinar la mejor manera de alcanzarlos. Además, en esta fase se encuentra la tarea inicial de revisión de la literatura existente sobre el conjunto de datos obtenido

- **Fase II. Estudio y comprensión de los datos**. El objetivo de esta etapa es transformar los datos crudos obtenidos en la fase anterior en información procesada y lista para ser utilizada en la construcción de modelos que permitan obtener una visión general de lo que se puede conseguir con los datos. En esta fase se encuentra la tarea de la primera toma de contacto de los datos, donde se comprende que significa cada campo y que información relevante puede aportar

- **Fase III. Preparación de los datos**. Esta etapa engloba todas las actividades necesarias para construir el conjunto final de datos que serán utilizados en las herramientas de modelado, a partir de los datos iniciales en bruto. Esta fase incluye una serie

de tareas importantes, tales como la selección cuidadosa de las tablas, registros y atributos que serán utilizados, así como la transformación y limpieza de los datos para su uso en las herramientas de modelado. El objetivo principal de esta etapa es garantizar la calidad y la coherencia de los datos finales que se utilizarán para el modelado. En esta fase se engloba todo el proceso de limpieza realizado, tanto la generación de scripts como la creación de las tablas pertinentes

- **Fase IV. Modelado**. Esta fase implica la selección y aplicación de las técnicas de modelado que sean pertinentes para el problema en cuestión, con el objetivo de obtener los mejores resultados posibles. En este caso, las tareas correspondientes con la Fase IV se centran en el diseño de la visualización, tales como los gráficos escogidos y qué datos van a mostrar cada uno de ellos

- **Fase V. Evaluación**. En esta fase del proyecto, se ha construido un modelo que cumple o trata de cumplir con los requisitos establecidos. Es fundamental llevar a cabo una evaluación exhaustiva del modelo, revisando cuidadosamente los pasos ejecutados para su construcción y comparando los resultados obtenidos con los objetivos de negocio establecidos. Además, es necesario asegurarse de que se hayan considerado todas las cuestiones importantes del negocio en el modelo construido. En esta fase se encuentran las validaciones de rendimiento de la base de datos analizando si esta cumple con los objetivos predefinidos. En el caso de la aplicación, también se analiza su rendimiento, así como si es capaz de cumplir con los objetivos de visualización de los datos

1.3 Estructura de la memoria

El contenido de la presente memoria se estructura en las siguientes secciones:

1. **Introducción**. En esta sección se ha descrito el contexto del problema, así como su conexión con la creación de una base de datos y de una aplicación. Además, se han expuesto los objetivos principales que se persiguen con este proyecto.

2. **Estado del arte**. Aquí se realiza una revisión teórica de los conceptos relevantes para el estudio, así como un análisis más detallado de los aspectos abordados en la introducción. En concreto, se analizan diversas fuentes de información relacionadas con la huella digital de las empresas en internet.

3. **Análisis del problema**. En esta sección, se hace un análisis de diversas consideraciones previas a la realización del trabajo, tales como el marco legal y ético, el plan de trabajo realizado y el presupuesto estimado para llevar a cabo el proyecto. En el caso del presupuesto, se analizan los recursos necesarios, tanto humanos como técnicos, y se asigna un valor económico a cada uno de ellos. Se consideran los gastos relacionados con la adquisición de hardware y software y el tiempo dedicado por el personal involucrado.

4. **Preparación y comprensión de datos**. Esta sección describe todo el procedimiento de recogida, transformación y limpieza del conjunto de datos. En la primera parte, se describe el conjunto de datos y como han sido recogidos, mientras que en la segunda se describen que técnicas se han utilizado para obtener nueva información a partir de combinar varios conjuntos de datos.

5. **Selección y diseño de la base de datos**. Esta sección sirve como resumen a alto nivel de los diferentes tipos de *datasets* disponibles, así como sus ventajas y desventajas.

Además, se justifica la elección de ElasticSearch como tecnología y se explica todo el diseño y desarrollo de esta base de datos.

6. **Diseño y desarrollo de la herramienta de visualización**. Se justifica la elección de Django como herramienta de visualización web y se detalla todo el procedimiento llevado a cabo para la creación y diseño de la aplicación, así como los gráficos empleados. Por último, se explica todo el procedimiento de despliegue de la aplicación en un servidor AWS.

7. **Interfaz gráfica**. En este apartado se presenta en detalle la interfaz gráfica desarrollada para la herramienta de visualización. Se describe la estructura y disposición de los elementos, así como las funcionalidades que ofrece. Se incluyen capturas de pantalla y se explican las interacciones posibles con la interfaz.

8. **Estimación de costes**. En este punto, se analizan todos los costes reales relacionados con el proyecto, tales como el coste económico de los servidores, el coste temporal de la limpieza, transformación y subida de datos al servidor, coste espacial y, por último, el coste energético y medioambiental, ya que es necesario que se tome consciencia sobre el coste que puede tener este trabajo sobre el medio ambiente.

9. **Conclusiones**. Aquí se hace una reflexión sobre el trabajo realizado y los resultados obtenidos, así como en qué cosas se han cometido errores, como se podrían haber evitado y, en caso de haberlo hecho, como solucionarlo. Además, se ha descrito el legado y se ha relacionado el trabajo con los estudios cursados.

10. **Bibliografía**. Esta sección contiene las referencias a los artículos, libros, conferencias, etc. utilizados para el desarrollo de la memoria.

CAPÍTULO 2
Estado del arte

La huella digital es el rastro que cada empresa deja tras su actividad en la red. Dicha huella se ha ido convirtiendo en una fuente importante de información económica durante las últimas décadas, debido al aumento del uso de internet en todos los negocios y la creciente digitalización de la economía. Esta cantidad ingente de información es una herramienta muy importante que puede ser utilizada por analistas y científicos de datos con el fin de obtener información relevante sobre la evolución del interés por ciertos temas.

2.1 Las páginas web de empresas como fuente de información

Generalmente, las empresas utilizan sus páginas web para promocionarse y describir que productos o servicios ofrecen, y también para interactuar con sus clientes y/o potenciales clientes, por lo que el contenido web está directamente relacionado con su actividad empresarial. Además, es una fuente de información que cambia diariamente, pero que gracias al registro que existe en internet, se pueden recuperar capturas históricas de los sitios web, pudiendo observar la evolución que han experimentado. Por estas razones, los sitios web de las empresas constituyen la cara visible e histórica de una empresa en internet. Algunos ejemplos del uso de esta fuente se pueden encontrar en [2].

A parte de este tipo de fuente, hay otros mecanismos para poder extraer información relevante de las empresas, como los metadatos web e información WHOIS [4, 5], el contenido de las redes sociales como Twitter, Instagram, Facebook y LinkedIn, entre otros [6, 7], y publicaciones en medios de comunicación u otros medios [4].

Sin embargo, es cierto que los datos de las páginas web son complejos de tratar, debido a la estructura variada y cambiante de los sitios web. Cada empresa puede tener su propio diseño y organización de la información en su página web, lo que dificulta la extracción y procesamiento automatizado de los datos. Todo esto implica dedicar una cantidad considerable de trabajo para recuperarlos, seleccionarlos, depurarlos y, finalmente, analizarlos en comparación con las fuentes de datos convencionales [8].

La gran ventaja que ofrece esta fuente de información es su amplitud y accesibilidad. A diferencia de otras fuentes de información sobre empresas, como informes financieros o bases de datos especializadas, la información obtenida a través de las páginas web y otros canales en línea está ampliamente disponible y puede ser consultada de una forma relativamente sencilla. Además, esta fuente de información difícilmente se verá desactualizada, ya que las empresas suelen mantener sus sitios web actualizados con regularidad, lo que permite obtener información en tiempo real sobre su actividad, productos y servicios [8]. Esto resulta especialmente útil para el seguimiento de empresas en sectores

dinámicos y en constante evolución. Otra ventaja es la gran diversidad de información disponible, ya que a través de las páginas web, se pueden obtener datos sobre la cultura corporativa [9], el desempeño empresarial [10], las estrategias comerciales [11] y el nivel de innovación [12], entre otros.

2.2 Google Trends como fuente de información

El creciente interés por parte de las empresas en posicionarse en internet ha llevado a una competencia feroz en la que las empresas necesitan diferenciarse para atraer a potenciales clientes mediante aplicaciones que ayuden a mejorar su posicionamiento en internet y, por tanto, ser más visible para su público objetivo. Las herramientas de análisis web, por ejemplo, son de gran valor ya que permiten registrar la actividad y el comportamiento de los usuarios en una página web y permiten realizar técnicas de SEO (*search engine optimization*). Un ejemplo es Google Trends, una herramienta de *search analytics* que muestra una medida de popularidad de los términos (palabras, frases, ideas) que han sido introducidos en la barra de búsqueda de Google.

Google Trends es una herramienta muy utilizada en el campo de la economía y la empresa con el objetivo de comprender a las personas o la sociedad a través de actividades de búsqueda (y así tratar de predecir el comportamiento) [13]. De hecho, hay estudios que tratan sobre las predicciones donde se señala que gracias a las búsquedas de los usuarios en línea podemos predecir su futuro comportamiento [14]. Unos ejemplos serían los beneficios del primer fin de semana para los cines o las ventas de videojuegos en su primer mes.

Otros estudios [15] también han descubierto la gran utilidad que tiene Google Trends mediante la identificación del punto de inflexión que ocurre repentinamente en el mercado para predecir la actividad económica relativa a las ventas de coches, ventas de casas y viajes. Si hay un aumento en las búsquedas de *concesionarios* o *vehículos de segunda mano* en una región en específico, se puede predecir que habrá un aumento en las ventas de vehículos. Esto también ha sido corroborado en otros trabajos [16] donde indican que el uso de la información obtenida a través de Google Trends puede ayudar a calcular la probabilidad de que los usuarios adquieran productos tecnológicos.

En suma, es por esto por lo que el uso de herramientas de análisis web contribuyen de cierta manera a la huella digital que cada empresa ha ido dejando en internet, debido a que registran y muestran la popularidad de los términos de búsqueda en un determinado período de tiempo y ubicación geográfica, permitiendo a las empresas conocer mejor a su público objetivo y adaptar su estrategia de marketing digital en consecuencia.

2.3 Crítica al estado del arte

Aunque la visualización de datos extraídos de internet no es un concepto nuevo, como se ha demostrado con herramientas como Google Trends, es un campo que aún no ha sido ampliamente explorado. Mientras que Google Trends facilita el análisis de la actividad económica mediante la información de las búsquedas de los usuarios en internet, el enfoque de este estudio es invertir el origen de los datos y centrarse en las propias empresas como fuente de información, utilizando los datos extraídos de sus páginas web. Es decir, en lugar de depender únicamente de las búsquedas de los usuarios, este enfoque se basa en aprovechar la gran fuente de información que las empresas proporcionan a través de sus sitios web.

CAPÍTULO 3
Análisis del problema

Con todo lo descrito anteriormente, queda claro que la huella digital puede ser una fuente importante de datos de la que poder nutrirse y obtener nueva información más relevante. Así pues, es necesario realizar un análisis más detallado del proyecto antes de comenzarlo estudiando su viabilidad, tanto legal como económica, y estableciendo una hoja de ruta.

3.1 Análisis del marco legal y ético

El marco legal es un conjunto de normas y principios que limitan el comportamiento de profesionales y organizaciones en la propiedad intelectual, el uso de datos sensibles y la ética. Por su parte, el marco ético hace referencia más bien a una guía sobre aspectos importantes a la hora de actuar en determinadas situaciones.

En el presente trabajo, se han utilizado datos cedidos por los autores de un proyecto anterior [2] relacionado con el tema tratado. A su vez, estos datos han sido extraídos en su totalidad de internet. Gran parte de ellos han sido obtenidos gracias a la herramienta "Wayback Machine"[1], que en sus términos de uso especifica que el acceso a las colecciones del archivo y su explotación se proporciona sin coste y únicamente se otorga con fines de investigación o educativos, lo que se cumple con el proyecto actual. Por otro lado, los datos obtenidos mediante las herramientas ORBIS[2] y SABI han sido bajo una licencia por suscripción, por lo que atendiendo a sus términos de uso también es legal el uso de estos datos por parte del proyecto.

En cuanto a la ética, la finalidad de este trabajo hace imposible que se puedan tomar decisiones discriminando por algún tipo de atributo protegido, ya que en este proyecto no procede. Los datos visualizados en el cuadro de mandos únicamente muestran información dividida por sectores, países, tamaños de empresas u otras casuísticas que no afectan al comportamiento humano ni influyen en su decisión. Además, cara al usuario final es imposible que extraiga información individual por una empresa en concreto, ya que esta información no está disponible desde el cuadro de mandos, haciendo improbable que se vea influenciado el comportamiento del usuario ante una empresa en específico.

Por último, el software principal utilizado en este trabajo ha sido Python, que se trata de un lenguaje de programación de código abierto. Python funciona bajo la licencia

[1] https://web.archive.org/
[2] https://www.bvdinfo.com/en-us/our-products/data/international/orbis

Python Software Foundation License (PSFL)[3], una licencia que permite la utilización de versiones derivadas sin la necesidad de que sean de código abierto.

3.2 Plan de trabajo

A continuación, en la Tabla 3.1 se presenta un plan de trabajo para el desarrollo del TFG, dividido en 7 fases o niveles y con una estimación aproximada de horas necesarias para cada una. Además, se ha incluido las horas reales de duración de cada una de las tareas. El día de inicio del trabajo se ha definido el 1 de febrero, mientras que la fecha límite se ha establecido en el 31 de mayo, con el objetivo de poder presentarse en la convocatoria de julio sin ningún tipo de problema:

Tabla 3.1: Plan de trabajo estimado y resultado real

Nivel	Descripción	Horas previstas	Horas reales
1	**Trabajo previo**	**15**	**20**
1.1	Definición del objetivo, alcance y motivación	5	5
1.2	Investigar y revisar la bibliografía relacionada	10	15
2	**Limpieza y transformación de datos**	**60**	**65**
2.1	Crear el script para unificar las bases de datos	30	35
2.2	Identificar datos relevantes y realizar transformaciones	10	10
2.3	Limpiar los datos resultantes	20	20
3	**Selección de la base de datos**	**30**	**25**
3.1	**Realizar una comparación de los tipos de base de datos**	**10**	**10**
3.1.1	Bases de datos convencionales	5	5
3.1.2	Bases de datos no convencionales	5	5
3.2	Seleccionar la base de datos más adecuada	20	15
4	**Diseño y desarrollo de la base de datos**	**60**	**65**
4.1	**Diseñar la estructura de la base de datos en local**	**30**	**40**
4.1.1	Insertar datos en la base de datos	10	20
4.1.2	Realizar pruebas de rendimiento	20	20
4.2	**Diseñar la estructura de la base de datos en la nube**	**30**	**25**
4.2.1	Insertar datos en la base de datos	5	10
4.2.2	Realizar pruebas de rendimiento	25	15
5	**Diseño y desarrollo de la aplicación web**	**90**	**110**
5.1	Definir requisitos de la aplicación	10	10
5.2	Escoger la tecnología más adecuada	10	10
5.3	Desarrollar la aplicación	60	80
5.4	Realizar pruebas de rendimiento	10	10
6	**Evaluación de costes**	**5**	**5**
7	**Redacción de la memoria de TFG**	**100**	**115**
	Horas totales	**360**	**400**

A continuación, se muestra el diagrama de Gantt resultante de la planificación prevista de tareas comentada anteriormente (figura 3.1). En este diagrama, se ha partido de una jornada laboral de 5 horas diarias los 5 días de la semana, con una única persona desarrollando el trabajo. Cada una de las actividades comienza cuando su antecesora ha terminado, es decir, no pueden empezar dos tareas a la vez (excepto la redacción del TFG,

[3] https://docs.python.org/3/license.html#terms-and-conditions-for-accessing-or-otherwise-using-python

que es una tarea que se realiza paralelamente a las demás). Este diagrama ha sido diseñado con la herramienta de Microsoft Project[4] teniendo en cuenta todas las restricciones anteriores.

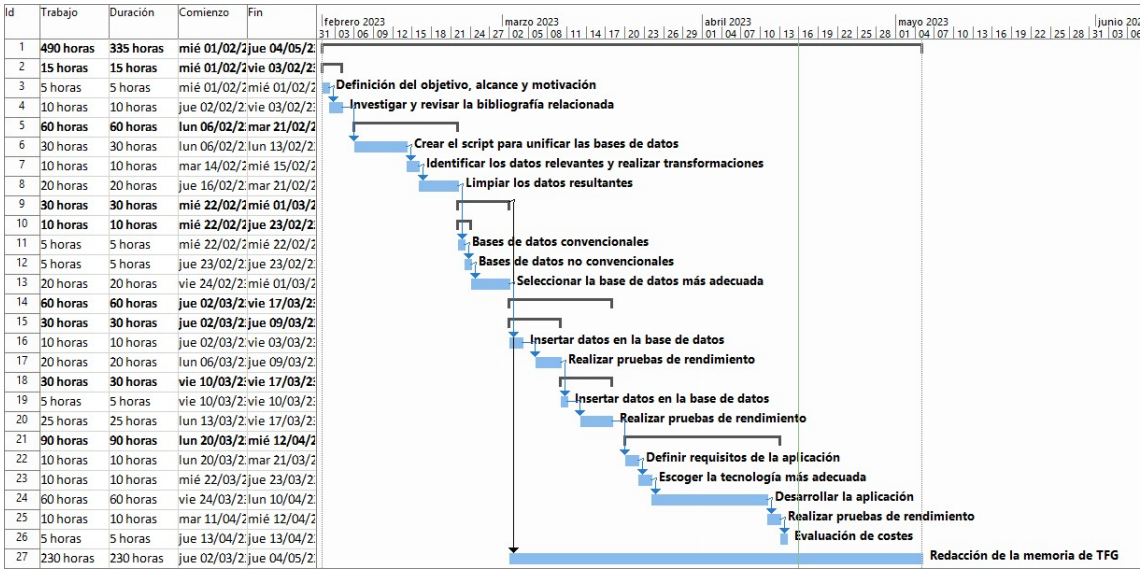


Figura 3.1: Diagrama de Gantt realizado a partir de las horas estimadas

Por otro lado, en la siguiente figura 3.2, se puede observar el tiempo real de ejecución de cada una de las tareas:

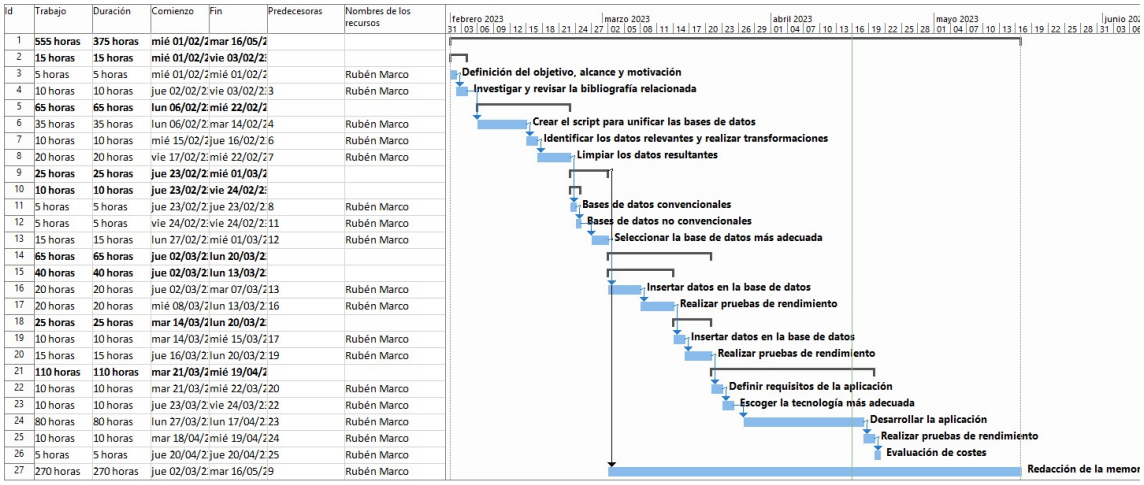


Figura 3.2: Diagrama de Gantt realizado a partir de las horas reales

En la planificación de este Trabajo de Fin de Grado, se han utilizado estimaciones basadas en la experiencia previa de las tecnologías empleadas anteriormente para asignar los tiempos de ejecución de cada tarea, así como un margen extra de 5 horas para cada una de ellas en caso de imprevistos. Sin embargo, se ha otorgado el mismo margen para todas las tareas, lo que puede inducir a error ya que no se tiene en cuenta su complejidad o los posibles inconvenientes que puedan surgir durante su ejecución. Es necesario tener en cuenta que cada tarea debe ser evaluada individualmente para determinar el margen necesario según su complejidad y los posibles imprevistos que puedan surgir. Por lo tanto, se debe establecer un margen de tiempo adecuado y suficiente para cada tarea. Al

[4] https://www.microsoft.com/es-es/microsoft-365/project/project-management-software?ms.url=mscomproject&rtc=1

hacer esto, se pueden evitar posibles retrasos en la ejecución del proyecto y garantizar su éxito. En el caso de los errores de cálculo en las tareas relacionadas con la creación de la aplicación, se debe a la falta de experiencia previa con esta tecnología (Django).

Por otro lado, la falta de experiencia en la realización de un Trabajo de Fin de Grado ha derivado en errores de subestimación y sobrestimación de horas invertidas en cada fase del trabajo. Estos errores de medición han resultado en un retraso de casi dos semanas, aunque es cierto que no ha sobrepasado la fecha límite definida en la estructuración del proyecto, en parte gracias a la formación académica recibida durante los cuatro años del grado en materias como Proyecto I, Proyecto II, Proyecto III y Gestión de Proyectos. A pesar de que estas materias se enfocan principalmente en el trabajo grupal, contribuyeron significativamente a que la estimación temporal fuese similar a la real para un trabajo individual.

3.3 Presupuesto

En primer lugar, la duración total del proyecto se estima inicialmente en 360 horas, aunque finalmente acaban siendo 400 horas. Cabe señalar que un TFG de Ciencia de Datos equivale a 12 créditos ECTS y cada ECTS a 25 horas de trabajo, lo que implica un total de 300 horas, por lo que ese umbral mínimo se supera sin complicaciones.

Por otro lado, es necesario tener en cuenta el coste asociado a la contratación de un analista de datos junior, cuyo salario bruto se estima en unos 21.000€ anuales[5] más un 23,6 % del salario destinado a las contingencias comunes, lo que se traduce en un coste anual de 25.956€ y por hora de 13,5€. Teniendo esto en cuenta, el presupuesto previsto para la contratación de un analista de datos junior asciende a 4.860€.

En lo que respecta al software utilizado en el proyecto, se ha optado por dos herramientas: Python y AWS. Es importante destacar que Python es un lenguaje de programación de código abierto, por lo que no supone ningún coste adicional. En cuanto a AWS, se ha realizado un análisis detallado del coste esperado en la Sección 8.1, previéndose un gasto mensual casi 100€ durante los dos meses de pruebas previstos en el proyecto, lo que implica un coste total de 200€. En el caso del hardware y gasto energético, suponiendo un gasto de 1500€ en un ordenador de sobremesa y una vida útil de 7 años se espera un gasto de 10€ durante todo el proyecto, mientras que el gasto energético se calcularía suponiendo un precio del KW/h de 0,35€ y una potencia de 130W durante 5 horas al día, sumando un total de 3,5€.

En suma, el presupuesto general que se ha realizado a base de previsiones asciende hasta un total de 5.063,5€.

[5] Estimación obtenida por experiencia propia del autor

CAPÍTULO 4
Preparación y comprensión de datos

4.1 Descripción del conjunto de datos y del proceso de recopilación

El conjunto de datos utilizado en este estudio se ha obtenido a partir de un trabajo anterior realizado por [2]. Para la recolección de datos en dicho trabajo se utilizaron las bases de datos ORBIS y SABI. Estos servicios de bases de datos por suscripción contienen información sobre empresas, y permiten al usuario filtrar por tamaño, sector, región, país, etc. La principal diferencia entre ORBIS y SABI es su alcance geográfico, ya que ORBIS tiene un alcance más mundial, mientras que SABI se enfoca únicamente en España y Portugal, lo que hace que la información sea más específica y correcta.

La población relevante para el estudio está compuesta por empresas ubicadas en España, Francia, Alemania e Italia que cuentan con un sitio web corporativo. Estas empresas se clasifican según las secciones del código NACE (Nomenclatura estadística de actividades económicas de la Comunidad Europea) [17] y su tamaño:

- Micro: empresas con menos de 10 empleados
- Pequeñas: empresas entre 10 y 49 empleados
- Medianas: empresas entre 50 y 249 empleados
- Grandes: empresas con más de 249 empleados

Para la definición de las secciones y el número de empleados se prefirió el muestreo aleatorio estratificado no proporcionado en lugar del proporcionado para la población mencionada anteriormente. El muestreo estratificado habría permitido reducir el tamaño total de la muestra, pero dificultaría la presentación de resultados desagregados por desglose NACE. Para un nivel de confianza del 95 % y un margen de error e=7 %, con P=Q=50 %, se seleccionaron alrededor de 200 empresas en cada grupo. En aquellos grupos donde la población total de empresas no alcanzó este número, se realizó un muestreo exhaustivo. En total, la muestra de datos contiene 32.248 empresas. Sobre el total, se reparten 8.331 en España, 6.940 en Francia, 8.200 en Italia y 8.777 en Alemania.

Con todo esto, se crea la base de datos de las empresas, donde cada empresa tiene informado su nombre, URL, país, región, principal sección de actividad y su tamaño.

Para obtener los datos del contenido de las páginas web de cada empresa se utiliza la herramienta "Wayback Machine". Esta herramienta fue creada por la compañía *The*

Internet Archive, que desde 1996 empezó a archivar páginas web y actualmente se ha convertido en el archivo de capturas de páginas web de acceso público más grande del mundo. "Wayback Machine" hace lo que comúnmente ellos llaman *capturas* y lo que los usuarios mejor entienden, ya que capturan las páginas webs tal y como eran en ese momento, con el mismo diseño e incluso imágenes, lo que lo hace mucho más visual y fácil de comprender, aunque realmente lo que se hace es un proceso llamado *crawling* o rastreo, en el que robots rastreadores visitan sitios web y guardan copias de las páginas en su base de datos. Luego, estas páginas se indexan y se hacen accesibles para que los usuarios las busquen y visualicen. La plataforma utiliza un conjunto de algoritmos para determinar qué páginas guardar y con qué frecuencia actualizarlas. A medida que los sitios web cambian y evolucionan, "Wayback Machine" sigue capturando y archivando nuevas versiones, lo que permite a los usuarios ver cómo han cambiado los sitios web con el tiempo [18].

Con todo esto, se accede a los datos web de cada empresa desde 2007 hasta 2021. Los datos analizados en este estudio suman un total de 483.720 observaciones, debido a las 32.248 empresas muestreadas durante 15 años.

Con la información de cada una de las extracciones, en el estudio se creó un software con el que poder descargar el contenido html y texto de las páginas web de las compañías y almacenarlo en un disco local. El contenido descargado sería aquel más cercano al ecuador de cada año (30 de Junio) y, en caso de no existir ese contenido, se trata como *Not Available* (NA).

En suma, los datos extraídos se dividen en dos bloques:

- Lista de empresas: fichero donde se especifica información de cada una de las empresas contempladas en el estudio, como el país, nombre, dominio web, sección y tamaño
- Contenido web: ficheros con el contenido de las páginas web durante 14 años de cada una de las empresas muestreadas en la lista de empresas

4.2 Transformación y limpieza

En el contexto del análisis de datos, una de las primeras tareas es la de tratar y limpiar los datos para garantizar su calidad y fiabilidad. En cuanto a la transformación de los datos, este es un paso crucial para garantizar la calidad y fiabilidad de la base de datos final.

La transformación de los datos se divide en dos bloques o fuentes diferentes. El primer bloque corresponde al fichero de los datos de cada empresa, donde se pueden hallar datos como el tamaño, la sección, su país, número de empleados, etc. Este fichero constituye una fuente importante de información para el análisis, ya que proporciona datos relevantes sobre cada empresa.

El segundo bloque corresponde a una gran cantidad de ficheros de texto correspondiente al contenido de las páginas web de cada empresa durante un periodo de años. Estos ficheros contienen información valiosa sobre el comportamiento de las empresas en línea y pueden proporcionar información sobre su presencia en internet o su estrategia de marketing, entre otros. Los nombres de los ficheros sirven como identificadores únicos y otorgan gran información, ya que indican el año de la captura del contenido, el tipo de archivo y el dominio al que hace referencia, pudiendo así identificar a que página web se le hizo esa captura (figura 4.1).

Figura 4.1: Estructura de los títulos de los ficheros de texto

Para construir la base de datos final, es necesario combinar ambos bloques de datos(figura 4.2). Para ello, se hace uso de la variable url del fichero de datos de las empresas, ya que en este campo se encuentra el dominio web de cada empresa. En el segundo bloque que corresponde a los ficheros de contenido utilizamos la url que podemos obtener del propio nombre del archivo. Con todo esto, se consigue un campo común con el que poder cruzar los datos.

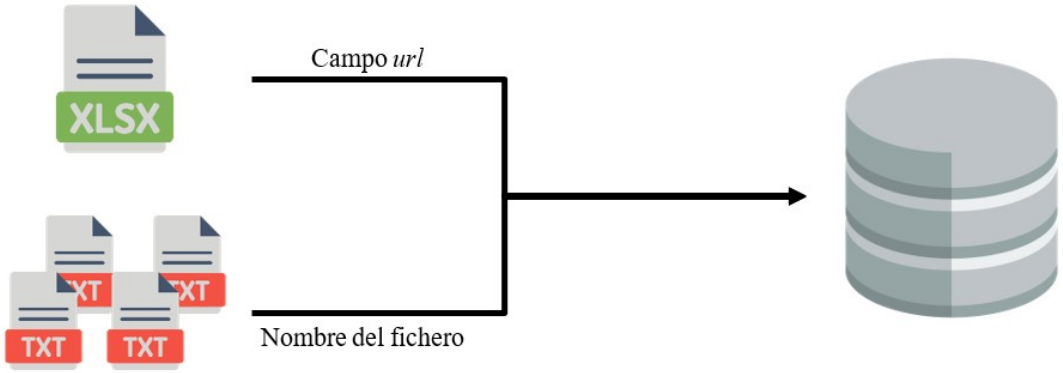


Figura 4.2: Esquema gráfico de la unión de ambos conjuntos de datos

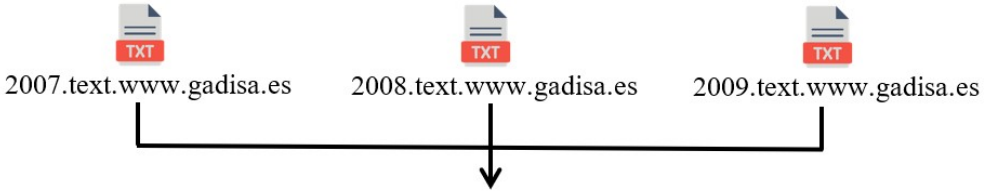
Con esta combinación de datos, se pueden extraer información relevante sobre cada empresa por años, lo que permite realizar análisis comparativos y de tendencias a lo largo del tiempo.

El siguiente paso realizado es el de transformar toda esta lógica a código Python. Para ello, es clave tener en cuenta que todos los ficheros de texto los tendremos repartidos en un directorio y que el fichero de datos de empresas puede estar en otro independiente. Es por eso, que se crearán dos funciones que extraigan la información necesaria.

Por un lado, se crea una función que tiene como objetivo extraer todos los dominios presentes en el campo url del Excel de las empresas. Por otro lado, se recorren todos los ficheros de texto y se obtendrá y guardará el título de estos. Tras este paso, se extrae el dominio web y el año de la captura a la que hace referencia ese fichero.

Con toda esta información obtenida, ya podemos crear la base de datos local en la que se contendrá toda la información necesaria. Esta base de datos se basará en el Excel de la información de las empresas y será simplemente una extensión de este fichero, añadiéndose dos campos nuevos. El primero de ellos es el campo *year*, que como se ha

comentado anteriormente, se obtiene del fichero de contenido, mientras que el segundo campo es *content*, que contendrá el contenido de cada fichero en su respectiva posición cuadrado por año y empresa a la que corresponde. Un ejemplo de cómo se obtiene la información de los ficheros y se generan nuevos datos a partir de ellos se puede visualizar en la figura 4.3.



Name	url	...	Year	Content
Gadisa SA	www.gadisa.es	...	2007	...
Gadisa SA	www.gadisa.es	...	2008	...
Gadisa SA	www.gadisa.es	...	2009	...

Figura 4.3: Creación de nueva información a partir de otros conjuntos de datos

Sin embargo, antes de introducir la información de cada fichero en el campo *content*, éste deberá sufrir un proceso de limpieza, ya que, con una simple observación de unos pocos ficheros, se puede contemplar la cantidad de espacios, saltos de líneas, tabulaciones e incluso palabras irrelevantes que encontramos en cada fichero. Es por ello por lo que el proceso de limpieza se centrará en estos aspectos.

Toda esta limpieza se puede realizar con código Python y la librería *re*. Esta librería[1] está ya incorporada en Python y proporciona soporte para el uso de expresiones regulares (*regular expressions*). Las expresiones regulares son patrones de búsqueda que se utilizan para identificar y manipular cadenas de texto.

En primer lugar, se identifican todas las palabras del fichero de texto, dejando fuera los valores numéricos y los caracteres especiales ya que no son relevantes para el proyecto. Además, se realizan otros procesos como el de pasar todas las palabras a minúsculas, eliminar saltos de línea, tabulaciones y espacios en blanco.

Por otro lado, se crea una lista de *stopwords* con el fin de eliminar palabras que tampoco son relevantes. Estas palabras no pertenecen realmente al contenido de la página web y se generan debido a la herramienta "Wayback Machine", utilizada para la extracción de la información. La lista de *stopwords* está compuesta por las palabras "wayback", "machine", "http", "https", "wm", "init", "archive", "wombat", "org", "static", "spatial", "alexa" y "crawls".

El último paso de la limpieza de los contenidos es el de eliminar duplicados, puesto que este dato no es relevante en el estudio. Esto se realiza mediante el uso de un set o conjunto. El set es un tipo de dato incorporado en Python que se utiliza para almacenar colecciones de datos y que no permite duplicados.

Con todo esto, la base de datos local final que se obtiene constará con un total de 11 campos, siendo 3 de ellos numéricos y 8 de tipo carácter (tabla 4.1).

Cabe destacar que por el momento y para el diseño de los gráficos, los campos *name*, *region*, *url* y *domain* no son necesarios y podrían eliminarse. A pesar de ello, cabe señalar que almacenar estos datos en la base de datos no supone un coste espacial significativo y

[1] https://docs.python.org/es/3/library/re.html

Tabla 4.1: Campos finales de la base de datos

Campo	Tipo	Descripción
id	NUMBER	Identificador de la empresa
name	VARCHAR	Nombre de la empresa
region	VARCHAR	Región de la empresa
url	VARCHAR	Dirección web de la empresa
domain	VARCHAR	Dominio de la empresa
employees	NUMBER	Número de empleados de la empresa
country	VARCHAR	País de la empresa
size	VARCHAR	Tamaño de la empresa
section	VARCHAR	Sección de la empresa
year	NUMBER	Año del contenido
content	VARCHAR	Contenido obtenido en la captura de la página web

pueden resultar útiles en futuras actualizaciones y cambios en los objetivos de visualización, por lo que por el momento van a permanecer en la base de datos.

CAPÍTULO 5
Selección y diseño de la base de datos

La selección de una base de datos apropiada para el trabajo en cuestión es crucial y primordial, puesto que este es el eje donde gira todo el proyecto. Se debe escoger un tipo de base de datos que trabaje bien sobre grandes volúmenes de datos y sea capaz de realizar búsquedas rápidas y eficientes sobre sus campos. Es probable que una cantidad de 300.000 registros no sea tan voluminosa como pueda parecer, pero sí que lo es el contenido de texto de cada uno de estos registros, puesto que cada uno de ellos cuenta con una gran cantidad de palabras. Es por eso por lo que es necesario realizar un estudio exhaustivo de las posibilidades, obteniendo sus ventajas e inconvenientes y comparándose entre sí.

5.1 Bases de datos relacionales

Una base de datos relacional es una colección de elementos de datos organizados en tablas formalmente descritas que permiten el acceso y la reorganización de datos de diversas formas. Las tablas, llamadas relaciones, contienen una categoría de datos descrita en columnas similares a las hojas de cálculo, y cada fila contiene una instancia única de datos para la correspondiente categoría. Al crear una base de datos relacional, se aplican dominios de valores posibles junto con restricciones a los datos. La relación entre las tablas hace que se consideren "relacionadas". Los sistemas de bases de datos relacionales requieren pocas suposiciones sobre cómo se extraerán los datos de la base de datos, por lo que la misma base de datos puede ser vista de varias formas diferentes. La mayoría de los sistemas de bases de datos relacionales utilizan el lenguaje SQL para acceder y modificar los datos almacenados. Los beneficios de la base de datos relacional incluyen la autodocumentación, la facilidad de actualización y eliminación de datos, la capacidad de resumen, recuperación y generación de informes, la estructura tabular y la simplicidad en los cambios del esquema de la base de datos [19].

5.1.1. Oracle vs MySQL

Unas de las bases de datos relacionales más utilizadas son Oracle SQL y MySQL. En líneas generales, MySQL es más eficaz y rápido en páginas webs, mientras que Oracle ofrece muchas más herramientas y robustez y suele ser utilizado en el caso de grandes cantidades de almacenamiento de datos por grandes compañías. Por último, es posible inferir que para los usuarios que manejan una base de datos con una gran cantidad de tablas y datos, Oracle es el sistema de gestión más efectivo, dado que tiene un mejor

desempeño al realizar consultas que involucran numerosos procesos. Por otro lado, si el objetivo es agregar datos y hacer pocas consultas, el más adecuado sería MySQL, ya que es más rápido al insertar o eliminar información. En conclusión, MySQL es más eficiente en inserción y eliminación de datos, mientras que Oracle se desempeña mejor al consultar registros [20].

5.1.2. Desventajas de las bases de datos relacionales

Como todas, las bases de datos relacionales tienen ciertas desventajas que son importantes comentar. La falta de escalabilidad es una de las principales limitaciones, ya que, después de cierto punto, es necesario distribuir la base de datos en múltiples servidores. Además, la estructura en forma de tablas puede generar complejidad si los datos no se pueden encapsular adecuadamente. Muchas funciones provistas por estas bases de datos no se utilizan, lo que aumenta tanto el coste como la complejidad del sistema. La base de datos relacional utiliza SQL, que funciona bien con datos estructurados, pero no tanto con datos no estructurados, lo que hace que el trabajo con grandes cantidades de datos pueda ser difícil. Además, el procesamiento de datos suele ser muy lento ante grandes cantidades de datos. Finalmente, si la cantidad de datos es muy grande, la partición de la base de datos en múltiples servidores puede plantear problemas para unir tablas [19].

Con todas estas desventajas, surgió la necesidad de crear una forma más eficiente de almacenar y acceder a grandes cantidades de datos, lo que llevó al desarrollo de bases de datos no relacionales.

5.2 Bases de datos no relacionales

Las bases de datos no relacionales o NoSQL no organizan los datos utilizando una estructura de tabla de filas y columnas, a diferencia de las bases de datos relacionales que lo hacen. En su lugar, las bases de datos no relacionales utilizan un modelo de almacenamiento específico para cada forma de base de datos. El término *NoSQL* se refiere a almacenes de datos que no utilizan SQL para consultar sus datos a pesar de que admiten solicitudes compatibles con SQL. Además, la ejecución de consultas varía significativamente de la de las bases de datos relacionales.

Las bases de datos no relacionales se utilizan a menudo cuando se necesita organizar grandes cantidades de datos complejos y diversos. Por ejemplo, una tienda grande puede tener una base de datos en la que cada cliente tenga su propio documento que contenga toda su información, desde el nombre y la dirección hasta el historial de pedidos y la información de la tarjeta de crédito. A pesar de sus formatos diferentes, cada una de estas piezas de información se puede almacenar en el mismo documento.

Además, las bases de datos no relacionales son más rápidas que las bases de datos relacionales, ya que una consulta no tiene que examinar varias tablas para obtener una respuesta. Por lo tanto, las bases de datos no relacionales son adecuadas para almacenar datos en constante cambio o para aplicaciones que manejan una amplia gama de tipos de datos. Pueden manejar grandes volúmenes de datos complejos y no estructurados, y ayudan a las aplicaciones en constante evolución que incluyen una base de datos dinámica que puede cambiar rápidamente [21].

Por otro lado, existen diferentes tipos de bases de datos no relacionales o NoSQL:

- Almacenes de datos de documentos: la forma más parecida a los almacenes de datos relacionales y utilizan estructuras como JSON o XML. Ejemplos de estos serían ElasticSearch, Solr y MongoDB

- Almacenes de datos clave/valor: almacenan una lista de pares de datos, donde cada par consta de una clave y su correspondiente valor, dentro de una entidad. Estos sistemas no almacenan los datos en tablas relacionales, lo que los hace altamente particionables y permite un escalado horizontal en escalas que otros tipos de bases de datos no pueden lograr. Además, destacan por ser muy eficientes tanto para lectura como para escritura. Ejemplos de bases de datos: Redis y BigTable

- Almacenes de datos en grafos: se trata de nodos y aristas que representan entidades y conexiones entre ellas respectivamente. El propósito principal es permitir que una aplicación realice consultas eficientes que exploren rápidamente la red de nodos y aristas, y analicen las relaciones entre las entidades. Un ejemplo sería FlockDB

- Almacenes de datos tabulares: Las bases de datos NoSQL tabulares combinan características de los sistemas de bases de datos tabulares y los de clave/valor. Utilizan un formato de tabla que permite la variación en la forma en que se nombran los datos dentro de cada fila de la misma tabla, traduciéndose en consultas más rápidas. Un ejemplo muy conocido es Cassandra

5.2.1. Desventajas de bases de datos no relacionales

Las bases de datos NoSQL también tienen algunas limitaciones que deben considerarse.

En primer lugar, la fiabilidad de las bases de datos NoSQL es menor que la de las bases de datos relacionales. Las bases de datos NoSQL carecen de un estándar universalizado, lo que significa que cada marca utiliza un esquema único. Algunos sistemas, como MongoDB, no tienen un esquema definido, mientras que otros, como ElasticSearch, tienen un esquema dinámico. En algunos casos, la estructura se asemeja a las bases de datos relacionales, como sucede con Cassandra. Esto hace que cada sistema tenga sus propias ventajas y desventajas, lo que requiere de un conocimiento previo para elegir la base de datos NoSQL adecuada para una situación específica. Por último, la función JOIN no está disponible, ya que las relaciones se establecen de manera diferente, lo que puede dificultar la realización de consultas complejas [21].

5.3 Comparativa y conclusiones

Al comparar bases de datos relacionales y no relacionales (tabla 5.1), es importante recordar que estos dos tipos de bases de datos son útiles por derecho propio, pero para diversos propósitos y casos de uso. No hay una base de datos que sea adecuada para todos, y tanto las bases de datos relacionales como las no relacionales tienen sus usos. Sin embargo, las bases de datos no relacionales tienen la escalabilidad y versatilidad para adaptarse a las necesidades del mercado en evolución, y se comporta mejor ante grandes cantidades de datos, por lo que esta es la opción más interesante para el estudio realizado [21].

Tabla 5.1: Ventajas y desventajas entre las bases de datos relacionales y no relacionales

	Ventajas	Desventajas
Relacional	Integridad de los datos	Procesamiento de datos lento
	Exactitud de los datos	Hardware caro
	Estructura básica fácil de entender	Difícil escalabilidad
	Fácil acceso a los datos	Gran complejidad
	Más seguro	Mal funcionamiento ante datos no estructurados
No relacional	Fácil manejo de datos no estructurados	Funcionalidad limitada en consultas complejas
	Gran rendimiento	Sin esquemas definidos
	Mayores facilidades en escalabilidad	Menor fiabilidad

5.4 Elección de la base de datos. ElasticSearch

Cuando se trata de escoger una base de datos para un proyecto, es importante considerar todas las características y funcionalidades de cada opción disponible. Como hemos visto, cada tipo de base de datos tiene sus propias fortalezas y debilidades, y es necesario seleccionar la que mejor se adapte a las necesidades del proyecto. Sin embargo, el proyecto ya está algo sesgado puesto que uno de los objetivos es el de buscar formas alternativas a las bases de datos tradicionales para almacenar nuestra base de datos y mejorar la eficiencia, por lo que por todo esto, se va a utilizar una base de datos no relacional.

Para un proyecto que requiere almacenar grandes cantidades de texto y realizar búsquedas rápidas y precisas, la mejor opción sería una base de datos de almacenamiento de datos de documentos. Esto se debe a que estas bases de datos están diseñadas para manejar datos no estructurados, como texto, y ofrecen capacidades de búsqueda avanzadas.

Entre las opciones de bases de datos de almacenamiento de datos de documentos, ElasticSearch es una excelente elección. Esta base de datos de búsqueda distribuida utiliza el motor de búsqueda de código abierto de Apache Lucene para indexar y buscar texto de manera eficiente. Además, ElasticSearch es altamente escalable y ofrece capacidades de búsqueda completas, lo que permite a los usuarios buscar cualquier palabra dentro de la base de datos, incluso si esa palabra está en un campo diferente. Por otro lado, permite la búsqueda de cadenas de texto completo. Su distribución y facilidad de uso hacen que sea muy accesible para comenzar a utilizar y escalar a medida que se agregan más datos.

Lo que diferencia a ElasticSearch de otros almacenes de documentos es que no es solo un almacén de documentos, sino un motor de búsqueda potente que permite realizar consultas en tiempo real. Su soporte para funciones como autocompletar, filtros geográficos y agregaciones de varios niveles, junto con su facilidad de uso, ha llevado a su amplia aceptación en la industria. Todo esto la hace ideal para proyectos que requieren el almacenamiento y búsqueda de grandes cantidades de texto [22].

Existen otras alternativas a ElasticSearch, tales como Solr, con resultados de rendimiento muy similares ya que se basan en Lucene, utilizan lenguajes de programación similares y tienen estilos y usos parecidos.

Sin embargo, ElasticSearch soporta más lenguajes de programación que Solr y, como se demuestra en el estudio [23], ElasticSearch tiene mejor rendimiento en cuanto la duración de la indexación en cantidades más pequeñas que Solr, aunque hay que tener en cuenta que en el estudio se cataloga cantidad pequeña a una muestra de 40 millones de

registros con entre 5 y 20 palabras cada uno, por lo que para el proyecto realizado es la opción con mejor rendimiento. Además, ElasticSearch obtiene mejores resultados de media en la búsqueda de palabras y, en las *queries per second* o consultas por segundo (QPS), ElasticSearch consigue realizar de media más consultas por segundo que Solr, lo que es muy interesante e importante para la aplicación que se desea desarrollar, debido a la necesidad de que ésta pueda ser capaz de soportar una gran cantidad de consultas simultáneamente por usuarios.

La información en ElasticSearch es organizada mediante índices y, a su vez, los documentos forman parte de los índices. Dentro de estos documentos se encuentran los campos, que harían referencia a cada una de las propiedades de ese documento. Por último, ElasticSearch utiliza los documentos indexados para desarrollar esquemas o *mappings*.

Con el objetivo de poner en contexto y familiarizarse con la estructura de ElasticSearch, la tabla 5.2 muestra las equivalencias entre los términos utilizados en las bases de datos relacionales y las no relacionales.

Tabla 5.2: Equivalencias entre las bases de datos relacionales y ElasticSearch

Bases de datos relacionales	**ElasticSearch**
Base de datos	Índice
Esquema	Mapping
Fila	Documento
Columna	Campo

Sin embargo, es necesario explicar más en profundidad alguno de los términos en ElasticSearch para facilitar su comprensión.

5.4.1. Documentos

La unidad básica de información que ElasticSearch indexa son los documentos, que están expresados en JSON. Un documento en ElasticSearch se puede entender como una fila en una base de datos relacional, que representa una entidad específica que se desea buscar. Además del texto, un documento en ElasticSearch puede incluir cualquier dato estructurado codificado en JSON, como números, cadenas y fechas. Cada documento tiene una identificación única y un tipo de datos específico. En resumen, los documentos en ElasticSearch son una forma flexible y diversa de representar datos estructurados.

Veamos en la siguiente tabla 5.3 un ejemplo de equivalencia de datos entre los tipos de bases de datos relacionales y, por otro lado, en la figura 5.1 la equivalencia en formato JSON para los no relacionales.

Tabla 5.3: Ejemplo de una tabla convencional en una base de datos relacional

Id	**Nombre**	**Edad**	**Género**
1	María	25	F
2	Pedro	49	M

5.4.2. Índices

En ElasticSearch, un índice es una colección de documentos relacionados entre sí. De hecho, el índice es la entidad de más alto nivel contra la que se puede consultar. Además, cada documento correlaciona un conjunto de claves con sus valores correspondientes.

```
{
  "id":1,
  "nombre":"Maria",
  "edad":25,
  "genero":"F"
},
{
  "id":2,
  "nombre":"Pedro",
  "edad":49,
  "genero":"M"
}
```

Figura 5.1: Ejemplo de datos no relacionales con formato JSON

Para facilitar la comprensión, el índice se puede asemejar a una base de datos relacional, donde todos los documentos dentro de un índice suelen estar lógicamente relacionados. Por ejemplo, en un sitio web de comercio, se puede tener un índice para Clientes, otro para Productos, y otro para Pedidos. Con todo esto, un índice en ElasticSearch permite a los usuarios consultar y manipular la información de manera efectiva.

5.4.3. Mapping

El mapping es el proceso de definir cómo se indexan y almacenan cada uno de los documentos y sus campos o atributos. En este tipo de esquema, se define el tipo y formato de los campos de los documentos. En la figura 5.2, se muestra el código del procedimiento API para crear el índice para los datos anteriores:

```
PUT /indice_nombre
{"mappings": {
        "properties": {
                "id": {"type":"integer"},
                "nombre": {"type":"text"},
                "edad": {"type":"integer"},
                "genero": {"type":"text"}
              }
        }
}
```

Figura 5.2: Ejemplo de creación de un índice en ElasticSearch

Al igual que en las bases de datos relacionales, es necesario definir el tipo de datos de cada campo. En ElasticSearch, el proceso para definir este tipo de dato se puede observar en el código anterior, donde para cada campo se escribe la secuencia *"type": valor*, donde en el valor se ha de especificar qué tipo de dato es. Los tipos de datos disponibles en ElasticSearch son muy variados[1], pero los más utilizados y comunes son *float*, *integer*, *text*, *date* y *boolean*.

5.4.4. Índice invertido

ElasticSearch utiliza lo que se conoce como un índice invertido, con el que puede manejar grandes cantidades de información y proporcionar resultados de búsqueda rápidos y precisos. Este tipo de índice almacena una asignación de contenido, como palabras o

[1] https://www.elastic.co/guide/en/elasticsearch/reference/current/sql-data-types.html

números, a sus ubicaciones en un documento o conjunto de documentos y está diseñado para permitir búsquedas de texto completo muy rápidas. Esencialmente, es una estructura de datos similar a un *hashmap* que dirige de una palabra a un documento o documentos, ya que el índice invertido crea una lista por cada palabra que aparece en cualquiera de los documentos y, tras esto, registra todos los documentos donde se encuentra esa palabra.

Un ejemplo de cómo indexa se puede observar en la figura 5.3, donde el término *rojo* aparece en el documento 1, por lo que se asigna a ese documento. Esto permite una búsqueda rápida de dónde encontrar términos de búsqueda en un documento determinado.

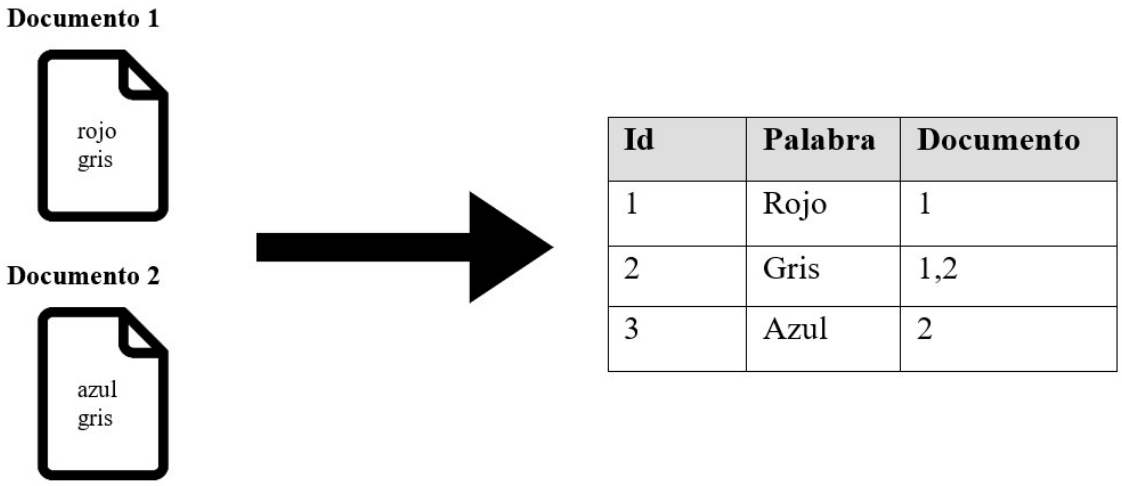


Figura 5.3: Ejemplo de indexación empleado por ElasticSearch

Para aumentar la estabilidad y distribuir la carga de cálculo en proyectos grandes, ElasticSearch utiliza una técnica llamada *sharding*. Mediante esta técnica, varios nodos se unen para formar un clúster y ElasticSearch divide el índice en partes individuales llamados *shards*, las cuales son distribuidas entre los nodos.

ElasticSearch está programado en Java, al igual que Lucene, y su motor de búsqueda proporciona resultados en formato JSON a través de un servicio web REST. La API de ElasticSearch permite la integración de la función de búsqueda en un sitio web.

Además, ElasticSearch ofrece el servicio ElasticStack, que cuenta con las aplicaciones Kibana, Beats y Logstash y que funcionan como herramientas adicionales para analizar la búsqueda de texto.

La contrapartida de ElasticSearch es que no es una herramienta gratuita, o por lo menos sus servicios en nube no lo son. Por otro lado, es cierto que ofrece un servicio gratuito que trabaja en local, lo que es más que suficiente para familiarizarte con esta base de datos. Hay varias razones por las que puede ser beneficioso empezar a trabajar con la versión de ElasticSearch en local antes de implementarla en un entorno de producción. Algunas de las razones incluyen:

- **Pruebas de funcionamiento**. Antes de implementar ElasticSearch en un entorno de producción, es importante asegurarse de que funciona correctamente. Al trabajar con la versión en local, se pueden realizar pruebas de funcionamiento y solucionar problemas sin afectar a los usuarios finales

- **Experimentación y exploración**. Trabajar con ElasticSearch en local permite experimentar y explorar diferentes configuraciones y características sin preocuparse por los impactos en el rendimiento o la disponibilidad de la base de datos

- **Desarrollo iterativo**. Al trabajar con ElasticSearch en local, se puede iterar rápidamente sobre nuevas funcionalidades y características. Esto permite probar nuevas ideas y ajustar el rendimiento antes de implementarlos en un entorno de producción
- **Coste**. Por último, uno de los alicientes más interesantes a la hora de utilizar esta base de datos NoSQL en local antes de pasar a producción es su coste nulo, lo que ayuda a que se pueda experimentar sin tener que preocuparse por los costes asociados a la implementación actual en un entorno de producción. Esto puede ser particularmente útil para proyectos pequeños o para desarrolladores que están aprendiendo sobre ElasticSearch y quieren adquirir experiencia práctica sin tener que incurrir en gastos adicionales

Estas características encajan particularmente con las características del proyecto, puesto que es necesario en los momentos iniciales de éste contar con un entorno de pruebas y testeo sin limitaciones económicas. Al trabajar de esta forma, se puede asegurar de que los datos se indexan y se buscan correctamente antes de implementarla en un entorno de producción. Además, también puedes probar la integración con tu aplicación de *dashboard* y ajustar la configuración de ElasticSearch según sea necesario para obtener un mejor rendimiento por lo que todo puede ser comprobado y validado antes de subir la versión definitiva.

5.4.5. Elección de la base de datos. OpenSearch

OpenSearch es un motor de búsqueda y análisis de datos de código abierto que se basa en el código fuente de ElasticSearch (hasta la versión 7.10 de esta), y que surge como protesta al cambio de licencia que sufrió ElasticSearch en 2021 a una menos amigable para la comunidad de código abierto, debido al aumento de costes por el uso de esta licencia. En respuesta, la comunidad de ElasticSearch bifurcó el código fuente original de ElasticSearch para crear OpenSearch, un proyecto de código abierto independiente.

OpenSearch ofrece muchas de las mismas características que ElasticSearch, como la capacidad de indexar y buscar datos en tiempo real y la capacidad de escalar horizontalmente para manejar grandes volúmenes de datos. Sin embargo, hay algunas diferencias importantes en la arquitectura y la implementación de OpenSearch en comparación con ElasticSearch, ya que se han agregado algunas características nuevas y se ha eliminado el código relacionado con la nueva licencia.

OpenSearch está respaldado por una comunidad activa de desarrolladores y se utiliza en muchos proyectos y aplicaciones de código abierto. Además, trabaja sobre AWS, lo que facilita su escalabilidad.

5.4.6. Elección de OpenSearch sobre ElasticSearch

Para el proyecto realizado, se ha utilizado OpenSearch frente a ElasticSearch por diversos motivos. En primer lugar, en cuanto a costes, OpenSearch te permite utilizar todas sus herramientas sin límite al pagar una tarifa por uso. Pasa lo mismo con ElasticSearch, pero esta última por su parte es algo más cara, lo que hace que aumente el coste del proyecto. Más adelante, en la Sección 8.1, se explicará más en detalle el desglose estimado de costes de cada alternativa. Además, la versión más básica de ElasticSearch de coste, incluye algunas limitaciones que lo pueden hacer menos atractivo. En términos de uso de AWS, OpenSearch es compatible con *Amazon Web Services* (AWS) y se ofrece como un servicio gestionado de AWS, lo que significa que AWS se encarga de la infraestructura

subyacente y la gestión del clúster. Esto puede reducir significativamente la complejidad de la gestión y administración de ElasticSearch en AWS.

Además, OpenSearch también ofrece características que pueden ser atractivas para los usuarios de AWS, como la integración con *Amazon CloudWatch* y *Amazon Kinesis*. Esto permite una mejor monitorización y procesamiento de datos en tiempo real.

Es por todo esto, por lo que se va a trabajar con la herramienta OpenSearch. Cabe recalcar que, a términos de la ejecución del proyecto, no existe ninguna diferencia entre utilizar ElasticSearch u OpenSearch más allá del coste y AWS. Además, como se comentaba anteriormente, OpenSearch es una ramificación de ElasticSearch a partir de la versión 7.10, por lo que guarda toda la semejanza de ElasticSearch hasta esa versión. Con todo esto, OpenSearch no va a limitar ni a poner en peligro la viabilidad de este proyecto.

5.4.7. Diseño y creación de la base de datos

En primer lugar, es necesario crear el entorno virtual de OpenSearch en AWS. Para ello, se accede a la página de AWS, se elige *Amazon OpenSearch Service* en la sección de *Analytics* y luego se selecciona la opción de *Crear dominio*. Tras esto, se proporcionará un nombre para el dominio. En nuestro caso, el dominio se llamará "btrends-elastic", haciendo referencia al nombre que recibe la aplicación.

Se deberá elegir la versión más reciente, establecer la implementación como *Development and testing*, así como configurar los nodos de datos con *t3.small.search* y tres nodos, ya que en los inicios de un proyecto es la opción más interesante y recomendable para ahorrar costes y funcionar perfectamente. Para simplificar, se utilizará un dominio de acceso público y se creará un usuario maestro con nombre de usuario y contraseña. En la política de acceso, se elegirá *Only use fine-grained access control*, ya que esto permite controlar el acceso a nivel de usuario y se utiliza para especificar qué usuarios o roles tienen acceso a qué índices y qué acciones se les permiten realizar en esos índices. De esta manera, se puede configurar y restringir el acceso a los datos de manera más granular. Además, al elegir esta opción, se pueden administrar los permisos de forma más detallada, lo que resulta en una mayor seguridad del dominio.

Después de crear el dominio, tendremos acceso al *Domain endpoint*, que corresponde a la URL del servidor de datos de OpenSearch. Esta URL es necesaria e importante porque se necesita para iniciar la conexión con el servidor y poder realizar tareas como crear índices, insertar datos y realizar búsquedas de esos datos. El formato de este dominio será parecido a la siguiente URL:

https://search-btrends-elastic-XXXXXXX.eu-west-1.es.amazonaws.com

5.4.8. Inserción de los datos

Como se ha comentado en apartados anteriores, el esquema que sigue OpenSearch difiere algo a los esquemas relacionales, debido a que es un tipo de base de datos NoSQL. Un esquema o *mapping* en OpenSearch define las propiedades de cada uno de los campos que va a tener un índice de nuestra base de datos. En el caso de este trabajo, es necesario definir los campos relevantes y que, por ello, vamos a tener en cuenta en la construcción de este *mapping*.

Todo el proceso de creación de los índices, *mappings* y subida de los datos ha sido realizado con código de Python, con el objetivo de facilitar y poder automatizar los procesos.

Cabe recordar que las variables finales de las que partimos se explican en apartados anteriores. Por lo que partiendo de esto, es necesario escoger qué variables o campos son los que interesan y más información pueden aportar, tratando de minimizar la inserción de datos que no aporten información al estudio, con el objetivo de disminuir lo máximo posible el tamaño de la base de datos y optimizar las búsquedas.

Las variables que se utilizarán en el estudio son: *id*, *name*, *country*, *size*, *section*, *year*, y *content*, ya que estas son las necesarias para los análisis realizados y descritos en apartados posteriores. Con lo cual, el *mapping* planteado tendría la siguiente forma:

```
mapping = {
        "properties": {
            "id": {"type": "integer"},
            "name": {"type": "text"},
            "country": {"type": "text"},
            "size": {"type": "text"},
            "section": {"type": "text"},
            "year": {"type": "integer"},
            "content": {"type": "text"}
    }
}
```

Figura 5.4: Definición del *mapping* utilizado para la creación del índice. Contiene todos los campos y su tipo de dato que van a ser utilizados en la base de datos

Una vez definido el *mapping*, es necesario realizar la conexión con la base de datos OpenSearch para poder empezar a crear el índice y subir los datos. Esto es muy fácil de realizar con la librería ElasticSearch disponible en Python, ya que permite con la función ElasticSearch establecer la conexión con el servidor. Para ello, basta con escribir la sentencia de la figura 5.5.

```
es = ElasticSearch("https://search-btrends-elastic-XXXXXXX.eu-west-1.es.
    amazonaws.com")
```

Figura 5.5: Código Python de conexión con la base de datos OpenSearch

Tras haber realizado la conexión con la base de datos, es necesario crear el índice en el que vamos a insertar todos estos datos. Esta misma librería también permite crear los índices sin necesidad de recurrir a las llamadas API. Simplemente como se visualiza en la figura 5.6, mediante la función "indices.create" añadiéndole el nombre que se desea y el *mapping* (por ejemplo, el definido anteriormente).

```
es.indices.create(index=index_name, mappings=mapping)
```

Figura 5.6: Secuencia de creación de un índice en OpenSearch donde se indica el nombre del índice y el esquema que se va a utilizar

En caso de que se quisiera eliminar ese índice (u otro cualquiera) y todo lo contenido en su interior, basta con usar la función "indices.delete'".

El último paso para la inserción de los datos es el de adaptarlos a lo requerido por OpenSearch. Si recordamos lo descrito anteriormente, OpenSearch utiliza el formato de fichero JSON para realizar consultas e insertar datos, por lo que se debe transformar el fichero de datos que se ha obtenido tras su limpieza a formato JSON. Para esto, se ha creado una función en Python que transforma este *dataframe* a JSON.

Una vez que los datos han sido transformados, están listos para ser cargados en el servidor de datos. Con este fin, se ha desarrollado una función específica, ya que aunque la librería ElasticSearch permite la creación de índices y la conexión al servidor utilizando la función "bulk", presenta restricciones en cuanto a la cantidad de datos que se pueden insertar simultáneamente. En nuestro caso, necesitamos introducir 300.000 registros.

Por ello, la función realizada se encarga de cargar los datos en formato JSON en el dominio de OpenSearch utilizando una solicitud de carga masiva (*bulk*). El proceso se realiza mediante la división del archivo JSON en trozos de 2000 líneas, ya que, tras realizar pruebas de inserción, se ha detectado que el servidor no acepta una carga tan grande y simultanea de archivos, debido a que devuelve el error 429 (demasiadas solicitudes). Por lo que tras realizar esas pruebas, se ha comprobado que dividiendo la carga en paquetes de 2000 inserciones no pone en peligro la inserción de estos. Para cada trozo, se realiza una solicitud POST al punto final del dominio especificado. La función utiliza la biblioteca "requests" para realizar la solicitud, y se proporcionan las credenciales de autenticación a través de los parámetros "auth" que contienen el nombre de usuario y la contraseña.

Además, con el objetivo de comprobar que todos los datos se insertan correctamente en la base de datos, la función devuelve una cadena que indica si la carga masiva de los datos se ha completado correctamente o no, basándose en los códigos de estado HTTP devueltos por cada solicitud. Si todos los códigos de estado son 200, la función devuelve "Se han subido correctamente todos los datos", de lo contrario devuelve "Algunos datos no se han subido correctamente", especificando que sentencia de ejecución ha fallado.

CAPÍTULO 6
Diseño y desarrollo de la herramienta de visualización

El objetivo principal es desarrollar una herramienta de visualización de datos que pueda proporcionar a los usuarios una forma interactiva y fácil de analizar el contenido de estos sitios web.

En segundo lugar, es necesario que la herramienta de visualización permita introducir formularios de búsqueda de palabras, con el objetivo de que el usuario introduzca el término o conjunto de términos que desea comprobar y analizar. A partir de aquí, la herramienta ha de ser capaz de conectarse a la base de datos de OpenSearch y descargar los datos necesarios según la búsqueda del usuario. Esto es debido a que es necesario descargar los resultados obtenidos a través de la consulta para así realizar los gráficos deseados.

Por último, la herramienta de visualización de datos debe de ser altamente escalable. A medida que la cantidad de usuarios empezase a crecer y, por consecuencia, el tráfico web aumenta, la herramienta puede adaptarse y manejar grandes volúmenes de solicitudes sin comprometer la velocidad o la calidad del análisis. Con todas estas características, se ha escogido Django[1] como herramienta para desarrollar la aplicación de visualización de los datos.

6.1 Django

Django es un marco web de alto nivel que utiliza Python para crear las páginas web. Esto es un software que facilita la creación de sitios web dinámicos, abstrae problemas comunes del desarrollo web y proporciona atajos para tareas de programación frecuentes.

Resumidamente, un sitio web dinámico es aquel en el que las páginas no son documentos HTML almacenados en algún lugar del sistema de archivos de un servidor. En un sitio web dinámico, cada página es generada por un programa informático, una llamada "aplicación web", que crea el desarrollador web. Una aplicación web puede, por ejemplo, recuperar registros de una base de datos o realizar una acción en función de la entrada del usuario.

Además, Django proporciona un método para asignar URL solicitadas a código que maneja solicitudes. En otras palabras, brinda una forma de designar qué código debe ejecutarse para qué URL. También facilita la visualización, validación y reenvío de formularios HTML (*templates*). Los formularios HTML son la forma principal de obtener

[1] https://www.djangoproject.com/

datos de entrada de los usuarios web. Convierte la entrada enviada por el usuario en estructuras de datos que se pueden manipular de manera conveniente y ayuda a separar el contenido de la presentación a través de un sistema de plantillas (*views*), para que pueda cambiar la apariencia y la sensación de su sitio sin afectar su contenido y viceversa. Se integra convenientemente con capas de almacenamiento, como bases de datos, pero no requiere estrictamente el uso de una base de datos [24].

Por último, y resumiendo el contenido anterior, se pueden destacar los siguientes elementos más característicos de Django:

- **Template**. Para presentar la información en un formato específico, se utilizan los *templates*, que son archivos HTML que permiten definir un formato que se repetirá varias veces en la web con diferentes contenidos. Gracias al uso de parámetros dentro de los *templates*, Django es capaz de completar el archivo con la información correspondiente, lo que facilita la presentación de la información de manera clara y organizada
- **Views**. Las vistas, por su parte, son funciones encargadas de completar la información de los *templates* a través del uso de los modelos. Es decir, las vistas toman la información almacenada en los modelos y la transforman en una estructura de datos que se puede manipular de manera conveniente, para luego ser presentada en el *template* correspondiente.
- **Urls**. Django también proporciona una forma conveniente de asignar URL a diferentes secciones de la aplicación. Esto permite que el código que maneja las solicitudes se ejecute para la URL solicitada correspondiente

La relación que siguen estos elementos puede ser visualizada en la figura 6.1:

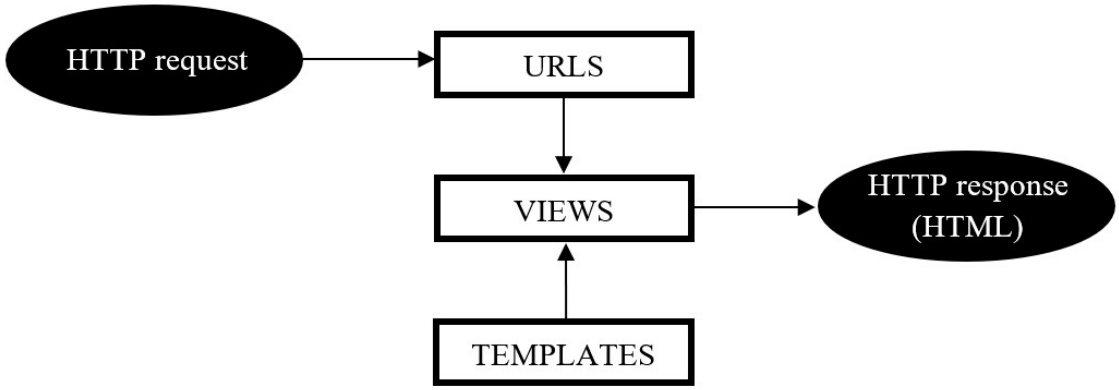


Figura 6.1: Relación de ficheros empleados en los proyectos Django

Esta herramienta no es nada nueva, ya que nació en 2003 y durante su vida ha sido y sigue siendo utilizada por grandes empresas como Instagram, Spotify, Pinterest, Prezi, etc. Django tiene diversas ventajas a tener en cuenta y por lo que es tan utilizada por empresas tan importantes:

- **Simplicidad de uso**. Django permite a los desarrolladores tomar partes del código base y reutilizarlos en otras partes de su programación
- **Facilidad**. Django depende y trabaja sobre Python, un lenguaje de programación conocido por su gran potencial y, sobre todo, facilidad de uso. Esta facilidad de uso se hace patente en factores como su sintaxis clara y legible, ya que la estructura de

bloques de código está delimitada por la indentación en lugar de por llaves, lo que ayuda a que el código sea más legible; una amplia biblioteca, la capacidad de trabajar sobre multiplataformas (puede trabajar con una amplia variedad de sistemas operativos) y, sobre todo, la gran comunidad que tiene este lenguaje, lo que ayuda mucho a la hora de solucionar errores y aprender a utilizar la herramienta

- **Escalable**. Django es compatible con la escalabilidad horizontal, lo que significa que se pueden agregar más servidores para manejar una mayor carga de trabajo. Esto se logra mediante el uso de herramientas como balanceadores de carga y clústeres de servidores. Además, Django proporciona facilidades para trabajar en un servidor local y luego escalar a servidores en la nube, permitiendo a los desarrolladores centrarse en el desarrollo de la aplicación en lugar de preocuparse por la configuración técnica de los servidores. Con su soporte para diferentes servidores web y sistemas de bases de datos, la configuración de variables de entorno y la integración con herramientas de automatización de despliegue, Django es una herramienta potente para el desarrollo de aplicaciones web escalables y robustas

6.2 Diseño de la interfaz de usuario

El diseño de la interfaz del usuario tiene como objetivo presentar diferentes gráficos en un *dashboard* intuitivo y fácil de usar. Este diseño se ha centrado en la simplicidad y la claridad, buscando ofrecer una experiencia de usuario óptima. Para ello, se ha construido una interfaz limpia y minimalista que no distrae al usuario de los datos que se están presentando.

A pesar de que ha sido diseñada con el objetivo de que funcione correctamente en ordenadores y los gráficos se adapten correctamente a los tamaños de los monitores, la aplicación web está también pensada para que funcione correctamente desde dispositivos móviles. Se han utilizado técnicas y herramientas específicas para optimizar la interfaz de usuario en dispositivos móviles, como el diseño adaptable para que los gráficos y la información se adapten automáticamente al tamaño de la pantalla del dispositivo.

Se ha puesto especial atención en garantizar que la navegación por la aplicación sea intuitiva y sencilla en dispositivos móviles, utilizando botones grandes y claros y evitando elementos que puedan dificultar la interacción en pantallas pequeñas.

Además, se han realizado pruebas en diferentes dispositivos móviles para asegurarse de que la aplicación funcione correctamente en todo momento, independientemente del tipo de dispositivo o sistema operativo que utilice el usuario. Todo esto asegura que la aplicación web ofrezca una experiencia de usuario óptima y eficiente, tanto en ordenadores como en dispositivos móviles.

El diseño de la aplicación web es, como se ha mencionado anteriormente, sencillo e intuitivo. Esta pantalla se divide en 3 zonas o bloques, los cuales se pueden visualizar en la figura 6.2 y se detallan a continuación:

6.2.1. Barra lateral

La barra lateral de la aplicación, que se encuentra en el lado izquierdo de la pantalla, está diseñada de manera vertical, y puede o no estar fija en su posición, según la preferencia del usuario. El objetivo de esta barra es proporcionar acceso rápido y sencillo a las diferentes secciones y funciones disponibles en la aplicación.

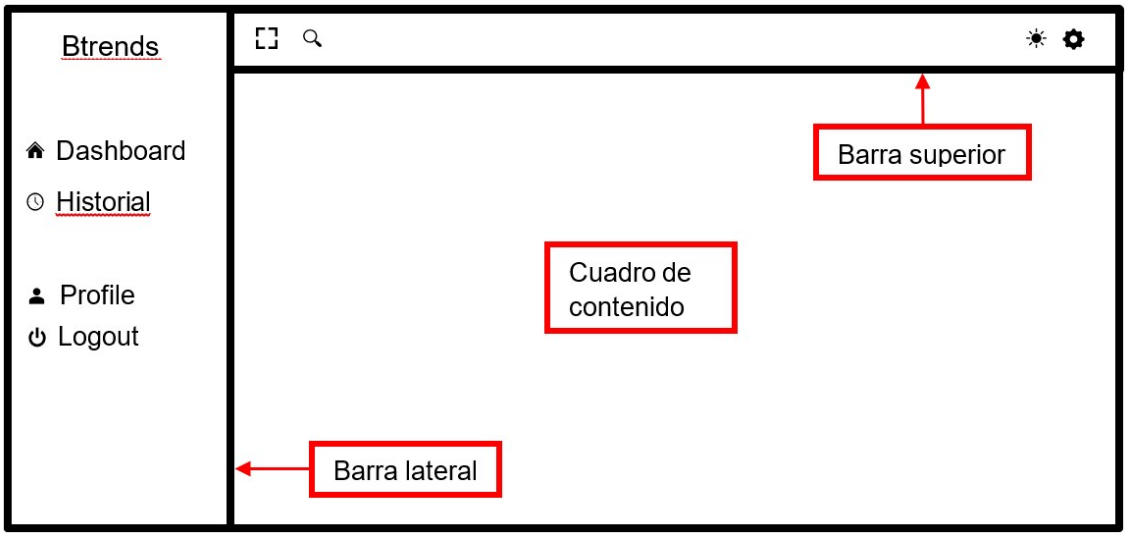


Figura 6.2: Boceto del diseño de la pantalla principal de la aplicación

Para permitir una mayor flexibilidad en la visualización de la aplicación, se ha incluido un botón que permite desactivar la fijación de la barra lateral y reducir su tamaño, viéndose solo los iconos de cada sección y el logo de la aplicación. De esta forma, los usuarios pueden prácticamente ocultar la barra lateral para aumentar el tamaño de las otras zonas de la aplicación. Esto les permitirá tener una vista más amplia y detallada de la información mostrada en la pantalla.

Para que el usuario pueda acceder a la barra lateral de nuevo, se ha incluido una funcionalidad que muestra la barra al posicionar el cursor del ratón sobre el lado izquierdo de la pantalla. Esto permite al usuario interactuar con la barra lateral sin tener que activarla de forma permanente, lo que podría resultar molesto o distraer la atención del usuario de la información que se está mostrando en la pantalla.

Asimismo, el botón que permite desactivar la fijación de la barra lateral también tiene la funcionalidad de volver a activarla, para que el usuario pueda volver a tenerla fija en su posición. Esta funcionalidad se activa pulsando nuevamente el botón.

Al lado de este botón, se encuentra el logo y el nombre de la aplicación, que funciona como un botón para regresar a la pantalla principal. Al hacer clic sobre este, los usuarios pueden acceder fácilmente a la página principal de la aplicación, donde pueden ver todas las opciones disponibles y navegar a través de ellas.

Este botón del logo y nombre de la aplicación es muy útil para aquellos usuarios que pueden haberse perdido en la navegación o en el uso de una función específica y necesitan regresar a la pantalla principal. Además, también puede resultar útil para aquellos que necesitan cambiar de sección o buscar una nueva opción para usar en la aplicación.

El propósito fundamental de la barra lateral es la organización y separación de las diversas secciones dentro de la aplicación, lo que permite una navegación intuitiva para el usuario. En total, existen cuatro secciones distintas divididas en dos grupos: visualización y perfil. No obstante, el último grupo de secciones solo estará disponibles para los usuarios que hayan iniciado sesión en la plataforma. En caso contrario, únicamente se podrá acceder a la sección de visualización, específicamente al panel de control (*dashboard*).

La primera de las secciones es precisamente la descrita anteriormente: el panel de control o *dashboard*. Esta es, a su vez, la pantalla de inicio o pantalla principal de la aplicación, y su función es la de visualizar los gráficos planteados dependiendo de la búsqueda

del usuario. Esta búsqueda sobre uno o varios términos también se realiza desde esta pantalla.

La siguiente sección en caso de no estar autenticado es la de "Login", donde al pulsar sobre esta se te abre otra pestaña diferente destinada al inicio de sesión. En esta pantalla, se muestra una ventana central donde hay situadas celdas de texto para introducir el usuario y la contraseña. En caso de no tener usuario, se puede crear una cuenta pulsando sobre el botón de "Sign Up". Este botón de registro lo que te permite es crear una cuenta, introduciendo datos como el nombre de usuario, el correo electrónico y la contraseña. En caso de que el usuario se cree una cuenta o ya la tenga creada, introduciendo los datos en la ventana anterior de Inicio de sesión, podrá iniciar la sesión y desbloquear las demás secciones reservadas para los usuarios autenticados.

Cabe recalcar que las demás secciones no son necesarias para el correcto funcionamiento de la aplicación y que es simplemente un valor añadido a esta. El objetivo principal de la app queda cubierto aún sin haber creado una cuenta.

Una vez que el usuario ha iniciado sesión, se mostrarán las otras tres secciones disponibles en la barra lateral. La primera de ellas es el historial, que permitirá al usuario acceder a su historial completo de búsquedas. Esta sección mostrará información detallada sobre cada búsqueda, incluyendo la fecha en que se realizó de forma clara y accesible. La segunda sección disponible para usuarios autenticados es el perfil, donde se puede acceder a la información del perfil del usuario. Esta sección permitirá al usuario ver y editar su información personal. La última sección de la barra lateral corresponde al cierre de sesión o "Logout". Al seleccionar esta opción, el usuario cerrará su sesión y pasará a no estar autenticado.

6.2.2. Barra superior

Esta barra se encuentra en la parte superior de la pantalla orientada horizontalmente, es fija y prácticamente no cambia en función de las acciones que se hagan por pantalla. Cuenta con un total de 4 botones con diferentes funciones orientadas a la personalización del espacio o la búsqueda de directorios. Estos botones están diseñados para ser intuitivos y fáciles de usar, lo que permite a los usuarios interactuar con la aplicación de manera eficiente y sin esfuerzo.

El primero de los botones sirve para activar y desactivar el modo de pantalla completa. Este modo es muy útil para aquellos usuarios que desean ver una página web en su totalidad, sin la presencia de la barra de herramientas del navegador, ni las pestañas que se encuentran en la parte superior de la pantalla. Asimismo, también se elimina la barra de herramientas de Windows que se sitúa en la parte inferior de la pantalla. De esta manera, los usuarios pueden disfrutar de una visualización más amplia y sin distracciones de la página web que estén visitando.

El segundo botón tiene forma de lupa, y sirve para buscar o filtrar sobre las diferentes secciones disponibles en la barra lateral, haciendo más sencilla la navegación entre funciones o secciones de la aplicación. Al presionar sobre este se expande el botón, abriéndose una celda de texto en la que poder escribir la sección que se desea buscar. Esta funcionalidad es muy útil para aquellos usuarios que necesitan acceder a una sección específica de la aplicación, ya que pueden hacerlo rápidamente a través de la barra de búsqueda. Por ejemplo, si el usuario acaba de abrir la aplicación y desea iniciar sesión, puede buscar en el botón de la lupa la palabra "Login" para filtrar y que solo aparezca esta sección en la barra lateral.

El tercer botón disponible sirve para activar o desactivar el modo oscuro y el icono puede ser un sol o una luna, en caso de querer desactivar o activar, respectivamente, el

modo oscuro. Esta opción permite cambiar el esquema de colores de la página web para que el fondo sea oscuro y el texto y elementos visuales sean claros. La principal ventaja de esta funcionalidad es que reduce la fatiga visual y el cansancio ocular, especialmente en entornos con poca iluminación. Todo esto hace que la funcionalidad sea muy importante, ya que permite a los usuarios personalizar la apariencia de la aplicación y ajustarla a sus preferencias de visualización, lo que a su vez aumenta la comodidad y la satisfacción del usuario.

El cuarto y último botón puede estar o no activado, dependiendo de si el usuario ha iniciado sesión o no lo ha hecho. Este corresponde a los ajustes y tiene forma de engranaje. Pulsando sobre este botón se despliega un menú que permite acceder al perfil o cerrar la sesión, por lo que para que aparezca este botón sobre la barra, el usuario debe haber iniciado sesión.

En resumen, la barra superior de la aplicación web está diseñada para ofrecer una experiencia de usuario intuitiva y personalizada. Los cuatro botones disponibles ofrecen diferentes opciones para adaptar la interfaz según las preferencias del usuario, permitiendo una navegación más fácil y cómoda. Además, la posibilidad de usar la aplicación en modo pantalla completa, el modo oscuro y los ajustes de la cuenta, ofrecen una mayor personalización de la experiencia de usuario y mayor control para el usuario sobre su cuenta. Todo esto hace que la aplicación sea más atractiva y fácil de utilizar para los usuarios.

6.2.3. Cuadro de contenido

El último bloque por describir es el del contenido de cada una de las pestañas. Este está relacionado con el descrito antes, puesto que generalmente el contenido variará según la selección que se realice en las pestañas de la barra lateral.

La pantalla de inicio de la aplicación constará de un buscador para que el usuario pueda introducir los términos que desea buscar, así como un mensaje explicativo debajo de la barra de búsqueda. Esta pantalla podrá ser accesible directamente al abrir la aplicación o pulsando sobre la sección *dashboard* en la barra lateral.

Si el usuario ha iniciado sesión, se mostrará el siguiente mensaje en la pantalla de inicio: "Para comenzar la búsqueda, escriba una palabra o conjunto de palabras en la barra superior. Si lo desea, también puede acceder a su historial de búsqueda." Al hacer clic en la palabra *historial*, el usuario podrá acceder a su propio historial de búsqueda. Si el usuario prefiere no acceder al historial, simplemente puede introducir los términos deseados y pulsar el botón azul de búsqueda o presionar la tecla *Enter*.

En el caso de que el usuario no haya iniciado sesión en la aplicación, se mostrará un mensaje distinto en la pantalla de inicio. Este mensaje indicará lo siguiente: "Para comenzar la búsqueda, escriba una palabra o conjunto de palabras en la barra superior. Si desea guardar su historial de búsqueda, por favor, inicie sesión o regístrese".

Al hacer clic en las palabras *iniciar sesión* o *regístrese*, el usuario podrá acceder a la pantalla correspondiente para realizar dicha acción. Sin embargo, si el usuario desea seguir utilizando el buscador sin iniciar sesión, podrá hacerlo sin ningún problema. Cabe destacar que en este caso el historial de búsqueda no será guardado.

Una vez que el usuario realiza la búsqueda, se mostrarán en pantalla los resultados de los gráficos que contienen los datos de la palabra o conjunto de palabras buscados. En total, la aplicación mostrará 13 gráficos, los cuales se dividen en cuatro secciones diferentes donde se presentan gráficos que muestran la evolución de los datos en el tiempo y la representación geográfica de estos. Estas secciones hacen referencia a los tipos de gráfi-

cos contenidos en cada una de ellas y son las siguientes: *gráficos de líneas*, *gráficos de barras*, *gráficos treemaps* y *gráfico de mapa*. Los gráficos han sido diseñados con la librería Plotly[2], ya que es una herramienta muy potente de visualización que permite al usuario interaccionar con los gráficos, realizando acciones como filtrar, seleccionar áreas del gráfico para poder ampliarlas o incluso descargarse las visualizaciones, entre otros.

La siguiente sección es la del historial de búsquedas, en la que se incluirá una tabla sencilla donde se mostrarán los términos buscados por el usuario, así como la fecha de la consulta. Al hacer clic en cualquier lugar de la fila correspondiente a una palabra, el usuario será redirigido a la sección de *dashboard*, donde se buscará automáticamente el término seleccionado. Esto se realiza para facilitar el acceso a las búsquedas anteriores y mejorar la experiencia del usuario en la aplicación.

La sección siguiente en la aplicación es la del perfil del usuario. Al hacer clic en ella, se abrirá la página correspondiente donde se mostrará la información del perfil. Esta sección incluirá la foto de perfil del usuario, su nombre, nombre de usuario y correo electrónico.

Además, se proporcionará un formulario que permitirá al usuario editar los datos personales relacionados con su perfil, como su dirección de correo electrónico, su contraseña, su foto de perfil, y cualquier otra información relevante. Esta funcionalidad permite a los usuarios mantener actualizada su información personal y personalizar su experiencia en la aplicación. Es importante destacar que el formulario de edición de perfil solo estará disponible para los usuarios autenticados, ya que se requiere acceso a su cuenta para hacer cambios en su información personal. Si un usuario no ha iniciado sesión, se le redirigirá a la página de inicio de sesión para que pueda hacerlo antes de acceder a esta sección de la aplicación. La sección del perfil también incluirá opciones para que el usuario pueda cerrar sesión. Al hacer clic en la opción de cerrar sesión, se finalizará la sesión del usuario y se le redirigirá a la página de inicio de sesión.

La última sección de la barra lateral corresponde al cierre de sesión o "Logout". Al seleccionar esta opción, el usuario cerrará su sesión y volverá a la pantalla principal, donde solo tendrá acceso al panel de control (*dashboard*). Las secciones reservadas para usuarios autenticados estarán bloqueadas y no estarán disponibles hasta que se inicie sesión nuevamente.

6.3 Funcionamiento de la interfaz

Como se comenta en apartados anteriores, Django permite trabar en un servidor local y luego poder escalar esto a servidores en la nube. Por este motivo, y para facilitar la etapa de pruebas y desarrollo, se ha diseñado toda la interfaz y su funcionamiento en local.

Django es un *framework* de desarrollo web que utiliza una estructura de archivos y carpetas bien definida para organizar una aplicación web. Al desarrollar una aplicación en Django, se debe crear un paquete de Python que sigue ciertas convenciones, incluyendo la creación de archivos como *views.py*, *urls.py* y *forms.py*. La creación de estas funciones y archivos puede ser un proceso repetitivo y tedioso, pero Django ofrece una solución para simplificar este proceso. Django cuenta con una herramienta que genera automáticamente la estructura básica de directorios y archivos de una aplicación, permitiendo centrarse en escribir el código de la aplicación en lugar de crear directorios.

En el caso de esta aplicación, se ha organizado de tal forma que se tienen dos secciones principales: *authentication* y *home*. Cada sección está contenida en un directorio separado

[2] https://plotly.com/python/

y contiene archivos específicos que manejan la lógica relacionada con su función correspondiente.

En la sección *home*, se encuentran los archivos:

- **urls.py**. Este archivo contiene la definición de las URL asociadas con la página de inicio
- **views.py**. Este archivo contiene la lógica que maneja las solicitudes HTTP relacionadas con la página de inicio

Mientras que en la sección *authentication*, se encuentran los archivos:

- **forms.py**. Este archivo contiene definiciones de formularios personalizados utilizados en la autenticación de usuarios
- **urls.py**. Este archivo contiene la definición de las URL asociadas con la autenticación de usuarios
- **views.py**. Este archivo contiene la lógica que maneja las solicitudes HTTP relacionadas con la autenticación de usuarios

La estructura de la aplicación Django sigue las convenciones estándar de Django, lo que facilita su organización y mantenimiento. Cada archivo tiene un propósito específico y está ubicado en el lugar adecuado para que sea fácil de encontrar y modificar.

A continuación, se detalla más específicamente que hace cada archivo creado en la sección home.

6.3.1. Archivo views.py. Home

El archivo *views.py* es aquel donde se definen las funciones que manejan las solicitudes HTTP entrantes. Cada función en este archivo se llama una vista y está diseñada para manejar una solicitud específica. Cuando un usuario hace una solicitud HTTP, Django busca la vista correspondiente en este archivo y la ejecuta.

Las vistas pueden aceptar argumentos opcionales, como el identificador de un objeto de base de datos o un parámetro de búsqueda, que se pasan en la URL de la solicitud. Las vistas pueden realizar diversas tareas, como recuperar datos de la base de datos, procesar formularios y renderizar plantillas HTML.

Además de procesar las solicitudes HTTP, las vistas también permiten crear funciones auxiliares que respaldan a las funciones HTTP. Estas funciones auxiliares pueden ser utilizadas para llevar a cabo operaciones complejas, como el procesamiento de datos o la validación de formularios.

En el archivo de vistas diseñado, se han creado funciones que permiten generar las visualizaciones planteadas. Para ello, se han creado diferentes funciones auxiliares que ayudan a realizarlo. La mayoría de las funciones creadas utilizan el objeto *request*. En Django, el objeto *request* es un argumento que reciben todas las funciones de las vistas. Este objeto representa la solicitud HTTP enviada por el cliente y contiene información como los datos del formulario, la dirección IP del cliente, la sesión actual, entre otros datos. Este objeto es creado automáticamente por el servidor web de Django cada vez que un cliente envía una solicitud HTTP a la aplicación y es enviado como argumento a la función de la vista correspondiente que se ha especificado en la URL.

En cuanto a las funciones, en primer lugar, se ha creado una función llamada "query-to-df", que transforma las consultas escritas por los usuarios en *dataframes* necesarios para realizar las gráficas. Esta función recibe el objeto request y el término buscado por el usuario y realiza una consulta al servidor de OpenSearch.

Para esto, se utiliza la librería *requests* de Python. La biblioteca "requests" es una herramienta muy popular en Python para hacer solicitudes HTTP y que, gracias a su API sencilla y fácil de usar, te permite interactuar con servicios y consumir datos en una aplicación sin tenerse que preocupar por los detalles técnicos de las solicitudes HTTP.

Para hacer la consulta con la librería requests, es necesario utilizar la función *get*. Con esta función se envía una solicitud GET a la URL especificada y se recuperan los datos de un recurso especificado. La función acepta varios parámetros diferentes que pueden ser usados para personalizar la solicitud. Para realizar esta consulta al servidor OpenSearch, se necesita especificar varios de estos parámetros que ofrece la función *get*:

- **url**. Este parámetro se utiliza para proporcionar la URL del recurso al que se desea enviar la solicitud. En este sentido, la dirección URL que se especifica es la relativa al dominio generado en AWS
- **auth**. El parámetro "auth" es generalmente utilizado para indicar las credenciales de autenticación para la solicitud. Estas credenciales son creadas por el administrador del servidor a la hora de generarlo en AWS y constan de un usuario y una contraseña
- **data**. Este último parámetro se utiliza para enviar los datos de la solicitud al servidor. Estos datos enviados corresponden a la consulta del usuario y son convertidos a formato JSON, ya que así es como trabaja la API de OpenSearch

El funcionamiento de las búsquedas en OpenSearch tienen limitaciones en cuanto al tamaño del espacio de búsqueda, ya que de forma predeterminada solo va a mostrar los 10 primeros registros que encuentre en la base de datos. Es decir, si el usuario busca el término *sostenibilidad* mediante la consulta de la figura 6.3, solo obtendrá los primeros 10 resultados.

```
query = {
    'query': {
     'match': {
         'content': 'sostenibilidad'
               }
         }
     }
```

Figura 6.3: Secuencia para las consultas en formato JSON sobre la API de OpenSearch

Es cierto que el límite puede ser aumentado a 10000, que es lo máximo que te permite OpenSearch. Sin embargo, esto sigue sin ser suficiente, ya que un término puede aparecer más de esa cantidad de veces. Por ejemplo, el término *cookies* aparece cerca de 50000 veces en la base de datos, por lo que no bastaría con ese límite.

No obstante, OpenSearch ofrece una solución a este inconveniente gracias a la paginación de resultados o *search after*. Este parámetro se usa para recuperar la siguiente página de resultados utilizando un conjunto de valores de ordenamiento realizando múltiples solicitudes de búsqueda, siendo siempre necesario que se trate de la misma consulta. Cada uno de los resultados devueltos contiene el parámetro *sort* (parámetro de ordenamiento), mediante el cual se puede recuperar la siguiente página de resultados, ya que

indica cuál ha sido el último registro que contenía esa palabra dentro de la página buscada. Es decir, si la última palabra de la primera página aparece en la observación 25000, la siguiente búsqueda (o lo que es lo mismo, la siguiente página) se tendrá que empezar a partir de la observación 25001. Se puede asemejar a un libro, donde la primera página está formada por 10000 observaciones que contienen la palabra buscada y cada una de las siguientes páginas pueden contener hasta 10000 observaciones más.

Sin embargo, no es posible por el momento en la API de OpenSearch obtener todos estos resultados mediante una sola solicitud, por lo que se ha generado un código Python que se encarga de generar tantas solicitudes como páginas tenga la búsqueda. Para ello, el código se encarga de enviar una primera solicitud que busca en la primera página de resultados y obtiene que valor ocupa el último registro de esa palabra en la base de datos. Con este dato recibido, se empieza a utilizar la paginación para obtener cada una de las siguientes páginas del espacio de búsqueda hasta que llega al final de este, por lo que la función hará tantas consultas al servidor como páginas de resultados existan.

Mediante estas consultas al servidor y una limpieza de los resultados obtenidos en formato JSON, se insertan los datos en un *dataframe*. Esto es lo que devuelve la función. Además, la función "query-to-df" también se encarga de guardar la búsqueda en el historial, utilizando la función auxiliar historial. Esta información solo se guardará en caso de que el usuario esté autenticado, y puede comprobarse gracias a la función "request.user.is-authenticated" propia de Django, que devuelve True si el usuario está autenticado y False en caso de que no lo esté. Volviendo a la función historial mencionada anteriormente, cabe destacar que ha sido desarrollada como parte de la aplicación Django. Esta función se encarga de registrar y almacenar la información de búsqueda del usuario en la base de datos de la aplicación. La función toma dos parámetros: *request*, que contiene información sobre la solicitud web del usuario, y *word*, que representa la palabra que el usuario ha buscado.

La función realiza una consulta a la tabla historial de la base de datos para comprobar si el usuario ha buscado previamente la palabra en cuestión. Si la palabra ya ha sido buscada, la función no realiza ninguna acción adicional. De lo contrario, la función agrega una nueva fila a la tabla historial con la información de búsqueda del usuario, incluyendo la palabra y la fecha de la búsqueda. Todo esto permite al usuario consultar su historial de búsquedas anteriores tal y como se ha explicado en secciones anteriores.

Otra función importante que se ha implementado en la aplicación Django es "extract-historial". Esta función tiene como objetivo recuperar la información de búsqueda previa del usuario desde la base de datos y devolverla en forma de lista para su posterior procesamiento y presentación en la interfaz de usuario. La función toma un parámetro *request*, que representa la solicitud web del usuario y con el que se podrá identificar el usuario, garantizando que solo se devuelvan los datos relevantes para el usuario que está realizando la solicitud.

Por otro lado, existen dos funciones que sirven como vistas en la aplicación y que son muy importantes en una aplicación Django. Son responsables de recibir las solicitudes del usuario, procesarlas y devolver una respuesta en función de las acciones solicitadas por el usuario. Cada función de vista es una función Python que toma un objeto de solicitud (*request*) como argumento y devuelve un objeto de respuesta (*response*). Las vistas determinan qué datos mostrar al usuario, cómo mostrarlos y cómo procesar las solicitudes del usuario.

La primera de las funciones de vista es la función *chart*, y su propósito es renderizar una página web que muestra gráficos relacionados con una palabra seleccionada por el usuario. En primer lugar, se realiza una consulta al servidor OpenSearch mediante la función "query-to-df" (comentada anteriormente) para obtener los datos necesarios.

Posteriormente, la función utiliza el *dataframe* obtenido para crear los diferentes tipos de gráficos que se muestran en la página web. Finalmente, se renderiza una plantilla HTML con los gráficos y los datos pertinentes.

Estos gráficos muestran una media ponderada de los datos obtenidos, ya que se ha considerado que cada combinación de variables tiene su propio peso (siendo las variables combinables país, sector y tamaño). Para el cálculo de esta media ponderada, lo que se ha realizado es crear una función llamada "suma-ponderada" que toma el valor de cada dato y lo multiplica por su peso correspondiente. Luego, se suman todos los productos obtenidos para las diferentes combinaciones de tamaño y sección.

La obtención de esos pesos es sencilla cuando solo se agrega por una variable, como en el caso de agregar solo por país. Sin embargo, si se desea agregar por más variables, es necesario combinar los pesos, como en el caso de agregar por tamaño y sección a la vez, donde se utiliza la función "calcula-peso". Esta función toma los vectores de pesos correspondientes a cada variable y los combina de manera adecuada para obtener el peso que corresponde a cada combinación de tamaño y sección en cada país. Por ejemplo, si tenemos un vector de pesos para el tamaño y otro vector de pesos para la sección, la función "calcula-peso" puede realizar una multiplicación elemento por elemento entre los dos vectores para obtener un nuevo vector de pesos que refleje la combinación de ambas variables. Una vez que se han calculado los pesos para cada combinación de tamaño y sección en cada país, se procede a la suma ponderada. Para ello, se multiplica cada dato por su peso correspondiente y se suman los productos resultantes. Este enfoque garantiza que la media ponderada refleje la importancia relativa de cada combinación de tamaño y sección en cada país, teniendo en cuenta los pesos asignados a cada variable.

Por último, la función "historial-view" se trata de una vista que se encarga de mostrar al usuario un historial de sus búsquedas previas. Esta vista utiliza la función "extract-historial" para extraer de la base de datos todas las palabras que el usuario ha buscado anteriormente. Luego, se crea un contexto que contiene la lista de palabras buscadas y se renderiza una plantilla HTML que muestra el historial en forma de lista. Una vez renderizada, se devuelve al usuario la página web con el historial de búsquedas.

6.3.2. Archivo urls.py. Home

El archivo *urls.py* contiene una lista de objetos de ruta URL que se utilizan para enrutar las solicitudes entrantes a las vistas correspondientes. Cada objeto de ruta URL consta de dos partes principales: el patrón de URL y el nombre de la vista que maneja esa URL. Al recibir una solicitud HTTP, Django busca el patrón de URL correspondiente en este archivo y redirige la solicitud a la vista apropiada.

Además de los patrones de URL estándar, que son simplemente cadenas de texto, las rutas URL también pueden contener variables y expresiones regulares para validar y filtrar las solicitudes entrantes. Por ejemplo, en el archivo *urls.py* se puede definir un patrón de URL que contenga una variable que represente el ID de un objeto en la base de datos. Al recibir una solicitud HTTP que coincida con ese patrón, Django extraerá automáticamente el valor de la variable y lo pasará como argumento a la vista correspondiente.

En este archivo en particular, se definen tres patrones de URL: uno para la vista de gráficos (*chart*), otro para la página de inicio (*home*) y otro para la vista de historial (historial). El primer patrón utiliza una cadena de texto vacía como patrón, lo que significa que se aplicará a la ruta raíz de la aplicación. Los otros dos patrones utilizan cadenas de texto que corresponden a los nombres de las vistas que manejan esas rutas. Además, el archivo *urls.py* también define una ruta URL genérica que se utilizará para cualquier

archivo HTML que se solicite y que no coincida con ninguna de las rutas URL definidas previamente.

En resumen, el archivo *urls.py* es fundamental para el funcionamiento de una aplicación web Django, ya que define las rutas URL y las vistas correspondientes que manejan las solicitudes entrantes. La estructura de cada objeto de ruta URL consta de dos partes principales: el patrón de URL y el nombre de la vista correspondiente. Las rutas URL también pueden contener variables y expresiones regulares para validar y filtrar las solicitudes entrantes.

6.3.3. Archivo forms.py. Authentication

El archivo *forms.py* es donde se definen los formularios web que se utilizarán en la aplicación. Un formulario es una representación de una página web que permite al usuario enviar datos al servidor. Los formularios se utilizan para validar y procesar los datos que se envían a través de una solicitud HTTP.

Los formularios pueden contener campos obligatorios, campos opcionales, campos de selección y campos de carga de archivos. Los formularios también pueden contener validadores personalizados para garantizar que los datos enviados cumplan con ciertos criterios.

En el caso de la aplicación, el archivo *forms.py* se encarga de definir formularios personalizados para la autenticación y registro de usuarios. Dentro de este archivo, se han definido dos clases: *LoginForm* y *SignUpForm*.

La clase *LoginForm* define un formulario personalizado para iniciar sesión en la aplicación, el cual cuenta con dos campos de tipo *CharField*: "username" y "password". La clase *SignUpForm*, por su parte, es una extensión de la clase *UserCreationForm* que permite que los usuarios se registren en la aplicación. Este formulario cuenta con tres campos adicionales: "email", "password1" y "password2". La clase *SignUpForm* tiene el modelo *User* de Django en su atributo *Meta*, lo que indica que este formulario está vinculado a ese modelo.

En resumen, el archivo *forms.py* proporciona una definición personalizada de formularios para la autenticación y registro de usuarios en tu aplicación Django.

6.3.4. Archivo urls.py. Authentication

La vista de *urls.py* en la sección de autenticación de la aplicación Django define una serie de rutas para las páginas relacionadas con la autenticación de usuarios. En particular, se han definido rutas para las páginas de inicio de sesión, registro y cierre de sesión, así como para la autenticación social utilizando la biblioteca de terceros "allauth".

Cada ruta está vinculada a una función de vista que se encuentra en el archivo *views.py* correspondiente a la sección de autenticación. Por ejemplo, la ruta "login/" está vinculada a la función "login-view" en views.py, mientras que la ruta "register/" está vinculada a la función "register-user".

Además, la ruta "logout/" está vinculada a una vista de Django predefinida llamada "LogoutView", que se encarga de cerrar la sesión del usuario actual.

6.3.5. Archivo views.py. Authentication

Este código es el controlador de la sección de autenticación de la aplicación Django y maneja las solicitudes HTTP para las páginas de inicio de sesión y registro. En él, se

han creado dos funciones que manejan las solicitudes de las páginas de inicio de sesión y registro respectivamente.

La primera función, llamada “login-view()”, renderiza la página de inicio de sesión y procesa la solicitud del usuario para iniciar sesión. Recibe una solicitud HTTP y devuelve una respuesta HTTP con el formulario de inicio de sesión y un mensaje de error si se proporcionan credenciales inválidas.

La segunda función, llamada “register-user()”, se encarga de la página de registro y procesa la solicitud del usuario para crear una cuenta en la aplicación. Recibe una solicitud HTTP y devuelve una respuesta HTTP con el formulario de registro y un mensaje de éxito si se crea la cuenta con éxito. Además, también valida los datos enviados por el usuario y muestra un mensaje de error si los datos son incorrectos o incompletos.

Ambas funciones utilizan los formularios personalizados definidos en el archivo *forms.py* y la autenticación proporcionada por el sistema de autenticación de Django. Además, se aseguran de que las solicitudes sean válidas y que los datos proporcionados sean correctos antes de procesarlos.

6.4 Producto final

A continuación, se va a adjuntar una muestra de capturas de la versión final de la herramienta de visualización para que se observen las funcionalidades descritas anteriormente. En el Apéndice B se adjuntan las demás capturas, donde se incluyen imágenes del funcionamiento de la herramienta en dispositivos móviles y otras funcionalidades como el inicio de sesión, ajustes, perfil y el modo oscuro, entre otros. Cabe destacar y, como se ha mencionado anteriormente, que los gráficos visualizados son interactivos, por lo que el usuario puede hacer zoom o desagregar la información.

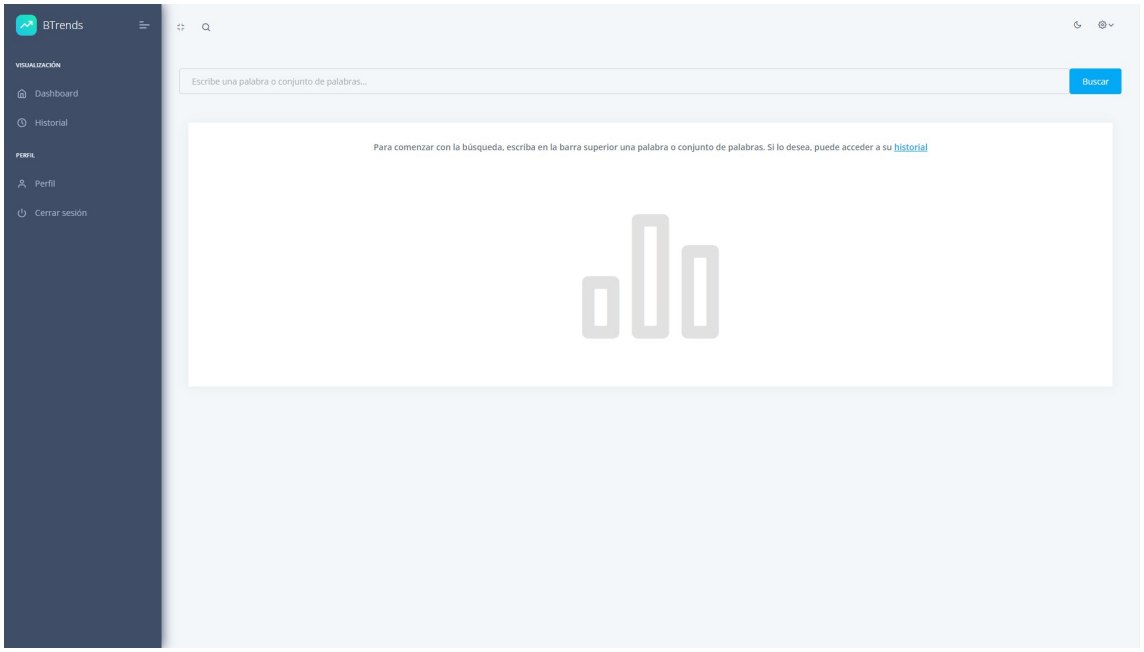


Figura 6.4: Pantalla principal de la herramienta de visualización

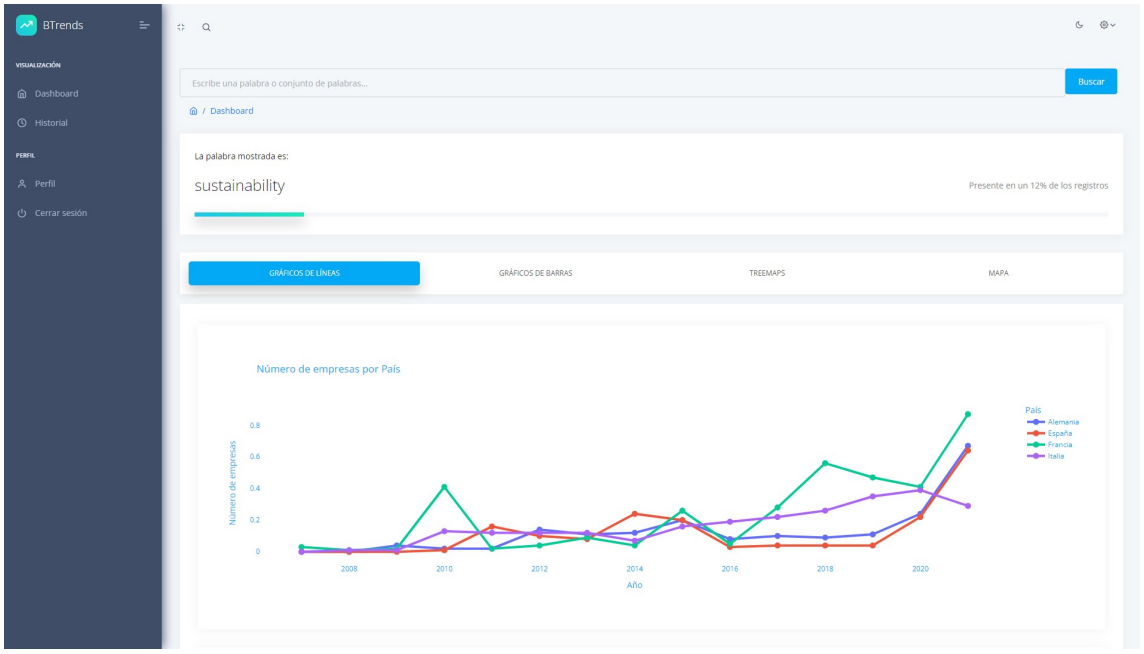


Figura 6.5: Pantalla de visualización de los gráficos de líneas

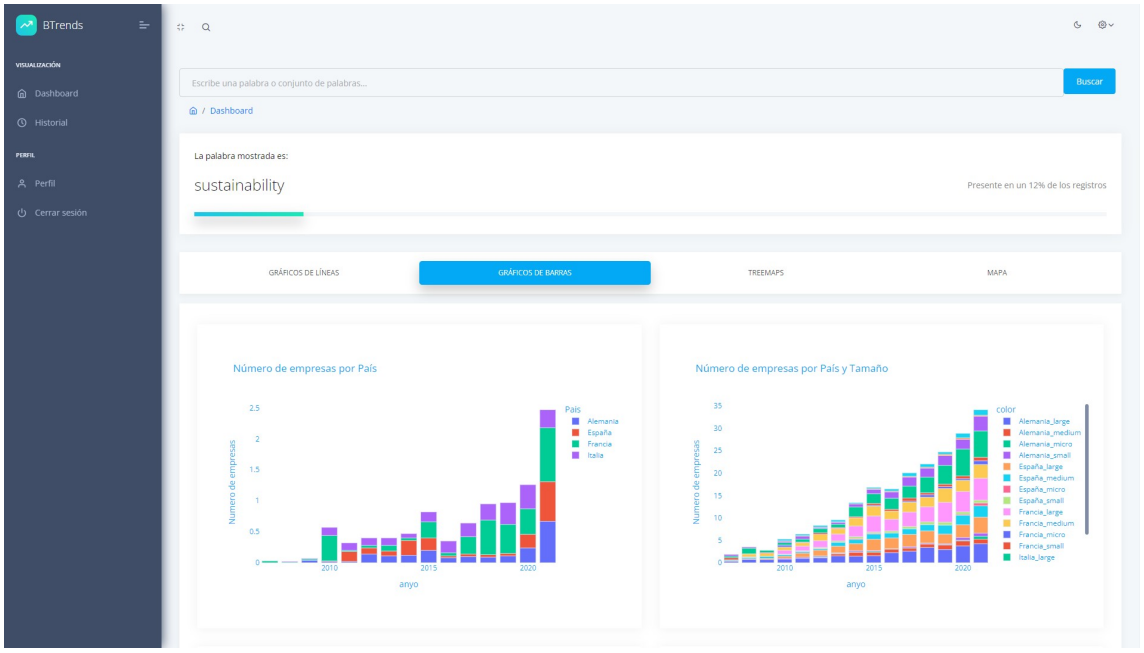


Figura 6.6: Pantalla de visualización de los gráficos de barras

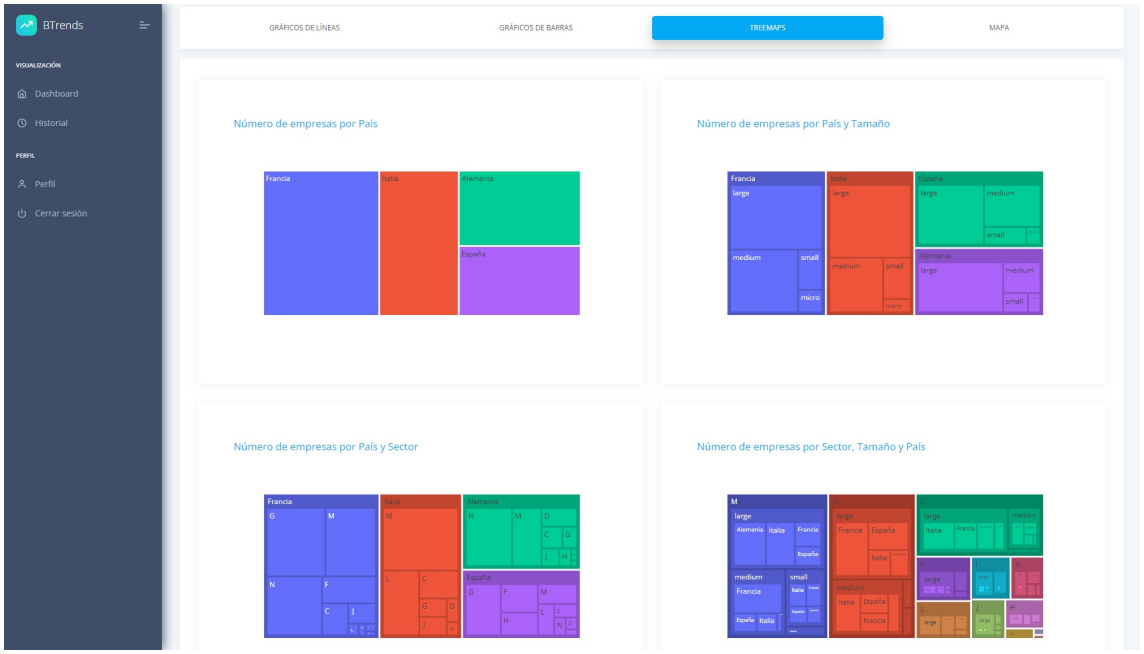


Figura 6.7: Pantalla de visualización de los gráficos treemap

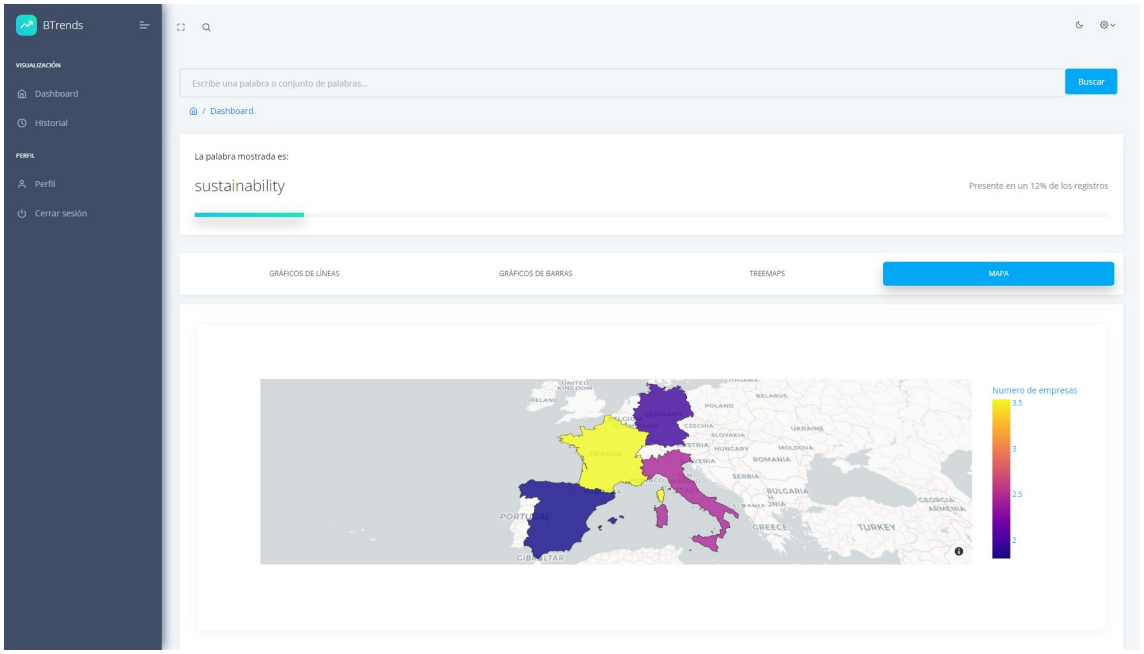


Figura 6.8: Pantalla de visualización del mapa

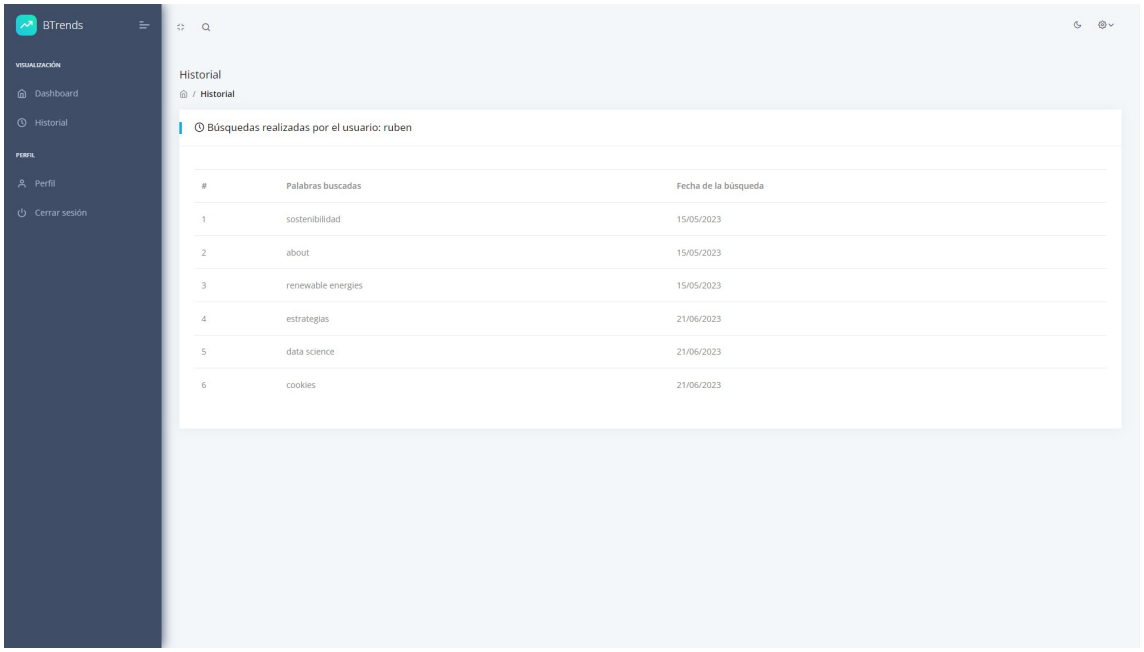


Figura 6.9: Pantalla de visualización del historial

6.5 Despliegue

Para el despliegue de la aplicación en un servidor, se ha utilizado la herramienta AWS y, más concretamente, una máquina virtual Amazon EC2. Esta máquina ha sido configurada para que solo permita el acceso a los usuarios conectados a la VPN de la Universidad Politécnica de Valencia. El proyecto se encuentra en una instancia **t2.micro** que ejecuta Ubuntu Server 22.04. Además, se han configurado reglas de entrada que permiten conectarse mediante SSH a la instancia y abrir los puertos necesarios para el funcionamiento de la aplicación.

Tras la creación de la instancia, se ha utilizado la herramienta Putty[3] para conectarse a esta. Para ello, es necesario contar con la dirección IP pública y con el par de claves generados con la instancia. En Putty, todo esto puede configurarse para conectarse con la instancia de forma casi automática mediante las diferentes opciones que ofrece esta herramienta, tales como indicar el puerto y la dirección IP de la instancia a la que se va a conectar (figura 6.10), el usuario de la instancia (figura 6.11) y el fichero de claves (figura 6.12).

Una vez conectado al servidor, el primer paso a realizar es importar la aplicación Django que se ha creado en local. Para ello se puede realizar mediante la herramienta WinSCP[4], que funciona como un cliente que permite traspasar archivos entre un ordenador local y un servidor remoto. Además, es compatible con Putty y es capaz de utilizar la información ssh almacenada en esta aplicación para conectarse al servidor en remoto de forma sencilla.

Otra forma de importar la aplicación y quizás la más fácil, es hacer uso de Github[5] y la función (figura 6.3) que tiene esta herramienta para clonar repositorios. Al estar alojando el proyecto en Github, por las ventajas de seguridad y de control de versiones que tiene esta página, es mucho más fácil hacer uso de este método para importar el proyec-

[3] https://putty.org/
[4] https://winscp.net/eng/index.php
[5] https://github.com/

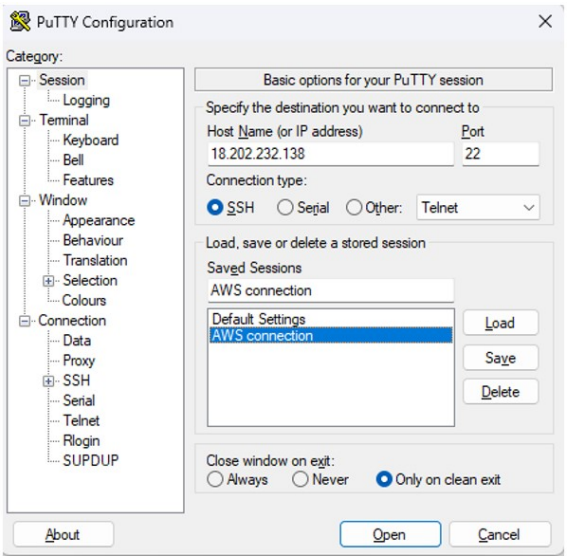


Figura 6.10: Configuración Putty para la conexión con el servidor AWS. Puerto e IP

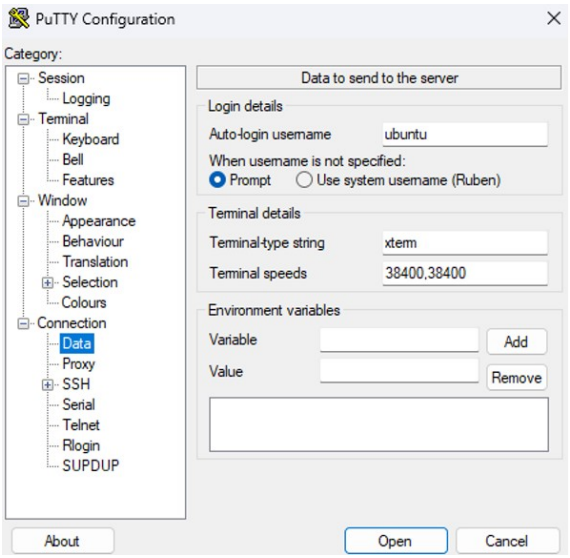


Figura 6.11: Configuración Putty para la conexión con el servidor AWS. Usuario

to. Mediante la siguiente línea de comando, se clona la aplicación entera en el servidor remoto:

Tras clonar el repositorio, se utiliza el fichero requirements.txt para instalar todas las librerías de Python necesarias en el servidor remoto. Este fichero debe ser creado con anterioridad mediante el comando *pip freeze >requeriments.txt* y guardado en la carpeta de la aplicación.

Una vez instaladas todas las librerías, se instalan las herramientas nginx[6], gunicorn[7] y supervisor[8]. Mediante la combinación de estas tres herramientas, es posible desplegar la aplicación en el servidor web y que funcione ininterrumpidamente. Como se observa en la figura 6.14, Nginx es el responsable de gestionar la solicitud web realizada por el cliente, redirigiéndola para que Gunicorn pueda procesarla utilizando *workers*. Cada *worker* o trabajador tiene la capacidad de comunicarse con la aplicación Django, lo que permite

[6] https://www.nginx.com/
[7] https://gunicorn.org/
[8] http://supervisord.org/

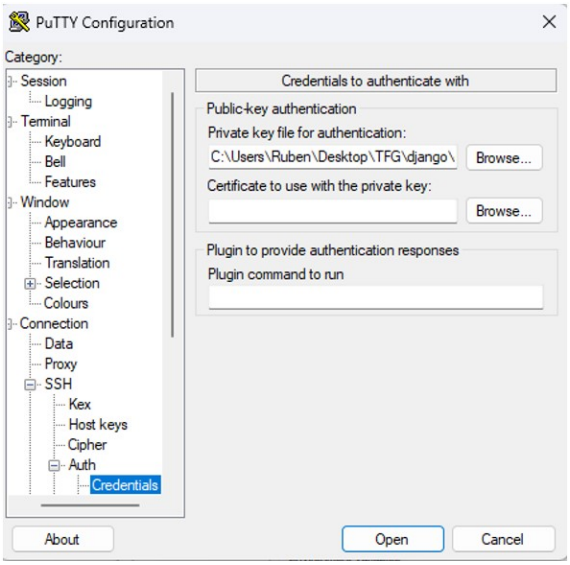


Figura 6.12: Configuración Putty para la conexión con el servidor AWS. Clave privada

```
1 git clone https://github.com/rubenmc2001/btrends.git
```

Figura 6.13: Función de Github para clonar repositorios

atender múltiples solicitudes simultáneamente y mantener un tiempo de respuesta óptimo. Por otro lado, Supervisor permite iniciar, detener o reiniciar la aplicación Django según sea necesario en cualquier momento, ofreciendo la capacidad de control sobre el proceso Gunicorn.

Para la configuración de gunicorn, y como se puede ver en la figura 6.15, se ha creado un fichero llamado ".deploy" que almacena la información del número de trabajadores que se desea, la dirección IP, el puerto en el que va a trabajar y la secuencia de ejecución, entre otros. Es importante tener en cuenta que la determinación de la cantidad óptima de trabajadores no solo se basa en el tráfico, sino también en las características del servidor, como la cantidad de procesadores y RAM disponible.

La configuración utilizada para Nginx se puede visualizar en la figura 6.16. En este caso, en la sección "upstream" se indica en qué servidor está disponible la aplicación, que corresponde al servidor local utilizado en la configuración de gunicorn. A continuación, la sección server indica sobre qué puerto está funcionando y el nombre del servidor mediante el cual se accede a la aplicación. Este nombre debe ser la dirección pública del servidor remoto creado en Amazon EC2. Las siguientes secciones redirigen las peticiones a Gunicorn.

Por último, se debe de crear un fichero que contenga la información de la figura 6.17. Este fichero sirve para que supervisor funcione y, como se puede observar, se le indica cual es el comando a ejecutar, el cual corresponde al fichero ".deploy" generado anteriormente y que contiene el comando de ejecución gunicorn. Además, se le indica el directorio de ejecución y otros comandos necesarios para su funcionamiento.

Una vez configurado estos tres servicios, la aplicación ya puede ser utilizada[9] y supervisada[10] (figura 6.18) por cualquier usuario que disponga de la VPN.

[9] http://34.245.212.249/
[10] http://34.245.212.249:9001/

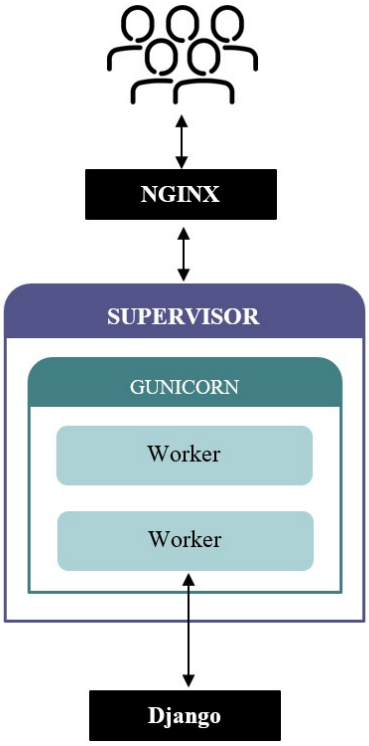


Figura 6.14: Funcionamiento de las herramientas nginx, gunicorn y supervisor

```
#!/bin/bash
NAME="btrends"
PATHHOME="/home/ubuntu/"
PATHENV="$PATHHOME/.venv"
PATHDJANGO="$PATHHOME/btrends"
USER="ubuntu"
GROUP="ubuntu"
NWORKERS=3
DJANGOWSGI="core.wsgi"
IP=127.0.0.1
PORT=8000
LEVEL="debug"
echo "Starting $NAME"
source $PATHENV/bin/activate
cd $PATHDJANGO
exec gunicorn $DJANGOWSGI --worker-class gevent --user=$USER --group=$GROUP --log-level=$level --bind=$IP:$PORT
```

Figura 6.15: . Fichero .deploy con la información necesaria para lanzar la aplicación

```
upstream btrends_app{
        server 127.0.0.1:8000;
}

server {
        listen 80;
        listen [::]:80;
        server_name 18.202.232.138;

        location / {
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_set_header Host $http_host;
                proxy_redirect off;
                proxy_pass http://btrends_app;
        }
}
```

Figura 6.16: Fichero de nginx donde se indica el servidor sobre el que trabaja

```
[program:btrends]
command=/home/ubuntu/.deploy
directory=/home/ubuntu/btrends
autostart=true
autorestart=true
stderr_logfile=/var/log/btrends.err.log
stdout_logfile=/var/log/btrends.out.log
```

Figura 6.17: Fichero de ejecución de gunicorn sobre el fichero .deploy

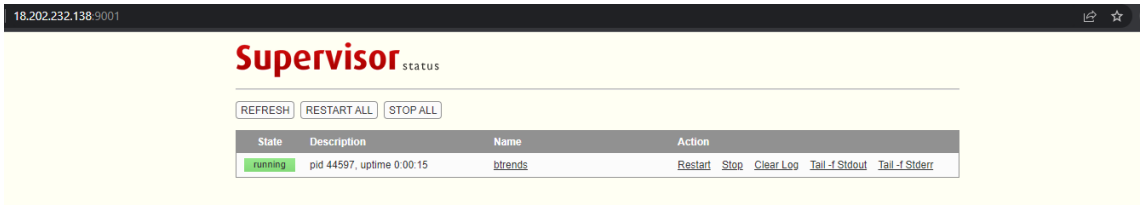


Figura 6.18: Portal que permite supervisar el funcionamiento de la aplicación

CAPÍTULO 7
Interfaz gráfica

En el marco del presente trabajo de fin de grado, se ha creado una base de datos que recopila las capturas extraídas hasta el año 2021. No obstante, existe la posibilidad de que se desee ampliar la información de la base de datos con nuevas capturas en el futuro. Para llevar a cabo esta tarea, sería necesario repetir el proceso de limpieza y subida de los nuevos datos al servidor. Con el fin de agilizar y simplificar este proceso, se ha desarrollado una interfaz gráfica que permite llevar a cabo la limpieza de los datos y su posterior subida al servidor de forma más sencilla y eficiente. De este modo, se asegura la actualización y ampliación de la base de datos de manera efectiva, ya que esta interfaz gráfica permite la automatización de gran parte del proceso, lo que reduce los tiempos y esfuerzos necesarios.

Además, la creación de esta interfaz gráfica conlleva una serie de ventajas, como la facilidad de uso, la automatización del proceso, la reducción de costes temporales y la mejora de la calidad de los datos, ya que se homogeniza todo el proceso, haciendo que cualquier dato insertado tras la carga inicial haya seguido el mismo procesamiento y limpieza.

7.1 Diseño de la interfaz

Para diseñar la interfaz de usuario, se ha utilizado el lenguaje Python, debido a que este ha sido utilizado durante todo el proyecto, lo que facilita la integración del proceso de limpieza y subida de datos. Concretamente, se ha hecho uso de la librería CustomTkinter[1] que, a su vez, se basa en la conocida librería Tkinter[2].

Tkinter (Tk) es una librería frecuentemente empleada en la realización de pruebas, prototipos y construcción de interfaces gráficas de usuario. Tk es un conjunto de herramientas de widgets multiplataforma de código abierto que es utilizado por diversos lenguajes de programación para el desarrollo de programas con interfaces gráficas.

Por otro lado, CustomTkinter es una biblioteca basada en Tkinter, que aporta widgets nuevos, modernos y totalmente personalizables. Hay pocas diferencias apreciables a la hora de escribir y ejecutar scripts entre Tkinter y CustomTkinter, ya que los widgets pueden ser empleados tanto como unos convencionales de Tkinter como en conjunto con otros elementos estándar de la misma librería. En la figura 7.1 se puede comprobar las diferencias entre cada una de las librerías a la hora de crear dos ventanas simples, compuestas únicamente por un botón.

[1] https://CustomTkinter.tomschimansky.com/
[2] https://docs.python.org/3/library/Tkinter.html

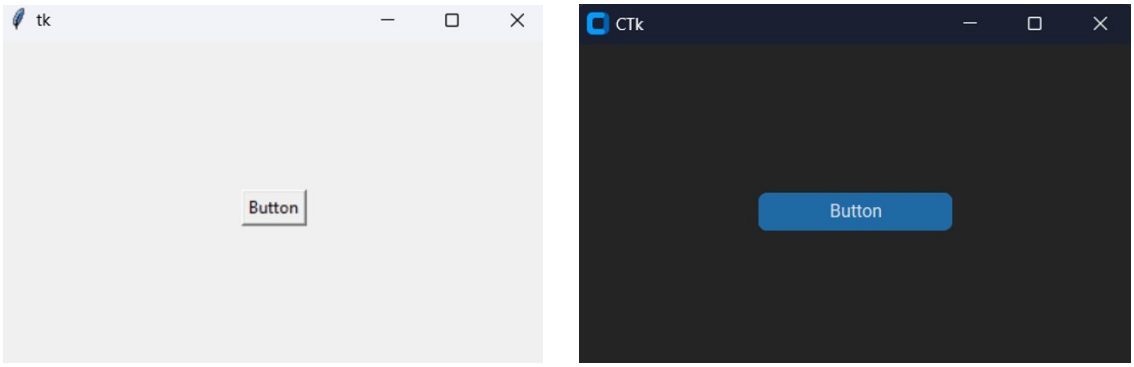


Figura 7.1: Ejemplos de interfaces gráficas simples con las librerías Tkinter (izquierda) y CustomTkinter (derecha)

En cuanto al diseño de la interfaz, este trata de ser lo más sencillo posible, con el objetivo de agilizar procesos y facilitar a los usuarios el uso de esta. Al ejecutar la interfaz, se desplegará una pantalla principal, que constará de una barra lateral y una pantalla principal.

En lo referente a la personalización de la interfaz de usuario, la barra lateral ofrecerá diversas funcionalidades. Una de las principales opciones es la posibilidad de modificar el tamaño general de la aplicación, incluyendo tanto el tamaño de las letras como el de los botones. Para llevar a cabo esta acción, se ha incorporado un botón desplegable que permite seleccionar entre diferentes porcentajes de aumento o disminución. El valor predeterminado es del 100 %, y la escala disponible oscila entre el 80 % y el 120 %. Por otro lado, también es posible personalizar el color de la interfaz, pudiendo escoger entre oscuro, claro o el predeterminado por el sistema. Esta funcionalidad está integrada en la librería CustomTkinter y es muy útil y fácil de integrar.

En cuanto al cuadro central de la interfaz, es el lugar donde se concentran las funcionalidades más importantes para el usuario. En particular, este cuadro está compuesto por tres cuadros de texto y dos botones de selección. En los primeros tres cuadros de texto, el usuario debe proporcionar información relevante, como su nombre de usuario, contraseña y la URL del servidor OpenSearch donde se almacenarán los datos. Esto es necesario para que los datos puedan ser subidos al servidor. Por otro lado, los dos botones de selección permiten al usuario elegir la ruta donde se encuentran los datos en su dispositivo, así como el archivo maestro con la información de las empresas. Una vez que toda esta información se haya proporcionado, el usuario puede hacer clic en el botón *Go* para iniciar el proceso de limpieza y subida de datos (figuras 7.2 y 7.3). Cuando haya finalizado el proceso, se desplegará una ventana que indicará que se ha completado la subida de datos, como se puede visualizar en la figura 7.4.

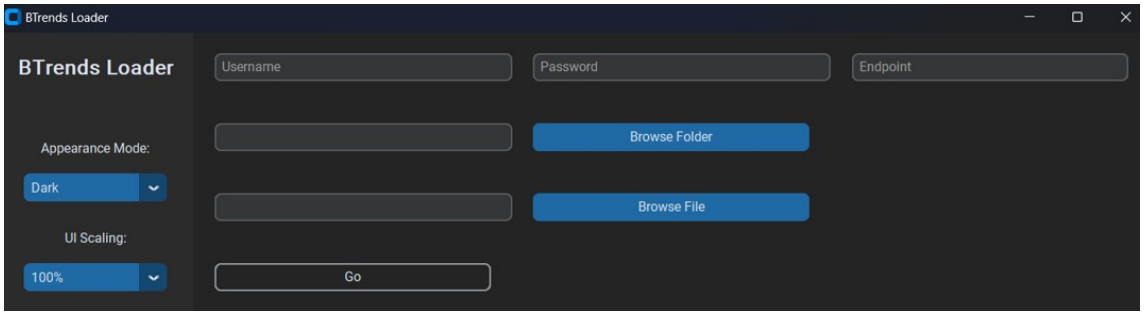


Figura 7.2: Diseño de la interfaz gráfica de usuario. Modo oscuro

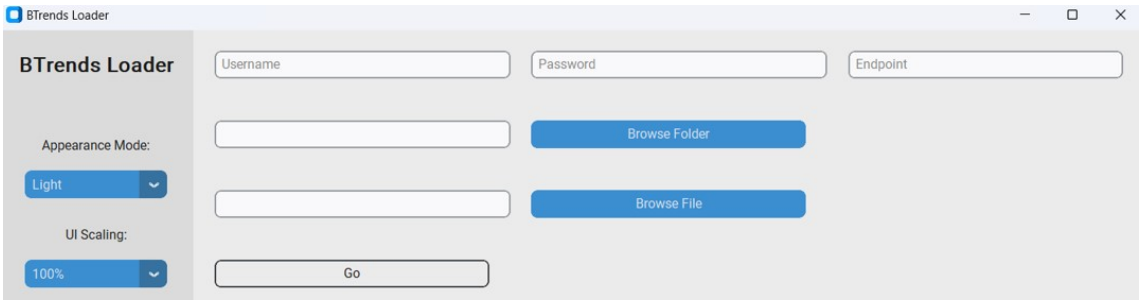


Figura 7.3: Diseño de la interfaz gráfica de usuario. Modo claro

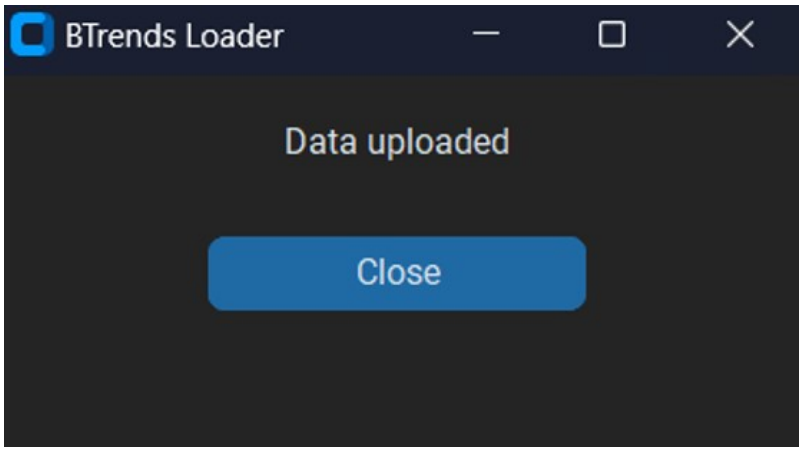


Figura 7.4: Ventana de confirmación de la subida de datos

CAPÍTULO 8
Estimación de costes

Uno de los factores a tener en cuenta es el de los costes del despliegue y mantenimiento de la base de datos OpenSearch y de la aplicación Django. A la hora de considerar los costes de los servidores, es importante tener en cuenta que los precios pueden variar ampliamente en función de la región en la que se aloje el servidor, el tipo de instancia elegido, el tipo y cantidad de almacenamiento utilizado, y otros factores. Además, al igual que con cualquier servidor, también se deben considerar los costes de mantenimiento y gestión del servidor.

Anteriormente se ha mencionado la elección de OpenSearch por encima de ElasticSearch debido a que ambas tecnologías no presentaban limitaciones que pudieran poner en peligro el proyecto, y se decidió por OpenSearch debido a que presentaba un menor coste. En este sentido, se realizará una comparación entre ambas tecnologías, utilizando los datos oficiales ofrecidos por cada una de las plataformas para poder analizar y evaluar las ventajas y desventajas de cada opción. De esta forma, se podrá tener un mejor contexto en relación con la elección de OpenSearch.

Por otro lado, también surge la necesidad de estimar los costes temporales del procesamiento de datos, ya que indirectamente están relacionados con el coste económico del proyecto. Estos costes temporales pueden incluir el tiempo necesario para preparar y limpiar los datos para su integración en la base de datos, así como el tiempo necesario para cargar los datos en el servidor y asegurarse de que estén disponibles para su uso en la aplicación. Es importante tener en cuenta que el tiempo necesario para estas tareas puede variar significativamente en función del tamaño y la complejidad de los datos, así como de la cantidad de recursos disponibles para realizar estas tareas.

Por último, en virtud del uso actual de servidores *cloud* con capacidad de almacenamiento limitado y de pago, es necesario llevar a cabo una evaluación de la cantidad de datos que se encuentran actualmente almacenados en los mismos, con el propósito de determinar si será necesario ampliar el espacio disponible en el futuro y, por ende, incurrir en un coste adicional.

8.1 Coste económico

La estimación del coste económico realizada es una mera predicción de lo que puede llegar a costar, a partir de la información que ofrece AWS en sus plataformas. Cabe recordar que se han utilizado dos servicios distintos de AWS, uno destinado al servidor que aloja los datos (OpenSearch) y otro donde funciona la herramienta de visualización. En el caso del servidor de datos, se ha realizado una comparación entre las tecnologías de OpenSearch y ElasticSearch

8.1.1. Servidor de almacenamiento de datos

Para la comparación, se va a hacer una suposición de un mismo escenario para ambas tecnologías, con el objetivo de poder comparar los costes y características de cada una de ellas de manera objetiva.

Para este escenario de comparación, se ha decidido seleccionar la región de Irlanda como ubicación para las instancias de ambas tecnologías, de manera que se pueda tener una comparación más precisa y objetiva en cuanto a los costes y el rendimiento. En este caso, se ha optado por utilizar un total de 3 instancias, cada una con 10 GB de almacenamiento y volúmenes de SSD (gp2).

Supongamos que se está iniciando en el uso de Amazon OpenSearch Service (tabla 8.1), creando un dominio en la región de Irlanda, con una instancia t3.small.search y 30 GB de almacenamiento de EBS. Considerando un uso mensual, cada una de las tres instancias del dominio tendría una duración de 720 horas, lo que sumaría un total de 2160 horas. A la fecha actual (22/03/2023), el precio de uso por hora es de 0,039$, lo que equivale a un coste mensual de 84,24$. En cuanto al almacenamiento de EBS, el precio se basa en la cantidad de GB seleccionada mensualmente, siendo de 0,135$ por GB en el caso de los volúmenes SSD gp2. Las tres instancias en conjunto tienen un almacenamiento total de 30 GB, lo que se traduce en un coste mensual adicional de 4,05$. Como resultado, el coste total mensual será de 88,28$.

Por otro lado, ElasticSearch te permite escoger entre tres proveedores de servidores: AWS (tabla 8.2), Azure y Google Cloud, siendo el más económico AWS. Suponiendo el mismo caso anterior, el servidor se situará en la región en Irlanda y se hará uso de 3 instancias de 10GB cada una. ElasticSearch ofrece el precio dependiendo de la suscripción en la que el usuario esté suscrito, siendo la de menos precio la Standard. El coste total de uso del servicio es de 0,0258$ por hora, que incluye la tarifa de almacenamiento. Aunque este precio por hora es más bajo que el ofrecido por OpenSearch, es importante tener en cuenta que solo se aplica a la suscripción Standard, que tiene un coste mensual de 95€. Por lo tanto, además del gasto de uso de los servidores, se debe agregar el coste de la suscripción. En este caso, se utilizaron un total de 2160 horas, lo que da como resultado un coste de 55,78$ multiplicando el precio por hora. Sumando el precio de la suscripción, el coste total es de 150,78$.

Tabla 8.1: Costes estimados del despliegue del servidor de ElasticSearch en AWS (OpenSearch)

AWS OpenSearch			
Tipo	**Precio**	**Uso facturado**	**Coste mensual**
Instancias	0,039$/h	2.160 horas	84,24$
Almacenamiento	0,135$/GB	30 GB	4,05$
			Total: 88,28$

Tabla 8.2: Costes estimados del despliegue del servidor de ElasticSearch en AWS (ElasticSearch)

AWS ElasticSearch			
Tipo	**Precio**	**Uso facturado**	**Coste mensual**
Instancias	0,0258$/h	2.160 horas	55,78$
Almacenamiento			95$
			Total: 150,78$

Después de analizar los costes y características de ambas tecnologías para el mismo escenario, se puede concluir que la opción de OpenSearch es más económica en compa-

ración con ElasticSearch, ya que aunque ElasticSearch ofrece un precio por hora más bajo en su suscripción Standard, el coste total mensual incluyendo la suscripción es significativamente mayor en comparación con OpenSearch. Por lo tanto, para este escenario en particular, la opción de OpenSearch es la más conveniente en términos de coste y características.

8.1.2. Servidor de la herramienta de visualización

La previsión de gastos del servidor que aloja la aplicación Django es algo más sencilla de realizar, puesto que se va a alojar sobre una instancia *t2.micro*. Hoy en día (22/02/2023), esta instancia tiene un coste de 0,0126$/hora, lo que multiplicado por 30 días suma un total de 9,2$.

Cabe destacar que estos costes son meras estimaciones previas al inicio del proyecto. El desglose real de los costes asociados al mantenimiento de los servidores se encuentra en el Apéndice C y se ha realizado una vez finalizado el despliegue y validación de la aplicación y el almacén de datos.

8.2 Coste temporal

El coste temporal proviene en gran parte del proceso de limpieza y preparación de los datos, y se explica más detalladamente en el Capítulo 4. Este proceso se divide en un total de 4 pasos:

- **Paso 1. Lista de urls del fichero de empresas**. Este proceso lo realiza una función Python que extrae todas las urls del fichero maestro de la información de empresas. El fichero Excel cuenta con un total de 32.248 empresas, por lo que el proceso no es costoso temporalmente (ya que únicamente ha de recorrer la columna url), con una duración media de menos de un segundo

- **Paso 2. Lista de urls en el directorio**. El siguiente proceso extrae las urls de los nombres de los ficheros de datos que se pretenden limpiar. Este proceso recorre todos los directorios de la carpeta de datos, por lo que depende del tamaño de esta. En total, la muestra con la que se trabaja contiene 4.091 directorios de datos, sumando un total de 302.181 ficheros de texto (con una media de 74 ficheros de texto por carpeta). El tiempo medio de obtención de estos datos es de unos 10 segundos

- **Paso 3. Extracción y limpieza del contenido de los ficheros**. Este paso se encarga de extraer todo el contenido que hay presente en cada uno de los más de 300.000 ficheros y realizarle una limpieza. De media, este proceso es bastante más costoso que los anteriores, ya que asciende a casi 40 minutos

- **Paso 4. Creación del fichero JSON con los datos**. Este paso es el último del procesamiento de los datos y es el previo a la subida. Aquí se genera el fichero formato JSON que se necesita para insertar los datos en la base de datos OpenSearch. De media, el tiempo que tarda este paso es cercano a 3 minutos

Por otro lado, también existe un cierto coste temporal al subir los datos anteriormente extraídos en formato JSON al servidor. Con las pruebas realizadas, se ha obtenido un promedio de 8 minutos para insertar los más de 300.000 datos con sus respectivos campos.

Tabla 8.3: Estimación de los costes temporales de proceso, limpieza y subida de datos

Paso	Tiempo de ejecución (s)	Tiempo de ejecución/ fichero (s)
Lista de urls del fichero de empresas	1	$3{,}177 * 10^{-6}$
Lista de urls en el directorio	10	$3{,}316 * 10^{-5}$
Extracción y limpieza	2400	0,008
Creación del fichero JSON con los datos	174	$5{,}759 * 10^{-4}$
Subida de datos	480	0,002
Tiempo total	**3065**	**0,010**

Un resumen de los pasos y del tiempo total empleado se puede observar en la siguiente tabla 8.3:

Por lo que, en total, el proceso de limpieza e inserción de datos conlleva unos 3065 segundos, que viene a ser 50 minutos. Es un proceso costoso temporalmente hablando, pero es cierto que se trata de una primera carga inicial, por lo que no es un problema tan crítico. Además, lo más normal es que si en un futuro se siguiesen insertando datos actualizados, sea menor la cantidad de ficheros que se tienen que tratar y, por consiguiente, menor el tiempo de ejecución.

Por ejemplo, si se desean realizar actualizaciones anuales, estaríamos hablando de solo 32.248 ficheros a tratar, por lo que multiplicándolo por el tiempo medio de ejecución por fichero ($\frac{0{,}010}{60} * 33592$), tardaría un total de 6 minutos en procesar y subir los datos.

8.3 Coste espacial

El archivo JSON diseñado para la inserción de los datos en el servidor de OpenSearch tiene un tamaño total de 1GB, lo cual es un factor crítico a considerar dado que es necesario contar con este espacio en el sistema local. Por otro lado, es posible verificar el tamaño de los índices creados a través de la consola proporcionada por OpenSearch y llamadas a su API. En nuestro caso, se ha creado un único índice denominado "btrends-index", que es la base de datos del contenido de todos los archivos. Para verificar el tamaño total ocupado por dicho índice, es posible utilizar el código de la figura 8.1:

```
1 GET _cat/indices?v&s=store.size:desc&h=index,store.size
```

Figura 8.1: Secuencia para las obtener el tamaño ocupado de cada índice sobre la API de OpenSearch

Con esta llamada obtenemos el siguiente resultado (figura 8.2), donde se observa que finalmente el índice ocupa 1,5GB en el servidor, lo que por el momento no representa ningún riesgo para la viabilidad del proyecto.

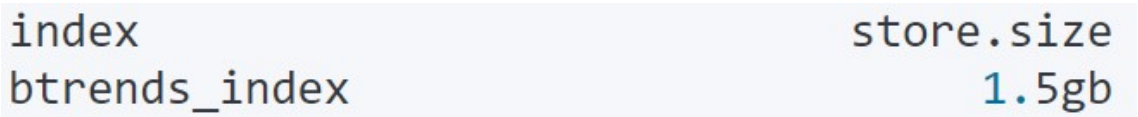


Figura 8.2: Espacio ocupado por la base de datos

8.4 Coste energético y medioambiental

Los centros de datos representan la columna vertebral de información de un mundo cada vez más digitalizado. La demanda de sus servicios ha aumentado rápidamente [25] y las tecnologías que necesitan datos para nutrirse, como la inteligencia artificial y los vehículos autónomos, amenazan con aumentar aún más la demanda. Dado que los centros de datos son empresas que consumen una gran cantidad de energía, se estima que representan alrededor del 1 % del consumo mundial de electricidad [26]. Varios análisis afirman que la energía utilizada por los centros de datos del mundo se ha duplicado en la última década y que su uso de energía se triplicará o incluso cuadruplicará en la próxima década [27, 28].

La expansión en la cantidad de grandes centros de datos y su demanda de electricidad asociada está teniendo un impacto significativo en el medio ambiente y en la economía global. Para abordar este problema de manera efectiva, es crucial que se adopten sistemas de energía renovable, limpios y de cero emisiones para satisfacer esta creciente demanda. En este contexto, se destaca la importancia de un sector eléctrico que dependa principalmente de fuentes renovables para lograr una UE climáticamente neutra para el año 2050[1].

En el trabajo realizado, este es un tema que se ha tenido muy en cuenta del mismo modo a la hora de elegir un centro de datos en el que alojar la base de datos y la aplicación. En este caso, y como se ha comentado anteriormente, se ha escogido trabajar sobre AWS por los beneficios derivados de un coste menor. Sin embargo, también se ha tenido en cuenta como de eficientes son sus centros de datos.

Según varios estudios llevados a cabo por *451 Research* [29], la infraestructura de AWS ha demostrado tener un ahorro de energía de casi cuatro veces más en comparación con la media de los centros de datos estadounidenses, llegando a cinco veces más en comparación con la media europea. Asimismo, se ha constatado que la implementación de AWS puede reducir la huella de carbono de los clientes en casi un 80 % y hasta en un 96 % cuando AWS funcione exclusivamente con energía renovable, un objetivo interpuesto por la compañía que se espera alcanzar antes de 2025[2].

[1] https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/topic-details/ee-20-2017

[2] https://sostenibilidad.aboutamazon.es/medioambiente/la-nube?energyType=true

CAPÍTULO 9
Conclusiones

En el presente trabajo, se ha creado una base de datos orientada a la búsqueda de texto, buscando optimizar los tiempos de búsqueda y minimizar los costes, así como una aplicación que muestra gráficos interactivos basados en la muestra almacenada en dicha base de datos. Todos los objetivos establecidos, tanto los principales como los secundarios, se han alcanzado en diferentes grados. Especialmente, se destaca el cumplimiento del objetivo principal, que consistía en crear una herramienta de visualización interactiva capaz de representar la evolución del interés por ciertos temas en los sitios web de las empresas. La aplicación web desarrollada se mantiene estable, ya que funciona en servidores AWS, y es capaz de generar visualizaciones de los términos deseados en tiempos adecuados, considerando las limitaciones existentes.

En cuanto a los objetivos secundarios, uno de los factores fundamentales para lograr los buenos tiempos de consulta en la aplicación ha sido la elección de una base de datos acorde a los requerimientos del trabajo. A través de un riguroso estudio de las posibilidades que había, se acabó escogiendo la tecnología de ElasticSearch para alojar el almacén de datos. Esto ha contribuido significativamente a la optimización de los tiempos de respuesta. Además, la herramienta de visualización propuesta a través de Django se ha desplegado correctamente, llegando a funcionar sobre ordenadores y dispositivos móviles.

Gran parte de estos objetivos pueden ser cubiertos de forma superficial y más sencilla con los conocimientos adquiridos por el autor durante el grado. Sin embargo, con el objetivo de realizar un trabajo provechoso y a la altura de un Trabajo de Fin de Grado, se han utilizado los conocimientos ya alcanzados como base para desarrollar soluciones más complejas. Por ejemplo, se ha utilizado la gran base teórica y práctica de Python para crear la herramienta de visualización con Django, así como la base que se tenía con AWS para desplegar tanto el almacén de datos como la aplicación en sus servidores. Por lo que respecta a dicha base de datos, se han utilizado los conocimientos obtenidos sobre bases de datos no relacionales como punto de partida.

Como se comenta en secciones anteriores, la inexperiencia con estas tecnologías ha supuesto retrasos en la confección del proyecto, debido en gran parte a la dificultad del despliegue de la aplicación Django en los servidores AWS, causado por la poca documentación existente para realizar tal tarea. Sin embargo, tras muchos intentos y comprendiendo cada uno de los pasos que había que seguirse, se ha conseguido completar ese punto.

Por otra parte, se han identificado errores en el diseño de la aplicación relacionados con la compatibilidad en dispositivos móviles. Inicialmente, se había diseñado y probado la visualización de los gráficos exclusivamente en ordenadores, lo que resultó en un desajuste en la presentación en dispositivos móviles. Ante esta situación, se ha llevado a

cabo un proceso de rediseño de las visualizaciones para asegurar su correcta visualización en ambos tipos de dispositivos. Estos obstáculos y errores encontrados a lo largo del desarrollo del proyecto dan lecciones y oportunidades de aprendizaje. Asimismo, han reforzado la capacidad para enfrentar y superar desafíos técnicos, así como para adquirir habilidades de adaptabilidad y resolución de problemas en un entorno real.

En conclusión, a pesar de los retrasos ocasionados por la inexperiencia con las tecnologías y los errores en el diseño, se ha logrado superar estos obstáculos y alcanzar los objetivos establecidos, dando como resultado final una aplicación con una presentación y un despliegue exitoso.

9.1 Legado

En primer lugar, el legado de este proyecto se resume en la propia herramienta de visualización desarrollada y, en relación a esta, se destaca la importancia de comunicar el impacto a corto, medio y largo plazo. La aplicación web estable y funcional, desplegada en servidores AWS, tiene el potencial de ser utilizada y beneficiar a diferentes organizaciones y empresas en la mejora de sus análisis de datos y la toma de decisiones, ya que dota a otros investigadores de una opción rápida y manejable para analizar las tendencias de los contenidos de las páginas web de las empresas. El acceso a estos gráficos interactivos desde diferentes dispositivos, incluyendo dispositivos móviles, amplía aún más el alcance y la utilidad de la aplicación, brindando flexibilidad y accesibilidad a los usuarios. Además, la creación de una interfaz gráfica que permite insertar nuevos datos en la base de datos permite seguir actualizando anualmente la información, evitando quedarse obsoleta. Sin embargo, debido a la rápida evolución de las diferentes tecnologías de almacenamiento y visualización, el autor es totalmente consciente de que este trabajo puede quedar obsoleto en un futuro.

Por otro lado, parte del legado de este proyecto resume en la optimización de los tiempos de búsqueda y la minimización de los costes asociados. Como se ha comentado a lo largo de este proyecto, la base de datos creada, utilizando la tecnología de ElasticSearch, ha demostrado ser una solución eficiente para lograr una respuesta rápida y precisa en las consultas realizadas. Esto permitirá a los beneficiarios del proyecto ahorrar tiempo y recursos al acceder a la información relevante de manera eficiente.

En términos de legado técnico, este proyecto promueve la transparencia y la reproducibilidad de la investigación y el análisis de datos. Se ha garantizado que el acceso a los datos y el código utilizado en el proyecto esté disponible en repositorios abiertos[1], lo que permite a terceros reproducir el análisis y obtener resultados similares. Además, la documentación y la descripción realizada durante todo el trabajo proporciona detalles adicionales sobre los procedimientos y metodologías empleados, facilitando la comprensión y la replicación de los pasos seguidos.

9.2 Relación del trabajo desarrollado con los estudios cursados

El trabajo realizado y presentado proviene del resultado de una reflexión sobre los estudios realizados y como poder aplicarlos en un Trabajo de Fin de Grado (TFG). Es por esto por lo que se ha analizado como pueden ser aplicados los conocimientos adquiridos durante la carrera en un entorno de producción y real, que permita resolver necesidades reales y que, a su vez, permita aumentar y completar la formación académica.

[1] https://github.com/rubenmc2001/btrends

Uno de los objetivos principales al realizar un TFG es el de demostrar que el alumno ha logrado adquirir los conocimientos recibidos durante los años de estudio universitario, dando soluciones reales a problemas del mundo laboral. Para demostrar esto, el trabajo actual aborda el estudio de la huella digital que las empresas dejan en internet. Así, el trabajo realizado se nutre de los estudios realizados en el Grado de Ciencia de Datos, ya que se han aplicado conocimientos teóricos y prácticos relacionados con el comportamiento económico, la programación en Python, la visualización de datos mediante librerías específicas de este lenguaje de programación, la estadística y el uso de herramientas de *cloud*.

Además, se han puesto en práctica habilidades y competencias transversales desarrolladas durante la carrera universitaria. Concretamente, las detalladas a continuación:

- **CT-01. Comprensión e integración**. Durante la formación en el grado, se han adquirido conocimientos profundos sobre las técnicas y herramientas utilizadas en el análisis de datos y la construcción de modelos predictivos. En este proyecto, se han aplicado parte de estos conocimientos para comprender y analizar la huella digital que las empresas dejan en la red, así como para integrar esta información tanto en una base de datos optimizada a texto como en un *dashboard* interactivo, funcionando ambos en la nube. Todo ello ha requerido una comprensión profunda de los conceptos teóricos y su aplicación práctica en un contexto real

- **CT-02. Aplicación y pensamiento práctico**. Se ha aplicado el conocimiento teórico adquirido y se ha establecido un proceso efectivo para alcanzar los objetivos propuestos, desde el procesamiento de datos hasta la creación de una aplicación web que permite visualizar gráficos

- **CT-03. Análisis y resolución de problemas**. Se han analizado y resuelto problemas de forma efectiva, identificando y definiendo los elementos significativos que los constituyen, tales como la identificación de los datos relevantes para la construcción de una base de datos optimizada a texto, la solución a la casuística del tiempo de ejecución a la hora de realizar consultas evitando altos costes temporales y la selección de las herramientas adecuadas para llevar a cabo el trabajo

- **CT-04. Innovación, creatividad y emprendimiento**. Se ha innovado y demostrado una actitud emprendedora para responder de forma original a las necesidades y demandas en el mundo laboral mediante la creación de un *dashboard* interactivo que muestra el impacto de la huella digital económica

- **CT-07. Responsabilidad ética, medioambiental y profesional**. Se ha actuado con responsabilidad ética, medioambiental y profesional durante todo el proceso de investigación y desarrollo, asegurando la privacidad de los datos el respeto a la normativa vigente

- **CT-08. Comunicación efectiva**. Se ha comunicado de manera efectiva utilizando adecuadamente los recursos necesarios y adaptándose a las características de la situación y de la audiencia de un Trabajo de Fin de Grado, tanto en la redacción de este como en su defensa

- **CT-11. Aprendizaje permanente**. Se ha utilizado el aprendizaje de manera estratégica, autónoma y flexible, a lo largo de toda la vida, en función del objetivo perseguido. Se ha buscado y utilizado diferentes recursos y fuentes de información para ampliar y profundizar el conocimiento sobre el tema, tanto a nivel teórico como práctico. Todo ello con el fin de mejorar continuamente la capacidad para dar respuesta a problemas reales en un futuro profesional

- **CT-12. Planificación y gestión del tiempo**. Se ha diseñado y seguido un plan detallado para llevar a cabo todas las etapas del proyecto de forma eficiente y eficaz, desde la definición del problema y los objetivos, hasta la planificación de las actividades y la asignación de tiempos. Esta competencia ha sido aplicada para poder cumplir con los plazos establecidos y alcanzar los resultados esperados
- **CT-13. Instrumental específico**. Se ha seleccionado y aplicado de forma adecuada las herramientas y tecnologías necesarias para llevar a cabo este trabajo, donde distintas herramientas de análisis y visualización de datos, programación y bases de datos han sido utilizadas

Bibliografía

[1] Weaver, Stephen D. y Gahegan, Mark. Constructing, visualizing, and analyzing a digital footprint. *Geographical Review*. 97:3:324–350, 2007.

[2] Doménech i de Soria, Josep, y otros. Assessing the green transition priorities of SMEs: A large scale web mining approach. *4th International Conference on Advanced Research Methods and Analytics (CARMA 2022)*. 278, 2022.

[3] Chapman Peter, y otros. CRISP-DM 1.0: Step-by-step data mining guide. *SPSS inc*. 9:13:1–73, 2000.

[4] Blazquez, Desamparados y Doménech i de Soria, Josep. Web data mining for monitoring business export orientation. *echnological and Economic Development of Economy*. 24:2:406–428, 2018.

[5] Doménech i de Soria, Josep, Martínez Gómez, Víctor y Mas Verdú, Francisco. Location and adoption of ICT innovations in the agri-food industry. *Technological and Economic Development of Economy*. 21:6:421–424, 2014.

[6] Orduna Malea, Enrique, Font, Cristina y Ontalba Ruipérez, José Antonio. From Universities to Private Companies: A Measurable Route of Linkedin Users. *Digital tools for academic branding and self-promotion*. IGI Global, 127–150, 2017.

[7] Cui, Ruomeng, y otros. The Operational Value of Social Media Information. *Production and Operations Management*. 27:10:1749–1769, 2017.

[8] Crosato, Lisa, Doménech i de Soria, Josep y Liberati, Caterina. Websites' data: a new asset for enhancing credit risk modeling. *Annals of Operations Research*. 1–16, 2023.

[9] Overbeeke, Marlies y Snizek, William. Web sites and corporate culture: A research note. *Business & Society*. 44:3:346–356, 2005.

[10] Meroño Cerdán, Angel L. y Soto Acosta, Pedro. External web content and its influence on organizational performance. *European Journal of Information Systems*. 16:1:66–80, 2007.

[11] Llopis, Juan, González, Reyes y Gasco, Jose. Web pages as a tool for a strategic description of the Spanish largest firms. *Information processing & management*. 46:3:320–330, 2010.

[12] Axenbeck, Janna y Breithaupt, Patrick. Innovation indicators based on firm websites-which website characteristics predict firm-level innovation activity?. *Annals of Operations Research*. 16:4, e0249583, 2021.

[13] Jun, Seung Pyo, Sun Yoo, Hyoung y Choi, San. Ten years of research change using Google Trends: From the perspective of big data utilizations and applications. *Technological Forecasting and Social Change*. 130:69–87, 2018.

[14] Goel, Sharad, y otros. Predicting consumer behavior with Web search. *Proceedings of the National academy of sciences*. 107:41:17486–17490, 2010.

[15] Choi, Hyunyoung y Varian, Hal. Predicting the Present with Google Trends. *Economic record*. 88:2–9, 2012.

[16] Jun, Seung Pyo, Park, Do Hyung y Yeom, Jaeho. The possibility of using search traffic information to explore consumer product attitudes and forecast consumer preference. *Technological Forecasting and Social Change*. 86:237–253, 2014.

[17] European Commission. Statistical Classification of Economic Activities in the European Community. *Eurostat: Methodologies and Working papers*. 2008.

[18] Bowyer, Surya. The Wayback Machine: notes on a re-enchantment. *Archival Science*. 21:1:43–57, 2021.

[19] Jatana, Nishtha, y otros. A Survey and Comparison of Relational and Non-Relational Database. *International Journal of Engineering Research & Technology*. 1:6:1–5, 2012.

[20] Portilla Tovar, Johan Alexander y Bernal Gómez, Mireya. Comparación del rendimiento de los comandos Insert, Select y Delet en los sistemas gestores de bases de datos Oracle y MYSQL. *Inventum*. 12:23:10–21, 2017.

[21] Thakur, Nimesh y Gupta, Nishi. Relational and non relational databases: A review. *Journal of University of Shanghai for Science and Technology*. 23:8:117–121, 2021.

[22] Andhavarapu, Abhishek. *Learning ElasticSearch*. Birmingham : Packt Publishing Ltd, 2017.

[23] Akca, Mustafa Ali, Aydogan, Tuncay y Ilkuçar, Muhammer. An Analysis on the Comparison of the Performance and Configuration Features of Big Data Tools Solr and ElasticSearch. *International Journal of Intelligent Systems and Applications in Engineering*. 4:8–12, 2016.

[24] Holovaty, Adrian y Kaplan Moss, Jacob. The definitive guide to Django: Web development done right. *Apress*. 2009.

[25] Cisco. Cisco Global Cloud Index: Forecast and Methodology. *Cisco Systems*. 2021.

[26] International Energy Agency. Digitalization & Energy. *IEA*. 2017.

[27] Belkhir, Lotfi y Elmeligi, Ahmed. Assessing ICT global emissions footprint: Trends to 2040 & recommendations. *Journal of cleaner production*. 177:448–463, 2018.

[28] Andrae, Anders y Edler, Tomas. On Global Electricity Usage of Communication Technology: Trends to 2030. *Challenges*. 6:1:117–157, 2015.

[29] Bizo, Daniel. The Carbon Reduction Opportunity of Moving to Amazon Web Services. *AWS*. 2019.

APÉNDICE A
Objetivos de desarrollo sostenible

Grado de relación del trabajo con los Objetivos de Desarrollo Sostenible (ODS).

Objetivos de Desarrollo Sostenible	**Alto**	**Medio**	**Bajo**	**No procede**
ODS 1. **Fin de la pobreza.**				**X**
ODS 2. **Hambre cero.**				**X**
ODS 3. **Salud y bienestar.**				**X**
ODS 4. **Educación de calidad.**				**X**
ODS 5. **Igualdad de género.**				**X**
ODS 6. **Agua limpia y saneamiento.**				**X**
ODS 7. **Energía asequible y no contaminante.**	**X**			
ODS 8. **Trabajo decente y crecimiento económico.**				**X**
ODS 9. **Industria, innovación e infraestructuras.**		**X**		
ODS 10. **Reducción de las desigualdades.**				**X**
ODS 11. **Ciudades y comunidades sostenibles.**				**X**
ODS 12. **Producción y consumo responsables.**		**X**		
ODS 13. **Acción por el clima.**				**X**
ODS 14. **Vida submarina.**				**X**
ODS 15. **Vida de ecosistemas terrestres.**				**X**
ODS 16. **Paz, justicia e instituciones sólidas.**				**X**
ODS 17. **Alianzas para lograr objetivos.**	**X**			

Reflexión sobre la relación del TFG con los ODS y con los ODS más relacionados.

La Agenda 2030 es un plan de acción global adoptado por los Estados miembros de las Naciones Unidas en 2015, que establece un conjunto de 17 Objetivos de Desarrollo Sostenible (ODS) y 169 metas asociadas a ellos, con el fin de erradicar la pobreza, proteger el planeta y garantizar la prosperidad para todas las personas[1]. Los objetivos de la Agenda 2030 incluyen desde la eliminación de la pobreza y el hambre, hasta la lucha contra el cambio climático, la promoción de la igualdad de género, el acceso a la educación, la salud y el agua potable, entre otros. En relación con el Trabajo de Fin de Grado (TFG) llevado a cabo, es necesario destacar ciertos ODS relacionados con el propósito del proyecto que se van a describir a continuación.

- **ODS 7. Energía asequible y no contaminante**. En el actual proyecto, se ha tomado en cuenta la eficiencia energética de los centros de datos al optar por la plataforma de AWS, la cual ha demostrado presentar un ahorro de energía significativo en comparación con la media de otros centros de datos especializados. Esto implica una reducción en el consumo de energía y, por ende, una menor emisión de gases de efecto invernadero y contaminación. El uso de la herramienta AWS por la base de datos y el servidor se alinea con las metas 7.2 y 7.3, ya que *Amazon Web Services* ha estado aumentando considerablemente la proporción de energía renovable para el funcionamiento de sus servidores, teniendo como objetivo para 2025 que el 100 % de la energía empleada en sus operaciones sea renovable. Además, al tratarse de un centro de datos, uno de los puntos más críticos que deben de optimizar es la eficiencia energética que normalmente está relacionada con la refrigeración de sus servidores, objetivo que han conseguido cumplir optimizando la longevidad y la circulación del aire del medio refrigerante empleado en los sistemas de refrigeración de los centros de datos, reduciendo en un 20 % el gasto energético
- **ODS 9. Industria, innovación e infraestructuras**. El análisis de la huella digital mediante un *dashboard* interactivo es una innovación tecnológica que puede incluso aumentar la productividad de las empresas. Esta aplicación está alineado con la meta 9.3, puesto que el *dashboard* permite el acceso a cualquier tipo de usuario, ya sea pequeña empresa o no, integrándolos en la cadena de valor y los mercados
- **ODS 12. Producción y consumo responsables**. Asimismo, el cumplimiento de este objetivo está algo relacionado con el ODS 7, ya que una elección consciente y responsable del centro de datos basada en criterios de eficiencia energética y costes reducidos se traduce en un consumo más responsable de los recursos. Además, el uso de servidores *cloud* se presenta como una excelente alternativa para ahorrar energía, debido a que los proveedores de servicios *cloud* organizan su infraestructura en centros de datos optimizados, pudiendo ser compartidos entre diferentes usuarios y logrando así un ahorro significativo de energía
- **ODS 17. Alianzas para lograr objetivos**. Por último, es posible que el uso del cuadro de mandos llegue a ser útil a la hora de tomar decisiones por parte de las empresas y/o usuarios. De este modo, el proyecto está alineado con las metas 17.7 y 17.18, ya que el uso de esta tecnología promueve el uso de plataformas ecológicamente racionales a cualquier parte del mundo y proporciona un aumento significativo de datos oportunos, fiables y de gran calidad

[1] https://www.un.org/sustainabledevelopment/es/development-agenda/

APÉNDICE B
Capturas de la herramienta de visualización

En el actual apéndice, se muestran las capturas de pantalla realizadas sobre la herramienta de visualización, tanto en el despliegue para ordenadores como para móviles. En estas capturas, se van a mostrar las pantallas de inicio de sesión, todos los gráficos, la opción de ajustes, el perfil y otras funcionalidades como el modo oscuro.

B.1 Herramienta web en ordenadores

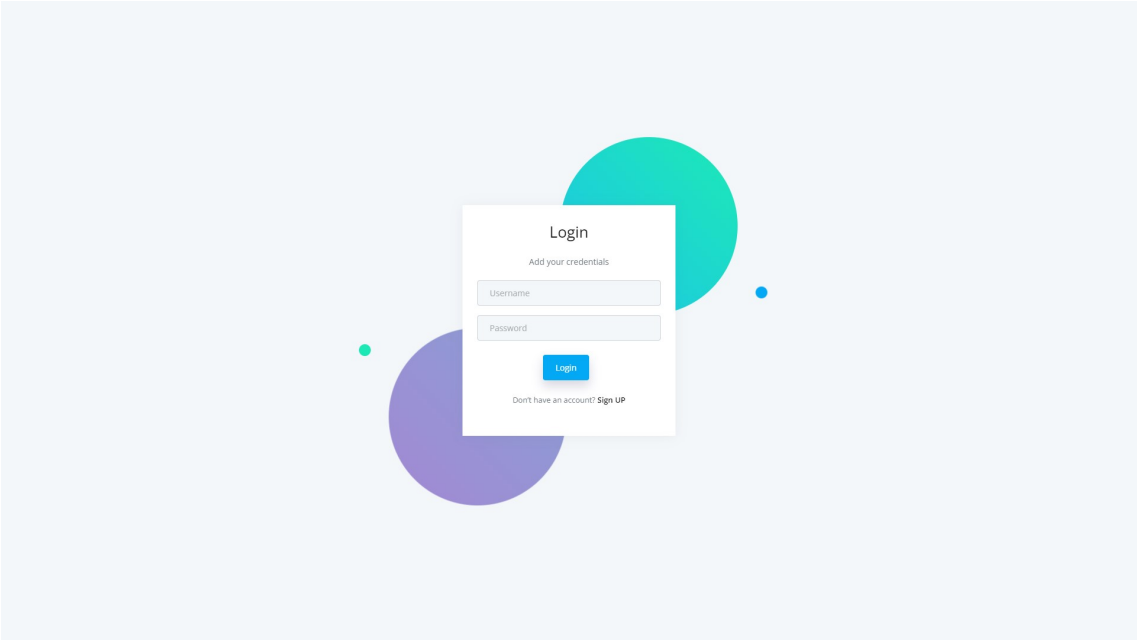


Figura B.1: Pantalla de inicio de sesión. Modo claro

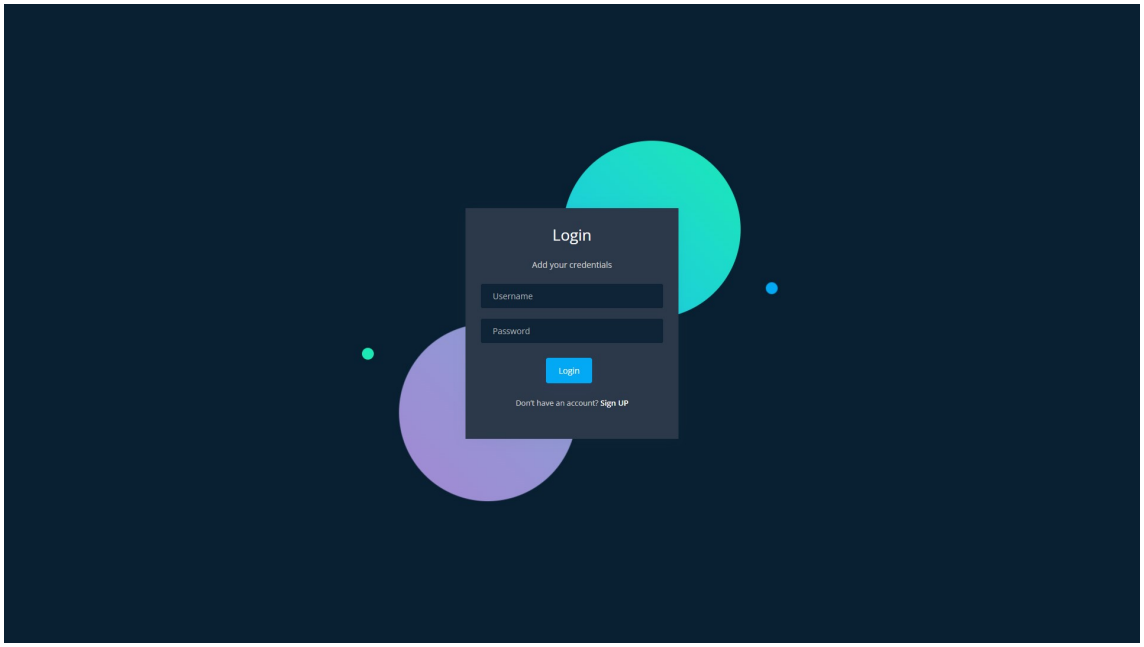


Figura B.2: Pantalla de inicio de sesión. Modo oscuro

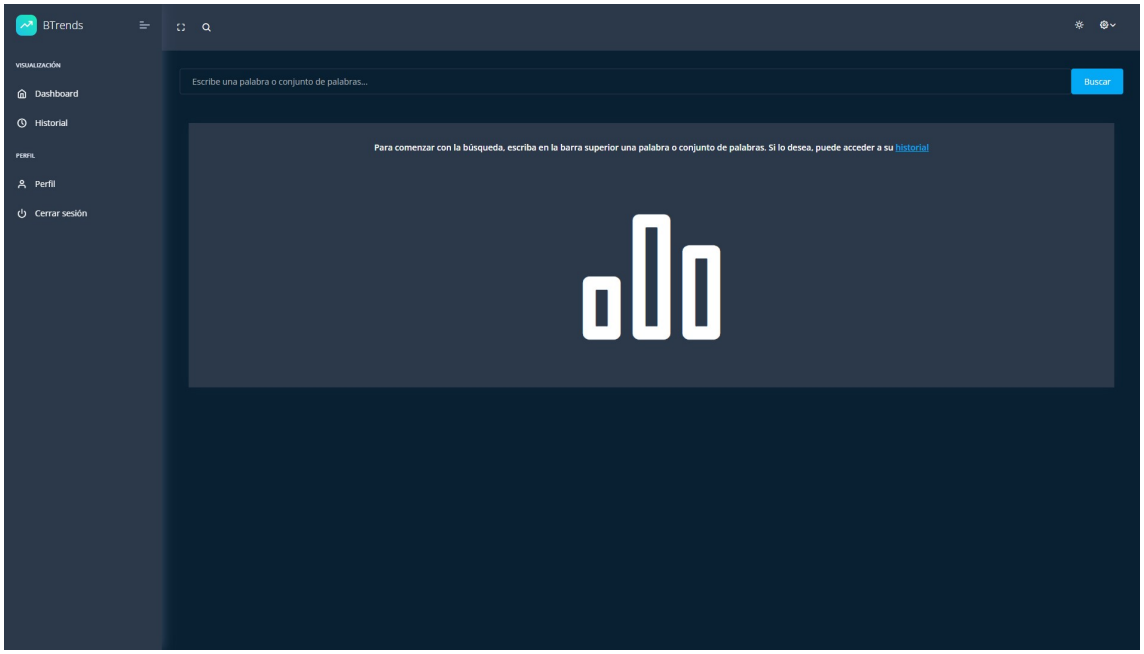


Figura B.3: Pantalla principal. Modo oscuro

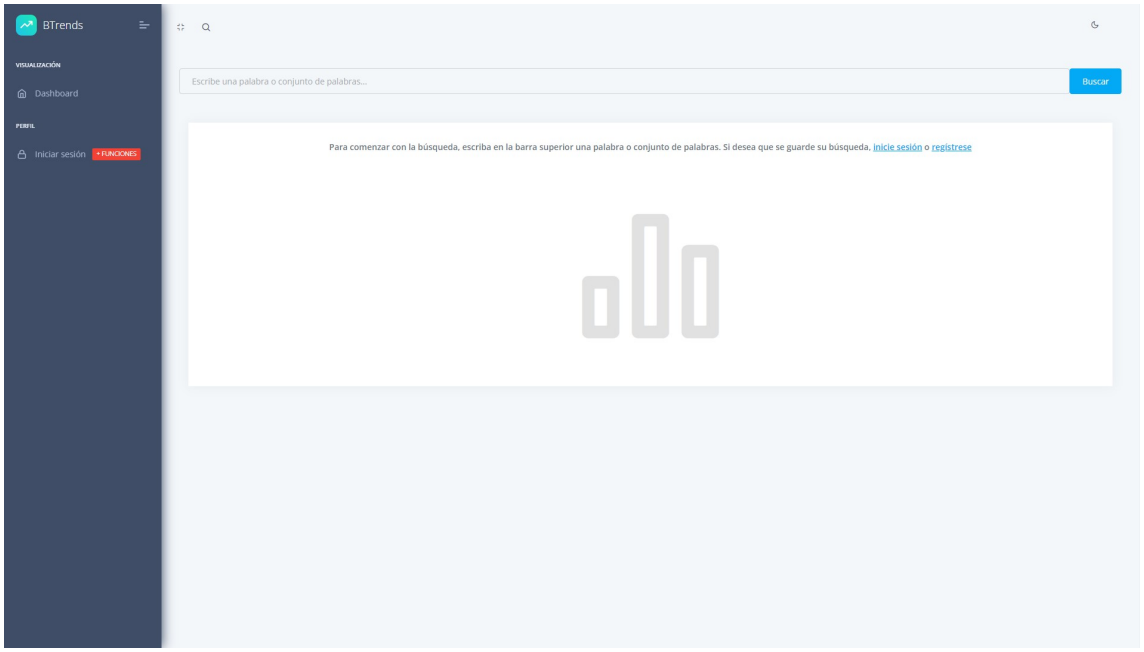


Figura B.4: Pantalla principal sin iniciar sesión. Modo claro

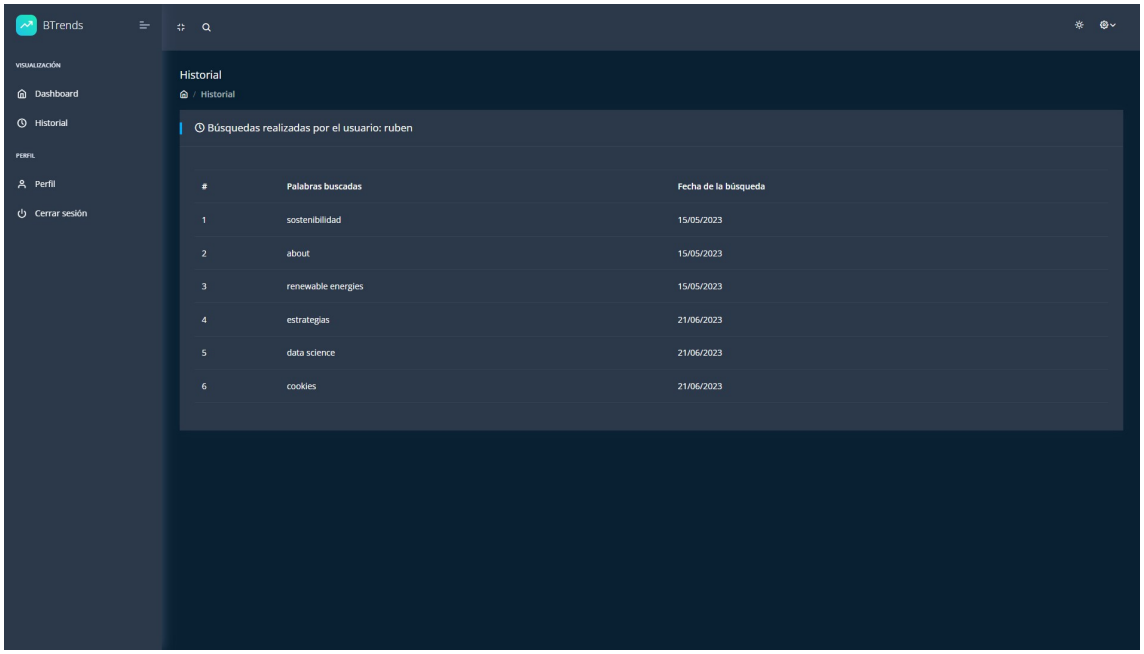


Figura B.5: Pantalla del historial. Modo oscuro

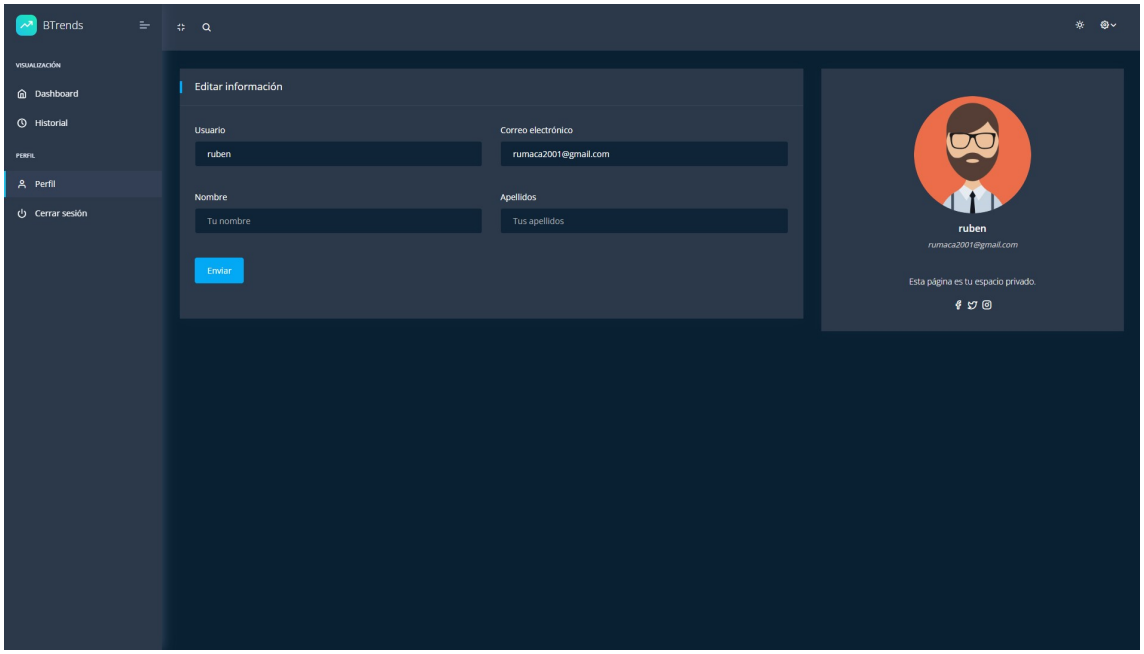


Figura B.6: Pantalla de perfil. Modo oscuro

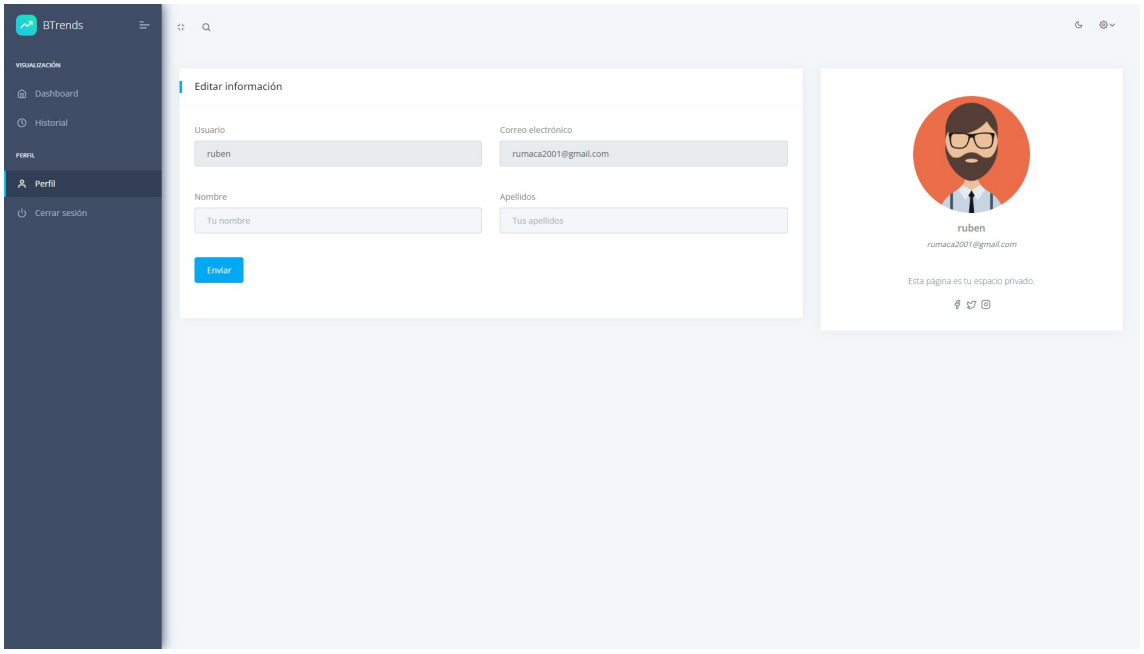


Figura B.7: Pantalla de perfil. Modo claro

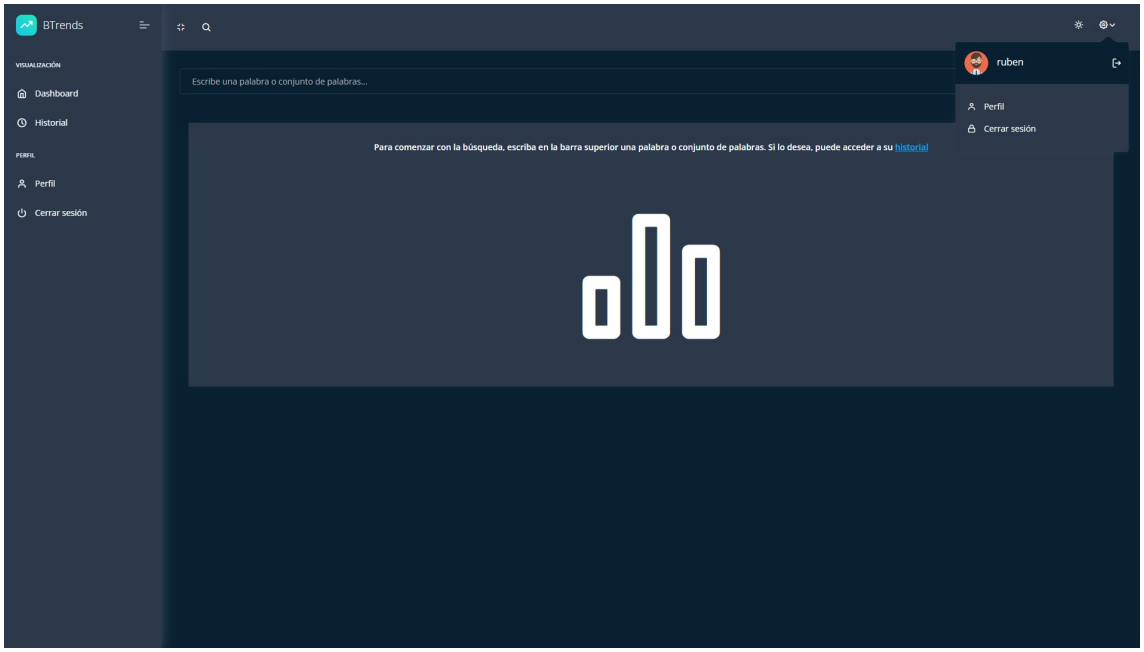


Figura B.8: Pantalla de ajustes. Modo oscuro

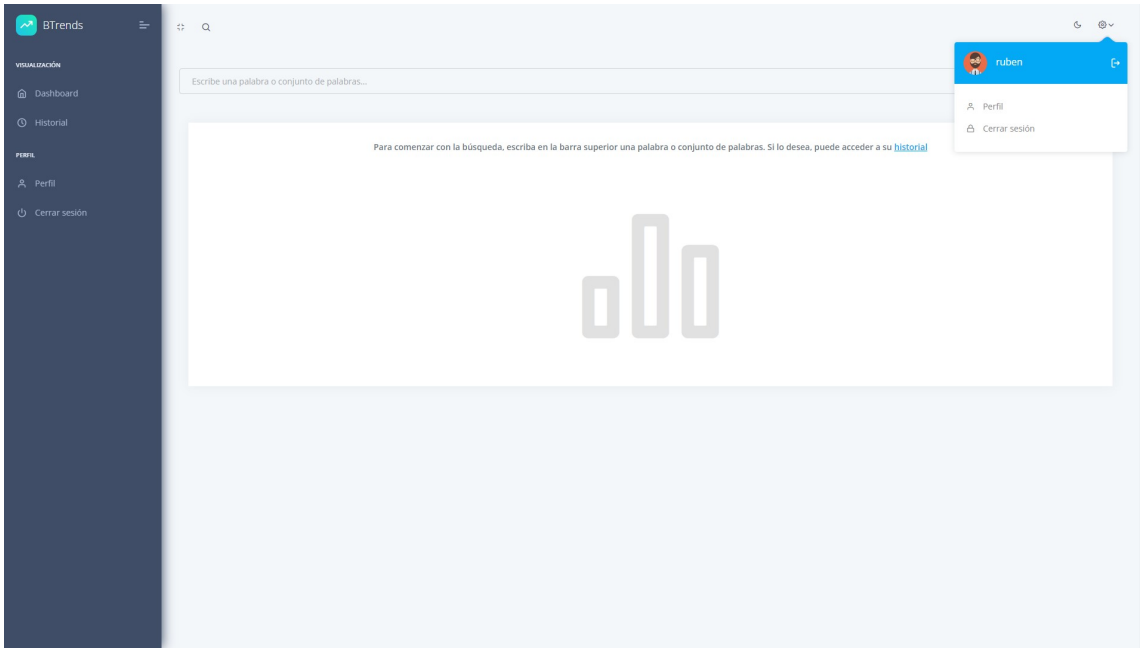


Figura B.9: Pantalla de ajustes. Modo claro

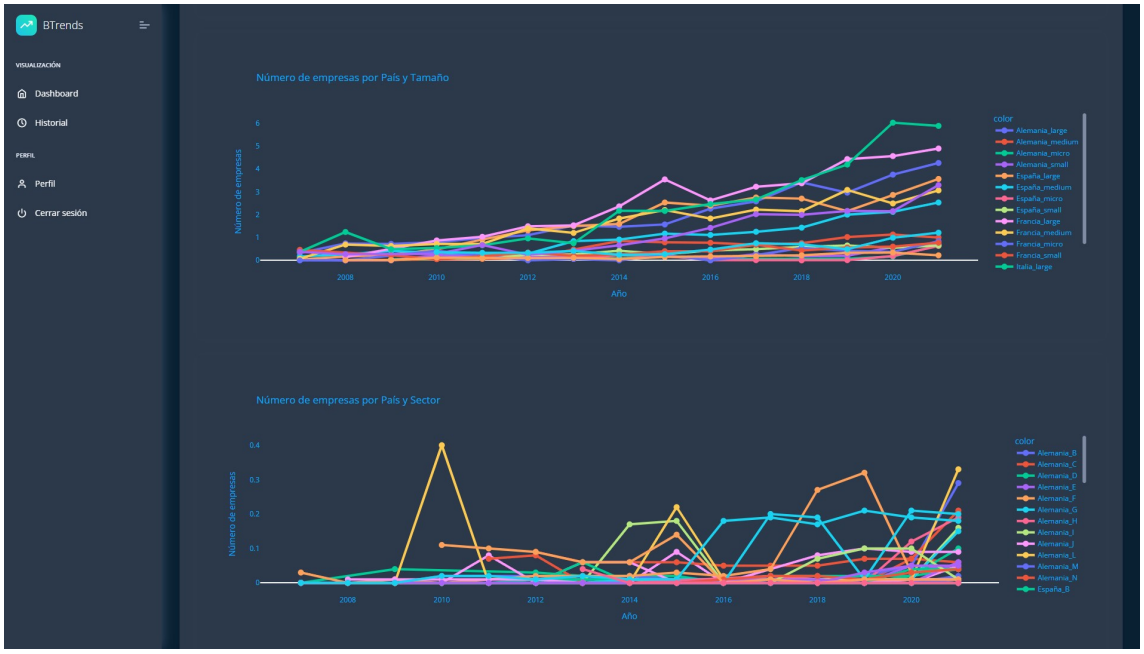


Figura B.10: Pantalla de visualización de los gráficos de líneas. Modo oscuro

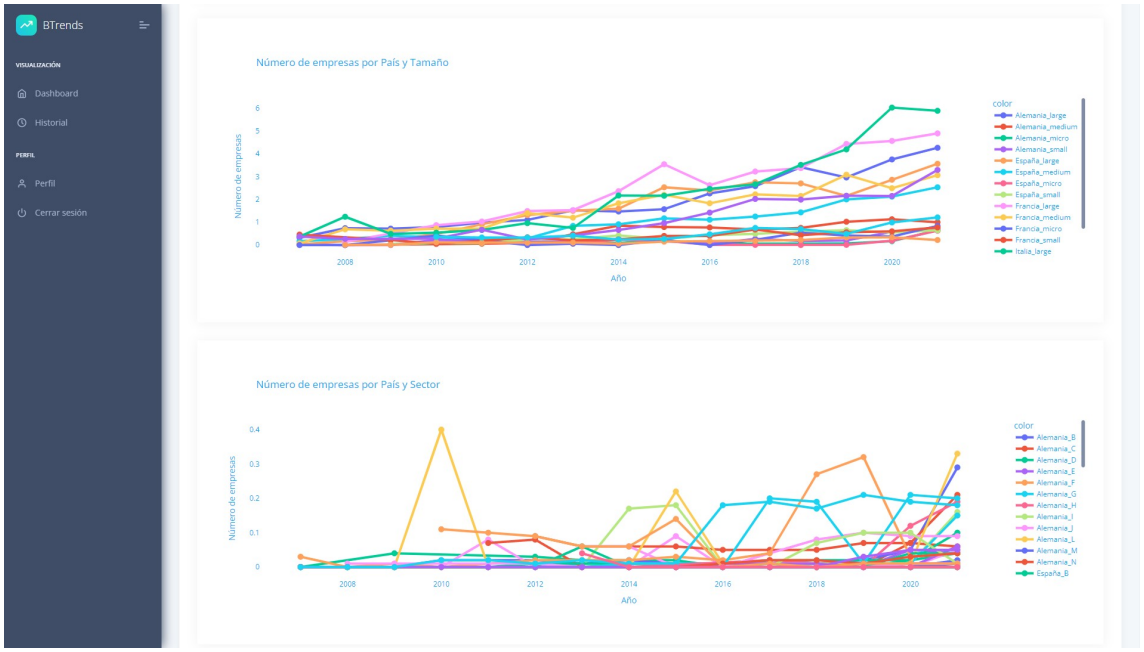


Figura B.11: Pantalla de visualización de los gráficos de líneas. Modo claro

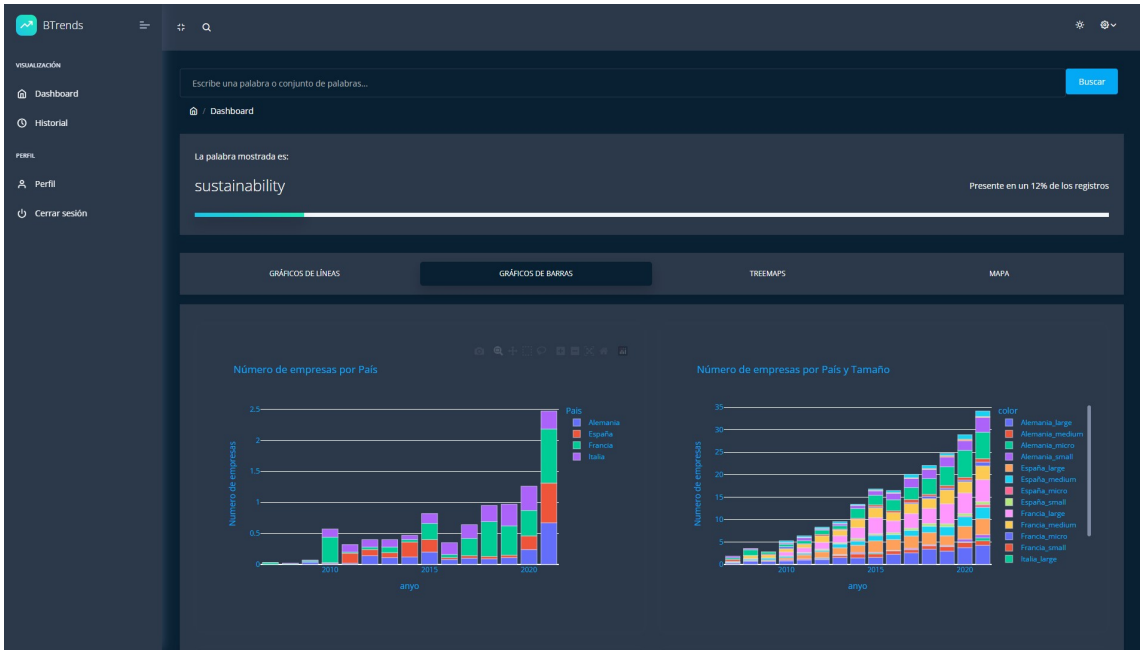


Figura B.12: Pantalla de visualización de los gráficos de barras. Modo oscuro

Figura B.13: Pantalla de visualización de los gráficos de barras. Modo claro

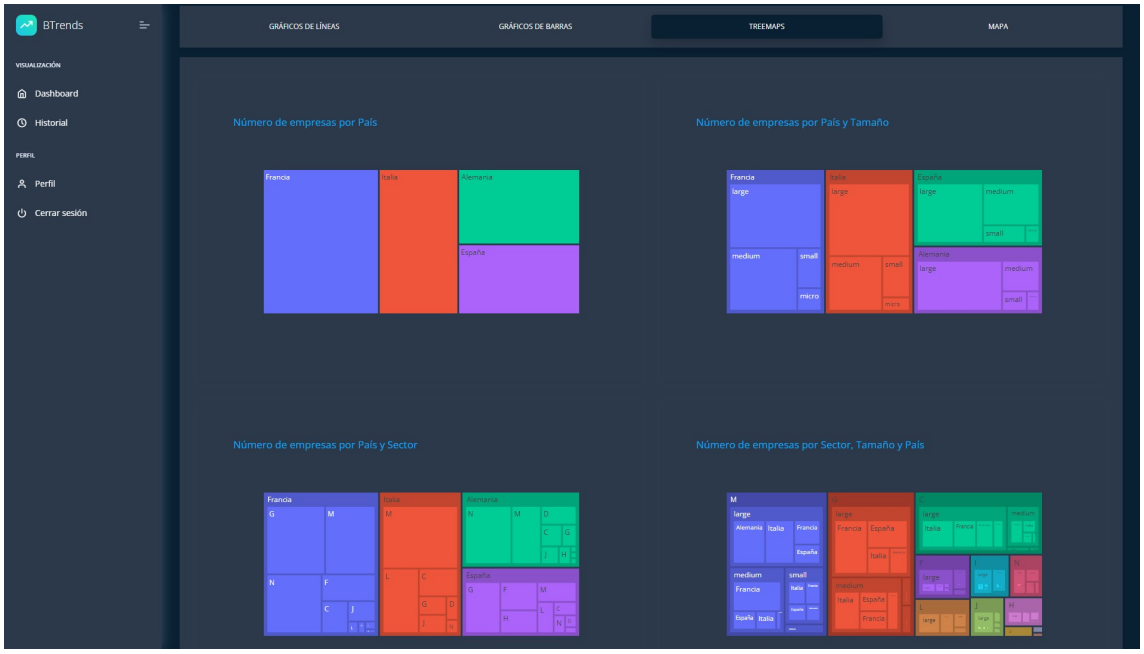


Figura B.14: Pantalla de visualización de los gráficos de barras. Modo oscuro

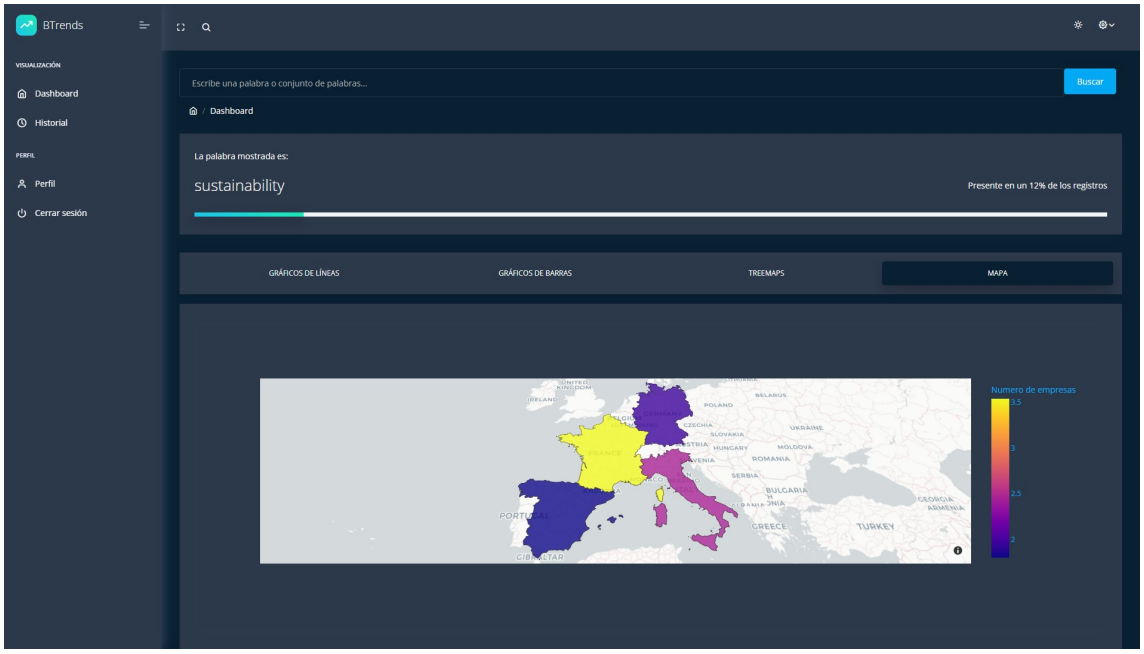


Figura B.15: Pantalla de visualización del mapa. Modo oscuro

B.2 Herramienta web en móviles

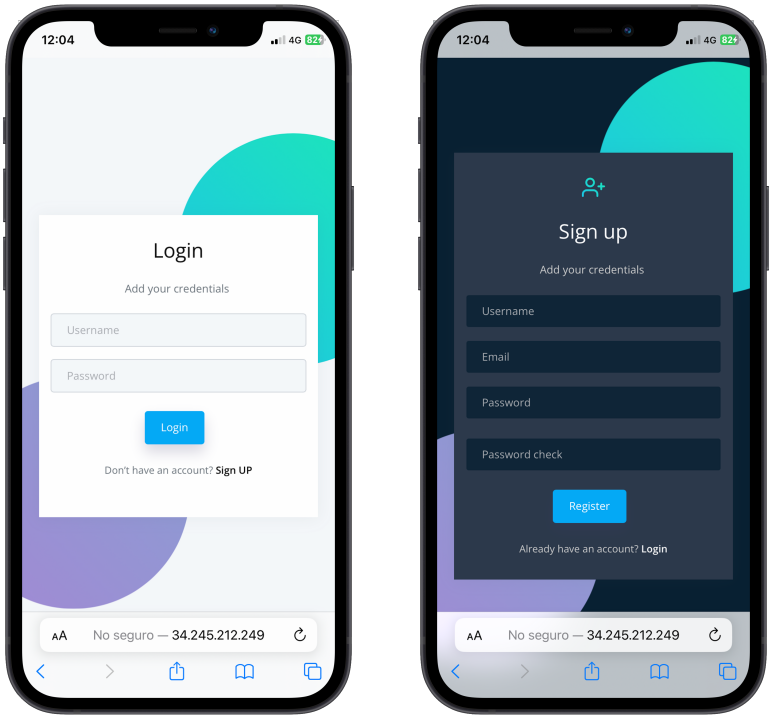


Figura B.16: Pantallas de inicio de sesión con modo claro activado (izquierda) y de registro con modo oscuro (derecha)

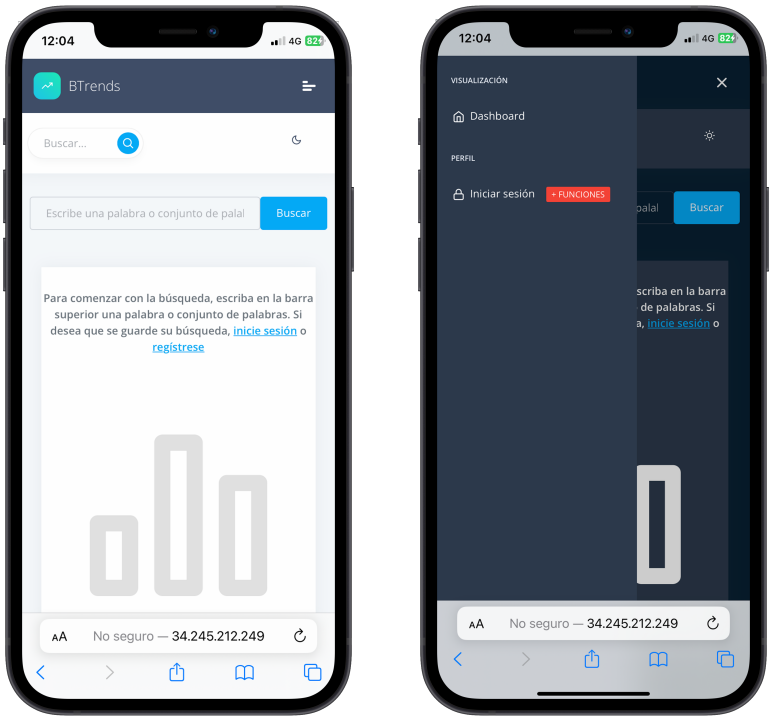


Figura B.17: Pantallas de inicio en modo claro (izquierda) y de la barra lateral de herramientas en modo oscuro (derecha)

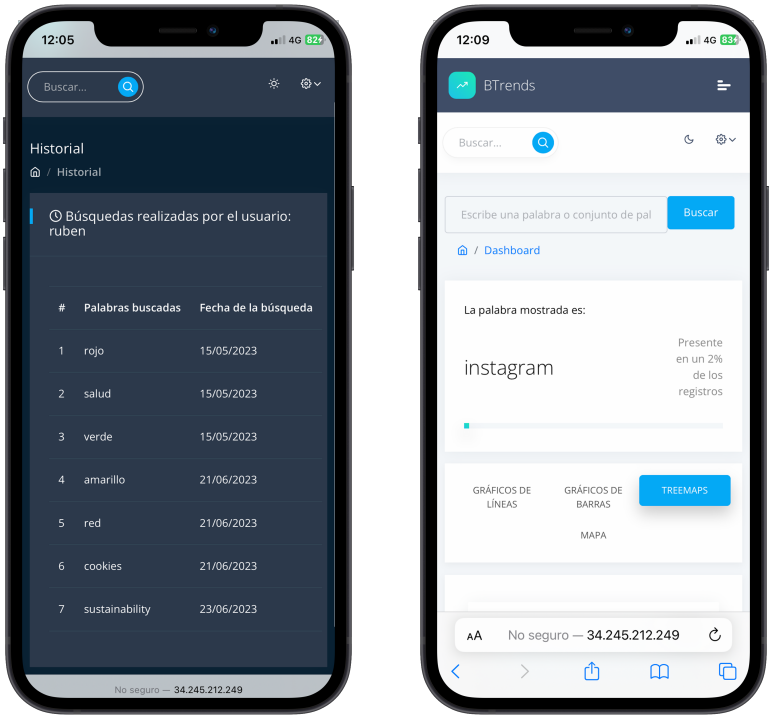


Figura B.18: Pantallas del historial de usuario en modo oscuro (izquierda) y de un ejemplo de búsqueda de la palabra "instagram" en el modo claro (derecha)

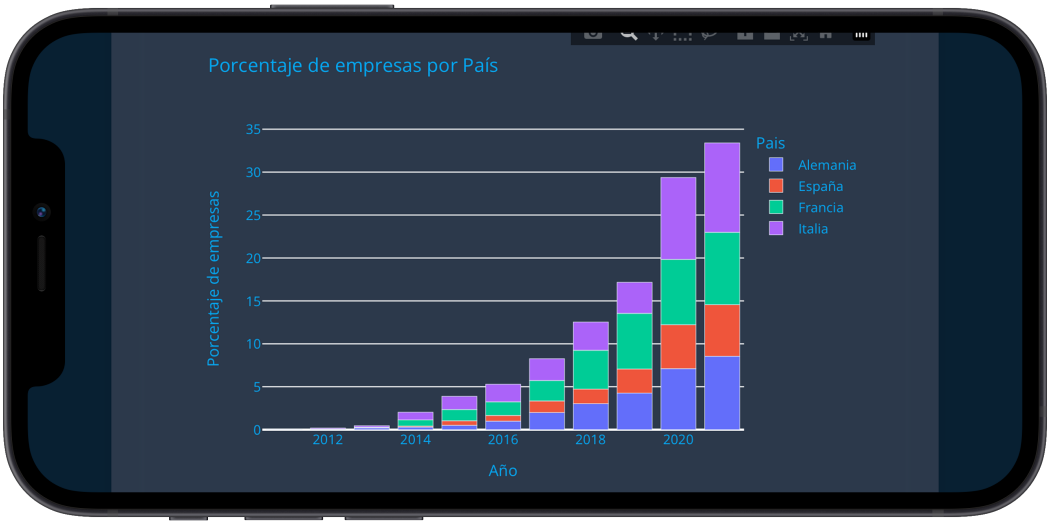


Figura B.19: Captura de los gráficos de barras

Figura B.20: Captura de los gráficos treemap

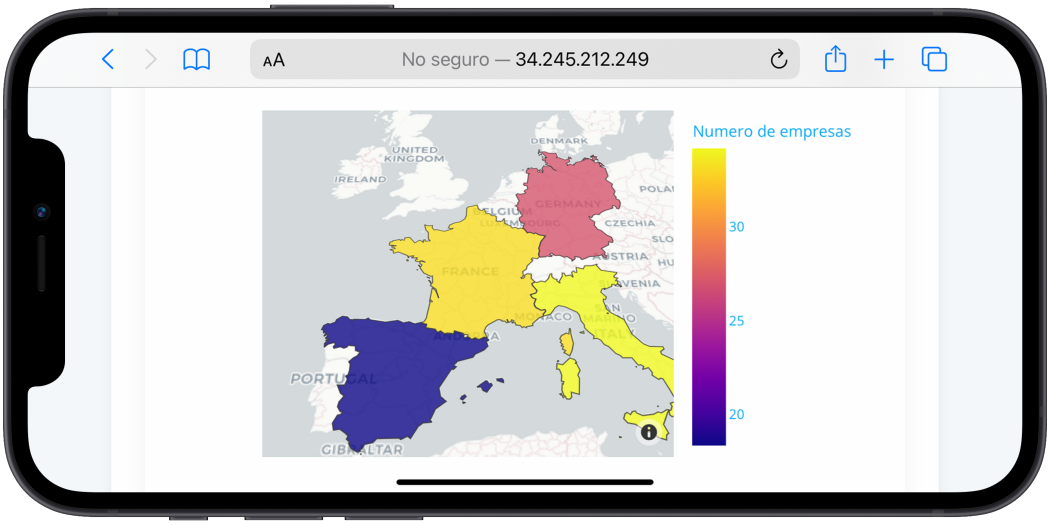


Figura B.21: Captura de la visualización del mapa

APÉNDICE C
Resumen de facturas

En este apéndice, se presentan las facturas reales correspondientes al coste de mantenimiento de los servidores AWS utilizados en el marco de este trabajo de fin de grado. Estas facturas ofrecen una visión detallada de los gastos incurridos durante el período de estudio y ofrecen una perspectiva transparente sobre los recursos económicos asignados para mantener la infraestructura. Para ello, se ha optado por incluir las tablas reales del desglose de las facturas.

Tabla C.1: Desglose de la factura real del servidor OpenSearch

Detail for Linked Account	
Amazon Elastic Compute Cloud	**USD 5,45**
Charges	USD 75,88
Savings Plan (Charges covered by Savings Plans)	-USD 71,38
VAT	USD 0,95
AWS Key Management Service	**USD 0,00**
Amazon Elastic File System	**USD 28,25**
Charges	USD 23,35
VAT	USD 4,90
AmazonCloudWatch	**USD 0,00**
Amazon Simple Notification Service	**USD 0,00**
Amazon Lightsail	**USD 0,00**
Amazon Simple Storage Service	**USD 0,06**
Charges	USD 0,05
VAT	USD 0,01
Amazon Simple Email Service	**USD 0,00**
Amazon CloudFront	**USD 0,00**
Amazon Relational Database Service	**USD 11,86**
Charges	USD 9,80
VAT	USD 2,06
Savings Plans for AWS Compute usage	**USD 0,00**
AWS Data Transfer	**USD 0,01**
Charges	USD 0,01
Amazon OpenSearch Service	**USD 73,47**
Charges	USD 60,72
VAT	USD 12,75
Total	**USD 119,10**

Tabla C.2: Desglose de la factura real del servidor que aloja la aplicación Django

Price per use	Total use	Total Price
$0,0126 per On Demand Linux t2.micro Instance Hour	720h	USD 9,072
$0,11 per GB-month	8GB	USD 0,88
Total	**USD 9,952**	