

Social Sentiment

Plataforma para la monitorización de redes sociales apoyada en análisis de sentimiento.



**VNiVERSIDAD
D SALAMANCA**

Proyecto de Fin de Grado

INGENIERÍA INFORMÁTICA

2021

Autor

Álvaro Martín López

Tutores

Roberto Casado Vara

Pablo Chamoso Santos

CERTIFICADO DE LOS TUTORES

Dr. Roberto Casado Vara, investigador en el Departamento de Informática y Automática y Dr. Pablo Chamoso Santos, profesor ayudante en el Departamento de Informática y Automática.

Hacen constar:

Que el trabajo titulado “Social Sentiment: Plataforma para la monitorización de redes sociales apoyada en análisis de sentimiento”, que se presenta, ha sido realizado por D. Álvaro Martín López para la superación de la asignatura Proyecto de Fin de Carrera del Grado en Ingeniería Informática de esta Universidad.

En Salamanca, a 6 de Julio de 2021.

Dr. Roberto Casado Vara

Dr. Pablo Chamoso Santos

RESUMEN

Este Trabajo de Fin de Grado tiene como meta el desarrollo de una plataforma de recolección y análisis de datos procedentes de redes sociales. Se pretende ofrecer un espacio donde los usuarios puedan consultar estadísticas y valoraciones de una red social concreta que no le pueda facilitar la web oficial de dicha red.

Así, el proyecto comienza con la investigación para determinar para qué red social se hará el caso de uso. La intención del proyecto es desarrollar una plataforma generalizada que haga de portal para varias redes sociales. Sin embargo, la casuística se ha reducido a una única red social (*Twitter*) por limitaciones temporales y de recursos. Una plataforma de la envergadura pretendida es prácticamente imposible de desarrollar en tan poco tiempo por un único trabajador, por lo que el TFG se centra en el caso de uso elegido y pretende desarrollar el proyecto de forma que sea fácil extenderlo y escalarlo.

Para la planificación temporal, dado el escaso tiempo y la imprevisibilidad del proyecto, se eligió la metodología *Scrum*. Esta metodología divide las fases del proyecto en *sprints*, haciendo “entregas” a menudo. Se ha elegido dada la flexibilidad que permite a la hora de implementar cambios.

Siguiendo el lenguaje de modelado *UML* se detallan los requisitos funcionales y no funcionales de la plataforma, así como los casos de uso de esta. Una vez se tuvo claro el diseño, se pensó y diseñó una arquitectura que satisfaga todos los objetivos de una forma simple y extensible.

Para implementar esta arquitectura se han utilizado principalmente los lenguajes *Python* y *JavaScript*, apoyados en una base de datos no relacional *MongoDB*. *Python* se eligió por su simplicidad y rapidez de implementación. La importancia de *JavaScript* se ve en la parte del cliente, pues se ha utilizado *Node.js* para implementar el *backend*. *JavaScript* es especialmente útil a la hora de responder a varias peticiones simultáneas de forma realmente cómoda: algo que le falta a *Python*. Por su parte, se ha elegido *MongoDB* sobre bases de datos relacionales como *MySQL* ya que se adapta mucho mejor a la estructura de los datos que persistiremos. Además, en nuestra base de datos apenas se harán modificaciones, solo creaciones y eliminaciones.

El funcionamiento general de la plataforma es el siguiente. Cuando un usuario se registra, debe proveer su nombre de usuario de la red social que desee analizar. Tras esto, la plataforma se encargará de recuperar periódicamente información acerca de su perfil, ofreciendo información acerca de las variaciones de seguidores, relevancia de sus *posts* en cuanto a los temas más relevantes de los que se hable o, como bien dice el nombre de la plataforma, el sentimiento u opinión de los comentarios de la gente en sus *posts* o perfil.

Para que la plataforma sea capaz de determinar dicho sentimiento, se pretende entrenar un modelo de Machine. Para ello se tiene que obtener primero un *dataset* de análisis de sentimiento en español. Así, la plataforma incorpora como servicio paralelo un bot de *Telegram* que recoge comentarios de redes sociales y se los muestra al usuario para que los catalogue en Positivo, Negativo o Neutro. Así, entre los datos recopilados y los ya disponibles en internet, se pretende construir un *dataset* en español de la mayor calidad posible. Es importante mencionar la dificultad del análisis de sentimiento en español al ser un idioma tan rico y variado.

Toda la documentación referente a cualquier apartado de este proyecto se reparte entre esta memoria y los anexos adjuntados.

PALABRAS CLAVE

En este apartado se citan las palabras clave presentes durante las distintas fases del proyecto:

Sentimiento, Inteligencia Artificial, Scrapper, MongoDB, Python, NodeJS, Twitter, Telegram.

ABSTRACT

This Final Degree Project aims to develop a platform for collecting and analyzing data de social networks. It is intended to provide a space where users can consult statistics and ratings of a particular social network that cannot be provided by the official website of that network.

Thus, the project begins with research to determine for which social network the use case will be made. The intention of the project is to develop a generalized platform that will serve as a portal for several social networks. However, the use case has been narrowed down to a single social network (Twitter) due to time and resource constraints. A platform of the intended scope is practically impossible to develop in such a short time by a single worker, so the project focuses on the chosen use case and aims to develop the project in a way that it is easy to extend and scale it.

For the time planning, given the short time and the unpredictability of the project, the Scrum methodology was chosen. This methodology divides the project phases into sprints, making "deliveries" often. It was chosen because of the flexibility it allows when implementing changes.

Following the UML modeling language, the functional and non-functional requirements of the platform are detailed, as well as its use cases. Once the design was clear, an architecture that satisfies all the objectives in a simple and extensible way was thought and designed.

To implement this architecture, Python and JavaScript languages were mainly used, supported by a non-relational database MongoDB. Python was chosen for its simplicity and speed of implementation. The importance of JavaScript is seen on the client side, as Node.js has been used to implement the backend. JavaScript is especially useful when it comes to responding to several simultaneous requests in a really convenient way: something that Python lacks. For its part, MongoDB has been chosen over relational databases such as MySQL as it is much better suited to the structure of the data we will be persisting. In addition, there will be hardly any modifications to our database, only creations and deletions.

The general operation of the platform is as follows. When a user registers, they must provide their username of the social network they wish to analyze. After this, the platform will periodically retrieve information about their profile, providing information about the variations in followers, relevance of their posts in terms of the most relevant topics being talked about or, as the name of the platform says, the sentiment or opinion of people's comments on their posts or profile.

In order for the platform to be able to determine such sentiment, it is intended to train a Machine model. To do this, a sentiment analysis dataset must first be obtained in Spanish. Thus, the platform incorporates as a parallel service a Telegram bot that collects comments de social networks and shows them to the user so that he can catalogue them as Positive, Negative or Neutral. Thus, between the data collected and those already available on the Internet, the aim is to build a dataset in Spanish of the highest possible quality. It is important to mention the difficulty of sentiment analysis in Spanish as it is such a rich and varied language.

All the documentation related to any part of this project is included in this report and the attached annexes.

KEYWORDS

This section cites the keywords present during the different phases of the project:

Sentiment, Artificial Intelligence, Scrapper, MongoDB, Python, NodeJS, Twitter, Telegram.

TABLA DE CONTENIDO

Certificado de los tutores	3
Resumen.....	5
Palabras clave.....	6
Abstract	7
Keywords.....	8
Tabla de ilustraciones.....	14
Tabla de tablas	17
Tabla de ecuaciones	18
1 Introducción	19
1.1 Las redes sociales	20
1.2 Análisis de sentimiento en twitter	21
2 Objetivos del sistema	23
2.1 Objetivos funcionales.....	23
2.2 Objetivos no funcionales.....	31
2.3 Objetivos personales.....	34
3 Conceptos teóricos.....	35
3.1 API-REST	35
3.2 Base de datos no relacional	35
3.3 Análisis de sentimiento	36
3.4 Arquitectura cliente-servidor	37
3.5 Telegram.....	37
4 Técnicas y herramientas.....	39
4.1 Módulo de utilidad.....	39
4.1.1 Visual Studio Code.....	40
4.1.2 Entornos utilizados.....	40

4.1.3	Git	43
4.1.4	Variables de entorno	43
4.1.5	Logging	46
4.1.6	Formatos	48
4.2	Técnicas y herramientas del módulo de la API	49
4.2.1	Flask.....	49
4.2.2	Flask-Cache.....	50
4.2.3	Postman.....	50
4.3	Técnicas y herramientas del módulo de ingesta.....	51
4.3.1	Twitter API.....	51
4.3.2	OAuth	52
4.3.3	Tweepy	53
4.4	Técnicas y herramientas del módulo de persistencia	53
4.4.1	MongoDB.....	53
4.4.2	PyMongo	55
4.4.3	Mongoose.....	55
4.4.4	MongoDB Compass	56
4.5	Técnicas y herramientas del módulo de análisis.....	57
4.5.1	Métricas seguidas.....	57
4.5.2	Analizador de sentimiento	60
4.6	Técnicas y herramientas del módulo de visualización	65
4.6.1	Frontend.....	65
4.6.2	Backend	66
4.7	Técnicas y herramientas del módulo supervisor.....	68
4.7.1	Request.....	68
4.7.2	Crontab.....	68
5	Aspectos relevantes	69
5.1	Planificación temporal.....	69

5.2	Arquitectura y diseño.....	71
5.3	Bot de Telegram.....	73
5.3.1	Diseño.....	73
5.3.2	Implementación.....	80
5.4	Despliegue.....	83
5.4.1	Pasos de despliegue (comunes para los tres entornos).....	83
5.5	Gestión de versiones.....	85
5.6	Pruebas realizadas.....	86
5.6.1	Bot de Telegram.....	86
5.6.2	Plataforma Social Sentiment.....	87
5.6.3	Datos de prueba.....	89
6	Conclusiones.....	92
6.1	Funcionalidad del sistema.....	92
6.2	Conclusiones.....	92
6.3	Líneas de trabajo futuro.....	93
7	Bibliografía.....	95

TABLA DE ILUSTRACIONES

Ilustración 1 Logo de la plataforma.....	19
Ilustración 2 Arquitectura del módulo de la API.....	24
Ilustración 3 Arquitectura del módulo de ingesta.....	26
Ilustración 4 Arquitectura del módulo de persistencia.....	27
Ilustración 5 Arquitectura del módulo de análisis	29
Ilustración 6 Arquitectura del módulo de visualización.....	30
Ilustración 7 Arquitectura del módulo supervisor	31
Ilustración 8 Creación de entorno virtual con Pipenv.....	45
Ilustración 9 Utilidades básicas de pipenv	46
Ilustración 10 Ejemplo de configuración de log de un módulo.....	46
Ilustración 11 Ejemplo de formatter del log	47
Ilustración 12 Ejemplo de manejadora del log.....	47
Ilustración 13 Ejemplos de logger	47
Ilustración 14 Get Request en Postman.....	50
Ilustración 15 Resultado de la petición GET.....	50
Ilustración 16 Vista para nombrar la aplicación en el panel de desarrollador de Twitter.....	51
Ilustración 17 Propiedades de la aplicación Twitter	52
Ilustración 18 Rótulo de MongoDB.....	53
Ilustración 19 Estructura de la base de datos	54
Ilustración 20 Uso básico de PyMongo	55
Ilustración 21 Conexión con base de datos vía Mongoose	56
Ilustración 22 Esquema e interfaz de Usuario mediante Mongoose	56
Ilustración 23 Interfaz de MongoDB Compass.....	57
Ilustración 24 Rótulo de SpaCy.....	61
Ilustración 25 Esquema de un Random Forest Classifier	64
Ilustración 26 Liberías de fuentes e iconos más utilizados en Internet en 2020	66

Ilustración 27 Utilidades básicas de crontab.....	68
Ilustración 28 Formato de crontab.....	68
Ilustración 29 Arquitectura del proyecto	72
Ilustración 30 Arquitectura del bot de Telegram	72
Ilustración 31 Diagrama de estados de la manejadora conversacional del bot	75
Ilustración 32 BOT: Pantalla de inicio.....	76
Ilustración 33 BOT: Ejemplo de votación	77
Ilustración 34 BOT: Panel de administrador	78
Ilustración 35 BOT: Gestión de oraciones inválidas	78
Ilustración 36 BOT: Gestión de oraciones etiquetadas.....	79
Ilustración 37 BOT: Información de la base de datos	80
Ilustración 38 Logo de The Botfather	81
Ilustración 39 Declaración de los diferentes estados	82
Ilustración 40 Arranque del bot	83
Ilustración 41 Instalación de MongoDB	84
Ilustración 42 Habilitación del servicio de mongod	84
Ilustración 43 Despliegue de la API	84
Ilustración 44 Despliegue de la página.....	85
Ilustración 45 Ejemplo de estructura de documento Mejores Tweets en base de datos. Visto a través de MongoDB Compass.....	89
Ilustración 46 Generación de fechas.....	90
Ilustración 47 Estructura del documento.....	90
Ilustración 48 Valores origen de los parámetros	90
Ilustración 49 Generación de datos diarios.....	90
Ilustración 50 Control de mínimos	91
Ilustración 51 Ejemplos de datos generados	91

TABLA DE TABLAS

Tabla 1 OBJ - 001	Gestión de la API	24
Tabla 2 OBJ - 002	Gestión del módulo de ingesta	25
Tabla 3 OBJ - 003	Gestión del módulo de persistencia.....	26
Tabla 4 OBJ - 004	Gestión del módulo de análisis	28
Tabla 5 OBJ - 005	Gestión del módulo de visualización.....	29
Tabla 6 OBJ - 006	Gestión del módulo supervisor	30
Tabla 7 NFR - 001	Modularidad.....	32
Tabla 8 NFR - 002	Escalabilidad.....	32
Tabla 9 NFR - 003	Extensibilidad	32
Tabla 10 NFR - 004	Privacidad.....	32
Tabla 11 NFR - 005	Formatos estándares de datos	33
Tabla 12 NFR – 006	Mantenibilidad	33
Tabla 13 NFR - 007	Uso de un entorno de Cloud Computing	33
Tabla 14 NFR - 008	Soporte.....	33
Tabla 15 NFR - 009	Simplicidad.....	34
Tabla 16 NFR - 010	Uso de un entorno de pruebas	34
Tabla 17	Comparativa nomenclatura bases de datos	36
Tabla 18	Especificaciones instancia EC2.....	42
Tabla 19	Variables de entorno	44
Tabla 20	Categorías de perfil en función de los seguidores.....	58
Tabla 21	Filtro de sentimiento aplicado a la puntuación	59
Tabla 22	Anexos	69
Tabla 23	Product Backlog.....	71
Tabla 24	Pruebas del bot de Telegram.....	86
Tabla 25	Pruebas de la web de Social Sentiment.....	87
Tabla 26	Pruebas de la API.....	87

Tabla 27 Pruebas del módulo de ingesta	88
Tabla 28 Pruebas del módulo de persistencia	88
Tabla 29 Pruebas del módulo de análisis	88
Tabla 30 Pruebas varias.....	89

TABLA DE ECUACIONES

Ecuación 1 Puntuación inicial.....	58
Ecuación 2 Puntuación intermedia	59

1 INTRODUCCIÓN

En el presente documento se expone la información necesaria para la memoria del Trabajo de Fin de Grado titulado “Social Sentiment: Plataforma para la monitorización de redes sociales apoyada en Inteligencia Artificial”, desarrollado por el alumno Álvaro Martín López durante cinco meses, de febrero de 2021 a junio de 2021.



Ilustración 1 Logo de la plataforma

Este Trabajo de Fin de Grado se ha llevado a cabo en colaboración con el grupo de investigación BISITE (Bioinformática, Sistemas Inteligentes y Teconología Educativa) de la universidad de Salamanca.

El constante crecimiento de las redes sociales implica que cada vez más sectores, entre ellos el de la informática, apuesten por ellas. Cada vez aparecen más aplicaciones que complementan el uso de redes sociales, como pueden ser *Hootsuite* [1] (que permite gestionar numerosos perfiles desde una sola plataforma), *Streamdn* [2] (para interactuar con localizaciones específicas del mundo) o *TweetAlarm* (que permite especificar frecuencias para alertas de tweets o menciones en Twitter). Así, en este trabajo se ha llevado a cabo la tarea de revisar el estado del arte para unir las redes sociales con una de las disciplinas que más crecimiento ha tenido recientemente en el mundo de la informática, la Inteligencia Artificial.

Así, las conclusiones y resultados obtenidos son pertinentes y permiten generar conocimiento entre los interesados. Este documento presenta los siguientes contenidos:

- Objetivos del Trabajo de Fin de Grado
- Conceptos teóricos
- Técnicas y herramientas

- Aspectos relevantes del desarrollo
- Conclusiones y líneas de trabajo futuras
- Bibliografía

1.1 LAS REDES SOCIALES

Las redes sociales forman una parte innegable del día a día de todos nosotros, llegando a ser incluso una parte crucial de la economía. Se mueven enormes cantidades de dinero entre patrocinios, publicidad, donaciones..., siendo los perfiles de mayor éxito aquellos que obtienen el mayor beneficio. Las redes sociales han evolucionado, llamando la atención de un abanico cada vez más grande de sectores, entre los que encontramos la informática. Son cada vez más numerosas y variadas las aplicaciones que complementan nuestro uso de las redes sociales, especialmente si se trata de crecer dentro de las mismas.

Para encuadrarnos en esta temática, es necesario conocer el concepto de capital social [3]. Éste consiste en el conjunto de recursos que una persona tiene debido a las relaciones que mantiene con otras. El capital social puede ser de tipo real o virtual. Este último es el que nos atañe. A través del mundo virtual, la información fluye más rápidamente, y las relaciones personales y las acciones conjuntas se forman y organizan de forma dinámica. Es por esta necesidad de relacionarse y de adquirir mayor capital social que las redes sociales adquieren cada vez más relevancia, ya que son el medio idóneo para conseguir dichos objetivos. Los “me gusta” y las opiniones que los demás tienen de nosotros en las redes sociales cada vez nos influye más y puede provocar incluso que cambiemos el contenido que subimos a éstas.

Otro punto importante que tratar respecto a las redes sociales es la economía. Debido al carácter global que posee actualmente la economía, el intercambio de información que ofrecen las redes sociales resulta muy útil. Las empresas necesitan aprovechar la transferencia de conocimiento, tanto para su mejora competitiva como para la innovación de sus productos o servicios [4]. Este hecho se ve ejemplificado principalmente en la publicidad y el marketing. Debido a la rapidez con la que las personas pueden volverse muy conocidas en las redes sociales y el gran poder de comunicación e influencia que poseen, estas empresas encuentran el medio perfecto para potenciar sus productos.

Debido a lo anteriormente expuesto, las redes sociales parecen un medio innovador y prometedor en numerosos ámbitos de la sociedad, pudiendo acelerar ciertos procesos y haciendo más efectiva la transmisión de la información.

1.2 ANÁLISIS DE SENTIMIENTO EN TWITTER

El análisis de sentimientos en *Twitter* es un área especializada dentro del análisis de sentimientos, un tema destacado de investigación en el campo de la lingüística computacional. Los enfoques del análisis de sentimientos identifican opiniones expresadas en texto gracias a métodos automatizados.

El análisis de sentimientos se ha llevado a cabo en una variedad de géneros de comunicación, incluidos los medios profesionales como los artículos de noticias [5], redes sociales las reseñas de productos [6], foros web [7] o *Facebook* [8]. El crecimiento de la investigación sobre el análisis de los sentimientos ha seguido al de los medios sociales, ya que los investigadores y las empresas persiguen el objetivo de mejorar la calidad de vida. Por ello, los investigadores y las empresas buscan las opiniones de grandes poblaciones de usuarios. Las tareas de análisis de sentimientos incluyen la clasificación de la polaridad del sentimiento expresado en el texto (por ejemplo, positivo, negativo, neutro), la identificación del objetivo o tema del sentimiento, la identificación del titular de la opinión e identificar el sentimiento de varios aspectos de un tema, producto u organización [9].

Los enfoques de aprendizaje automático desempeñan un papel importante en la extracción de opiniones. Por lo general, el análisis de sentimiento a nivel de documento y de frase puede formularse como un problema de clasificación que determinan si se expresa un sentimiento positivo o negativo. Los clasificadores están capacitados para determinar las polaridades de los textos que se les proporcionen. El clasificador *Naive Bayes*, el clasificador de entropía máxima y la máquina vectorial de apoyo (*SVM*) [10] son los modelos más utilizados.

Sin embargo, no es fácil satisfacer el requisito de tener un corpus correctamente etiquetado, especialmente en situaciones de dominios e idiomas cruzados. En la situación de dominio cruzado, dado que existen diferencias entre los distintos dominios, el clasificador entrenado de un dominio no siempre logra un rendimiento comparable en otro. Es más grave en la situación interlingüística que en la situación de dominio cruzado ya que los clasificadores entrenados no pueden aplicarse a los textos en otro idioma directamente debido a las diferencias lingüísticas. Los métodos semisupervisados, que entrenan a los clasificadores corpus etiquetados y no etiquetados en diferentes dominios e idiomas, se desarrollan para hacer frente a la falta de corpúsculos anotados. En la minería de opinión de nivel fino, se requieren más esfuerzos para la extracción del sentimiento sus relaciones. Además, los corpus etiquetados y de calidad son difíciles de obtener. Se han propuesto varios métodos no supervisados basados en la Asignación de Dirichlet Latentes (*LDA*) [11] para liberar la dependencia de los cuerpos anotados.

2 OBJETIVOS DEL SISTEMA

Como se especifica en la memoria, en este Trabajo de Fin de Grado se ha utilizado la metodología ágil de *Scrum* [12]. Por tanto, el desarrollo de la plataforma se ha dividido en *sprints*, consistiendo el primero en la fase de iniciación. Dicha fase consiste en definir claramente los objetivos de la plataforma, así como sus prioridades en el desarrollo. Posteriormente, se definen los requisitos del sistema y las necesidades que pueda tener el cliente potencial.

La especificación de requisitos actúa como medio de comunicación entre las partes del proyecto, siendo estas: clientes, usuarios, ingenieros y desarrolladores. Así, se recogen las soluciones que aporte el equipo de desarrollo junto a las necesidades de clientes y usuarios.

El objetivo de este apartado es especificar los objetivos de la plataforma para satisfacer necesidades de los usuarios de redes sociales. Estas necesidades se establecieron en la investigación previa en base al estado del arte de las aplicaciones actuales, incluyendo las estadísticas públicas que ofrecen las propias redes sociales.

Primero detallaremos los objetivos funcionales del sistema, puesto que son los principales. A continuación, se listarán los objetivos no funcionales requeridos y, por último, los objetivos personales del proyecto.

2.1 OBJETIVOS FUNCIONALES

Los objetivos funcionales son aquellos directamente derivados de las necesidades que pueden plantear los posibles usuarios y cómo la plataforma puede satisfacerlas.

OBJ - 001	Gestión de la API
Versión	V 1.0
Autores	Álvaro Martín López
Descripción	El sistema debe incorporar una API central que gestione el acceso al sistema
Subobjetivos	Ninguno
Urgencia	Alta
Estado	Validado
Estabilidad	Alta

Comentarios	Ninguno
-------------	---------

Tabla 1 OBJ - 001 Gestión de la API

La parte troncal de este sistema es la API. Esta debe ser capaz de acceder a los distintos módulos, de forma que estos sean completamente independientes entre sí. De esta manera, se facilitará bastante el mantenimiento de la plataforma, pudiendo buscar alternativas en caso de que fuera necesario un cambio grande en alguno de los módulos [13] [14].

Además, la API debe ofrecer suficientes *endpoints* para satisfacer las necesidades de la página web, haciendo las labores de controlador central y gestionando los errores que pudiesen producirse.

La API debe ser capaz también de ofrecer *endpoints* para los distintos módulos externos que quisieran hacer uso de la plataforma, como son el supervisor o el bot de *Telegram*.

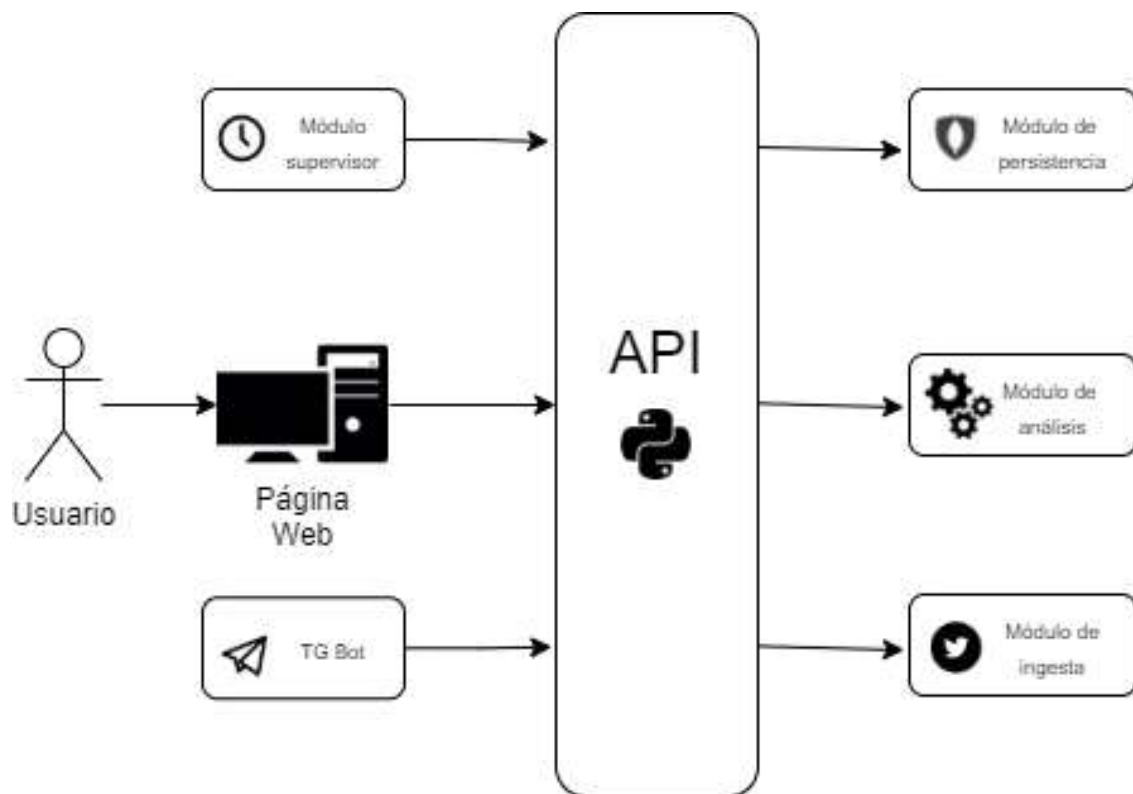


Ilustración 2 Arquitectura del módulo de la API

OBJ - 002	Gestión del módulo de ingesta
Versión	V 1.0
Autores	Álvaro Martín López
Descripción	El sistema debe ser capaz de gestionar la obtención de datos de Twitter.
Subobjetivos	Ninguno
Urgencia	Alta
Estado	Validado
Estabilidad	Alta
Comentarios	Ninguno

Tabla 2 OBJ - 002 Gestión del módulo de ingesta

Para poder analizar las estadísticas de un perfil es necesaria primero una forma de obtenerlas. Así, es imprescindible un módulo capaz de extraer la información necesaria de Twitter. Entre la información recabada encontramos:

- Información referente al perfil del usuario, como su descripción, enlaces a otras redes, posible verificación de perfil...
- Registro de tweets del usuario, de forma que se pueda monitorizar aquellos de mayor éxito y establecer relaciones.
- Respuestas a los tweets del usuario, puesto que son la fuente directa de opinión hacia el usuario. La parte de la inteligencia artificial se dedicará a trabajar con estas respuestas.
- Los temas más mencionados, *Trending Topic*, que servirán para las distintas métricas del módulo de análisis.

Es pertinente recalcar que se necesitan ciertos recursos para calcular toda la información en tiempo real. Debido a las limitaciones de hardware con las que se cuenta, algunas funciones del módulo de ingesta quedarán reservadas para ciertos momentos del día, obteniendo dicha información de la base de datos.

Para obtener estos datos, *Twitter* ofrece una API con la que se podrá autenticar la aplicación y los usuarios podrán vincular su cuenta a la plataforma. Sin embargo, en lugar de hacer peticiones directamente a la API, se buscará una librería de Python que facilite el acceso a la misma [15].

En el apartado de Técnicas y herramientas se explica el proceso necesario para obtener las claves que se utilizarán para autenticar la plataforma.

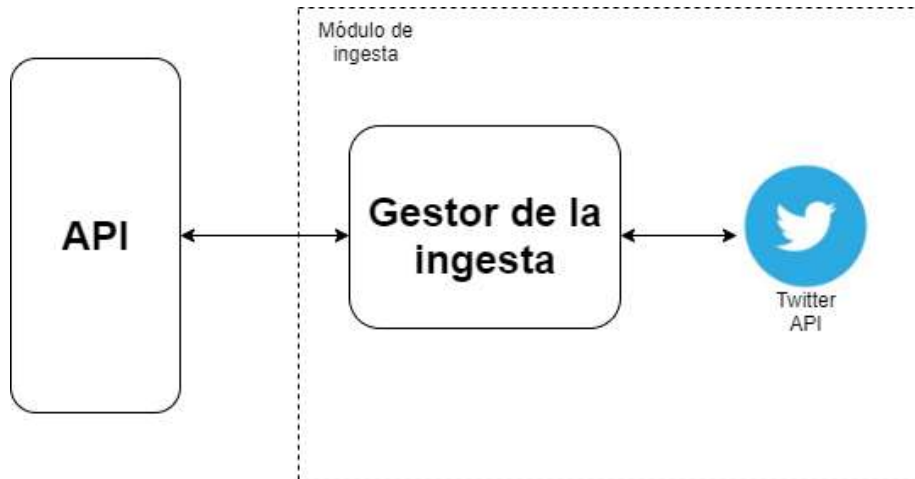


Ilustración 3 Arquitectura del módulo de ingesta

OBJ - 003	Gestión del módulo de persistencia
Versión	V 1.0
Autores	Álvaro Martín López
Descripción	El sistema debe ser capaz de gestionar una base de datos que almacene tanto los usuarios como la información necesaria para los análisis.
Subobjetivos	Ninguno
Urgencia	Alta
Estado	Validado
Estabilidad	Alta
Comentarios	Ninguno

Tabla 3 OBJ - 003 Gestión del módulo de persistencia

Es necesario almacenar en una base de datos la información obtenida mediante el módulo de ingesta, de manera que se pueda mostrar al usuario la evolución de su perfil. Además, algunas métricas del análisis precisan también de los históricos de la base de datos.

Sumado a la información de las cuentas de Twitter, es necesario almacenar también información de los usuarios referente a la plataforma web, como pueden ser su correo electrónico, contraseña o identificadores de las diversas redes sociales.

Para almacenar todos estos datos se utilizará una base de datos no relacional. Se ha elegido este tipo frente a alternativas como *MySQL*, *MariaDB* u otras bases de datos relacionales ya que es necesario cierto grado de velocidad en las lecturas y escrituras para mitigar el tiempo de respuesta de las técnicas de *scraping* y análisis de sentimientos [16]. En el caso de Social Sentiment, se ha escogido *MongoDB* [17].

Para utilizar la base de datos se implementa un módulo en Python que actuará de intermediario, ofreciendo distintos *endpoints* a la API para facilitar el manejo de los datos.

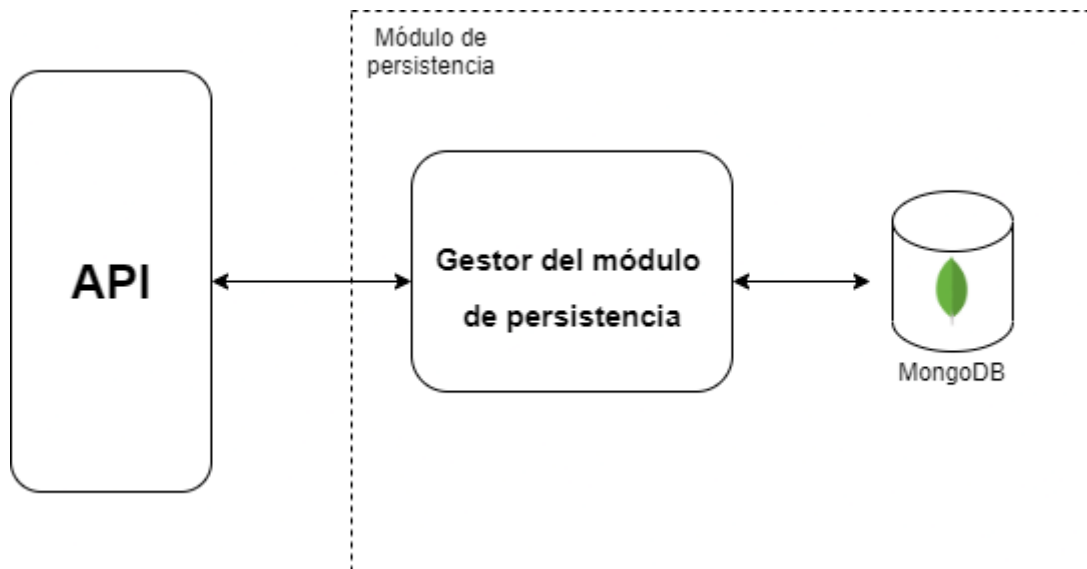


Ilustración 4 Arquitectura del módulo de persistencia

OBJ - 004	Gestión del módulo de análisis
Versión	V 1.0
Autores	Álvaro Martín López
Descripción	El sistema debe ser capaz de analizar los datos de los usuarios, proporcionando los resultados del análisis de sentimientos y otras métricas
Subobjetivos	Ninguno
Urgencia	Alta
Estado	Validado
Estabilidad	Alta
Comentarios	Ninguno

Tabla 4 OBJ - 004 Gestión del módulo de análisis

El sistema debe incorporar un módulo al que proporcionar datos, ya sea persistidos o en tiempo real, que devuelva diversos resultados que informen al usuario del estado de su cuenta.

Este módulo debe proporcionar a la API los resultados de las distintas métricas en un formato específico, informando adecuadamente en caso de que se produzca algún error, de forma que la API pueda responder en función.

Además, debe incorporar las funcionalidades necesarias para realizar el análisis de sentimientos, que puede implementarse en un submódulo dado el propósito más específico respecto al resto del módulo.

El analizador, por su parte, debe tener una forma clara de acceder al modelo entrenado para el análisis de sentimientos, así como el resto de los ficheros necesarios para su correcto funcionamiento.

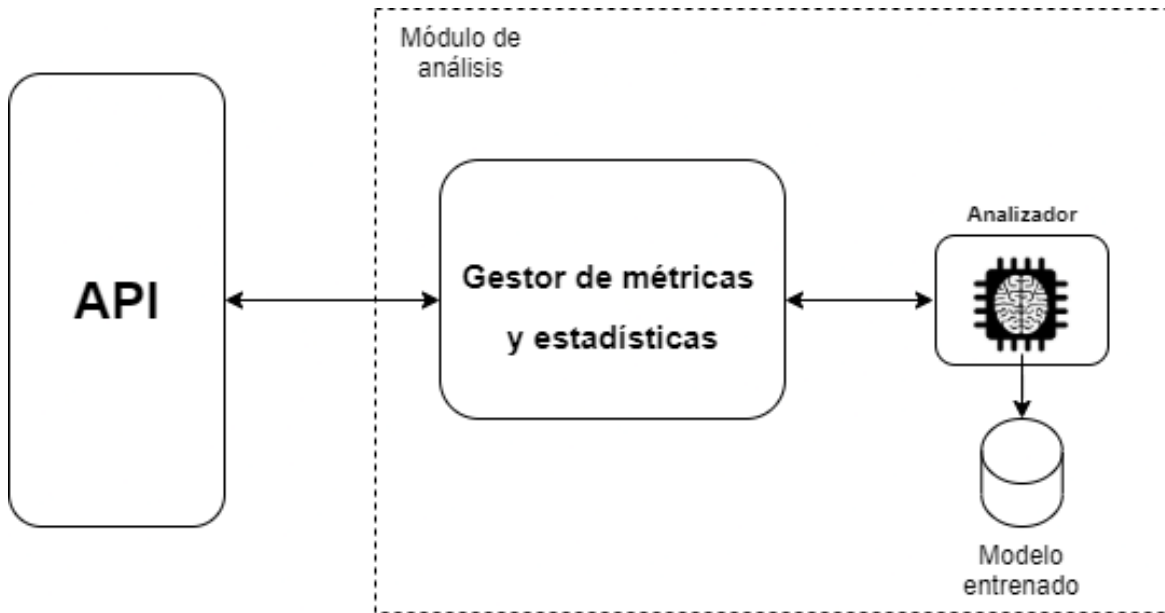


Ilustración 5 Arquitectura del módulo de análisis

OBJ - 005	Gestión del módulo de visualización
Versión	V 1.0
Autores	Álvaro Martín López
Descripción	El sistema debe ofrecer a los usuarios una forma de visualizar su información
Subobjetivos	Ninguno
Urgencia	Alta
Estado	Validado
Estabilidad	Alta
Comentarios	Ninguno

Tabla 5 OBJ - 005 Gestión del módulo de visualización

Es necesario que los usuarios puedan visualizar de alguna manera la información recabada en los análisis. Para ello, se debe implementar una plataforma web que ofrezca una serie de paneles en los que el usuario pueda ver el estado actual de sus perfiles, así como un histórico de los datos recabados.

La plataforma debe incorporar también un sistema de usuarios que permita registrarse e iniciar sesión adecuadamente. También debe ser capaz de controlar la vinculación de redes. Es decir, que el perfil de Twitter que introduzca el usuario le pertenezca realmente.

El *backend* hará uso de la API para obtener la información que mostrará al usuario. Si esto supusiera muchos recursos, se puede contemplar obtener ciertos datos mediante el módulo de persistencia en lugar de dinámicamente.

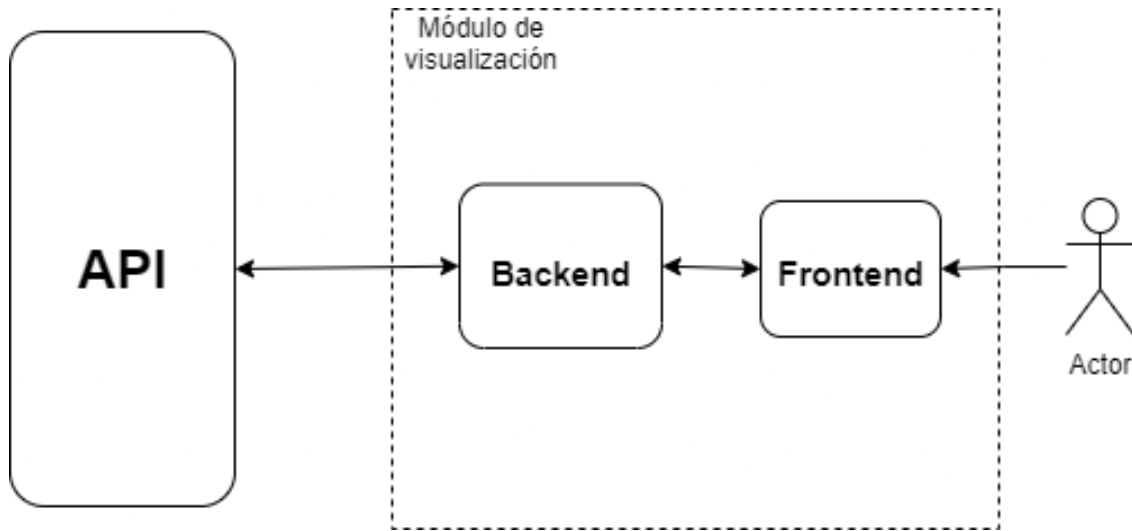


Ilustración 6 Arquitectura del módulo de visualización

OBJ - 006	Gestión del módulo supervisor
Versión	V 1.0
Autores	Álvaro Martín López
Descripción	El sistema debe ser capaz de actualizar periódicamente la información de los usuarios
Subobjetivos	Ninguno
Urgencia	Media
Estado	Validado
Estabilidad	Alta
Comentarios	Ninguno

Tabla 6 OBJ - 006 Gestión del módulo supervisor

El sistema debe contar con un módulo supervisor controlable a través de cualquier mecanismo de planificación de tareas.

De esta manera, el módulo supervisor será encargado de hacer las llamadas necesarias a la API para actualizar los perfiles de los usuarios. Esto se llevará a cabo mediante un módulo externo que haga llamadas a la API para facilitar el manejo de las actualizaciones periódicas, separándolo del resto del sistema. Sin embargo, en este trabajo todos los módulos se han desplegado en el mismo nodo.

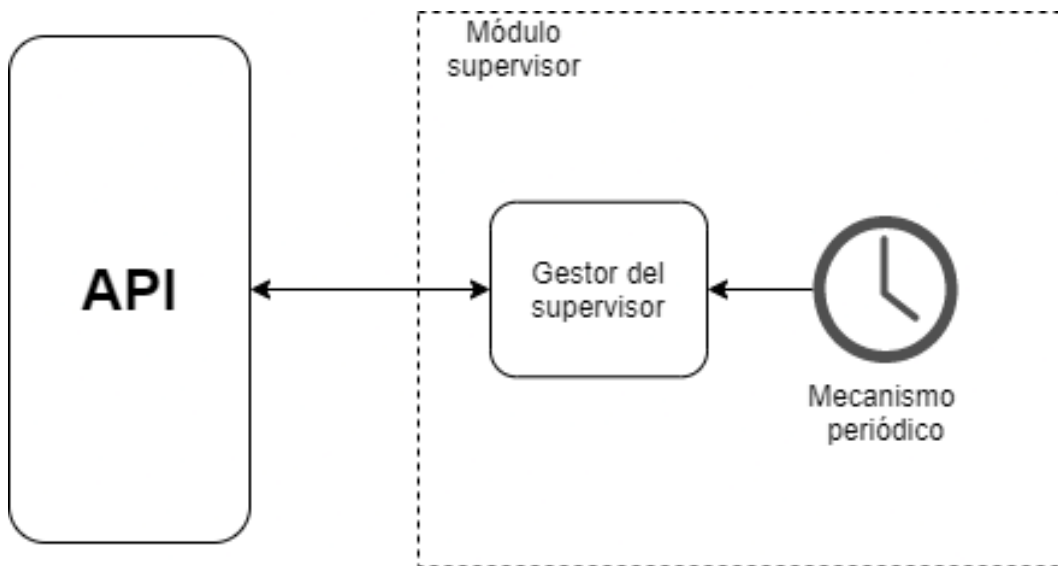


Ilustración 7 Arquitectura del módulo supervisor

2.2 OBJETIVOS NO FUNCIONALES

Una vez se ha definido cada objetivo principal del sistema, se deben establecer las condiciones y restricciones del sistema, esto es, requisitos no funcionales. A continuación, se muestra un listado de los mismos:

NFR - 001		Modularidad	
Versión		V 1.0	
Autores		Álvaro Martín López	
Descripción		El sistema debe estar separado en módulos identificables e independientes que faciliten el mantenimiento del sistema.	

Tabla 7 NFR - 001 Modularidad

NFR - 002		Escalabilidad	
Versión		V 1.0	
Autores		Álvaro Martín López	
Descripción		El diseño del sistema debe permitir modificaciones en su carga de trabajo sin afectar al funcionamiento.	

Tabla 8 NFR - 002 Escalabilidad

NFR - 003		Extensibilidad	
Versión		V 1.0	
Autores		Álvaro Martín López	
Descripción		El diseño del sistema debe permitir la implementación de funcionalidades adicionales de forma sencilla.	

Tabla 9 NFR - 003 Extensibilidad

NFR - 004		Privacidad	
Versión		V 1.0	
Autores		Álvaro Martín López	
Descripción		Los datos personales de los usuarios se utilizarán únicamente para el cálculo de estadísticas que ellos mismos verán e información sensible, como pueden ser contraseñas, debe estar correctamente encriptada.	

Tabla 10 NFR - 004 Privacidad

NFR - 005	Formatos estándares de datos
Versión	V 1.0
Autores	Álvaro Martín López
Descripción	El sistema debe utilizar un formato constante para los datos, como puede ser <i>JSON</i> , de forma que los módulos entreguen la información correctamente procesada

Tabla 11 NFR - 005 Formatos estándares de datos

NFR – 006	Mantenibilidad
Versión	V 1.0
Autores	Álvaro Martín López
Descripción	El sistema debe tener una arquitectura que facilite su mantenimiento, permitiendo corregir errores sin afectar al resto de partes

Tabla 12 NFR – 006 Mantenibilidad

NFR - 007	Uso de un entorno de Cloud Computing
Versión	V 1.0
Autores	Álvaro Martín López
Descripción	El sistema debe estar desplegado en una máquina que permita su máxima disponibilidad.

Tabla 13 NFR - 007 Uso de un entorno de Cloud Computing

NFR - 008	Soporte
Versión	V 1.0
Autores	Álvaro Martín López
Descripción	La página web debe incorporar medios de contacto con el administrador para reportar posibles errores.

Tabla 14 NFR - 008 Soporte

NFR - 009		Simplicidad	
Versión		V 1.0	
Autores		Álvaro Martín López	
Descripción		El diseño de la página web debe ser suficientemente sencillo como para obtener toda la información de un rápido vistazo.	

Tabla 15 NFR - 009 Simplicidad

NFR - 010		Uso de un entorno de pruebas	
Versión		V 1.0	
Autores		Álvaro Martín López	
Descripción		Se debe disponer de un entorno de pruebas que simule lo más fielmente al entorno final	

Tabla 16 NFR - 010 Uso de un entorno de pruebas

2.3 OBJETIVOS PERSONALES

La plataforma tendrá como objetivo principal informar al usuario tanto del estado actual de sus redes sociales como de los eventos pasados. Así, se pretende que el usuario pueda entender qué variables influyen en su crecimiento dentro de una red social y ayudarle a ascender dentro de la misma.

La plataforma está destinada a público de todo tipo, desde usuarios con cuentas promedio hasta grandes *influencers* que quieran ascender a lo más alto. Puede aplicarse incluso a compañías, pudiendo analizar, por ejemplo, una campaña de marketing a través de los históricos que ofrece la página web.

Dado que hay muchas redes sociales que un usuario querría monitorizar, se pretende hacer el sistema lo más modular y extensible posible, de forma que se puedan implementar funcionalidades de nuevas redes sociales de forma sencilla sin tener a penas impacto en el diseño de la plataforma.

Dado que no hay muchos conjuntos de datos etiquetados para el análisis de sentimiento en español, se pretende incorporar un *dataset* propio a aquellos que se encuentren y consideren válidos para satisfacer los objetivos del sistema. Para ello, se usará la plataforma de *Telegram* y las funcionalidades que ofrece para crear *bots* propios.

3 CONCEPTOS TEÓRICOS

En este apartado se definirán y explicarán los conceptos más importantes y frecuentes tratados en este documento.

3.1 API-REST

Las siglas de API significan *Application Programming Interface*, definiéndose como un conjunto de definiciones y de protocolos que se permiten la comunicación entre dos aplicaciones.

Gracias a este tipo de software, se puede ahorrar tiempo de desarrollo, decrementando el coste total. En el ejemplo de este trabajo, se utiliza por ejemplo la API de *Twitter* para obtener información, en lugar de hacer un módulo de *scraping* desde cero, leyendo y formateando el código *html* de la página.

También se pueden crear APIs propias, como es el caso de este trabajo, de forma que se facilite la comunicación entre módulos, facilitando el desarrollo y mantenimiento de la plataforma.

Implementar una API en la plataforma presenta así varias ventajas:

- Permite la reutilización de código, reduciendo costes y tiempos asociados al desarrollo
- Incrementa la interoperabilidad entre las aplicaciones.
- Suele ser sencilla de construir
- No consume muchos recursos

La parte de REST hacen referencia a la forma de interactuar con el servidor HTTP. Sus siglas significan *REpresentational State Transfer*. Es decir, es un servicio que no tiene estado. Esto significa que, dadas dos peticiones a la API, estas son completamente independientes. Para que el servidor recordase de algún tipo la sesión sería necesario un usuario, token u otro tipo de credencial [14].

3.2 BASE DE DATOS NO RELACIONAL

Una base de datos no relacional almacena información sin utilizar lenguaje *SQL* como herramienta de consulta (aunque sí pueden usarlo como apoyo). Por este motivo, también se les llama bases de datos *NoSQL*.

Este tipo de bases de datos tiene la peculiaridad de no utilizar tablas, sino colecciones que engloban documentos. Se podría realizar la siguiente analogía entre la terminología relacional y la no relacional:

Base de datos relacional	Base de datos no relacional
Base de datos	Base de datos
Tabla	Colección
Registro	Documento

Tabla 17 Comparativa nomenclatura bases de datos

Las bases de datos no relacionales no trabajan con estructuras definidas, estando pensadas para grandes volúmenes de datos, permitiendo trabajar con datos no estructurados o semi-estructurados. Los documentos de estas bases de datos pueden tener además un identificador único dentro de la base de datos, que facilita la búsqueda global de información, supliendo la carencia de estructuración.

Como contraparte, este tipo de base de datos no cumplen las propiedades de integridad, atomicidad y consistencia con las que si cumplen las relacionales [16].

3.3 ANÁLISIS DE SENTIMIENTO

El análisis de sentimiento, también conocido como minería de opinión, es un tipo de procesamiento del lenguaje natural cuyo objetivo principal es extraer el tipo de opinión o estado de ánimo sobre un producto. Dicho producto puede ser la intención general de un documento de texto, la opinión sobre una película, una campaña publicitaria o incluso frases sueltas.

Esta disciplina es una de las que trata la Inteligencia Artificial, como puede ser el *Machine Learning*. Para que estos algoritmos aprendan y se pueda extraer un modelo con el que clasificar información, es necesario aportar a este un conjunto de datos de entrenamiento.

Este conjunto de entrenamiento está normalmente conformado por sentencias asociadas a etiquetas particulares. Las etiquetas pueden variar en función del resultado que se quiere obtener. Se puede partir de una polaridad básica positivo-negativo en cuanto a la opinión y complicarlo hasta el punto de que el modelo distinga entre felicidad, asco, tristeza, miedo...

Los datos deben ser procesados antes de ser aportados como conjunto de entrenamiento., de forma que se puedan extraer las características fundamentales del corpus. Después, se proporciona al algoritmo, que debe ajustar sus pesos de cara a obtener la mayor precisión posible.

Así, se requiere cierta capacidad de cómputo para llevarlos a cabo todo este proceso, lo cual puede suponer una limitación.

Otras posibles limitaciones para obtener una precisión adecuada pueden ser:

- Subjetividad excesiva en la catalogación de oraciones, como puede ser etiquetar una frase como negativa únicamente por expresar una idea contraria
- Sarcasmo o negaciones en las oraciones, difícil de detectar
- Calidad y tamaño del conjunto de entrenamiento

3.4 ARQUITECTURA CLIENTE-SERVIDOR

Cuando se habla de arquitectura cliente-servidor, se hace referencia al modelo informático consistente en un nodo (servidor) que gestiona recursos y servicios de cara a otro nodo (cliente) que pretende hacer uso de los mismos.

Así, la parte del servidor cuenta con la capacidad de cómputo y los datos almacenados, comunicándose con el cliente para resolver sus peticiones. Los clientes, por su parte, se conectan a través de una red como puede ser Internet al servidor. El cliente, además de pedir datos, también puede querer entregarlos o modificar algunos ya existentes, de forma que el servidor tiene que ofrecer las funcionalidades adecuadas.

Este modelo ofrece como ventaja un alto grado de fiabilidad, permitiendo centralizar recursos, mejorar la seguridad, la administración del servidor o escalar el sistema fácilmente. Sin embargo, que los recursos se encuentren centralizados puede ser una desventaja también, pudiendo subsanarse con cierto grado de distribución.

3.5 TELEGRAM

Telegram [18] es una aplicación de mensajería enfocada a la velocidad y seguridad, con más de 500 millones de usuarios activos anuales, convirtiéndola en una de las 10 apps más descargadas del mundo.

Además de los básicos mensajes, fotografías, vídeos y archivos generales, también se pueden crear grupos con un límite de 200.000 usuarios, donde se pueden establecer una serie de reglas. También permite el uso de canales en caso de querer difundir a una audiencia ilimitada, como puede ser el caso de un canal de noticias. Soporta llamadas de voz y videollamadas, así como mensajes de voz y video.

Sumado a todo esto ofrecen dos tipos de APIs destinadas a desarrolladores: la Bot API y TDLib, que permite crear clientes de *Telegram* customizados. Las APIs están abiertas, ofreciendo un proceso muy simple para utilizarlas.

La *Bot API* de *Telegram* permite a los desarrolladores crear de forma simple programas que utilicen mensajes de *Telegram* como interfaz. Para utilizarla ponen a disposición pública un *bot* conocido como *The Botfather*, gracias al cual se pueden registrar *bots* y editar la parte pública de los mismos, como puede ser el nombre del *bot*, la descripción o los comandos que mostrará la ayuda. La funcionalidad del *bot* la implementa el propio programador.

4 TÉCNICAS Y HERRAMIENTAS

En este apartado se recogen las técnicas y herramientas utilizadas en el desarrollo de la plataforma.

Es importante tener claro que lenguaje utilizar para el desarrollo, ya que debe ser adecuado para las necesidades de la plataforma. Para ello, debemos tener en cuenta el tiempo y los objetivos de la plataforma.

Como hay varios módulos que desarrollar, *Python* [19] es una buena opción, ya que es un lenguaje que permite un desarrollo rápido de software. Esto permitirá tener conectados los módulos rápidamente, estableciendo los puntos de entrada a cada uno, para implementar posteriormente la funcionalidad más específica.



Python es uno de los lenguajes más utilizados en el mundo, disponiendo de una inmensa cantidad de librerías que serán muy útiles en el desarrollo de los módulos.



Sin embargo, *Python* tiene también desventajas, como ser más lento que otros lenguajes como C++ o la forma en la que trata la asincronía. Para realizar una página web, son necesarias llamadas asíncronas, por lo que se debe escoger un lenguaje más adecuado para el módulo de visualización. De esta forma se ha elegido Node.js [20], un entorno de ejecución multiplataforma asíncrono y basado en *JavaScript*, líder en desarrollo web. Además, se ha utilizado *Express.js* como *framework* por su comodidad a la hora de organizar el código, crear el proceso de enrutamiento, identificación y validación de sesiones, cabeceras automáticas en las respuestas... Por último, se ha decidido utilizar *TypeScript* [21], un lenguaje compilado en JavaScript que añade objetos basados en clases y tipos estáticos, aumentando su entendibilidad y calidad.

A continuación, se listarán los módulos, librerías, *frameworks*, técnicas o herramientas divididas según el módulo con el que estén relacionadas.

4.1 MÓDULO DE UTILIDAD

Se ha elegido dar este nombre a aquellas técnicas y herramientas que sirven de apoyo para el resto de los módulos, pero no están relacionados entre sí como para conformar un módulo por sí mismas.

4.1.1 Visual Studio Code

Visual Studio Code [22] es un editor de texto plano gratuito y de código abierto desarrollado por Microsoft. Pese a que consume poca memoria (pues tiene que cargar todo el *core* de Chrome), es un programa muy sencillo de utilizar y muy potente, ofreciendo una gran cantidad de *plugins* que facilitan el desarrollo y ahorran bastante tiempo.

Este editor es compatible con multitud de lenguajes, ofreciéndoles soporte mediante *IntelliSense* [23] para resaltar sintaxis, autocompletar, entre otros

4.1.2 Entornos utilizados

Se listan y describen en este punto los entornos utilizados para el desarrollo, pruebas y despliegue de la plataforma.

4.1.2.1 Entorno local

El sistema operativo en el que se ha desarrollado la plataforma es un *Linux Manjaro 20*, puesto que es el sistema operativo instalado en el ordenador personal del desarrollador.

4.1.2.1.1 Manjaro

Manjaro Linux [24] es una distribución *GNU/Linux*, con *Xfce*, *KDE* o *GNOME Shell* como interfaz de usuario por defecto, aunque hay otras versiones no oficiales que ofrecen también gran soporte. Se trata de un sistema operativo libre para ordenadores personales y enfocado en la facilidad de uso. Está basado en *Arch Linux* [25] y usa un modelo de desarrollo *rolling release*, consistente en actualizaciones constantes de software que generan menor riesgos al actualizar componentes principales.

Esta distribución pretende ser amigable con el usuario sin perder las características de *Arch*, como el gestor de paquetes *Pacman* y la compatibilidad con AUR (*Arch User's Repository*). *Manjaro* utiliza tres repositorios:

- El *unstable*, que contiene los más recientes paquetes de *Arch*, probablemente con un retraso de uno a dos días.
- El repositorio *testing* que agrega paquetes desde el repositorio *unstable* cada semana.
- El repositorio *stable*, que contiene sólo paquetes que son considerados estables por el equipo de desarrollo, además de scripts que ayudan a que las actualizaciones conflictivas se configuren automáticamente sin necesidad de intervención del usuario, como en el caso de las otras ramas, o en *Arch Linux*.

4.1.2.2 Entorno virtual

Puesto que en la máquina principal funciona el *bot* desde prácticamente el principio y se va desplegando gradualmente la plataforma, resulta de utilidad construir un entorno que emule en lo posible la máquina principal, de forma que las pruebas de la implementación no afecten a un comportamiento que ya se ha considerado correcto.

4.1.2.2.1 Oracle VM Virtualbox

Actualmente, *VirtualBox* [26] se ejecuta en hosts *Windows*, *Linux*, *Macintosh* y *Solaris* y es compatible con un gran número de sistemas operativos como *Solaris* y *OpenSolaris*, *OS/2* y *OpenBSD*.

VirtualBox se está desarrollando activamente con frecuentes lanzamientos y tiene una lista cada vez mayor de características, sistemas operativos invitados soportados y plataformas en las que funciona.

Así, tanto en la máquina virtual creada como en la máquina final que proporciona *AWS*, se instala el sistema operativo *Ubuntu*, basado en *Debian*.



4.1.2.2.2 Ubuntu

Ubuntu [27] es uno de los sistemas operativos basados en *Linux* más utilizado en la actualidad, algo que ha conseguido gracias a su rendimiento, a la posibilidad de descargarse gratis y a tratarse de software libre que cualquiera puede modificar para mejorar su código, algo que ha propiciado la aparición de otros entornos gráficos como *Kubuntu* o *Xubuntu* entre muchos otros. El periodo aproximado de actualizaciones de *Ubuntu* es de seis meses y es Canonical la que se encarga de aportar los *updates* de seguridad para solucionar errores críticos y bugs.

4.1.2.3 Entorno principal

Por último, para el despliegue final de la plataforma, se necesita un host que pueda ser accesible durante el mayor tiempo posible. Además, este equipo debía ser obtenido rápido para poder desplegar el *bot* de *Telegram*, ya que es inviable tener un equipo personal ejecutando el servidor y el *bot* durante todo el día.

4.1.2.3.1 AWS (Amazon Web Services)

Se trata de una subsidiaria de Amazon que ofrece plataformas de computación en nube y APIs bajo de manda a particulares, empresas y gobiernos, en régimen de pago por uso.

La tecnología de AWS se implementa en granjas de servidores de todo el mundo y es mantenida por la filial de Amazon. Las tarifas se basan en una combinación de uso (conocido como modelo "Pay-as-you-go"), hardware, sistema operativo, software o características de red elegidas por el suscriptor que requieren disponibilidad, redundancia, seguridad y opciones de servicio. Los suscriptores pueden pagar por un único ordenador virtual de AWS, por un ordenador físico dedicado o por *clusters* de ambos. Como parte del acuerdo de suscripción, Amazon proporciona seguridad a los sistemas de los suscriptores [28].



4.1.2.3.2 Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) [29] proporciona capacidad informática escalable en la nube de *Amazon Web Services* (AWS). El uso de Amazon EC2 elimina la necesidad de invertir en hardware por adelantado, por lo que se puede desarrollar e implementar aplicaciones más rápidamente. Es posible utilizar Amazon EC2 para lanzar tantos o tan pocos servidores virtuales como se necesite, configurar la seguridad y la red y administrar el almacenamiento. Amazon EC2 permite escalar hacia arriba o hacia abajo para manejar los cambios en los requisitos o los picos de popularidad, reduciendo su necesidad de prever el tráfico. Las especificaciones de la máquina se recogen en la siguiente tabla:

Tipo de Instancia	t2.micro
Sistema Operativo	Ubuntu 20.03
CPU	1 CPU,
Memoria RAM	1 Gb
Disco Duro	20 Gb (30Gb R/W)

Tabla 18 Especificaciones instancia EC2

4.1.3 Git

Git [30] es un sistema de control de versiones distribuido, gratuito y de código abierto, diseñado para manejar todo tipo de proyectos, desde los más pequeños hasta los más grandes, con rapidez y eficiencia.

Es fácil de aprender y ocupa poco espacio con un rendimiento rapidísimo. Supera a las herramientas *SCM* como *Subversion*, *CVS*, *Perforce* y *ClearCase* con características como la ramificación local barata, cómodas áreas de preparación y múltiples flujos de trabajo.

Gracias a esta herramienta, se puede separar el desarrollo de la plataforma en varias ramas en función del entorno en el que estemos trabajando, incluyendo los incrementos del código de forma gradual.



4.1.3.1 GitHub

GitHub es un proveedor de alojamiento en Internet para el desarrollo de software y el control de versiones mediante *Git*. Ofrece la funcionalidad de control de versiones distribuidas y gestión de código fuente de *Git*, además de sus propias características.



Proporciona control de acceso y varias características de colaboración, como seguimiento de errores, solicitudes de características, gestión de tareas, integración continua y wikis para cada proyecto.

No se debe confundir *GitHub* con *Git*, puesto que *Git* es una herramienta y *GitHub* una plataforma de almacenamiento y control de versiones que hace uso de *Git*.

El código completo de este Trabajo de Fin de Grado se ha colgado en la plataforma de *GitHub*, siguiendo el siguiente enlace: <https://github.com/Alburrito/social-sentiment/>. La tarea de controlar el flujo de versiones mediante *Git* ha sido facilitada gracias a los *plugins* disponibles en el IDE *Visual Studio Code*.

4.1.4 Variables de entorno

Las variables de entorno son un conjunto de valores dinámicos con nombre que se almacenan en el sistema y se utilizan en las aplicaciones lanzadas en *shells*. En otras palabras, son variables con nombre y valor asociado, normalmente escritas en mayúsculas.

Las variables de entorno suponen una serie de beneficios a la plataforma, como son:

- Facilidad de configuración. Solo se debe configurar la plataforma la primera vez y es sencillo realizar cambios, ya que se pueden establecer los parámetros de configuración (como direcciones IP, URL, contraseñas, etc.) en un fichero aparte.
- Mayor seguridad al poder establecer los valores desde una Shell o un fichero aparte. Así, al usar *Git* podemos establecer como regla ignorar la subida del fichero para no exponer información sensible.
- Menos errores en producción. Al trabajar con varias ramas, es fácil confundir las configuraciones de los entornos. Por ejemplo, supongamos un servidor de correo en cuya rama principal se cuelen correos de *debug*. Estos llegarían a los clientes, siendo un resultado no deseable.

Para establecer las variables de entorno de la plataforma Social Sentiment, se ha utilizado un fichero `.env` con la siguiente información:

Ámbito	Parámetros
Página Web (<i>Node.js</i>)	<ul style="list-style-type: none"> ● Puerto de la aplicación (1)
Base de Datos (<i>MongoDB</i>)	<ul style="list-style-type: none"> ● Host, puerto y nombre de la BD ● Nombres de las colecciones utilizadas
API (<i>Flask</i>)	<ul style="list-style-type: none"> ● Modo <i>Debug</i> ● Host y puerto de la API ● Entorno de <i>Flask</i>
Ingesta (<i>Twitter</i>)	<ul style="list-style-type: none"> ● <i>Tokens</i> de la App y de la cuenta personal de Twitter (2) ● URL de la función de <i>Callback</i> (3) ● WOEID de España (4)
Otros	<ul style="list-style-type: none"> ● TESTENV. Permite comprobar el entorno.

Tabla 19 Variables de entorno

1. El puerto que sirve la página web cambia en función de la rama.
2. Se obtienen en la página de desarrolladores de *Twitter*, aunque el proceso de obtención se explicará en las técnicas y herramientas correspondientes al módulo de ingesta.

3. Para la autorización de los usuarios en *Twitter* es necesario aportar en la página de la app una url de *callback*, aunque este proceso se explica en el *Anexo V. Documentación técnica*.

4. El WOEID es un identificador que se usa para obtener el *Trending Topic* de *Twitter* España, aunque este proceso se explica en la parte de implementación del punto de *Aspectos relevantes* y, más a fondo, en el *Anexo V. Documentación técnica*.

Estas variables se recogen mediante un fichero de *Python* que gestiona las configuraciones tratado más adelante, excepto en el módulo de visualización, donde se recogen a través de un fichero propio.

Dado el propósito del *bot* de *Telegram*, este módulo está pensado para ser desplegado lo más rápido posible y en un nodo distinto al resto de la plataforma, por lo que incorpora un fichero de configuración propio escrito en *Python*. Dicho fichero incorpora todos los mensajes que puede enviar el *bot* y parámetros de configuración, facilitando bastante los posibles cambios.

4.1.4.1 *Pipenv*

Para activar el entorno de la plataforma Social Sentiment y cargar así las variables se utiliza el módulo de *Python* llamado *Pipenv* [31]. *Pipenv* combina la herramienta *Pip*, que permite la instalación de paquetes de *Python*, con *Virtualenv*, que facilita la gestión de entornos virtuales.

De esta forma, se puede crear un entorno para la plataforma que cargue automáticamente las variables e instalar las dependencias de *Python* con la misma herramienta.

Para crear el entorno, se navega hasta el directorio del proyecto y se ejecuta:

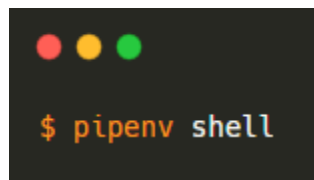


Ilustración 8 Creación de entorno virtual con *Pipenv*

Esto creará el entorno si no existía ya y lo activará en ambos casos. Una vez en el entorno se pueden ejecutar las siguientes órdenes básicas:

```
$ pipenv --rm # Elimina el entorno
$ pipenv install pandas # Instala un paquete de python
$ pipenv graph # Muestra un grafo de dependencias
$ pipenv check # Comprueba vulnerabilidades en las dependencias

# También se puede seleccionar una versión específica de python
$ pipenv --python 3.7
```

Ilustración 9 Utilidades básicas de pipenv

4.1.5 Logging

Para poder seguir el flujo de ejecución fuera de *Visual Studio Code*, se ha implementado en los módulos de *Python* un sistema de *logs* utilizando el módulo *logging* [32].

Este módulo implementa un sistema flexible de registro ya que, al ser parte de la biblioteca estándar de *Python*, cualquier módulo puede participar en el registro. Este módulo implementa cuatro clases principales:

- *Loggers*, que ofrecen la interfaz para usar en el código.
- Manejadoras, que envían los registros al destino correspondiente
- Filtros, para determinar con mayor precisión los registros que se mostrarán
- Formateadores, para especificar el formato de la salida final.

De esta forma, en el fichero de *resources/utills.py*, explicado por partes según el módulo en el punto de *Aspectos Relevantes*, se definen, para cada módulo, sus correspondientes parámetros de log. En la figura inferior se puede observar un ejemplo de parámetros para el log del gestor.

```
136 PYSTATS_HANDLER = 'pystats_handler'
137 PYSTATS_LOGGER = 'pystats_log'
138 PYSTATS_LOG_DIR = LOG_DIR
139 PYSTATS_LOG_LVL = logging.INFO
140 PYSTATS_LOG_FILE = os.path.join(PYSTATS_LOG_DIR, 'pystats.log')
```

Ilustración 10 Ejemplo de configuración de log de un módulo

Estos parámetros recogen los nombres y ficheros de destino de los logs de cada uno de los módulos. Una vez establecidos se crea el directorio indicado por *LOG_DIR* si no existiese.

Posteriormente se utilizan en un `dictConfig()`, indicando los formatos, manejadores, *loggers* y filtros.

- Para el formato se usa la siguiente sintaxis:

```
'formatters': {'default': {
    'format': '[%(asctime)s %(levelname)s %(name)s %(threadName)s : %(message)s',
}},
```

Ilustración 11 Ejemplo de formatter del log

- Para las manejadoras, se permite la opción de indicar el máximo de bytes que puede ocupar un fichero de log, a partir de los cuales empezará a escribir en un fichero nuevo. Esto se complementa muy bien con otra funcionalidad que permite establecer el número máximo de logs que puede haber. Una vez superado este número, se escribirá sobre el primero, formando un ciclo:

```
ANALYZER_HANDLER: {
    'class': 'logging.handlers.RotatingFileHandler',
    'filename': ANALYZER_LOG_FILE,
    'level': ANALYZER_LOG_LVL,
    'formatter': 'default',
    'maxBytes': 65536,
    'backupCount': 5
}
```

Ilustración 12 Ejemplo de manejadora del log

- Para el *logger*, únicamente es necesario indicar el nombre del mismo, su nivel de log y las manejadoras que tendrá

```
SCRAPPER_LOGGER: {
    'level': SCRAPPER_LOG_LVL,
    'handlers': ['default', SCRAPPER_HANDLER]
},
PYSTATS_LOGGER: {
    'level': PYSTATS_LOG_LVL,
    'handlers': ['default', PYSTATS_HANDLER]
},
```

Ilustración 13 Ejemplos de logger

En cuanto al nivel de *log*, este módulo implementa cinco niveles. Estos son, ordenados de menor a mayor importancia:

1. Nivel de *Debug*
2. Nivel de Información
3. Nivel de Advertencia
4. Nivel de Error
5. Nivel Crítico

4.1.6 Formatos

Se listan a continuación los dos formatos utilizados para los datos, así como su descripción, diferencias y ventajas y desventajas.

4.1.6.1 JSON (*JavaScript Object Notation*)

Este formato de texto es uno de los más utilizados en el intercambio de datos. En primer lugar, surgió como un subconjunto de la notación literal de objetos de *JavaScript*, pero acabó convirtiéndose un formato independiente reconocido dado su amplia aceptación como alternativa a *XML*.

Es un formato legible y escribible por humanos, pero también por máquinas. Un archivo en formato *JSON* se caracteriza por dos estructuras principales:

- Colección de pares clave-valor. En muchos lenguajes, es la representación de un objeto, adoptando también otros nombres como registro, *hashtable*, diccionario...
- Lista ordenada de valores. En muchos lenguajes, se conoce como array, vector o lista.

Así, un objeto es un conjunto desordenado de pares nombre-valor y utiliza como delimitador llaves de apertura y cierre {}.

Cada nombre es seguido por dos puntos y los pares nombre/valor están separados por un símbolo de coma. De esta forma, es fácil crear todo tipo de estructuras utilizando los distintos tipos de datos. Esto provoca que, de una forma u otra, casi todos los lenguajes puedan soportar este formato, convirtiéndolo desde hace unos años en uno de los más utilizados del mundo [33].



```
{
  "frame": -1816911578,
  "get": false,
  "lovely": [
    false,
    false,
    true,
    false
  ],
  "mixture": "bare"
}
```


4.1.6.2 *Pickle*

El lenguaje *Python* tiene un módulo llamado *Pickle* [34], que implementa protocolos binarios para serializar y deserializar estructuras de objetos de *Python*. El “*pickling*” es un proceso por el que una jerarquía de objetos *Python* se convierte en un flujo de bytes, siendo el “*unpickling*” el proceso inverso en el que, dado un flujo de bytes, se puede convertir en una jerarquía de objetos. Este formato es bastante útil a la hora de guardar y cargar modelos de *Machine Learning* o similares.

4.1.6.3 *Diferencias entre JSON y Pickle*

Estos dos formatos presentan entre sí las diferencias mostradas a continuación:

- La principal diferencia que podemos encontrar es que, mientras que el primero es un formato de serialización de texto, el segundo es un formato de serialización binario. Esto provoca que un humano no pueda leer un archivo *pickle*, pero sí uno *JSON*.
- *Pickle* es específico de *Python*, mientras que *JSON* es interoperable y muy utilizado también fuera de *Python*.
- *Pickle* puede representar un número enorme de tipos de *Python* (muchos de ellos automáticamente), mientras que *JSON* solo puede representar un subconjunto de los tipos que incorpora *Python*.
- El módulo de *Pickle* no incorpora seguridad, por lo que hay que tener cuidado con los ficheros que se quiera deserializar. Por otro lado, deserializar un *JSON* no confiable no crea en sí mismo una vulnerabilidad de ejecución de código arbitrario.

4.2 TÉCNICAS Y HERRAMIENTAS DEL MÓDULO DE LA API

Para llevar a cabo la construcción de la API, se han utilizado las siguientes técnicas y herramientas.

4.2.1 Flask

Flask [35] es un *framework* de aplicaciones web *WSGI* ligero. Está diseñado para que empezar sea rápido y fácil, con la capacidad de escalar a aplicaciones complejas. Comenzó como una simple envoltura alrededor de *Werkzeug* y *Jinja* y se ha convertido en uno de los marcos de aplicaciones web de *Python* más populares.

Flask ofrece sugerencias, pero no impone ninguna dependencia o disposición del proyecto. Depende del desarrollador elegir las herramientas y bibliotecas que quiere utilizar. Hay muchas extensiones proporcionadas por la comunidad que hacen que añadir nuevas funcionalidades sea fácil.

4.2.2 Flask-Cache

Flask-Caching [36] es una extensión de *Flask* que añade soporte de caché para varios *backends* a cualquier aplicación *Flask*. Además de proporcionar soporte para todos los *backends* de caché originales de *werkzeug* a través de una API uniforme, también es posible desarrollar un propio *backend* de caché.

4.2.3 Postman

Postman [37] es una herramienta utilizada especialmente para el *testing* de *API REST* [14], aunque también ofrece otras funcionalidades a mayores. Gracias a *Postman* es posible testear y depurar la API, además de monitorizarla, escribir pruebas automáticas, documentarlas...

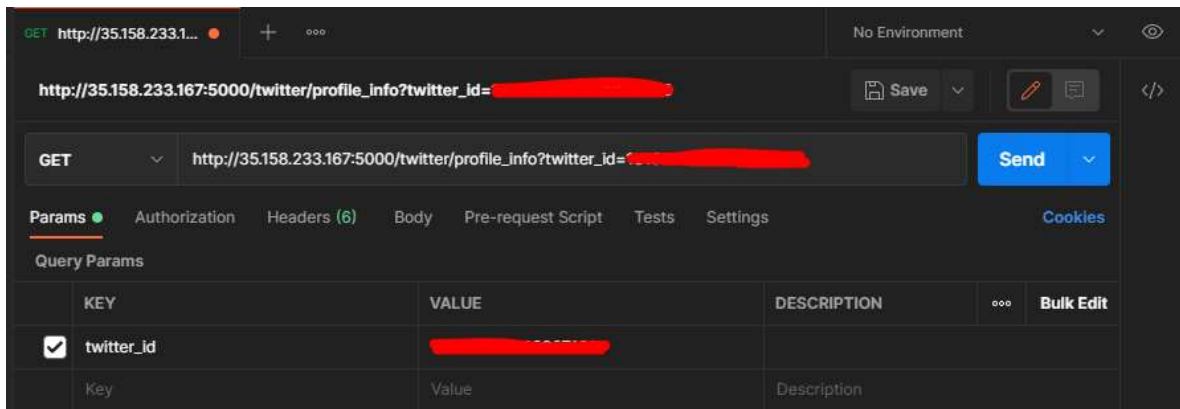


Ilustración 14 Get Request en Postman

Como se ve en la figura, utilizar *Postman* es tan simple como especificar la *URL* de la API a la que se quiere acceder. Las funcionalidades que ofrece para el control de argumentos agilizan mucho las pruebas. En el ejemplo se llama a un *endpoint* de la API que devuelve el perfil de un usuario dado su identificador.

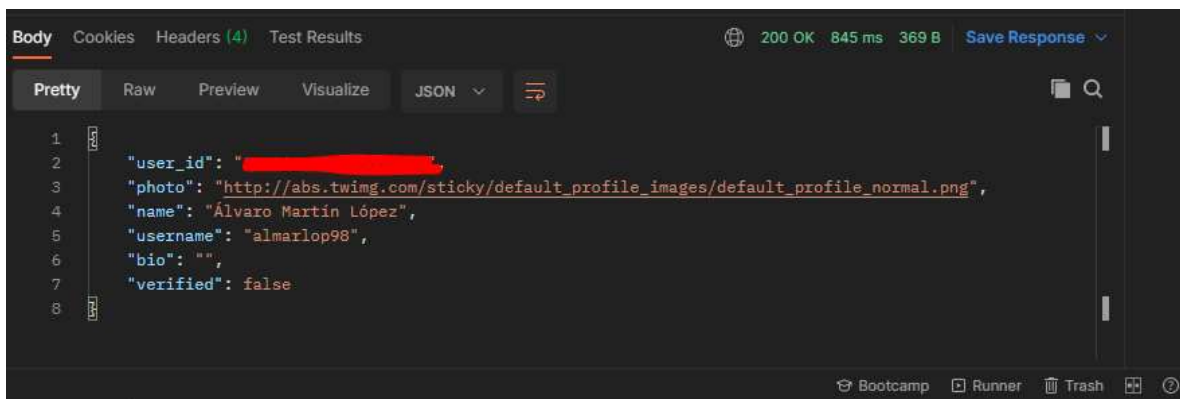


Ilustración 15 Resultado de la petición GET

El resultado de la petición puede mostrarse en distintos formatos según interese, como pueden ser *JSON*, *XML* o *HTML*.

4.3 TÉCNICAS Y HERRAMIENTAS DEL MÓDULO DE INGESTA

Los datos de *Twitter* son únicos y reflejan información que los usuarios deciden compartir de forma pública. Si plataforma de API ofrece acceso amplio a los datos de *Twitter* que los usuarios han decidido compartir con el mundo. *Twitter* también es compatible con API que permiten a los usuarios administrar su propia información de *Twitter* que no es pública (como los mensajes directos) y proporcionarla a los desarrolladores que ellos han autorizado a administrarla.

4.3.1 Twitter API

Cuando alguien quiere acceder a la API de *Twitter* [15], tiene que registrar una aplicación. De forma predeterminada, las aplicaciones solo pueden acceder a información pública en *Twitter*. Ciertos puntos de conexión, como aquellos responsables del envío o recepción de mensajes directos, requieren permisos adicionales antes de poder acceder a la información. Estos permisos no se otorgan de forma predeterminada; el usuario elige según cada aplicación si desea otorgar este acceso y puede controlar todas las aplicaciones autorizadas en su cuenta.

Para poder registrar una aplicación se debe ingresar a la página oficial de desarrolladores [38]. Una vez allí, se debe ingresar con la cuenta de *Twitter* o registrarse en caso de no tener una.

Una vez *logueados* y en el panel de desarrolladores, se debe ingresar a crear una App, lo que ofrecerá una serie de vistas que incorporan formularios para poder registrar la app.

En primer lugar, preguntará el nombre de la aplicación:

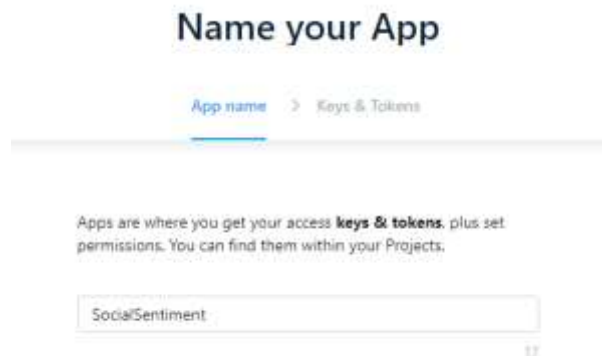


Ilustración 16 Vista para nombrar la aplicación en el panel de desarrollador de *Twitter*

Una vez registrada y especificado su propósito, se puede establecer una URL de *Callback* (utilizada durante la vinculación de cuentas) y una URL del sitio web, junto a otros parámetros.

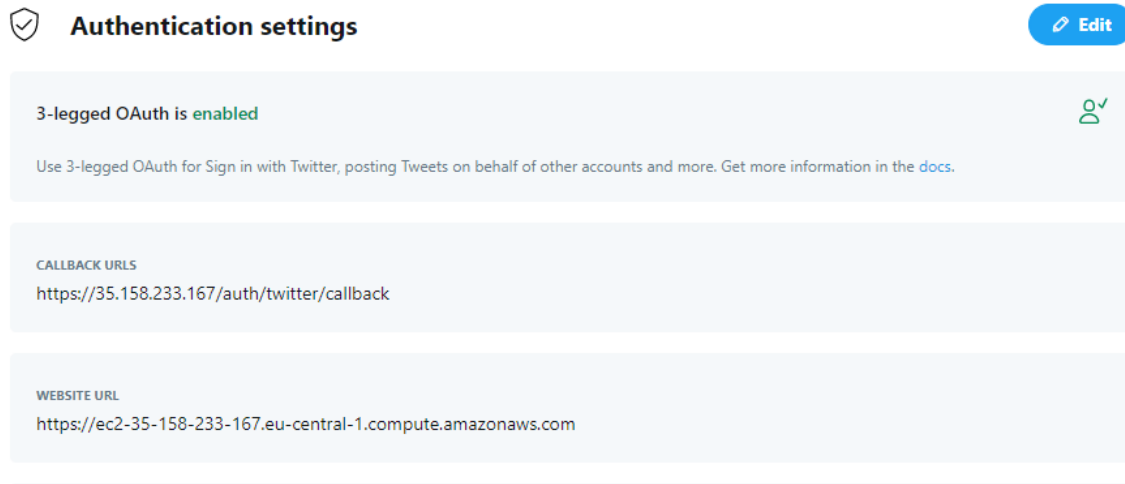


Ilustración 17 Propiedades de la aplicación Twitter

Por último, se pueden generar los *tokens* (públicos y secretos) de la aplicación creada y de la cuenta de Twitter en la sección *Keys and Tokens*. Una vez generados es importante guardarlos con mucho cuidado ya que no es posible volver a verlos en la plataforma y, en caso de perderlos, se deben generar claves nuevas.

Los datos obtenibles de la API de *Twitter* se dividen en cinco categorías:

- Cuentas y usuarios.
- *Tweets* y respuestas
- Mensajes directos
- Anuncios
- Herramientas y SDK del editor

Gracias a la API es posible no solo recuperar información, sino también interactuar enviando mensajes, bloqueando cuentas, siguiendo otras cuentas...

4.3.2 OAuth

OAuth [39] es un protocolo y *framework* de autorización abierto que proporciona a las aplicaciones la capacidad de "acceso designado seguro". Por ejemplo, se puede decir a *Facebook* que está bien que una aplicación acceda a un perfil o publique actualizaciones en la línea de tiempo del interesado sin tener que darle a esa aplicación la contraseña de *Facebook*. Esto minimiza el riesgo de manera importante: En el caso de que la aplicación sufra una brecha, la contraseña de Facebook permanece segura.

OAuth no comparte los datos de las contraseñas, sino que utiliza tokens de autorización para probar una identidad entre los consumidores y los proveedores de servicios. *OAuth* es un protocolo de autenticación que permite aprobar que una aplicación interactúe con otra en nombre de un usuario sin revelar su contraseña.

La API de *Twitter* ofrece *OAuth1.0 (Application-user)* y *OAuth2.0 (Application-only)*, aunque el módulo de *Tweepy* [40] solo ofrece soporte para *OAuth1.0* de momento.

4.3.3 Tweepy

Tweepy es una librería de *Python* que facilita enormemente la comunicación con la API de *Twitter*, encapsulando las llamadas a los distintos *endpoints* en métodos de una clase.

Para utilizarla únicamente se debe conectar nuestra aplicación mediante *OAuth*, gracias a un método que implementa este proceso y, posteriormente, se puede acceder a las funcionalidades de la API de *Twitter*.

4.4 TÉCNICAS Y HERRAMIENTAS DEL MÓDULO DE PERSISTENCIA

4.4.1 MongoDB

MongoDB [17] es una base de datos de documentos con la escalabilidad y la flexibilidad que desea con la consulta y la indexación que necesita en una plataforma como la perseguida. Provee un modelo de documentos sencillo de aprender y utilizar, con drivers oficiales para más de diez idiomas, aunque la comunidad ha desarrollado aún más.

MongoDB almacena los datos en documentos en formato *JSON* [33], permitiendo cambiar la estructura de los datos con el tiempo. El modelo de los datos se asigna a los objetos del código, facilitando trabajar con los datos en los módulos correspondientes.



Ilustración 18 Rótulo de MongoDB

4.4.1.1 Estructura de la base de datos

Para estructurar la información necesaria se han creado dos bases de datos, una para la plataforma en sí y otra para el Bot. La segunda se explicará en el apartado correspondiente al *Bot* de *Telegram* (apartado 4.3).

En cuanto a la base de datos principal, destinada a la plataforma en sí misma, se han creado tres colecciones. La estructura de estas colecciones se representa en los diagramas inferiores:

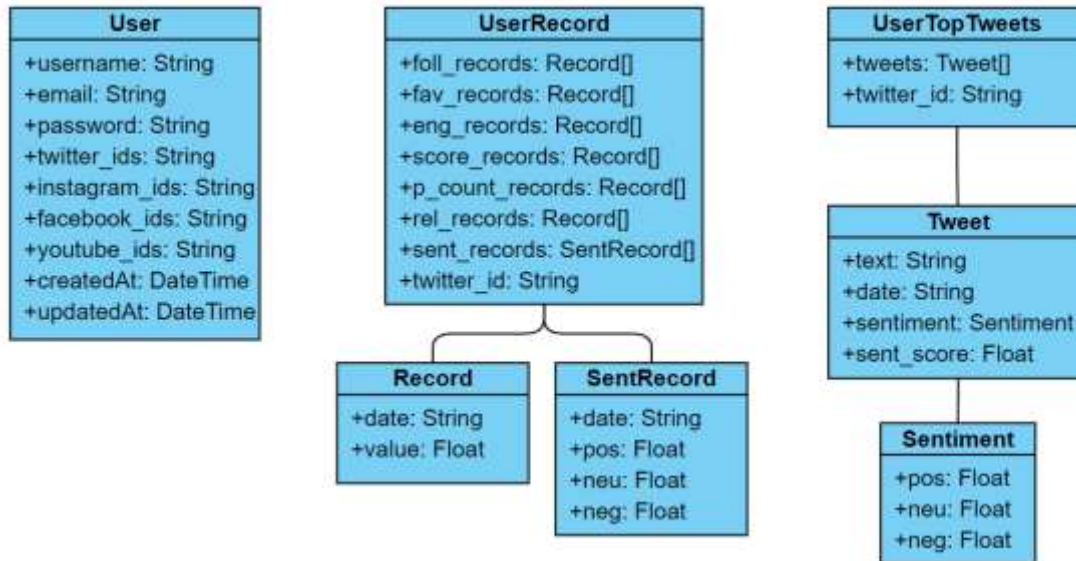


Ilustración 19 Estructura de la base de datos

1. La colección **users** recoge en cada documento *User* información básica de la cuenta del usuario junto a los identificadores de sus respectivas redes sociales
2. La colección **user_records** recoge en cada documento *UserRecord* el histórico de valores de la cuenta de Twitter de cada usuario. Cada registro de estos históricos consta de una fecha y un valor asociado, a excepción de los referentes al sentimiento, que utilizan porcentajes de positivo (pos), neutro (neu) y negativo (neg).
3. La colección **user_top_tweets** recoge en cada documento *UserTopTweets* los cinco tweets del usuario que mejor recepción hayan tenido. Cada tweet de este tipo de documento se representa como un objeto con los campos referenciados en el diagrama superior.

Todos los documentos tienen un campo *twitter_id* que permite hacer *queries* para usuarios específicos.

4.4.2 PyMongo

PyMongo [41] es una distribución de *Python* que contiene herramientas para trabajar con *MongoDB*, y es la forma recomendada para trabajar con *MongoDB* desde *Python*. Para trabajar con esta librería, basta con seguir el flujo indicado en la figura siguiente:

```
from pymongo import MongoClient

# Crear el cliente. 27017 es el puerto por defecto de MongoDB
client = MongoClient('urlBD', 27017)

# Conectar con la base de datos
db = client['nombreBD']

# Utilizar una colección
collection = db['nombreColeccion']

# Ya se pueden utilizar las herramientas de MongoDB
res = collection.find()
res = collection.insert_one({'id':6, 'name': 'Alvaro'})
res = collection.update_one({'id':6}, {'$set': {'name': 'almarlop'}})
```

Ilustración 20 Uso básico de PyMongo

Los métodos de *PyMongo* devuelven en general cursores que tienen formato *JSON*, fácilmente transformables a objetos o tratables como diccionarios

4.4.3 Mongoose

Mongoose [42] proporciona una solución sencilla, basada en esquemas, para modelar los datos de la aplicación de la página web. Incluye el *casting* de tipos integrado, la validación, la creación de consultas, y mucho más.

En el caso de la gestión de usuarios el módulo de visualización se comunica directamente con la base de datos a través de *Mongoose*, ya que realiza esta función de forma más cómoda que conectando a través de la API.

Mongoose se instala a través de *npm* como se verá posteriormente. De forma general, lo primero que se debe hacer para comenzar a usarlo es implementar la conexión mediante una función asíncrona, a la cual se le debe especificar la *URI* de la base de datos junto a una serie de parámetros para su correcto funcionamiento:

```
import mongoose from "mongoose";

(async () => {
  const mongoose = await mongoose.connect('mongodb://localhost/nombreBD'), {
    useNewUrlParser: true,
    useFindAndModify: false,
    useCreateIndex: true,
    useUnifiedTopology: true
  }
})
```

Ilustración 21 Conexión con base de datos vía Mongoose

Una vez conectado a la base de datos se debe definir un esquema o modelo que define la estructura del documento con el que se trabajará y una interfaz que permita su uso:

```
const UserSchema = new Schema<UserDocument>({
  username: { type: String, required: true, unique: true },
  email: { type: String, required: true, unique: true },
  password: { type: String, required: true },
  twitter_ids: { type: String, required: true },
  instagram_ids: { type: String, required: true },
  facebook_ids: { type: String, required: true },
  youtube_ids: { type: String, required: true },
}, {
  timestamps: true,
});

export type UserDocument = Document & {
  username: string;
  email: string;
  password: string;
  twitter_ids: string;
  instagram_ids: string;
  facebook_ids: string;
  youtube_ids: string;
  encryptPassword(password: string): Promise<string>;
  matchPassword(password: string): Promise<boolean>;
};
```

Ilustración 22 Esquema e interfaz de Usuario mediante Mongoose

En las figuras superiores se muestra como ejemplo el esquema e interfaz utilizados para los usuarios. Como se puede observar, coincide con los campos mostrados en los diagramas de las colecciones mostrados anteriormente.

Como se puede ver, es posible definir para cada campo una serie de parámetros para indicar si es obligatorio o de valor único, entre otras características, además de incluir las fechas de creación y modificación.

En cuanto a la interfaz, se puede implementar métodos necesarios para completar la funcionalidad.

4.4.4 MongoDB Compass

Por último, se ha utilizado como herramienta complementaria para monitorizar la base de datos MongoDB Compass [43]. Esta herramienta supone una alternativa con interfaz gráfica a las órdenes clásicas desde *Shell* de *MongoDB*.

Esta herramienta permite recordar las conexiones, pudiendo gestionar la base de datos de los entornos de desarrollo, pruebas y producción fácilmente. Además, incorpora todas las funcionalidades básicas de *CRUD* y muchas más.

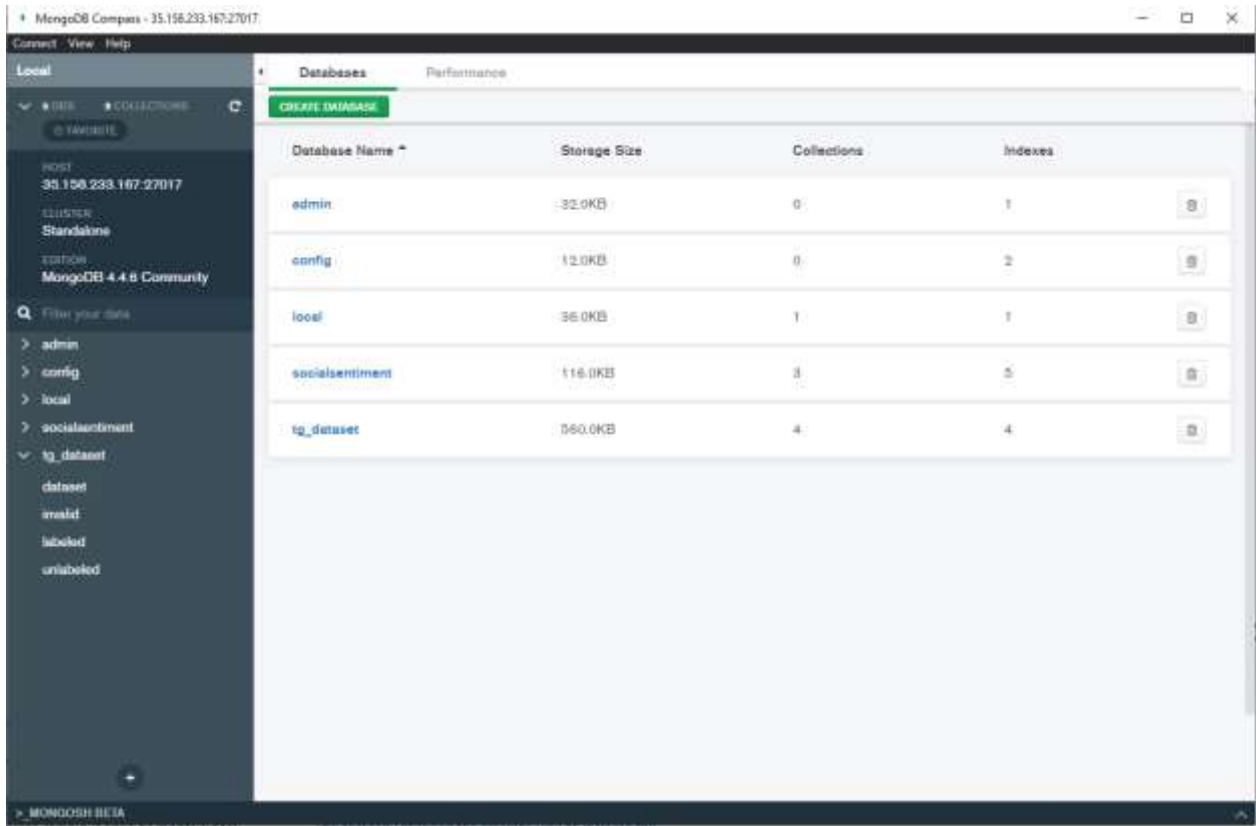


Ilustración 23 Interfaz de MongoDB Compass

En la figura superior se puede ver una captura del *dashboard* que ofrece *MongoDB Compass*. Como se puede observar, se trata de una conexión a la base de datos de producción. Ofrece una forma muy cómoda de navegar entre bases de datos y colecciones a través de su *sidebar*.

4.5 TÉCNICAS Y HERRAMIENTAS DEL MÓDULO DE ANÁLISIS

El propósito general de este módulo es aplicar análisis de sentimiento y calcular distintas métricas. Pese a que parte de la información también se muestra al usuario, esta sirve en su práctica totalidad en el cálculo de una puntuación. Gracias a esta puntuación, el usuario se podrá comparar con el resto de las cuentas.

4.5.1 Métricas seguidas

Aunque la implementación de las mismas se explica en el *Anexo V. Documentación técnica del programador*, en este punto se explicará el proceso de obtención de la puntuación final que se muestra al usuario.

4.5.1.1 Puntuación inicial

En primer lugar, se obtiene una puntuación en base a tres parámetros fáciles de obtener, que conformarán entre sí una puntuación inicial más significativa. Cada uno de estos parámetros obtiene un puntaje sobre 100. Estos parámetros son:

- Redes sociales (RS) alternativas que muestre el usuario en su perfil. Así, obtendrá mayor puntuación cuanto más impacto tengan las redes sociales que ofrezca.
- Verificado (V). Esta propiedad supone que *Twitter* reconoce al usuario como persona pública reconocida, por lo que se valora directamente con los cien puntos que un perfil esté verificado. En caso contrario, no se llevará puntaje alguno en este parámetro.
- Seguidores (S). En función de la cantidad de seguidores se ofrecerá también una mayor o menor puntuación. Para ello se ha creado la siguiente clasificación:

Tipo de perfil	Seguidores	Puntaje sobre 100
Usuario común	Menos de 5000	0
Micro-influencer	Entre 5000 y 25.000	15
Influencer pequeño	Entre 25.000 y 100.000	30
Influencer grande	Entre 100.000 y 1.000.000	50
Macro-influencer	Entre 1.000.000 y 7.000.000	75
Celebridad	Más de 7.000.000	100

Tabla 20 Categorías de perfil en función de los seguidores

Tras conocer el valor de cada uno de estos parámetros, se calcula la puntuación inicial atendiendo a la siguiente métrica:

$$P_{inicial} = w_{rs} * RS + w_v * V + w_f * S$$

Ecuación 1 Puntuación inicial

Donde w_x representa los pesos de las distintas métricas de la media ponderada. Estos pesos deben sumar 1 para que la puntuación inicial se mantenga sobre 100 y pueden editarse en el fichero *resources/utills.py* que recoge todos los pesos.

4.5.1.2 Puntuación intermedia

Una vez se tiene la puntuación inicial, se calculan métricas más complejas para obtener una segunda puntuación más completa al tomar en cuenta un número mayor de factores. Las métricas intermedias son:

- Frecuencia de posteo (FP). Premiando un nivel intermedio que aboga por no hacer ni muchas ni pocas publicaciones diarias, persiguiendo no aburrir a su *target* ni dejarlo colgado.
- Relevancia (R). Se da un mayor puntaje a aquellos usuarios con un mayor porcentaje de similitud entre el *Trending Topic* español y los términos más frecuentes de sus *Tweets*.
- Participación (P). Se define como el número de interacciones (véase *retweets* o favoritos) por usuario. Se calcula sobre los tweets que comprenden el cuarto más nuevo y el décimo más nuevo. Ya que los tres primeros se consideran en crecimiento y a partir del décimo puede no representar una participación reciente.

Junto a la puntuación inicial, a los valores sobre 100 obtenidos se le aplican los pesos asociados a la métrica correspondiente como en el paso anterior, calculando una nueva media ponderada:

$$P_{intermedia} = w_{P_{inicial}} * P_{inicial} + w_{fp} * FP + w_r * R + w_p * P$$

Ecuación 2 Puntuación intermedia

4.5.1.3 Puntuación final

Por último, a la puntuación obtenida se le aplica un filtro con el sentimiento obtenido a partir de las respuestas de sus *tweets*. Para ello, se comprueba el sentimiento predominante y se aplica una función acorde a la siguiente tabla:

Sentimiento predominante	Función
Positivo	Se suma un 15% de la puntuación obtenida
Neutro	No se hace nada
Negativo	Se resta un 15% de la puntuación obtenida

Tabla 21 Filtro de sentimiento aplicado a la puntuación

4.5.2 Analizador de sentimiento

Para la creación del modelo que utiliza el analizador de sentimiento se han utilizado las siguientes herramientas:

4.5.2.1 Exploración de los datos

Antes de utilizar los datos, se debe comprobar que estos son válidos y hacer las modificaciones convenientes de lo contrario. Para ello, se han utilizado las siguientes plataformas y módulos.

4.5.2.1.1 Kaggle

Kaggle [44], una subsidiaria de *Google LLC*, es una comunidad en línea de científicos de datos y profesionales del aprendizaje de máquinas. *Kaggle* permite a los usuarios encontrar y publicar conjuntos de datos [45][46], explorar y construir modelos en un entorno de ciencia de datos basado en la web, trabajar con otros científicos de datos e ingenieros de aprendizaje de máquinas, y participar en competiciones para resolver los desafíos de la ciencia de datos.

4.5.2.1.2 Jupyter Notebook

Los *Notebook* son documentos producidos por la aplicación *Jupyter Notebook* [47] y contienen tanto código informático, por ejemplo *Python*, como elementos de texto enriquecido, por ejemplo texto plano, ecuaciones, figuras, enlaces, etc.

Son documentos tanto legibles y exportables a formato PDF como ejecutables, lo que facilita mucho el análisis de datos al poder ejecutar código e ir mostrando los resultados poco a poco.

4.5.2.1.3 Pandas

Pandas [48] es una herramienta de análisis y manipulación de datos de código abierto rápida, potente, flexible y fácil de usar, construida sobre el lenguaje de programación *Python*. Utiliza estructuras como *Series* o *DataFrames*, con funcionalidades que facilitan enormemente trabajar con los *datasets* de entrenamiento.

4.5.2.1.4 Matplotlib

Matplotlib [49] es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o *arrays* en el lenguaje de programación *Python* y su extensión matemática *NumPy* [50]. Gracias a esta librería se puede ver, por ejemplo, la relación entre cantidad de frases positivas, negativas y neutras de un rápido vistazo.

4.5.2.2 Preprocesamiento

Una vez se tiene un conjunto de datos aceptable, se somete a un preprocesamiento en el que se eliminan valores nulos, convierte el texto a minúscula, se obtienen los lexemas... Todo para reducirlo a una serie de tokens que contengan el significado. Se pretende que la lista de tokens sea lo más significativa posible. Para ello se ha hecho uso de la siguientes técnicas y librerías

4.5.2.2.1 NLTK

Natural Language ToolKit [51] es una plataforma líder para construir programas en *Python* que trabajen con datos del lenguaje humano. Ofrece interfaces fáciles de usar para más de 50 *corpus* y recursos léxicos como *WordNet* [52], junto con un conjunto de librerías de procesamiento de texto para la clasificación, la *tokenización*, el *stemming*, el etiquetado, el análisis sintáctico y el razonamiento semántico, *wrappers* para librerías de *NLP* de uso industrial y un foro de debate activo.

4.5.2.2.2 SpaCy

SpaCy [53] es una biblioteca para el procesamiento avanzado del lenguaje natural en *Python*. Se basa en las investigaciones más recientes y fue diseñada desde el primer día para ser utilizada en productos reales.

Viene con pipelines preentrenados y actualmente soporta la *tokenización* y el entrenamiento de más de 60 idiomas. Cuenta con modelos de velocidad y redes neuronales de última generación para el etiquetado, el análisis sintáctico, el reconocimiento de entidades con nombre, la clasificación de textos y mucho más, el aprendizaje multitarea con transformadores preentrenados.



Ilustración 24 Rótulo de SpaCy

4.5.2.2.3 Eliminación de caracteres ajenos

Las frases preprocesadas son sometidas a una serie de filtros utilizando expresiones regulares con el objetivo de eliminar:

- Enlaces a otras páginas.
- Caracteres de control o de HTML

4.5.2.2.4 Eliminación de stop-words

Cuando se trabaja con lenguaje natural hay multitud de palabras que son muy frecuentes, pero no portan ningún sentimiento, como pueden ser preposiciones o conjunciones. Aunque no solo estas, también muchos verbos, sustantivos y adjetivos tienen ausencia de sentimiento y pueden provocar que el algoritmo aprenda incorrectamente. Para evitarlo, se utilizan listas predefinidas de *stop-words* en función del idioma, controlando que no aparezcan en los datos de entrenamiento [54].

4.5.2.2.5 Lematización

Del conjunto inicial de datos se pretende extraer una lista de tokens que contengan el sentimiento de la oración. Además, se pretende que esta lista de tokens sea concreta y significativa. Es decir, que las palabras “odiaron”, “odian” y “odio” se interpreten como un único concepto y no como tres distintos. Para ello se hace uso de la lematización, que permite obtener la raíz de las palabras [55].

4.5.2.2.6 One-hot encoding

Puesto que las etiquetas de las oraciones no son números sino cadenas de caracteres que expresan una idea, se deben transformar estas etiquetas a un lenguaje que pueda entender la máquina. Así, el *one hot encoding* es un proceso de conversión de variables de datos categóricos para que puedan proporcionarse a los algoritmos de aprendizaje automático para mejorar las predicciones. Esta técnica es una parte crucial de la ingeniería de características para el aprendizaje automático [56].

4.5.2.3 Extracción de características y entrenamiento

Una vez se tiene un conjunto de oraciones procesadas, se deben extraer las características de las mismas y proporcionar el resultado a un algoritmo de clasificación para que aprenda.

4.5.2.4 Scikit-Learn

Scikit-learn [57] es una biblioteca de aprendizaje automático de software libre para el lenguaje de programación Python. Presenta varios algoritmos de clasificación, regresión y agrupación,

incluyendo máquinas de vectores de apoyo, bosques aleatorios, refuerzo de gradiente, *k-means* y *DBSCAN*, y está diseñado para interoperar con las bibliotecas numéricas y científicas de *Python* *NumPy* y *SciPy*.

4.5.2.5 TF-IDF

TF-IDF [58] significa frecuencia de términos-frecuencia de documentos inversa (*Term Frequency-Inverse Document Frequency*), y el peso *TF-IDF* es un peso que se utiliza a menudo en la recuperación de información y la minería de textos. Este peso es una medida estadística utilizada para evaluar la importancia de una palabra en un documento de una colección o *corpus*. La importancia aumenta proporcionalmente al número de veces que una palabra aparece en el documento, pero se compensa con la frecuencia de la palabra en el *corpus*.

El *TF-IDF* puede utilizarse con éxito para filtrar las palabras clave en varios campos temáticos, como la clasificación y el resumen de textos.

4.5.2.6 Random Forest Classifier

El bosque aleatorio (*Random Forest*) es un método de conjunto que combina múltiples árboles de decisión para clasificar, por lo que el resultado del bosque aleatorio suele ser mejor que el de los árboles de decisión.

Los bosques aleatorios son un algoritmo de aprendizaje supervisado. Puede utilizarse tanto para la clasificación como para la regresión. También es el algoritmo más flexible y fácil de usar. Un bosque se compone de árboles. Se dice que cuantos más árboles tiene, más robusto es un bosque. Los bosques aleatorios crean árboles de decisión sobre muestras de datos seleccionadas al azar, obtienen la predicción de cada árbol y seleccionan la mejor solución mediante una votación. También proporciona un indicador bastante bueno de la importancia de las características.

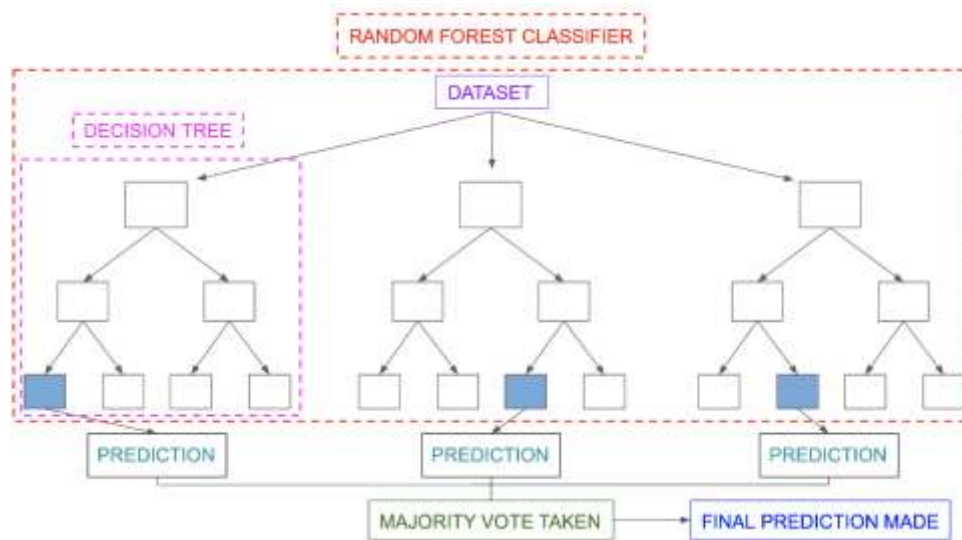


Ilustración 25 Esquema de un Random Forest Classifier

Ventajas:

- Mayor precisión que la mayoría de los clasificadores no lineales.
- Algoritmo muy robusto gracias a los múltiples árboles de decisión que incorpora.
- No enfrenta problemas de *overfitting*, ya que al tomar la media de todas las predicciones anula los sesgos, solucionando el problema
- La posibilidad de utilizarlo tanto en regresión como en clasificación le hace muy versátil

Desventajas:

- Es bastante más lento que otros algoritmos de clasificación, ya que hace uso de varios árboles de decisión.
- Esta lentitud puede provocar que no sea posible usarlo en tiempo real.

Es más difícil interpretar el modelo de un *RFC* que de un solo árbol de decisión [59].

4.5.2.7 GridSearch

La validación cruzada es una técnica de ajuste que intenta calcular los valores óptimos de los hiperparámetros. Es una búsqueda exhaustiva que se realiza sobre los valores específicos de los parámetros de un modelo. Esta técnica puede tardar bastante tiempo ya que prueba cada posible combinación de hiperparámetros que se implemente [60].

4.6 TÉCNICAS Y HERRAMIENTAS DEL MÓDULO DE VISUALIZACIÓN

Se lista a continuación todas las herramientas utilizadas en la implementación del módulo de visualización, es decir, la página web.

4.6.1 Frontend

4.6.1.1 HTML5

Se puede definir *HTML5* como un estándar que sirve para definir la estructura y el contenido de una página Web.

Sirve como referencia del *software* que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado *HTML*) para la definición de contenido de una página web, como texto, imágenes, vídeos, juegos, entre otros [61].

4.6.1.2 CSS y Bootstrap

Como *HTML*, *CSS* [62] (*Cascading Style Sheets*) u hojas de estilo en cascada en español, no es realmente un lenguaje de programación, tampoco es un lenguaje de marcado. Es un lenguaje de hojas de estilo, es decir, permite aplicar estilos de manera selectiva a elementos en documentos *HTML*.

Bootstrap [63], por su parte, es una colección gigantesca de útiles y fragmentos de código reutilizables escritos en *HTML*, *CSS* y *JavaScript*. Es también un *framework* de *frontend* que permite a los desarrolladores crear sitios web responsivos rápidamente. Es decir, *Bootstrap* ahorra escribir bastante código y es gratis. La versión utilizada es la cuarta, ya que la quinta aún no es tan estable.

4.6.1.3 FontAwesome

Font Awesome [64] es un conjunto de herramientas de fuentes e iconos basado en *CSS* y *Less*. A partir de 2020, *Font Awesome* fue utilizado por el 38% de los sitios que utilizan scripts de fuentes de terceros, colocando a *Font Awesome* en segundo lugar después de *Google Fonts* [65]

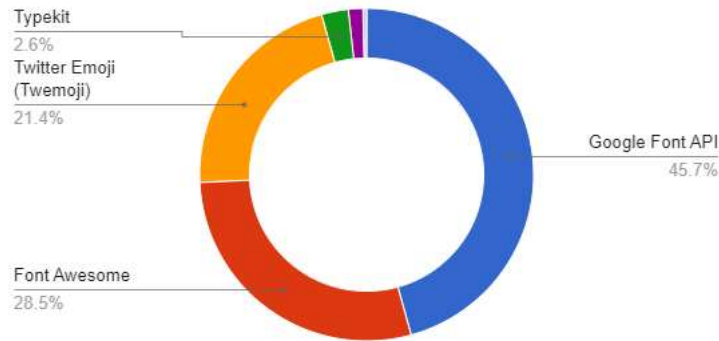


Ilustración 26 Liberías de fuentes e iconos más utilizados en Internet en 2020

4.6.1.4 Bootswatch

Boostwatch [66] es una plataforma que ofrece una serie de plantillas para *Bootstrap* que modifican parámetros como la tipografía, forma de botones, colores primarios, etc. Ofrece formas sencillas de utilizar las plantillas, como son las redes de entrega de contenido (*CDN*). Estas son un conjunto de servidores distribuidos geográficamente que trabajan juntos buscando la transferencia rápida de contenido en Internet.

4.6.2 Backend

4.6.2.1 Node.js

Node.js [20] es un entorno de ejecución JavaScript de código abierto y multiplataforma. Es una herramienta popular para casi cualquier tipo de proyecto. Ejecuta el motor V8 de *JavaScript*, el núcleo de *Google Chrome*, fuera del navegador. Esto permite que *Node.js* tenga un gran rendimiento. Una aplicación *Node.js* se ejecuta en un solo proceso, sin crear un nuevo hilo para cada solicitud. *Node.js* proporciona un conjunto de primitivas de E/S asíncronas en su biblioteca estándar que evitan que el código *JavaScript* se bloquee y, en general, las bibliotecas en *Node.js* están escritas utilizando paradigmas de no bloqueo, haciendo que el comportamiento de bloqueo sea la excepción y no la norma.

4.6.2.2 Express.js

Express [67] es un *framework* de aplicaciones web *Node.js* mínimo y flexible que proporciona un sólido conjunto de características para aplicaciones web y móviles.

4.6.2.3 TypeScript

TypeScript [21] es un lenguaje de código abierto que se basa en JavaScript, una de las herramientas más utilizadas del mundo, añadiendo definiciones de tipos estáticos. Los tipos proporcionan una manera de describir la forma de un objeto, proporcionando una mejor documentación, y permitiendo a *TypeScript* validar que su código está funcionando correctamente.

4.6.2.4 Nodemon

Nodemon [68] es una utilidad perfecta para el desarrollo que vigila cualquier cambio en el código fuente y reiniciará automáticamente el servidor.

4.6.2.5 Amcharts

La biblioteca *JavaScript* de *amCharts* [69] admite gráficos clásicos como los de líneas, áreas, columnas, barras, tarta, XY, dispersión, velas y OHLC, así como otros más "exóticos" como los diagramas de *Gauges*, *Funnels*, *Gantt*, *Chord* y *Sankey*, *TreeMap* y mapas del mundo o del país totalmente interactivos y adaptables. Los productos totalmente funcionales están disponibles de forma gratuita para cualquier tipo de uso, así como en forma comercial de pago para todo tipo de escenarios de uso.

4.6.2.6 Passport

Passport [70] es un middleware de autenticación para *Node.js*. Extremadamente flexible y modular, *Passport* puede incorporarse a cualquier aplicación web basada en *Express*. Un amplio conjunto de estrategias soporta la autenticación utilizando un nombre de usuario y una contraseña, *Facebook*, *Twitter*, y más.

4.6.2.7 Otros módulos

Además de los listados anteriormente, que se pueden considerar como principales, se ha hecho uso de:

- *bcrypt*: para encriptado de contraseñas [71]
- *connect-flash*: para enviar mensajes flash entre páginas al redirigir [72]
- *fs*: para acceder a ficheros [73]
- *https*: para crear un servidor https [74]
- *morgan*: para seguir el flujo de trabajo de la página [75]
- *request*: para hacer peticiones a la API de Social Sentiment [76]

- `serve-favicon`: para utilizar el `favicon.ico` de la página [77]
- Tipos correspondientes, `@types`, de los middlewares utilizados para poder funcionar con `TypeScript`.

4.7 TÉCNICAS Y HERRAMIENTAS DEL MÓDULO SUPERVISOR

En el caso del supervisor, su funcionamiento se basa en el módulo de `requests` para `Python` y la utilidad `crontab`.

4.7.1 Request

Este módulo [78] permite enviar peticiones HTTP de forma muy sencilla. No es necesario codificar los datos `PUT` y `POST` ni añadir manualmente cadenas de consulta, solo se necesita utilizar el formato `JSON`.

Es uno de los paquetes de `Python` más descargados con unos 14 millones de descargas a la semana, siendo utilizado en más de medio millón de repositorios en `GitHub`.

4.7.2 Crontab

`Crontab` [79] es el nombre del programa utilizado para manejar tablas que usará el demonio `cron`. Un archivo `crontab` especifica, en lenguaje humano, “ejecuta X comando en Y momento”. Cada usuario puede tener un `crontab` propio y están pensados para ser editados utilizando las utilidades del propio `crontab`.

```
$ crontab -u alvaro -e # Selecciona el usuario para editar el crontab
$ crontab -l # Lista las entradas del crontab
$ crontab -e # Edita el crontab para el usuario logeado.
```

Ilustración 27 Utilidades básicas de `crontab`

El formato utilizado en las tablas de `cron` es el siguiente:

```
* * * * * Comando a ejecutar
|   |   |   |   |
|   |   |   |   | Dia de la semana ( 0 - 6 ) ( Domingo = 0 )
|   |   |   |   |
|   |   |   |   | Mes ( 1 - 12 )
|   |   |   |   |
|   |   |   |   | Dia del mes ( 1 - 31 )
|   |   |   |   |
|   |   |   |   | Hora ( 0 - 23 )
|   |   |   |   |
|   |   |   |   | Minuto ( 0 - 59 )
```

Ilustración 28 Formato de `crontab`

5 ASPECTOS RELEVANTES

Se recogen en este apartado algunos de los aspectos más relevantes en cuanto a la gestión y al desarrollo del proyecto. Los anexos entregados junto a este documento relatan con más detalle los siguientes temas:

Anexo	Tema tratado
Anexo I	Planificación temporal
Anexo II	Especificación de los requisitos
Anexo III	Análisis de los requisitos
Anexo IV	Diseño del sistema
Anexo V	Documentación técnica
Anexo VI	Manual de usuario

Tabla 22 Anexos

5.1 PLANIFICACIÓN TEMPORAL

Pese a que en un primer momento se intentó realizar la planificación temporal haciendo uso de los métodos tradicionales vistos durante el transcurso del grado, la naturaleza del proyecto (horarios variables, incertidumbre en las tecnologías por utilizar y su tiempo de investigación, posibles retrasos, otras asignaturas...) se decidió utilizar una metodología ágil [80]. La razón es que estas permiten un cálculo más realista de los tiempos dedicados a cada actividad.

ID	Nombre	Tiempo Estimado	Tiempo real	Importancia	Estado
1	Sprint 1- Fase de iniciación	15	15	10	✓
1.1	Definición de la visión del proyecto	2	2	10	✓
1.2	Definición del equipo de trabajo	1	1	7	✓
1.3	Desarrollo del "Product Backlog"	5	5	10	✓
1.4	Plan de lanzamiento	2	2	9	✓
1.5	Anexo I	3	3	10	✓

2	Sprint 2- Fase de planificación	15	15	10	✓
2.1	Definición de objetivos	3	3	10	✓
2.2	Identificación de los casos de uso	3	3	10	✓
2.3	Identificación de las funcionalidades del sistema	4	5	10	✓
2.4	Investigación sobre métodos de persistencia, ingesta y análisis	5	5	8	✓
2.5	Anexo II	5	6	10	✓
3	Sprint 3 – Fase de implementación I	15	15	10	✓
3.1	Descomposición del dominio del problema	3	3	8	✓
3.2	Módulos de análisis y servicios	3	4	8	✓
3.3	Diagrama de módulos	2	2	8	✓
3.4	Modelo de análisis	3	4	8	✓
3.5	Bot de <i>Telegram</i>	10	8	10	✓
3.6	Estructura básica de la API y módulos de persistencia e ingesta	10	10	10	✓
3.7	Anexo III	15	15	10	✓
4	Sprint 4 – Fase de implementación II	15	20	10	✓
4.1	Modelo de diseño	3	3	9	✓
4.2	Vista arquitectónica	3	3	10	✓
4.3	Implementación de los módulos de estadísticas y finalización de la API	10	10	10	✓
4.4	Implementación del módulo de visualización y el supervisor	15	20	10	✓
4.5	Anexo IV	10	10	10	✓
5	Sprint 5 – Fase de revisión	15	10	10	✓
5.1	Documentación técnica del sistema	5	5	10	✓
5.2	Estructura del sistema	3	3	8	✓
5.3	Despliegue del sistema	5	5	10	✓
5.4	Anexo V	15	10	10	✓

6	Sprint 6 – Fase de lanzamiento	15	15	10	✓
6.1	Documentación práctica del sistema	2	2	10	✓
6.2	Pruebas del sistema	10	10	10	✓
6.3	Colgar versión final	2	2	10	✓
6.4	Retrospectiva y conclusiones	2	2	9	✓
6.5	Anexo VI	3	3	10	✓
6.5	Revisión de la documentación	5	5	10	✓

Tabla 23 Product Backlog

En la figura superior se muestra el *Product Backlog* de la planificación utilizando *Scrum*, aunque se explica con más detalle en el *Anexo I, Planificación temporal*

5.2 ARQUITECTURA Y DISEÑO

La característica que se ha perseguido principalmente al diseñar la arquitectura es la modularidad. Se pretende que la plataforma incorpore módulos especializados independientes entre sí, de forma que sea la API central quien los gestione.

La idea detrás de esta decisión es poder responder a los cambios fácilmente. El software está en constante desarrollo y, al depender en gran parte de librerías externas y no tener excesivo tiempo para modificar diseños, es bueno que los cambios no tengan gran impacto en la arquitectura. De este modo, se puede responder a problemas como podría ser librerías no disponibles al cambiar de host, versiones no actualizadas, cambios en la forma de utilizar las librerías. Por ejemplo, si el módulo utilizado para utilizar la API de Twitter quedara desactualizado respecto a algún cambio en la API, podríamos sustituirlo sin afectar al resto de la plataforma.

Respondiendo a la arquitectura planteada, la plataforma puede dividirse en seis módulos principales, expuestos en los objetivos principales del sistema. Estos son:

- Módulo de la API: Parte troncal del sistema.
- Módulo de ingesta: Encargado de obtener la información necesaria de Twitter.
- Módulo de persistencia: Encargado de la base de datos no relacional.
- Módulo de análisis: Encargado de los cálculos y de aplicar el análisis de sentimiento.
- Módulo de visualización: Consistente en una página web que se comunica con la API.
- Módulo supervisor: Encargado de las actualizaciones periódicas de la base de datos.

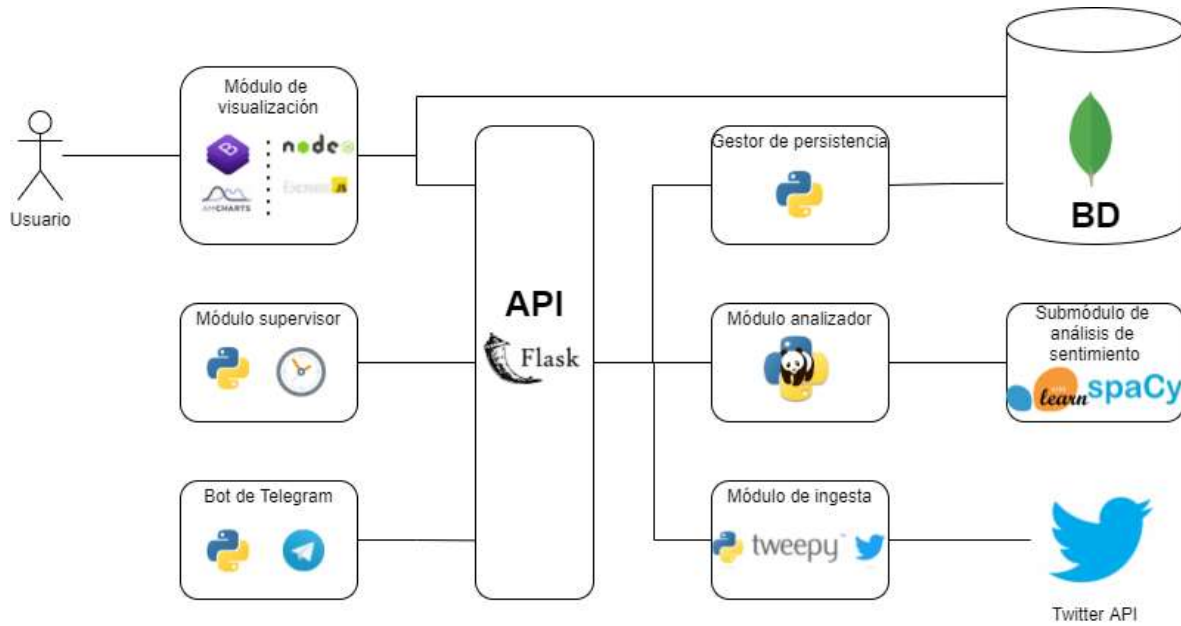


Ilustración 29 Arquitectura del proyecto

Además de estos módulos principales, también se debe tener en cuenta el módulo del bot de *Telegram* utilizado para la construcción de un conjunto de datos de entrenamiento para el modelo de Machine Learning. Este se despliega en la misma máquina que la plataforma, el funcionamiento de la misma es totalmente independiente del bot.

El funcionamiento de este bot se basa en una manejadora conversacional que por detrás recupera o modifica una base de datos especial para él. Básicamente muestra frases al usuario, que las cataloga para guardarlas en distintas colecciones. Cuando la batería de frases a mostrar es baja, se puede rellenar de nuevo especificando palabras clave.



Ilustración 30 Arquitectura del bot de Telegram

5.3 BOT DE TELEGRAM

Cuando se quiere entrenar un modelo orientado al análisis de sentimiento, uno se encuentra en el paradigma del aprendizaje supervisado. Concretamente nos encontramos ante un problema de clasificación, donde el módulo responsable obtendrá un texto como entrada y devolverá una etiqueta positiva, negativa o neutra como salida.

Para que un modelo dentro de este paradigma aprenda, es necesario dotarle primero de un conjunto de datos etiquetados que sirvan de ejemplo. Como ya se ha explicado, se pretende generar un conjunto de datos de aprendizaje entre los datos recopilados por la plataforma y otros conjuntos ya existentes.

Por tanto, es necesario implementar un espacio donde catalogar frases cómodamente. Se llega así a dos alternativas: crear este espacio como una página web normativa con sus códigos HTML/CSS/JS o crearlo dentro de una plataforma ya existente. A la hora de decidirse, es importante tener en mente los recursos y el tiempo necesarios, la importancia de este espacio y el público al que irá dirigido.

En primer lugar, crear una página web implica hacer un diseño previo de la misma, programar el backend, las vistas que verá el usuario y otras tareas que requieren cierto tiempo. Si bien es cierto que, una vez finalizada, la página es más accesible que el bot de *Telegram* (que implica descargar *Telegram*), hay que tener en cuenta que clasificar frases es un trabajo constante que puede resultar pesado para el resto de gente ajena al proyecto. Esto implica que, por desgracia, se estimó que no se obtendría mucha ayuda.

Así, teniendo en mente que la gente colaborará de forma constante no tendrá problema en descargar *Telegram*, se optó por implementar este espacio a través de un *bot*.

Influyen también hechos como:

- La experiencia previa programando más *bots* que páginas web.
- La capacidad de usar *Python*, que es más cómodo que *Node.js*
- Las funcionalidades más que suficientes que ofrece la API de *Telegram*

5.3.1 Diseño

La arquitectura del *bot* es muy simple: consta de un *script* en *Python* con toda la funcionalidad del *bot*. Este se conecta con la base de datos, permitiendo añadir o borrar frases de las distintas colecciones.

La base de datos, por su parte, es una BD de *MongoDB* llamada **tg_dataset**. En esta base podemos encontrar cuatro colecciones:

- **Unlabeled**, cuyos documentos son meras frases obtenidas de *Twitter*
- **Invalid**, donde van aquellas frases inválidas.
- **Labeled**, donde van aquellas frases procedentes de *Unlabeled* tras ser votadas.
- **Dataset**, que recoge la colección definitiva de frases.

La separación en cuatro colecciones sigue la siguiente lógica:

- En primer lugar, las frases recién *scraped*, sin voto, se guardan en *Unlabeled* a espera de ser votadas
- Las frases etiquetadas con “Positivo”, “Negativo” o “Neutro” pasan a la colección *Labeled*, mientras que las inválidas (aquellas que no aportan ningún sentimiento completo), se pasan a la colección *Invalid*.
- Estas colecciones intermedias existen dado que en esta parte del proyecto puede participar gente que conozco o, simplemente, puede votar gente que no ha acabado de entender lo que tiene que hacer. Así, tanto *Labeled* como *Invalid* permiten al administrador una revisión para determinar si dar un voto válido o borrarlas (en caso de las inválidas) y para aceptarlas, eliminar su etiqueta o borrarlas directamente (en caso de las etiquetadas)
- Si las frases son aceptadas, se guardan finalmente en la colección *Dataset*.

Aun así, dado el gran volumen de datos es necesario un programa que se ejecute periódicamente (en este caso mediante un *crontab* para los lunes a las 7:30 de la mañana) y que rellene la colección de *Unlabeled* (en caso de estar bajo mínimos) y que acepte y borre las frases de *Labeled* e *Invalid*, respectivamente.

Como se explicará posteriormente, se utiliza para el *bot* una estructura conversacional [81], caracterizada por establecer estados y varias manejadoras para cada estado. De esta forma, se establece el siguiente mapa de estados:

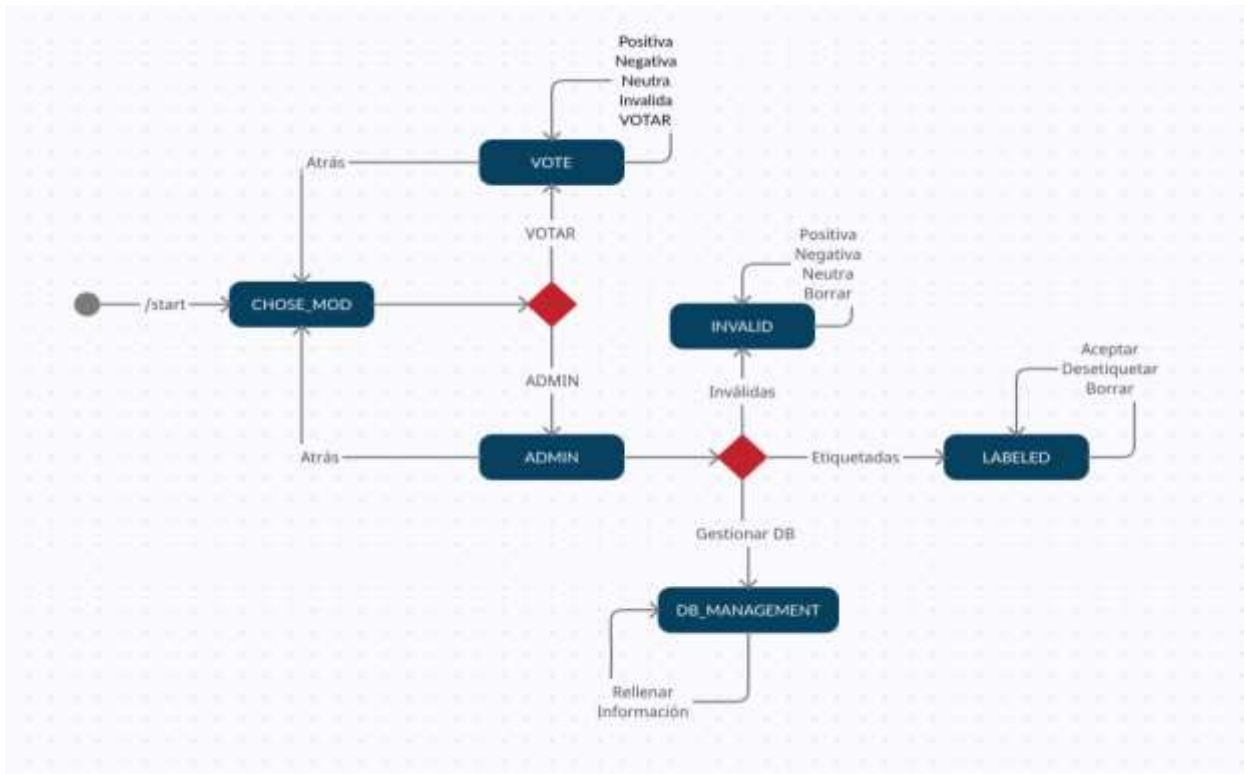


Ilustración 31 Diagrama de estados de la manejadora conversacional del bot

Definimos así lo siguientes estados con la funcionalidad descrita:

- **Punto de entrada.** Cuando el usuario inicia el *bot*, se ejecuta automáticamente */start*, lo que muestra un mensaje de bienvenida y ofrece votar o acceder al panel de administrador. Podemos acceder además a ejemplos de votos.

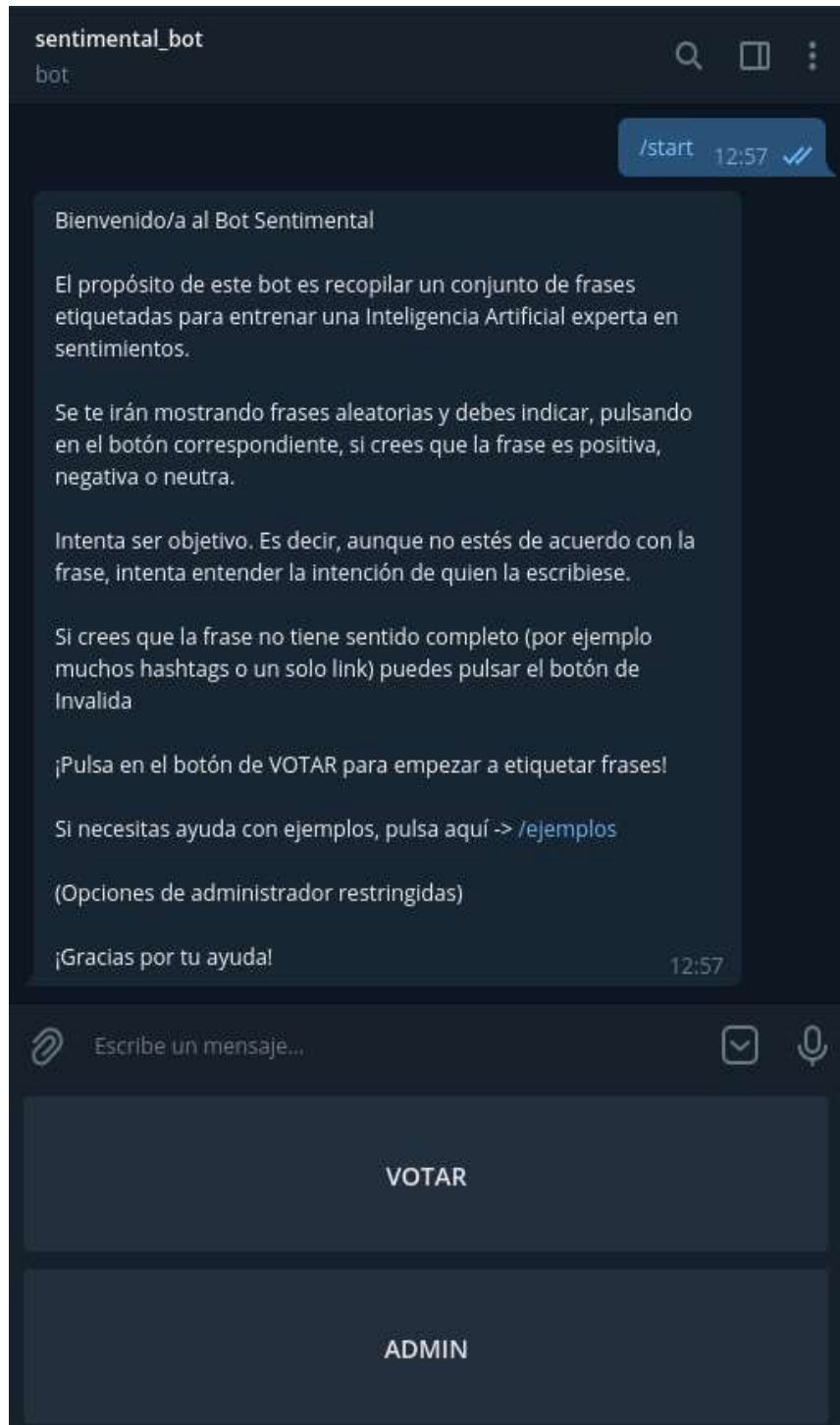


Ilustración 32 BOT: Pantalla de inicio

- **CHOSE_MOD.** En este estado se ingresa a las pantallas de voto o al panel del administrador, en función del botón pulsado. Es el estado al que se ingresa tras el comando */start*

- **VOTE.** En este estado se ejecuta `vote_sentence()`, que mostrará al usuario tweets sin etiquetar de la colección *Unlabeled*. Se seguirán enviando frases indefinidamente. El usuario puede acceder a ejemplos o a la pantalla anterior en cualquier momento.

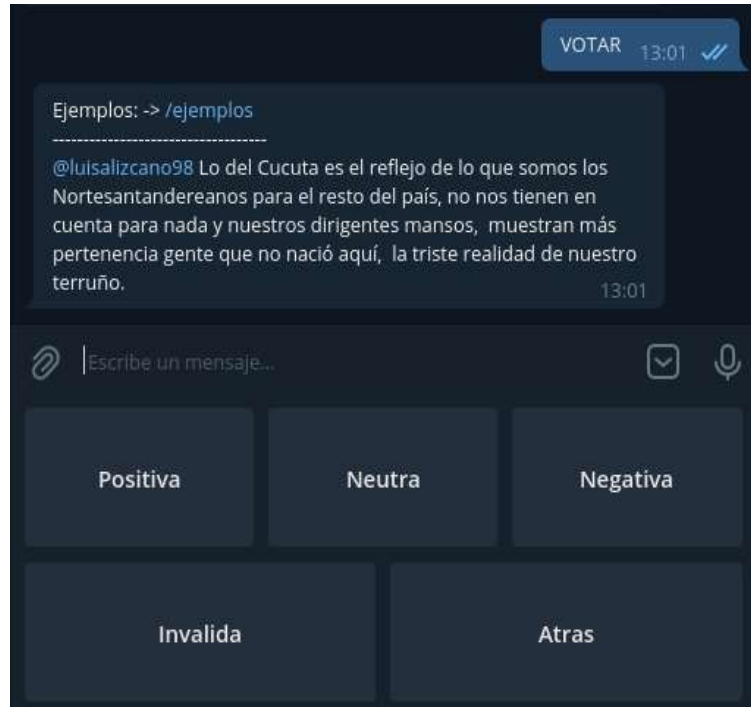


Ilustración 33 BOT: Ejemplo de votación

Como se puede ver, los botones proporcionan las opciones de voto (positiva, negativa, neutra, que guardarán las frases en *Labeled*), la opción “*Inválida*” (que la guardará en *Invalid*) y la opción “*Atrás*” para volver a la pantalla anterior en caso de, por ejemplo, querer acceder al panel del administrador

- **ADMIN.** Solo se puede acceder a este panel en caso de ser el usuario `@alvarito_lml`. Esto se controla gracias a un código único de usuario que no cambia y no es secreto.

Como se puede ver en la figura, las opciones desplegadas corresponden a los modos de revisión y a una gestión manual de la base de datos.



Ilustración 34 BOT: Panel de administrador

- **INVALID.** Este estado corresponde a uno de los métodos de revisión manual en el que se muestran las frases que se han etiquetado como inválidas. Ofrece botones para establecer un voto que resultará definitivo y se enviará a la colección *Dataset* o, si se desea, eliminar el *tweet* definitivamente.

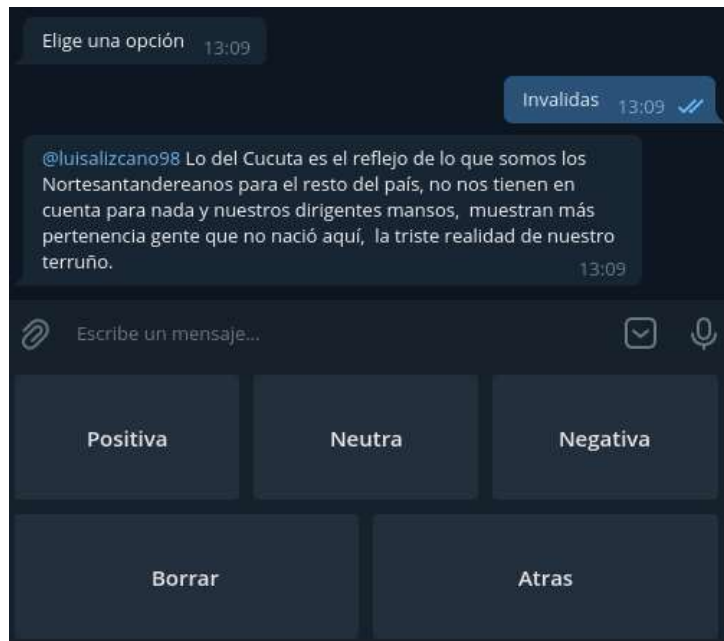


Ilustración 35 BOT: Gestión de oraciones inválidas

- **LABELED.** El caso de este estado es análogo a *Invalid*. Representa uno de los modos de revisión. En él se muestran frases que han sido etiquetadas como Positiva, Negativa o Neutra y se ofrecen botones para:
 - Aceptar el voto definitivamente y enviar la frase a la colección *Dataset*
 - Desetiquetar la frase y que vuelva a la colección *Unlabeled*
 - Eliminar la frase definitivamente. No pasa por la colección *Invalid* ya que no tendría sentido revisar la frase dos veces.



Ilustración 36 BOT: Gestión de oraciones etiquetadas

- **DB_MANAGEMENT.** Puesto que la batería de sentencias no es infinita, se puede rellenar manualmente especificando el número de tweets a recuperar y un conjunto de *keywords* con la forma *NUMERO@KEYWORDS*. Por ejemplo: *20@felicidad* recuperaría veinte tweets acerca del término “*felicidad*”.

Además, permite consultar el estado de la base de datos con el botón “*Información*”. Pese a que esto es fácilmente consultable desde, por ejemplo, un gestor gráfico, se decidió implementar esta funcionalidad para poder consultar el estado de la base de datos en cualquier momento, ya que esta se debe rellenar manualmente.

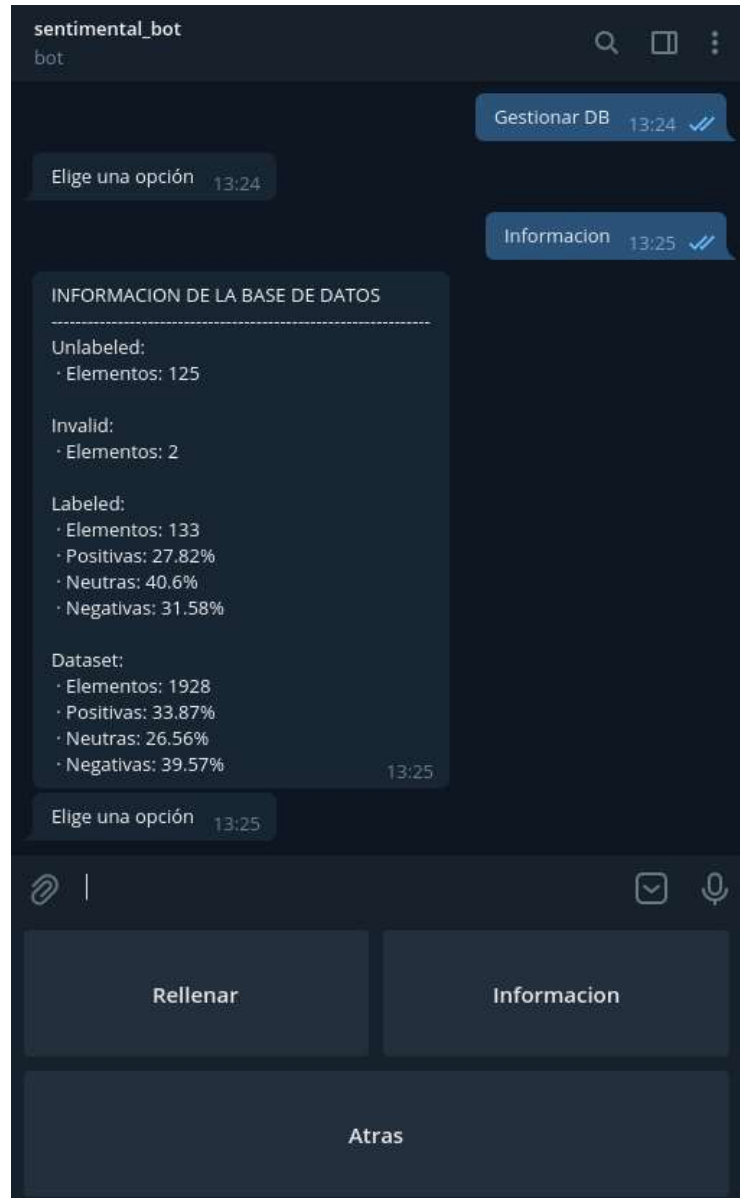


Ilustración 37 BOT: Información de la base de datos

5.3.2 Implementación

La API que ofrece *Telegram* para desarrollar *bots* es muy completa, pero a la hora de desarrollar no resulta cómoda para programar. Por suerte, existe un módulo de Python llamado *python-telegram-bot* [81], que hace de *wrapper* para esta API y facilita mucho las cosas. Además, cuenta con una documentación extensa que nunca deja lugar a dudas, incorporando ejemplos de las distintas funcionalidades. Para instalar dicho módulo solamente es necesario utilizar *pip* de la siguiente forma:

```
pip3 install python-telegram-bot
```


Junto a este módulo se ha utilizado también *tweepy* y *pymongo*, para recuperar tweets y para gestionar la base de datos, respectivamente. Sin embargo, se encuentran documentados en los apartados correspondientes (Técnicas y herramientas)

Antes de escribir el código con las funcionalidades del *bot*, se debe registrar al mismo. Para ello existe el *@BotFather*, un *bot* oficial de *Telegram* que nos proporcionará un *Token* con el que identificar nuestro *bot*. A mayores nos ofrece herramientas para modificar los comandos mostrados al usuario, cambiar el nombre o descripción del *bot*, etc.



Ilustración 38 Logo de The Botfather

Simplemente se debe buscar en la propia aplicación de *Telegram* al usuario *@BotFather* y ejecutar el comando */newbot*. A partir de este punto el proceso para conseguir el token es un formulario trivial con datos identificativos del *bot*.

De esta forma, se registra *@sentimental_tw_bot*.

En los *bots* de *Telegram* podemos encontrar distintos tipos de manejadoras (de comandos, de mensajes, conversacionales...) escogiendo una manejadora conversacional, que ofrece una gran facilidad para diseñar las funcionalidades requeridas. Estas manejadoras utilizan los estados. Así, para un mismo estado se definen distintas manejadoras. Esto permite gestionar la entrada del usuario en función de la pantalla en la que nos encontremos:

```

conv_handler = ConversationHandler(
    entry_points=[CommandHandler('start', bot_start)],
    states={
        CHOSE_MOD: [MessageHandler(Filters.regex('^VOTAR$'), vote_sentence), # -> VOTE
                    MessageHandler(Filters.regex('^ADMIN$'), admin_options), # -> ADMIN
                    MessageHandler(Filters.regex('^Atras$'), chose_mod), # -> CHOSE_MOD
                    CommandHandler('ejemplos', vote_examples)], # -> CHOSE_MOD

        VOTE: [MessageHandler(Filters.regex('^VOTAR|Positiva|Neutra|Negativa|Invalida$'), vote_sentence), # -> VOTE
              MessageHandler(Filters.regex('^Atras$'), chose_mod), # -> CHOSE_MOD
              CommandHandler('ejemplos', vote_examples)], # -> CHOSE_MOD

        ADMIN: [MessageHandler(Filters.regex('^Invalidas$'), admin_manage_invalid), # -> INVALID
               MessageHandler(Filters.regex('^Etiquetadas$'), admin_manage_labeled), # -> LABELED
               MessageHandler(Filters.regex('^Gestionar DB$'), admin_manage_database), # -> DB_MANAGEMENT
               MessageHandler(Filters.regex('^Atras$'), chose_mod)], # -> CHOSE_MOD

        INVALID: [MessageHandler(Filters.regex('^Invalidas|Positiva|Neutra|Negativa|Invalida|Borrar$'), admin_manage_invalid), # -> INVALID
                 MessageHandler(Filters.regex('^Atras$'), admin_options)], # -> ADMIN

        LABELED: [MessageHandler(Filters.regex('^Aceptar|Desetiquetar|Borrar$'), admin_manage_labeled), # -> LABELED
                 MessageHandler(Filters.regex('^Atras$'), admin_options)], # -> ADMIN

        DB_MANAGEMENT: [MessageHandler(Filters.regex('^Rellenar|Informacion$'), admin_manage_database), # -> DB_MANAGEMENT
                       MessageHandler(Filters.regex('^Atras$'), admin_options), # -> ADMIN
                       MessageHandler(Filters.text, fill_unlabeled)],
    },
    fallbacks=[
        MessageHandler(Filters.text, bot_start)
    ],
    allow_reentry = True,
)

```

Ilustración 39 Declaración de los diferentes estados

En la figura se pueden apreciar varios ejemplos. Pongamos el estado VOTE, que corresponde a la pantalla en la cual se muestran tweets para que el usuario las catalogue. En este estado encontramos tres manejadoras:

- Una manejadora de mensajes, que establece que para los mensajes indicados (“VOTAR”, “Positiva”, “Neutra”, etc.) se ejecutará el método `vote_sentence()`,
- Una manejadora de mensajes que establece que para el mensaje “Atrás” se ejecutará el método `vote_sentence()`
- Una manejadora de comandos para mostrar ejemplos de votos, que se llamará al escribir o pulsar en `/ejemplos`

En cada uno de los métodos se comprueba qué mensaje ha escrito el usuario. En función de dicho mensaje y la función que se esté ejecutando es posible saber de dónde viene el usuario y gestionar así el texto a mostrar, si hay que persistir un voto... Cuando un método ha terminado su ejecución, devuelve el estado al que pasa la conversación.

La funcionalidad de los métodos se limita a recoger y analizar el texto escrito por el usuario y actuar en base a ello para pasar a otro estado o enviar otra frase, por lo que no se entrará en excesivo detalle. Para la interfaz gráfica, consistente en un *layout* de botones, se ha utilizado la clase `ReplyKeyboardMarkup` que ofrece el paquete de *Python de Telegram*.

Para resumir el código, se han seguido los siguientes pasos:

1. Instanciar un *Updater*, que gestiona las actualizaciones del chat. Es a este objeto al que se le proporciona el token del bot
2. Instanciar un *Dispatcher*, al que se añaden todas las manejadoras necesarias. En este caso, se añade la manejadora conversacional

```
logging.info('Iniciando bot...')
updater = Updater(token=TOKEN, use_context=True)
logging.info('Updater iniciado.')
dispatcher = updater.dispatcher
logging.info('Dispatcher iniciado.')

logging.info('Iniciando manejadoras...')
init_handlers(dispatcher)
logging.info('Manejadoras iniciadas.')

logging.info('Comenzando a sondear...')
updater.start_polling()
updater.idle()
```

Ilustración 40 Arranque del bot

3. Configurar la manejadora conversacional explicada anteriormente con los estados correspondientes, de forma que se siga el diagrama explicado en la sección del diseño.
4. Implementar los métodos necesarios. En general, difieren en las colecciones que tratan o en los mensajes que puedan recibir del usuario.

5.4 DESPLIEGUE

En este punto se explica cómo se ha desplegado el sistema en los distintos entornos.

5.4.1 Pasos de despliegue (comunes para los tres entornos)

1. Instalar si no lo está ya *MongoDB*. Para ello, en un sistema operativo basado en *Debian* que use *apt*, como es *Ubuntu*, se utilizarán las siguientes órdenes:

```

# Importar la clave pública
$ wget -qO - https://www.mongodb.org/static/pgp/server-4.4.asc | sudo apt-key add -
# Añadir el repositorio y actualizar la lista de repositorios
$ echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/4.4 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-4.4.list
$ sudo apt-get update
# Instalar MongoDB
$ sudo apt-get install -y mongodb-org

```

Ilustración 41 Instalación de MongoDB

En el caso de estar en *Manjaro*, donde se utiliza *pacman* en lugar de *apt*, se puede descargar *MongoDB* desde el repositorio *AUR* siguiendo el siguiente enlace <https://aur.archlinux.org/packages/mongodb-bin/>. Una vez descargado solo habría que hacer uso de la utilidad *makepkg* para construir el paquete.

Por último, puesto que ambos son *Linux* que usan *SystemD*, se debe habilitar y arrancar el servicio de *MongoDB* utilizando *systemctl*.

```

# Habilitar servicio
$ sudo systemctl enable mongod
# Arrancar servicio
$ sudo systemctl start mongod
# Comprobar estado del servicio
$ sudo systemctl status mongod

```

Ilustración 42 Habilitación del servicio de mongod

2. Activar el entorno virtual para cargar las variables de entorno, lanzando posteriormente la API en el puerto 5000. Para ello, se utilizan las siguientes órdenes:

```

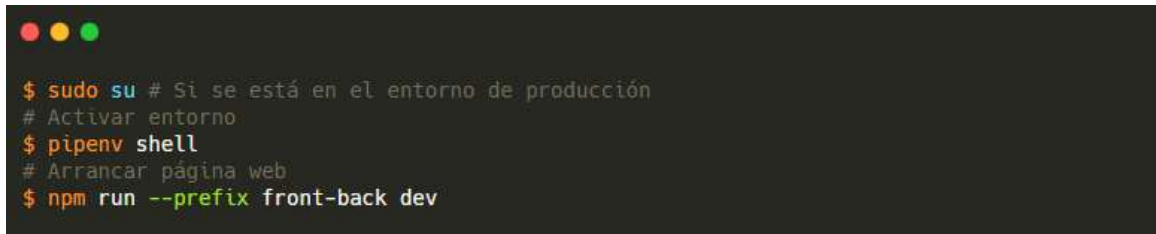
# Estando en la carpeta raíz del proyecto.
# Crear u obtener shell del entorno virtual
$ pipenv shell
# Instalar requisitos
$ pipenv install # o bien pipenv sync para sincronizar con Pipfile.lock
# Arrancar la API
$ python app.py

```

Ilustración 43 Despliegue de la API

La API es accesible así en <http://localhost:5000>. El puerto es el mismo independientemente del entorno. Sin embargo, la variable de entorno *FLASK_ENV* cambia de valor entre *production* y *development* en función del entorno.

3. Activar el entorno virtual para cargar las variables de entorno, lanzando posteriormente la página web haciendo uso de la utilidad de *npm*. Para ello, desde la carpeta raíz del proyecto, se utiliza la siguiente orden:



```
$ sudo su # Si se está en el entorno de producción
# Activar entorno
$ pipenv shell
# Arrancar página web
$ npm run --prefix front-back dev
```

Ilustración 44 Despliegue de la página

La razón de lanzar el servidor como superusuario en caso de estar en producción es el cambio de puerto. En la rama de desarrollo el servidor utiliza el puerto 3500, mientras que en producción utiliza el puerto protegido 443, correspondiente al protocolo HTTPS

En el momento del despliegue final, para dejar el servidor y la API activos sin tener *shell* activa, se utiliza la orden *nohup* delante de los comandos:

- *python app.py*
- *npm run --prefix front-back dev*

Gracias a esta orden se puede mantener la ejecución de un comando proporcionado como argumento tras salir de una terminal, ya que hace que se ejecute de forma independiente a la sesión al ignorar la señal HUP (enviada a un proceso cuando se cierra la terminal que lo controla)

5.5 GESTIÓN DE VERSIONES

Para no mezclar variables de entorno, código de distintas ramas y seguir una organización general se ha utilizado la herramienta Git y la plataforma *GitHub* junto a los *plugin* de *GitHub* para el *IDE Visual Studio Code*. Esto ha permitido además hacer un seguimiento de las versiones del proyecto, pudiendo colgar los incrementos de cada sprint en el repositorio de *GitHub* o volver a *commits* anteriores en caso de algún error [30].

En el repositorio de *GitHub* se han creado dos ramas, *main* y *dev*, representando los entornos de producción y desarrollo. La implementación y las pruebas se han ido realizando en la rama *dev*, haciendo el correspondiente *merge*, es decir, fusionando la rama *dev* con *main* cuando se llegaba a un estado estable.

Así, en el entorno local y la máquina virtual se ha utilizado la rama *dev* mientras que en la instancia *EC2* se ha utilizado la funcionalidad *git clone* y *git pull* para ir actualizando la plataforma.

Llegado un punto avanzado del desarrollo, para las últimas pruebas en producción se ha creado una última rama intermedia, *maindev*, que permite probar el código y realizar cambios sin afectar el estado estable de la rama *main*.

Por último, el código se ha colgado en un repositorio nuevo debido a que el repositorio que se ha utilizado durante el desarrollo contiene información personal, como pueden ser las claves de *Twitter* y *Telegram*, a las que se puede acceder revisando el historial de *commits*. Este repositorio se encuentra siguiendo el siguiente enlace: <https://github.com/Alburrito/social-sentiment>

5.6 PRUEBAS REALIZADAS

Se listan a continuación el conjunto de pruebas realizadas junto a los resultados que se deben obtener para considerar la prueba como pasada.

5.6.1 Bot de Telegram

Test	Resultado esperado	Estado
Votos de usuarios	Las oraciones se guardan junto a su voto en la correspondiente base de datos	✓
Routing	Los botones llevan correctamente a cada vista.	✓
ADMIN: Restringir demás usuarios	Solo el desarrollador puede acceder al panel de administración.	✓
ADMIN: Labelled	Las oraciones de la colección <i>Labelled</i> se aceptan, borran y desetiquetan correctamente	✓
ADMIN: Invalid	Las oraciones se aceptan con el voto indicado o se borran definitivamente de forma correcta.	✓
ADMIN: Gestión de BD	La batería de oraciones se rellena correctamente según los parámetros proporcionados y se muestra el estado de la BD al administrador.	✓
Crontab	La base de datos se autogestiona correctamente	✓

Tabla 24 Pruebas del bot de Telegram

5.6.2 Plataforma Social Sentiment

5.6.2.1 Web

Test	Resultado esperado	Estado
Accesibilidad	La web es accesible desde cualquier punto.	✓
Registro	Un usuario puede registrarse en la plataforma a través de la web con el correspondiente control de errores del formulario	✓
Login	Un usuario registrado puede autenticarse en la plataforma a través de la web con el correspondiente control de errores del formulario	✓
Vincular cuenta de Twitter	Un usuario registrado puede vincular su cuenta Social Sentiment con su cuenta de <i>Twitter</i>	✓

Tabla 25 Pruebas de la web de Social Sentiment

5.6.2.2 API

Test	Resultado esperado	Estado
Accesibilidad	La API es accesible únicamente desde los hosts especificados y por la web.	✓
Endpoints Twitter	Los <i>endpoints</i> que la API proporciona a la web para acceder a los datos de Twitter devuelven los resultados adecuados.	✓
Endpoints Supervisor	Los <i>endpoints</i> que la API proporciona al Supervisor para actualizar los perfiles funcionan correctamente y actualizan los mismos.	✓

Tabla 26 Pruebas de la API

5.6.2.3 Módulo de ingesta

Test	Resultado esperado	Estado
Datos Perfil	El módulo de ingesta obtiene correctamente la información de un perfil dado un identificador.	✓
Últimos tweets	El módulo de ingesta obtiene correctamente los últimos N <i>tweets</i> de un perfil (junto a su información) dado un identificador.	✓

Respuestas	El módulo de ingesta obtiene correctamente respuestas a un <i>tweet</i> dado el identificador del <i>tweet</i> y del usuario responsable.	✓
Trending Topic	El módulo de ingesta obtiene correctamente el <i>Trending Topic</i> actual de España	✓

Tabla 27 Pruebas del módulo de ingesta

5.6.2.4 Módulo de persistencia

Test	Resultado esperado	Estado
Recuperar records	El módulo de persistencia recupera correctamente cada histórico proporcionándole un identificador de usuario.	✓
Recuperar top tweets	El módulo de persistencia recupera correctamente los mejores <i>tweets</i> proporcionándole un identificador de usuario.	✓
Actualizar registros	El módulo de persistencia actualiza correctamente la base de datos acorde a los valores especificados por la API tras la orden del Supervisor.	✓

Tabla 28 Pruebas del módulo de persistencia

5.6.2.5 Módulo de análisis

Test	Resultado esperado	Estado
Obtener puntuación	El módulo de análisis realiza correctamente todas las métricas hasta obtener la puntuación final.	✓
Pipeline de predicción	El submódulo de análisis de sentimiento realiza correctamente los pasos para obtener el sentimiento de cada <i>tweet</i> .	✓

Tabla 29 Pruebas del módulo de análisis

5.6.2.6 Otros

Test	Resultado esperado	Estado
Sistema de logging	Cada módulo tiene su propio <i>logger</i> , permitiendo seguir el flujo de ejecución del programa y localizar posibles	✓

	errores.	
Despliegue	El sistema queda en ejecución tras cerrar la terminal en la que se despliega.	✓
Accesibilidad	La API y base de datos no son accesibles desde hosts no especificados.	✓

Tabla 30 Pruebas varias

5.6.3 Datos de prueba

Algunas de las pruebas no pueden realizarse sin un procedimiento previo. Por ejemplo, para las pruebas de accesibilidad se ha pedido ayuda a voluntarios para que intenten ingresar al panel de administrador del *bot* de *Telegram*, API, base de datos... Para las pruebas del módulo de ingesta se ha pedido permiso a voluntarios para obtener información de sus perfiles de *Twitter*.

Por su parte, para las pruebas de persistencia y análisis, es necesario un generar un conjunto de datos de prueba. De otra forma, se necesitaría estar obteniendo datos durante un periodo de tiempo prolongado, paralizando ciertas partes del desarrollo.

Para los datos de prueba de mejores tweets del usuario, simplemente se ha utilizado la herramienta *MongoDB Compass* [43] para introducir un documento con los campos. El sentimiento asociado a las supuestas respuestas al tweet se ha calculado en base a cinco frases aleatorias que hacen la función de respuesta. Estas frases aleatorias proceden de una batería de oraciones con la misma proporción de positivas, neutras y negativas.

```

    _id: ObjectId("60b675801b34a6f4e7fdb378")
  tweets: Array
    0: Object
      text: "Esto es un ejemplo de tweet"
      date: "2020-05-21 00:00:00"
      sentiment: Object
        pos: 80
        neu: 15
        neg: 5
      sent_score: 50
    1: Object
    2: Object
    3: Object
    4: Object
  twitter_id: "██████████"

```

Ilustración 45 Ejemplo de estructura de documento Mejores Tweets en base de datos. Visto a través de MongoDB Compass

Para generar los datos de los históricos, se ha utilizado *Jupyter Notebook* y *Python* de la siguiente forma:

1. Se genera una lista de fechas del 1 al 31 de mayo

```
dates = list(rrule(DAILY, @start=datetime.datetime(2020, 5, 1), until=datetime.datetime(2020,5,31)))
print(dates[13], '\n', dates[-3:])
[datetime.datetime(2020, 5, 1, 0, 0), datetime.datetime(2020, 5, 2, 0, 0), datetime.datetime(2020, 5, 3, 0, 0)]
[datetime.datetime(2020, 5, 29, 0, 0), datetime.datetime(2020, 5, 30, 0, 0), datetime.datetime(2020, 5, 31, 0, 0)]
```

Ilustración 46 Generación de fechas

2. Se crea la estructura que tendrá el documento (el campo *user_id* se cambió posteriormente a *twitter_id*, siendo este una cadena)

```
doc = {
    "user_id" : 1,
    "foll_records" : [],
    "fav_records" : [],
    "eng_records" : [],
    "score_records" : [],
    "sent_records" : [],
    "p_count_records": [],
    "rel_records": []
}
```

Ilustración 47 Estructura del documento

3. Se fija un valor de partida de los datos

```
foll = 5000
favs = 5000
eng = 60
score = 50
og_post_count = 10
og_relevancy = 50
```

Ilustración 48 Valores origen de los parámetros

4. Para cada fecha, se incrementa o decrementa aleatoriamente cada valor respecto a la fecha anterior. Se fijan unos márgenes para especificar cuánto se suma o resta en cada parámetro.

```
pos = 100 - randint(40,51)
neg = 100 - pos - randint(10,31)
neu = 100 - pos - neg
entry = {
    "date" : date,
    "pos" : pos,
    "neg" : neg,
    "neu" : neu
}
doc["sent_records"].append(entry)

foll = foll + randint(0,16)
doc["foll_records"].append({
    "date":date,
    "value":foll})
```

Ilustración 49 Generación de datos diarios

- Si el valor de alguno de los parámetros decreciese demasiado, se establece un nuevo valor algo más alto.

```
if favs < 2500:  
    favs = 2750
```

Ilustración 50 Control de mínimos

- Se obtiene así un documento con varios *arrays* representando los valores de cada día del mes de mayo. Los valores de los parámetros no presentan una relación real entre sí, pero la forma de su evolución permite obtener resultados más realistas en el módulo de análisis que si fueran totalmente estrictamente aleatorios.

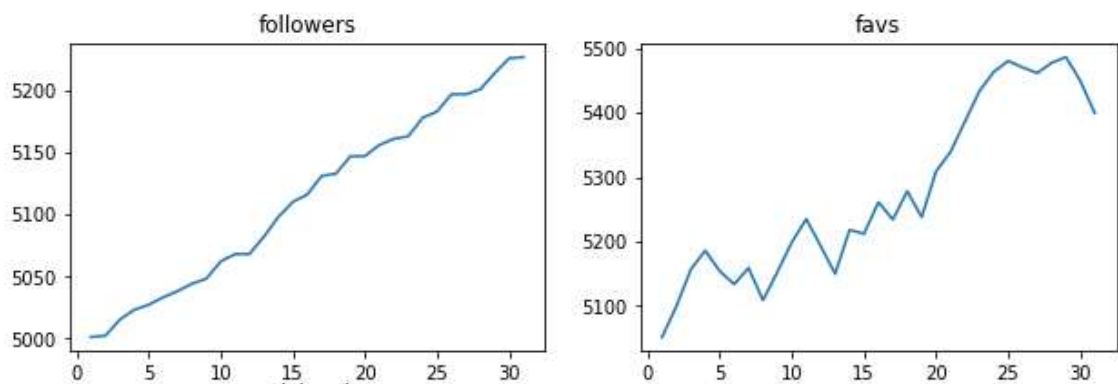


Ilustración 51 Ejemplos de datos generados

Los márgenes se han establecido tras ejecutar el código varias veces hasta encontrar el conjunto más convincente.

6 CONCLUSIONES

6.1 FUNCIONALIDAD DEL SISTEMA

Este Trabajo de Fin de Grado ha consistido en el diseño, desarrollo y documentación de una plataforma de monitorización de redes sociales, concretamente el caso de uso de *Twitter*. Para ello pone a disposición del usuario una plataforma web donde registrarse y visualizar distintas estadísticas, parte de ellas obtenidas tras realizar análisis de sentimiento. Se ha desarrollado paralelamente a la plataforma un *bot* de *Telegram* para obtener un conjunto de datos de aprendizaje que mejore la precisión del modelo de análisis de sentimiento.

6.2 CONCLUSIONES

En este apartado se exponen las conclusiones obtenidas como resultado del desarrollo del proyecto.

- Una de las motivaciones principales de este trabajo ha sido ampliar los conocimientos adquiridos durante el grado, aplicándolos a tecnologías dominantes en el sector. Dentro de las válidas para el proyecto, he intentado escoger lenguajes y herramientas con las que tuviera al menos cierta base, como *Python*, *Git* o la API de *Telegram*, para adquirir conocimientos más avanzados. Sin embargo, también he querido probar herramientas totalmente nuevas para mí, como son los servicios de AWS (ampliamente utilizados hoy en día) o crear una página web desde cero con *NodeJS*.
- He hecho hincapié en perfeccionar la estructura y el diseño de la página web, ya que es una rama bastante extendida en el mundo laboral. Esto me ha permitido incluso realizar una plantilla que incorpora la gestión de usuarios utilizada en el Trabajo de Fin de Grado, alojada en el siguiente repositorio bajo licencia GPL: <https://github.com/Alburrito/nodejs-express-users-template>
- Aunque es una tarea costosa, la participación voluntaria en el *bot* de *Telegram* ha permitido obtener la cantidad de 2000 frases etiquetadas que incorporar al modelo de análisis de sentimiento.
- Gracias al hincapié hecho en la arquitectura modular de la plataforma se ha podido cumplir con éxito todos los objetivos marcados en el proyecto.
- He ampliado mis conocimientos en la rama del *Machine Learning* y el procesamiento de lenguaje natural, *NLP*, ramas de la informática que me llaman mucho la atención

- La plataforma tiene gran carga de trabajo entre extracción de datos, análisis, *tokenización*, actualizaciones... Esto provoca que se hayan restringido las actualizaciones a la madrugada de cada día, franjas horarias en las que hay menos tráfico. Se ha tenido que utilizar un fichero *swap* en la máquina de Amazon, ya que con 1 Gb de memoria no era suficiente para correr la API con los distintos módulos y la página web. Se han proporcionado 2 Gb más, alcanzando fácilmente un 75 % del total.
- He obtenido experiencia en un caso de uso realista del desarrollo de una plataforma software. Esto incluye la gestión, planificación, análisis, diseño, desarrollo, implementación, pruebas y despliegue de la misma. Así, he obtenido una visión realista del esfuerzo, costes y tiempo que suponen las diferentes tareas para crear un producto software, repasando la mayor parte de competencias del grado.

6.3 LÍNEAS DE TRABAJO FUTURO

Gracias a la arquitectura modular de la plataforma, es sencillo incorporar nuevas funcionalidades a la misma.

En primer lugar, se puede modularizar aún más sin excesivo coste, diseñando módulos que engloben funcionalidades aún más separadas según la red social. Dada la presencia de la API, no es difícil diseñar un sistema de clases que abstraiga la funcionalidad concreta de cada *scraper*, cada analizador o cada actualización del supervisor. Además, el módulo de *Passport*, utilizado tanto en la autenticación local como en la vinculación de *Twitter*, proporciona estrategias tanto para las otras redes propuestas de la plataforma como para muchas otras.

La distribución modular del software del proyecto permite que en un futuro los distintos módulos pudiesen estar distribuidos, balanceando la carga de trabajo. Si el número de usuarios de la plataforma aumenta, la máquina de Amazon no podrá soportar toda la carga de trabajo sin imponer costes adicionales o contratando de mejores prestaciones.

En cuanto al modelo de *Machine Learning*, este es mejorable en cualquier momento. Si se obtiene un *dataset* mejor, se puede entrenar un nuevo modelo con él y simplemente serializarlo y guardarlo en la carpeta correspondiente. Si se decide realizar otro preprocesamiento, únicamente habría que cambiar los métodos privados del submódulo encargado acorde al nuevo preprocesamiento que use el modelo.

El módulo de análisis se puede mejorar modificando los pesos de las distintas métricas. El valor óptimo de estos pesos se puede obtener tras la monitorización prolongada de la plataforma.

Tras la defensa del Trabajo de Fin de Grado se pretende continuar con el desarrollo de la plataforma como proyecto personal, aumentando aún más los conocimientos en las distintas áreas y publicando el código en su repositorio de *GitHub*, permitiendo que cualquiera pueda colaborar en la mejora del mismo. Además, se dejará funcionando el *bot* de *Telegram* con el objetivo de crear progresivamente un *dataset* aún mayor que unir al ya obtenido.

En conclusión, se ha desarrollado una plataforma web modular capaz de gestionar la recolección, persistencia, análisis y visualización de la evolución de perfiles en redes sociales. Concretamente se ha desarrollado para el caso de uso de *Twitter*.

7 BIBLIOGRAFÍA

- [1] Hootsuite: social media marketing & management dashboard. (n.d.). Recuperado en Julio, 6, 2021, de <https://www.hootsuite.com/>
- [2] Streamd.in (@streamdin) | twitter. (n.d.). Recuperado en Julio, 6, 2021, de <https://twitter.com/streamdin>
- [3] Mirabell, O. (2014). Redes sociales, economía y empresa. *Oikomics*, (2), 3-5.
- [4] Ros-Martín, M. (2009). Evolución de los servicios de redes sociales en internet. *El profesional de la información*, 18(5), 552-557.
- [5] Tetlock, P. C. (2007). Giving content to investor sentiment: The role of media in the stock market. *The Journal of finance*, 62(3), 1139-1168.
- [6] Pang, B., Lee, L., & Vaithyanathan, S. (2002). Thumbs up? Sentiment classification using machine learning techniques. *arXiv preprint cs/0205070*.
- [7] Abbasi, A., Chen, H., & Salem, A. (2008). Sentiment analysis in multiple languages: Feature selection for opinion classification in web forums. *ACM transactions on information systems (TOIS)*, 26(3), 1-34.
- [8] Ortigosa, A., Martín, J. M., & Carro, R. M. (2014). Sentiment analysis in Facebook and its application to e-learning. *Computers in human behavior*, 31, 527-541.
- [9] Abbasi, A., Chen, H., & Salem, A. (2008). Sentiment analysis in multiple languages: Feature selection for opinion classification in web forums. *ACM transactions on information systems (TOIS)*, 26(3), 1-34.
- [10] J. Shawe-Taylor, S. Sun, A review of optimization methodologies in support vector machines, *Neurocomputing* 74 (2011) 3609–3618
- [11] D. M. Blei, A. Y. Ng, M. I. Jordan, Latent Dirichlet allocation, *Journal of Machine Learning research* 3 (2003) 993–1022.
- [12] Trigás Gallego, M. (2012). *Metodología scrum* 32-54
- [13] Malka, L. (2011, October). Vmccrypt: modular software architecture for scalable secure computation. In *Proceedings of the 18th ACM conference on Computer and communications security* (pp. 715-724).

- [14] Zhou, W., Li, L., Luo, M., & Chou, W. (2014, May). REST API design patterns for SDN northbound API. In 2014 28th international conference on advanced information networking and applications workshops (pp. 358-365). IEEE.
- [15] Makice, K. (2009). Twitter API: Up and running: Learn how to build applications with the Twitter API. " O'Reilly Media, Inc
- [16] Jatana, N., Puri, S., Ahuja, M., Kathuria, I., & Gosain, D. (2012). A survey and comparison of relational and non-relational database. International Journal of Engineering Research & Technology, 1(6), 1-5
- [17] MongoDB, I. (2014). Mongoddb. URL <https://www.mongodb.com/>. Cited on (2014), 9.
- [18] Telegram web. (n.d.). Recuperado en Julio, 6, 2021, de <https://web.telegram.org/>
- [19] Chazallet, S. (2016). Python 3: los fundamentos del lenguaje. Ediciones ENI.
- [20] Tilkov, S., & Vinoski, S. (2010). Node. js: Using JavaScript to build high-performance network programs. IEEE Internet Computing, 14(6), 80-83.
- [21] Bierman, G., Abadi, M., & Torgersen, M. (2014, July). Understanding typescript. In European Conference on Object-Oriented Programming (pp. 257-281). Springer, Berlin, Heidelberg.
- [22] Visual studio code - code editing. redefined. (n.d.). Recuperado en Julio, 6, 2021, de <https://code.visualstudio.com/>
- [23] Intellisense in visual studio code. (n.d.). Recuperado en Julio, 6, 2021, de <https://code.visualstudio.com/docs/editor/intellisense>
- [24] Manjaro - enjoy the simplicity. (n.d.). Recuperado en Julio, 6, 2021, de <https://manjaro.org/>
- [25] Arch linux. (n.d.). Recuperado en Julio, 6, 2021, de <https://archlinux.org/>
- [26] Oracle vm virtualbox. (n.d.). Recuperado en Julio, 6, 2021, de <https://www.virtualbox.org/>
- [27] Ubuntu: enterprise open source and linux. (n.d.). Recuperado en Julio, 6, 2021, de <https://ubuntu.com/>
- [28] Amazon web services (aws) - cloud computing services. (n.d.). Recuperado en Julio, 6, 2021, de <https://aws.amazon.com/>
- [29] Guillermo, M., Billones, R. K., Bandala, A., Vicerra, R. R., Sybingco, E., Dadios, E. P., & Fillone, A. (2020, November). Implementation of Automated Annotation through Mask RCNN Object Detection model in CVAT using AWS EC2 Instance. In 2020 IEEE region 10 conference (TENCON) (pp. 708-713). IEEE.

- [30] Vuorre, M., & Curley, J. P. (2018). Curating research assets: A tutorial on the Git version control system. *Advances in Methods and Practices in Psychological Science*, 1(2), 219-236.
- [31] Pipenv: python dev workflow for humans — pipenv 2021.5.29 .. (n.d.). Recuperado en Julio, 6, 2021, de <https://pipenv.pypa.io/en/latest/>
- [32] Logging — logging facility for python — python 3.9.6 documentation. (n.d.). Recuperado en Julio, 6, 2021, de <https://docs.python.org/3/library/logging.html>
- [33] Pezoa, F., Reutter, J. L., Suarez, F., Ugarte, M., & Vrgoč, D. (2016, April). Foundations of JSON schema. In *Proceedings of the 25th International Conference on World Wide Web* (pp. 263-273).
- [34] Pickle — python object serialization — python 3.9.6 documentation. (n.d.). Recuperado en Julio, 6, 2021, de <https://docs.python.org/3/library/pickle.html>
- [35] Grinberg, M. (2018). *Flask web development: developing web applications with python*. "O'Reilly Media, Inc."
- [36] Flask-caching — flask-caching 1.0.0 documentation. (n.d.). Recuperado en Julio, 6, 2021, de <https://flask-caching.readthedocs.io/>
- [37] Postman | the collaboration platform for api development. (n.d.). Recuperado en Julio, 6, 2021, de <https://www.postman.com/>
- [38] Use cases, tutorials, & documentation | twitter developer platform. (n.d.). Recuperado en Julio, 6, 2021, de <https://developer.twitter.com/en>
- [39] Leiba, B. (2012). OAuth web authorization protocol. *IEEE Internet Computing*, 16(1), 74-77.
- [40] Roesslein, J. (2009). tweepy Documentation. Online] <http://tweepy.readthedocs.io/en/v3>, 5.
- [41] Gálvez Lao, M. (2019). Construcción de infraestructura Big Data para el procesamiento y visualización de datos de Twitter.
- [42] Satheesh, M., D'mello, B. J., & Krol, J. (2015). *Web development with MongoDB and NodeJs*. Packt Publishing Ltd.
- [43] Mongodb compass. (n.d.). Recuperado en Julio, 6, 2021, de <https://www.mongodb.com/products/compass>
- [44] Kaggle: your machine learning and data science community. (n.d.). Recuperado en Julio, 6, 2021, de <https://www.kaggle.com/>
- [45] Spanish airlines tweets sentiment analysis | kaggle. (n.d.). Recuperado en Julio, 6, 2021, de <https://www.kaggle.com/c/spanish-airlines-tweets-sentiment-analysis>

- [46] Dataset alternativo. (n.d.). Recuperado en Julio, 6, 2021, de https://doctoradofs.files.wordpress.com/2016/05/2clases_es_generaltassisol_pub.xls
- [47] Project jupyter | home. (n.d.). Recuperado en Julio, 6, 2021, de <https://jupyter.org/>
- [48] Pandas - python data analysis library. (n.d.). Recuperado en Julio, 6, 2021, de <https://pandas.pydata.org/>
- [49] Tosi, S. (2009). *Matplotlib for Python developers*. Packt Publishing Ltd.
- [50] Oliphant, T. E. (2006). *A guide to NumPy* (Vol. 1, p. 85). USA: Trelgol Publishing.
- [51] Natural language toolkit — nltk 3.6.2 documentation. (n.d.). Recuperado en Julio, 6, 2021, de <https://www.nltk.org/>
- [52] Miller, G. A. (1995). WordNet: a lexical database for English. *Communications of the ACM*, 38(11), 39-41.
- [53] Spacy · industrial-strength natural language processing in python. (n.d.). Recuperado en Julio, 6, 2021, de <https://spacy.io/>
- [54] Saif, H., Fernández, M., He, Y., & Alani, H. (2014). On stopwords, filtering and data sparsity for sentiment analysis of twitter.
- [55] Plisson, J., Lavrac, N., & Mladenic, D. (2004, May). A rule based approach to word lemmatization. In *Proceedings of IS* (Vol. 3, pp. 83-86).
- [56] Rodríguez, P., Bautista, M. A., Gonzalez, J., & Escalera, S. (2018). Beyond one-hot encoding: Lower dimensional target embedding. *Image and Vision Computing*, 75, 21-31.
- [57] Scikit-learn: machine learning in python — scikit-learn 0.24.2 .. (n.d.). Recuperado en Julio, 6, 2021, de <https://scikit-learn.org/>
- [58] Kantrowitz, M., Mohit, B., & Mittal, V. (2000, July). Stemming and its effects on TFIDF ranking. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 357-359).
- [59] Gupte, A., Joshi, S., Gadgul, P., Kadam, A., & Gupte, A. (2014). Comparative study of classification algorithms used in sentiment analysis. *International Journal of Computer Science and Information Technologies*, 5(5), 6261-6264.
- [60] Chen, Y., Luo, M., Peng, J., Wang, J., Zhou, X., & Li, S. (2017). Classification of land use in industrial and mining reclamation area based grid-search and random forest classifier. *Transactions of the Chinese Society of Agricultural Engineering*, 33(14), 250-257.

- [61] Prescott, P. (2015). HTML 5. Babelcube Inc..
- [62] McFarland, D. S. (2012). CSS3: the missing manual. " O'Reilly Media, Inc.".
- [63] Spurlock, J. (2013). Bootstrap: responsive web development. " O'Reilly Media, Inc.".
- [64] Font awesome. (n.d.). Recuperado en Julio, 6, 2021, de <https://fontawesome.com/>
- [65] Font scripts market share, websites and contacts - wappalyzer. (n.d.). Recuperado en Julio, 6, 2021, de <https://www.wappalyzer.com/technologies/font-scripts/>
- [66] Bootswatch: free themes for bootstrap. (n.d.). Recuperado en Julio, 6, 2021, de <https://bootswatch.com/>
- [67] Express.js. (n.d.). Recuperado en Julio, 6, 2021, de <https://expressjs.com/>
- [68] Nodemon - npm. (n.d.). Recuperado en Julio, 6, 2021, de <https://www.npmjs.com/package/nodemon>
- [69] Amcharts: javascript charts & maps. (n.d.). Recuperado en Julio, 6, 2021, de <https://www.amcharts.com/>
- [70] Gálvez Lao, M. (2019). Construcción de infraestructura Big Data para el procesamiento y visualización de datos de Twitter.
- [71] Bcryptjs - npm. (n.d.). Recuperado en Julio, 6, 2021, de <https://www.npmjs.com/package/bcryptjs>
- [72] Connect-flash - npm. (n.d.). Recuperado en Julio, 6, 2021, de <https://www.npmjs.com/package/connect-flash>
- [73] File system | node.js v16.4.2 documentation. (n.d.). Recuperado en Julio, 6, 2021, de <https://nodejs.org/api/fs.html>
- [74] Https | node.js v16.4.1 documentation. (n.d.). Recuperado en Julio, 6, 2021, de <https://nodejs.org/api/https.html>
- [75] Morgan - npm. (n.d.). Recuperado en Julio, 6, 2021, de <https://www.npmjs.com/package/morgan>
- [76] Making http requests with node.js. (n.d.). Recuperado en Julio, 6, 2021, de <https://nodejs.dev/learn/making-http-requests-with-nodejs>
- [77] Serve-favicon - npm. (n.d.). Recuperado en Julio, 6, 2021, de <https://www.npmjs.com/package/serve-favicon>

[78] Python. (n.d.). Recuperado en Julio, 6, 2021, de <https://realpython.com/python-requests/>

[79] Crontab(5) - linux manual page. (n.d.). Recuperado en Julio, 6, 2021, de <https://man7.org/linux/man-pages/man5/crontab.5.html>

[80] Fowler, M., & Highsmith, J. (2001). The agile manifesto. *Software development*, 9(8), 28-35.

[81] Nufusula, R., & Susanto, A. (2018). Rancang Bangun Chat Bot Pada Server Pulsa Menggunakan Telegram Bot API. *JOINS (Journal of Information System)*, 3(1), 80-88.