INSTITUTO TECNOLÓGICO DE BUENOS AIRES
Escuela de Ingeniería y Gestión

Calidad de datos contextual en Big Data
Calidad de Datos de Twitter

Autor: Cortés Rodríguez, Kevin Imanol (Legajo 54.775)

Tutor de Tesis: Ph.D. Vaisman, Alejandro

Trabajo Final presentado para la obtención del título de
INGENIERO EN INFORMÁTICA

Lavarden 315, 1437 CABA, Buenos Aires, Argentina

PREFASE 3

INTRODUCCIÓN, MOTIVACIÓN y CONTRIBUCIONES 4

QUÉ ES LA CALIDAD DE DATOS 6
DIMENSIONES Y MÉTRICAS 9

IMPLEMENTACIÓN 12
ARQUITECTURA v1 12
PROBLEMAS HALLADOS EN LA IMPLEMENTACIÓN 14
NUEVA IMPLEMENTACIÓN 17
IMPLEMENTACIÓN DE API REST 21
BASE DE DATOS RELACIONAL 25
EXPERIENCIA AL USUARIO Y USABILIDAD 27
NUEVAS FUNCIONALIDADES 30
TIPOS DE DASHBOARDS 32

CONCLUSIONES, LIMITACIONES Y MEJORAS 39

BIBLIOGRAFÍA 42

PREFASE

En cada una de las fases del análisis en los procesos relacionados a Big Data, la calidad de datos juega un papel importante. La obtención de la calidad de datos, basados en las dimensiones de la calidad y métricas, deben ser adaptados en pos de capturar las nuevas características que el Big Data nos afronta.

Este documento trata de profundizar dicho problema, redefiniendo las dimensiones y métricas de la calidad de datos en un escenario de Big Data, donde el dato llega en tiempo real en formato JSON y es procesado por distintos componentes para obtener métricas de calidad de datos. En particular, este proyecto estudia el caso concreto de mensajes de usuarios de la red social Twitter.

Por otra parte, también se detalla la implementación de una nueva arquitectura continuando el proyecto de *Data quality in a big data context: about Twitter's data quality*[1] basada en microservicios, desde el momento que se procesa un tweet, llega desde la interfaz al usuario y todas las mejoras agregadas en pos de mejorar la experiencia al usuario.

INTRODUCCIÓN, MOTIVACIÓN y CONTRIBUCIONES

Para entender cuál es la motivación, primero debemos reflexionar de por qué debemos tener en cuenta al Big Data como principal protagonista de este informe. Cuando hablamos de Big Data, estamos hablando de un conjunto de datos y combinaciones cuyo volumen, complejidad y velocidad de crecimiento dificulta la captura, procesamiento o análisis mediante herramientas y tecnologías convencionales como las bases de datos relacionales. La naturaleza que bordea su concepto se debe principalmente a que no está estructurada, por lo que para su análisis y explotación debemos utilizar previamente distintas técnicas para ordenarlas y luego, si así se requiere, combinarlas con otro tipo de datos que bien pueden ser estructurados.

Desde otra perspectiva, el análisis y uso de técnicas de Big Data ha sido de gran aporte y valor agregado a empresas, entidades y organismos ya que basados en dicha premisa, permite clarificar y abordar la mejora de soluciones a problemas existentes, solucionar problemas que a simple vista no había respuesta o inclusive hallar nuevos problemas que antes no se sabian que existian.

El estudio de Big Data se caracteriza por utilizar cinco dimensiones conocidas como las cinco V's: *volumen*, *velocidad*, *variedad*, *veracidad* y *valor*. Estas 5 características del Big Data provoca que existan problemas para extraer datos reales y de alta calidad, de conjuntos de datos tan masivos, cambiantes y complicados. Basados en estas cinco dimensiones, se han creado distintas métricas para analizar la calidad de los datos. El concepto de la calidad de datos debería ser revisado y analizado en un contexto de Big Data ya que este último afronta problemas que los datos convencionales no presentaban en casos con base de datos relacionales. Intuitivamente, cada una de las cinco V's define un contexto diferente para el análisis de los datos y consecuentemente, para la calidad de los datos a nivel general.

El efecto del contexto en el campo del análisis de la calidad de datos no ha sido investigado en detalle en el pasado puesto a las dificultades presentadas para obtener métricas en datos de gran cantidad de volumen y no estructurados. El contexto, también conocido como la condición comunitaria en un entorno, se refiere a un proceso que implica la evaluación de la información que tendrá un efecto sobre la probabilidad de éxito como resultado de este hecho que la comunicación efectiva depende de la masa de conocimiento común compartido entre varias personas. Para que una persona entienda una unidad de comunicación a nivel técnico, debe comprender el proceso con respecto al resultado de la transmisión con consideración sobre si el receptor lo entendió. Este informe también trata de cómo el contexto de los mensajes obtenidos en tiempo real de la red social Twitter influye en los valores de la calidad de los datos.

En particular, este informe tratará la calidad de datos basados en el contexto en un escenario en tiempo real, donde los datos llegan en tiempo real de forma desestructurada y con fiabilidad y utilidad muy volátiles. Todas estas características son opuestas a un escenario de análisis con datos estructurados, ordenados, transformados y analizados offline. Además, en este informe abordaremos las diferencias, pros y contras de la arquitectura explicada en [1] y nuevos complementos agregados para aumentar la experiencia de uso al usuario final.

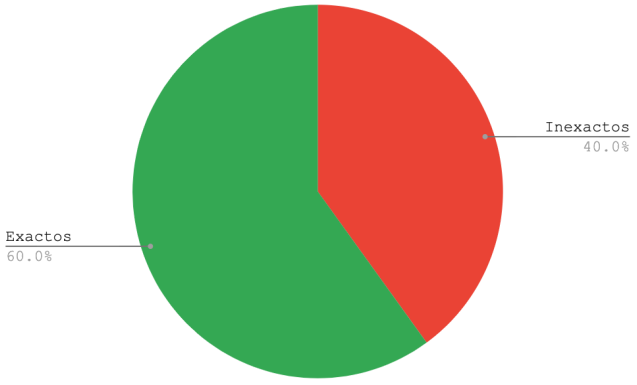
A lo largo de este documento también abordaremos los distintos problemas hallados en la implementación de un sistema que obtiene tweets en tiempo real, computa la calidad del dato en tiempo real, aplicando métricas de calidad definidos formalmente en el informe mencionado en [1] y descritos formalmente nuevamente en este. La implementación incluye una interfaz web que permite el filtrado de tweets, por ejemplo, por keywords, computando su calidad de datos y visualizando no solo la calidad en su totalidad, sino por cada una de sus dimensiones.

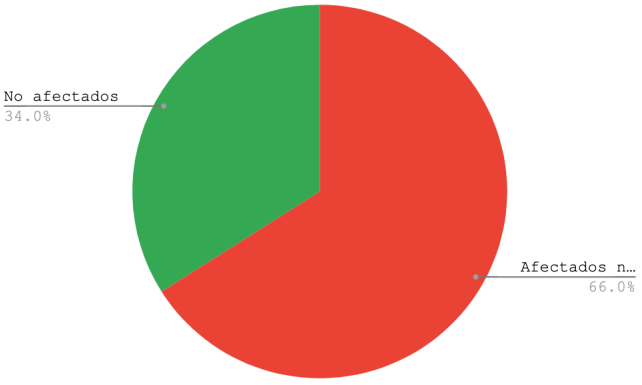
QUÉ ES LA CALIDAD DE DATOS

La falta de calidad de los datos es uno de los principales problemas a la hora de realizar un análisis basados en ellos. En efecto, lo que llamamos datos con "buena calidad" es un activo más potente, ya que permite sacar conclusiones y hallar nuevas preguntas a problemas sobre situaciones de la realidad que con un simple análisis no se podría haber obtenido.
Según la serie de normas ISO 9000[1], la calidad se podría definir como "*el grado en el que un conjunto de características inherentes cumple con los requisitos, esto es, con la necesidad o expectativa establecida, generalmente implícita u obligatoria*".

Para muchas personas que toman decisiones basadas en estadísticas o situaciones pasadas, los datos son su principal motor de búsqueda. Administrar la calidad de los datos y aplicar análisis son clave para implementar con éxito las metas a alcanzar. Para ello, debemos entender cuál es el impacto de la calidad de datos en números.

Cerca del 40% de las compañías que recopilan datos de sus usuarios dicen que no es la adecuada. De 100 empresas involucradas en una iniciativa de calidad de datos, se descubrió que el 23% de la información que utilizan para tomar decisiones comerciales eran inexactos, mientras que los peores vieron un 43% que sus datos obtenidos eran erróneos. Otro hecho sorprendente fue que el 92% de las empresas admiten que sus datos de contacto no son precisos, mientras que el 66% de las organizaciones creen que se ven afectados negativamente por datos inexactos.





[1] Conjunto de enunciados en los cuales se especifican los elementos a integrar el sistema de Gestión de Calidad

Para poder obtener un buen control de calidad se requiere cubrir todo un procedimiento o etapas en todo el ciclo de vida del dato.

1. Detección, descubrimiento y análisis de fuentes de información
   La generación de los datos puede tener muchísimos orígenes diferentes. Por otra parte, el descubrimiento es un proceso iterativo que no requiere una amplia creación de modelos iniciales, aunque sí las habilidades necesarias para comprender las relaciones entre la información. Para este proyecto, las fuentes de información provienen de una única red social: Twitter, aunque se debe hacer una distinción ya que los usuarios son aquellos quienes publican los mensajes. En contraste, la red social es el medio y estos últimos son nuestra fuente de información.

2. Almacenamiento y organización de los datos
   El objetivo del almacenamiento es generar una medida tangible de la calidad de los datos al inicio que permitirá aclarar las condiciones actuales de la información, aportando visibilidad sobre aspectos tan relevantes como la existencia de duplicidades o redundancias en los datos. Además, tener los datos almacenados proporciona una gran variedad de procesos analíticos diferentes implementables a futuro. El objetivo es que los datos se almacenan en "crudo" para ser en el momento de realizar análisis sobre ellos cuando se procesan y se aplican las transformaciones necesarias.
   Una de las mejoras aplicadas en esta versión del proyecto es la de poder persistir el 100% de los datos recogidos por Twitter para hacer consultas históricas basadas en el contexto.

3. Preprocesado y calidad del dato
   Este apartado está relacionado en conocer la calidad de nuestros datos desde el punto de vista técnico (formatos, completitud, disponibilidad, integridad de fuentes, etc.). Como mencionaremos más adelante en este informe, aplicaremos cuatro distintas calidades basados en el usuario, la confiabilidad y usabilidad.

4. Analitica predictiva y prescriptiva
   La analítica predictiva y prescriptiva pretende anticipar lo que ocurrirá (predictiva) para proporcionar alternativas de actuación sobre esta previsión (prescriptiva). Es en esta fase donde los algoritmos de Machine Learning e Inteligencia Artificial entran en juego ya que son capaces de aprender de todos los históricos de información para abordar objetivos concretos. Particularmente este informe no se centrará en este tipo de análisis.

5. Explotación de resultados
   Una vez que el proceso de calidad de datos se ha implementado, es importante que se informe sobre los resultados. La explotación de los resultados de los análisis la adaptamos a las necesidades de cada usuario. Esta explotación de resultados puede ser a través de informes interactivos o herramientas de visualización.

Por medio de este ciclo de vida, tenemos la oportunidad de detectar cualquier anomalía que se pudiera presentar en los datos durante cualquiera de nuestros procesos antes de alcanzar nuestro fin, por ello es importante llevar a cabo un seguimiento adecuado, correcto y de mejora continua.

Debido a que el Big Data tiene las características de las 5 V's, cuando se emplean y procesan, extrayendo datos reales y de alta calidad de conjuntos de datos masivos, variables y complicados, esto se convierte en un problema. En la actualidad, la calidad de datos de Big Data se enfrenta a la diversidad de fuentes de datos ya que aporta abundantes tipos de datos y estructuras de datos complejas y aumenta la dificultad de la integración de datos. Con un gran volúmen de datos es difícil juzgar la calidad de los datos dentro de un tiempo razonable, éstos cambian muy rápido y no hay suficientes estándares de calidad de datos unificados y aprobados y la investigación sobre la calidad de datos de big data.

La producción y el almacenamiento de datos se ha conceptualizado como un sistema íntegro de fabricación de datos. Un aspecto central de esto es el concepto de un proceso de producción de datos que transforma los datos en información útil para los consumidores de datos.
Bajo este criterio, identificamos tres roles:

1. **productores de datos** (personas, grupos u otras fuentes que generan datos);
2. **quienes persisten o almacenan los datos**;
3. **consumidores de datos** (personas o grupos que usan datos).

Cada rol está asociado con un proceso o tarea: los productores de datos están asociados con procesos de producción de datos; luego quienes persisten los datos, hacen mantenimiento de los mismos y velan por la seguridad e integridad; y consumidores de datos con procesos de utilización de datos, que pueden involucrar agregación e integración de datos adicionales.

Esto significa que la utilidad y la usabilidad son aspectos importantes de la calidad.

DIMENSIONES Y MÉTRICAS

Esta sección estudiará cómo se pueden usar las dimensiones calidad de datos en un escenario de Big Data. El estudio se centrará en los datos generados de origen humano y se describirán cómo estas dimensiones se pueden aplicar para abordar la calidad en mensajes de Twitter.

1. *Legibilidad / Readability* (r): Dado un diccionario D, y una colección de palabras consideradas válidas en un documento x, la legibilidad de x, denotado r(x) se define como el cociente entre el palabras válidas en x y todas las palabras en x, si las hay, de lo contrario es cero. Es decir, dado un conjunto W de las palabras (válidas y no válidas) que están presentes en el documento x, la legibilidad de x es, donde en este caso particular x representa un tweet.

$$r(x) = \begin{cases} \frac{\#\{w \in W \wedge w \in D\}}{\#\{w \in W\}} & if \quad W \neq \emptyset \\ 0 & if \quad W = \emptyset \end{cases}$$

2. *Completitud / Completeness* (c): Considerando un objeto x en el dominio y una matriz props que contiene los nombres de las propiedades requeridas para describir x para un problema dado p; se asume que x se representa como una colección de pares (propiedad, valor) de la forma: $\{(p_1, v_1), ..., (p_i, v_i)\}$ en la que v_i es un valor para p_i. Si la propiedad $p_i \varepsilon x$ tiene asociada a valor no null v_i, es lo que se define como "bien definido". A su vez, tambien existe una funcion llamada "valodPropsOf" que, dado un objeto x devuelve un set de propiedades bien definidas. La completitud de x, denotado *c(x)* informa si todas las propiedades de props están bien definidas en x; y es computado de la siguiente forma:

$$c(x) = \begin{cases} 1 & if \quad props_p \subset validPropsOf(x) \\ 0 & otherwise \end{cases}$$

Dado un tweet x, como por ejemplo *x = { text: 'I like Bitcoin', user: null }* y *props=[text]*, el valor de *c(x) = 1*. En caso de que *props=[text, user]*, *c(x) = 0* ya que no está bien definida

3. *Utilidad / Usefulness* (u): Como se trata de un proyecto que obtiene datos en tiempo real de una red social, el factor humano es inapelable. Se asumirá que esta propiedad está directamente relacionada con la posibilidad de:
    a. Detectar sentimientos, que puede ser positiva o negativa en un objeto x que puede ser un tweet o post. Si x refleja un sentimiento positivo o negativo acerca de cierto tópico o persona, x se la considerara útil. Si el sentimiento es neutral o si no puede ser computable por una herramienta del procesamiento del lenguaje natural, se la considerara no útil.
    b. Detectando el dominio del tópico de x, por ejemplo, política, marketing, deportes, etc.

$$u(x) = \begin{cases} 1 & if \quad (sentiment(x) = P \vee sentiment(x) = N) \\ 0 & otherwise \end{cases}$$

4. *Confiabilidad / Trustworthiness* (t): En una red social (o en general, con datasets creados por seres humanos) cualquiera puede publicar cualquier tipo de información; por lo que en lo que en estos ámbitos respecta, la confiabilidad es un rol clave para la calidad del dato.

DATA QUALITY BASADOS EN EL CONTEXTO

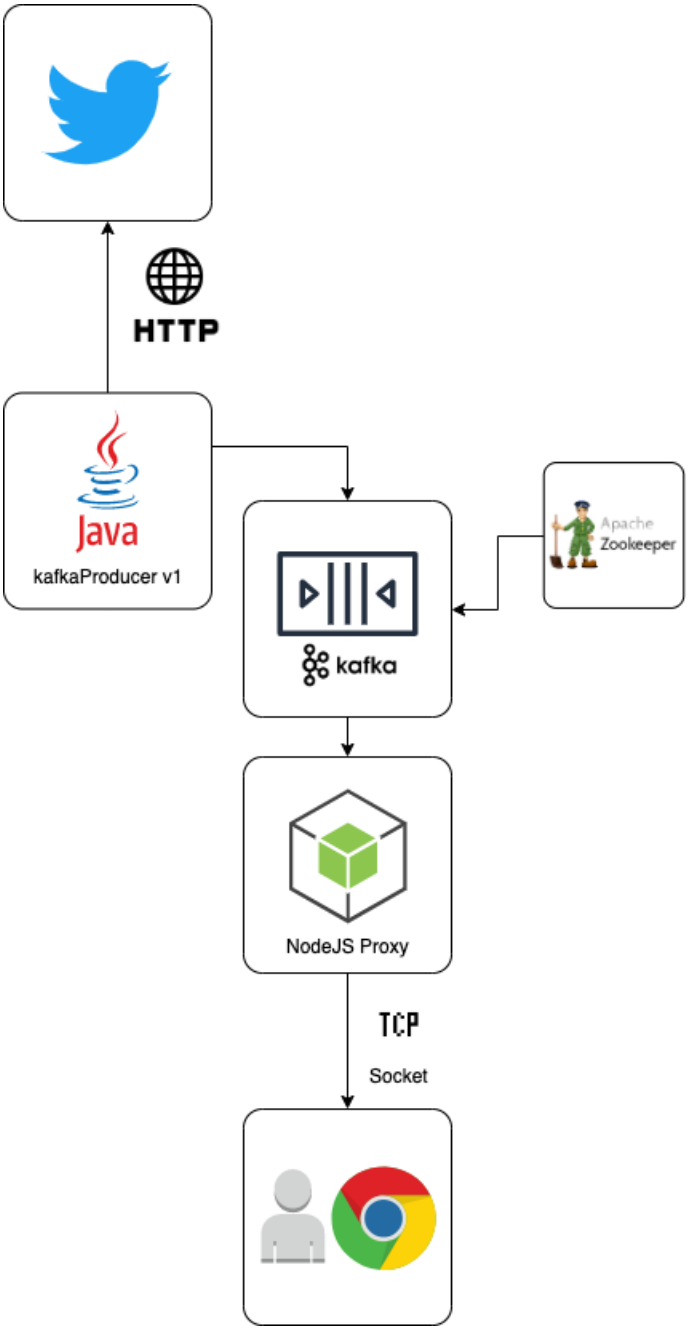
Para crear una estrategia efectiva de calidad de datos, se debe tener en cuenta el contexto. La razón es que si la calidad de los datos se define como la capacidad de un determinado conjunto de datos para cumplir con un propósito, entonces la cuestión de si tiene o no datos de calidad depende del contexto de uso que se le vaya a dar.

La evaluación de la calidad de una fuente de datos depende del contexto, es decir, las nociones de datos "buenos" o "pobres" no pueden separarse del contexto en el que los datos son producidos o utilizados. Por ejemplo, los datos sobre las ventas anuales de un producto con variaciones durante el año puede ser un dato de calidad por alguien que evalúa los ingresos anuales de dicho producto. Sin embargo, los mismos datos pueden que no sea lo suficientemente buenos para un almacén que está tratando de estimar el pedidos para el siguiente mes. Además, la calidad de los datos está relacionada con la discrepancia entre los datos reales, los valores almacenados y los valores "reales" que se suponía o se esperaba que se almacenarán. Por ejemplo, si una medición de temperatura se toma con un termómetro defectuoso, el valor almacenado (la medida) diferirá del valor correcto (el valor real temperatura), que se suponía que debía almacenarse. Esto es un ejemplo de datos semánticamente inexactos.

IMPLEMENTACIÓN

Esta sección presenta y describe la implementación de la arquitectura y sus componentes aplicados para luego analizar la calidad de un set de tweets en tiempo real. Primero detallaremos la arquitectura presentada en [1] y luego las modificaciones aplicadas en esta segunda versión del proyecto. También se realizará un análisis de los problemas existentes con la arquitectura inicial y cómo fueron corregidos. Por otra parte, se enunciarán las nuevas funcionalidad agregadas que mejoran la usabilidad y experiencia al usuario.

ARQUITECTURA v1



El core del sistema es una instancia de Apache Kafka[13]. Éste es un software de código abierto que proporciona un marco para almacenar, leer y analizar datos de transmisión. Está diseñado para ejecutarse en un entorno "distribuido", lo que significa que puede ejecutarse en varios servidores, aprovechando la potencia de procesamiento adicional y la capacidad de almacenamiento que esto brinda.

Por otra parte, Kafka funciona con Zookeeper[14], que actúa como un servicio centralizado y se utiliza para mantener los datos de nombre y configuración para proporcionar una sincronización flexible y robusta dentro de los sistemas distribuidos. Zookeeper realiza un seguimiento del estado de los nodos del clúster de Kafka[14] y también realiza un seguimiento de las particiones. Por sí mismo permite que varios clientes realicen lecturas y escrituras simultáneas y actúa como un servicio de configuración compartida dentro del sistema.

Cuando un usuario final hace un request al sitio web (1) para hacer búsqueda de calidad de datos en Twitter, todos los request están siendo capturados por un proxy desarrollado en NodeJS[13] y luego es derivado al servicio de "*kafkaProducer*". Este servicio está desarrollado en Java 8 exponiendo una API REST usando el framework de Spring Boot[15]. La API comienza la búsqueda creando un stream de conexión con Twitter mediante la librería de *hbc*[16] e instanciando un nuevo tópico con un identificador único universal (UUID v6) en Kafka[13]. A medida que los tweets persisten en la cola de mensajes de Kafka, estos quedan en espera a ser "consumidos". Una vez que alguien, suscripto al tópico de esa cola de Kafka comienza a recibir el evento, desencola y envía al cliente por medio de un túnel TCP. Para mostrar el resultado al cliente en la interfaz visual, la UI necesita de un proxy que consume los datos del productor y los envía a la UI mediante un web socket utilizando el framework socket.io. Por otra parte, el servicio de NodeJS[12] usa el framework ExpressJS[12] para exponer la API REST, mediante el cual la interfaz del usuario puede hacer request para una nueva búsqueda.

Finalmente, la interfaz de usuario web está construida sobre la biblioteca dc.js[2], que utiliza un framework de procesamiento llamado crossfilter.js y la biblioteca de visualización de datos d3.js. Toda la lógica para el procesamiento se encuentra de este lado; por lo que es el navegador del usuario quien recibe todos los datos y luego hace promedios y cálculos.

[2] **dc.js** es una biblioteca de gráficos para Javascript con soporte nativo de filtro cruzado, que permite una exploración altamente eficiente en grandes conjuntos de datos. Aprovecha la librería de d3 para representar gráficos en formato SVG compatible con CSS. Los gráficos representados con dc.js están basados en datos y son reactivos y, por lo tanto, proporcionan información instantánea sobre la interacción del usuario

PROBLEMAS HALLADOS EN LA IMPLEMENTACIÓN

El trabajo antes descrito presenta algunos problemas que la presente propuesta resuelve, a través, principalmente de una nueva arquitectura, que describiremos más adelante.
Durante la revisión de la implementación se han encontrado algunas limitaciones no solo en lo que respecta a la funcionalidad, sino que también en cuanto a los datos y valores observados finalmente por el usuario. Dichos errores funcionales deben ser arreglados previo a continuar con el plan de mejoras y agregar nueva funcionalidad al sistema.

1. UI monolítica

   Si bien toda la interfaz del usuario había sido hecha por medio del framework de AngularJS, no se hacía uso de las ventajas de usar este tipo de herramientas: trabajar por componentes. Toda la interfaz está diseñada sobre un mismo componente, haciéndolo poco escalable y difícil de modularizar en caso de querer agregar nuevas funciones y visualizaciones.

2. UI no adaptable a otros dispositivos

   Tampoco se hacía uso de las ventajas de usar AngularJS en cuanto a adaptación de tipos de pantallas, por lo que la experiencia al usuario estaba limitada en dispositivos mobile. Desde otra perspectiva, tampoco era adaptable a dispositivos mobile dado que el core de procesamiento de las visualizaciones se resuelven desde el lado del cliente, por lo que en dispositivos con poca memoria o poca capacidad de cómputo era imposible de reproducir.

3. Problemas de performance y saturación de memoria del navegador por procesamiento

   El principal problema hallado fue que luego de una cierta cantidad de tweets, el navegador del usuario se ralentizaba, haciendo que se deba forzar el cierre del proceso por medio del sistema operativo. Esto se debía a que la lógica en el cálculo de las métricas estaba del lado del cliente y no del servidor, mientras que el navegador seguía recibiendo tweets por medio del Websocket enviado por el proxy (y consecuentemente desencolado del tópico correspondiente de Kafka)

4. Pérdida de los datos

   Como los resultados desencolados del tópico de Kafka eran recibidos por el cliente, en ningun momento habia un agente intermediario capaz de persistir los resultados eficientemente; por lo que no se podían hacer consultas, comparaciones ni análisis de búsquedas pasadas.

5. Seguridad de usuarios

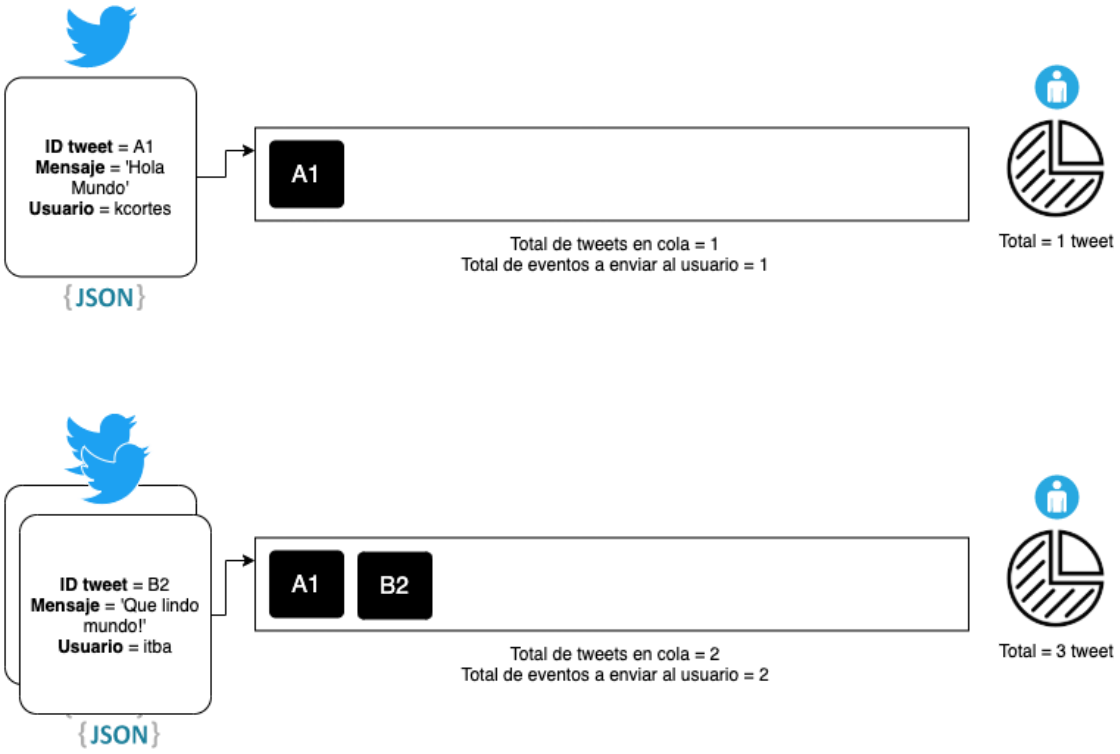
   Cualquier usuario que contaba con la URL podía ingresar y hacer consultas sobre la plataforma.

6. Análisis basados en previas búsquedas

   En correlación con el ítem 4 de pérdida de los datos, tampoco se podían hacer comparaciones de métricas de calidad de datos con el contexto del tiempo. Las métricas de la calidad de los datos puede variar significativamente si se toma en cuenta algunos sucesos en el tiempo.

7. Datos repetidos en la cola de Kafka

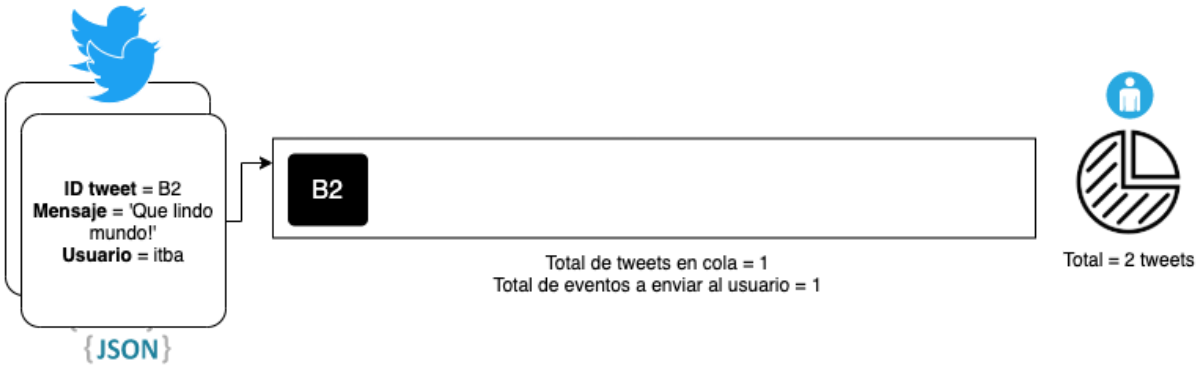
   Inicialmente, la cola de tweets de un tópico, traía datos repetidos. Lo que sucedía era que el proxy NodeJS, encargado de hallar eventos y enviarlos al cliente, enviaba siempre datos que ya habían sido enviados anteriormente porque la configuración no estaba contemplado para borrar el mensaje. Para poner un ejemplo, se puede apreciar el siguiente gráfico de correlación de eventos



Siguiendo el ejemplo del gráfico anterior, suponemos que llega un tweet de id A1 al tópico de Kafka. Luego, el proxy detecta el evento y lo envía al cliente. Este último observa que hay 1 tweet y hace la composición de todos los gráficos con dicho dato.

Si el servicio de *kafkaProducer* recibe un segundo tweet, lo encola al mismo tópico de Kafka. Consecuentemente, el consumer detecta un segundo evento y envía todos los datos al cliente: el ya previamente enviado y el nuevo. Este error hace que las métricas, si bien individualmente son correctas, en su conjunto no lo son ya que los promedios y medias toman casos repetidos.

Lo que debe hacer la cola de Kafka es limpiar el mensaje de id A1 cuando envia satisfactoriamente el mensaje por el Websocket. De esta forma, lo correcto luego de enviar el mensaje de id A1 está ejemplificado en el siguiente diagrama:

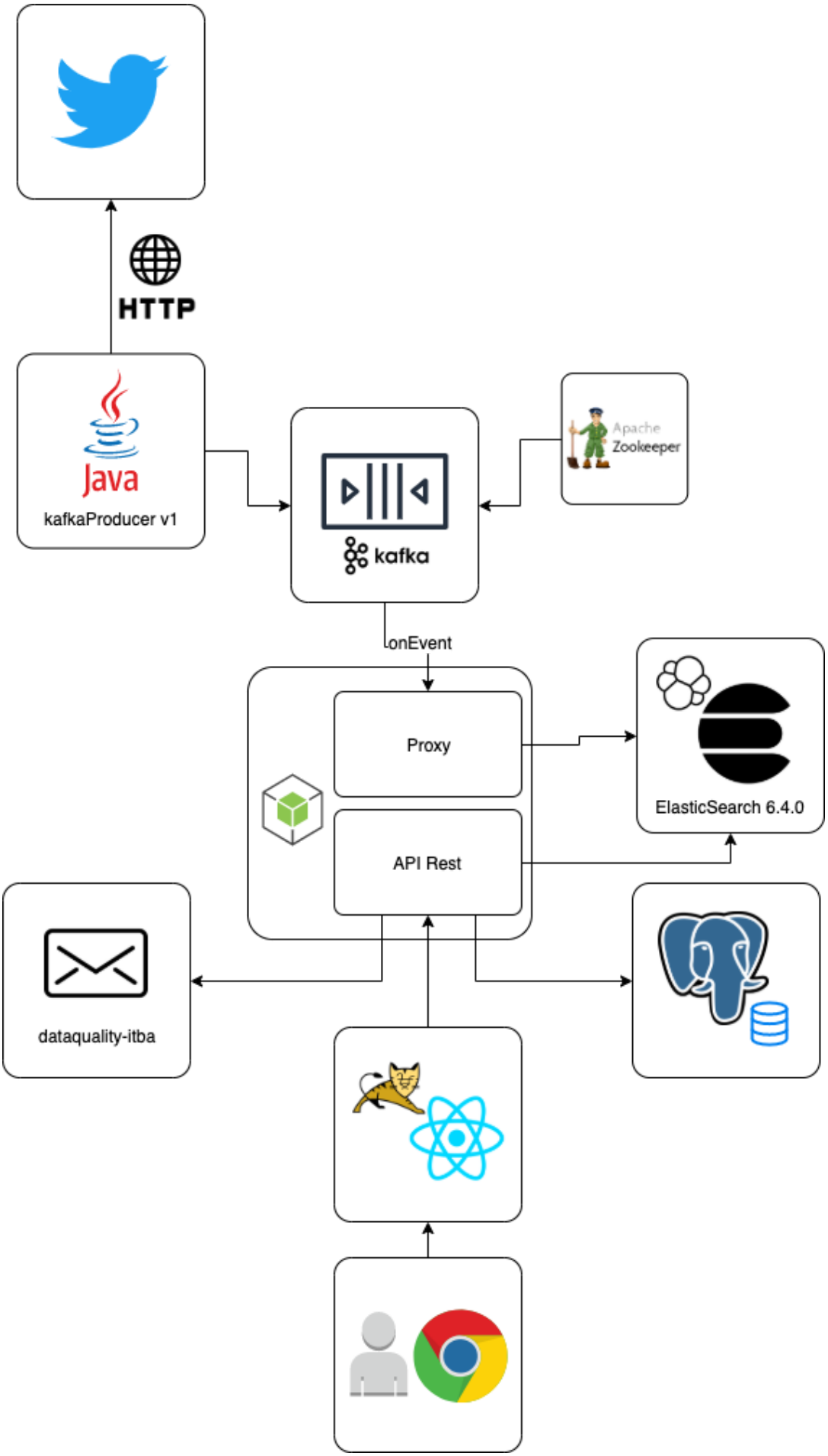


NUEVA IMPLEMENTACIÓN

Los objetivos del proyecto son:

1. Mejorar la implementación existente mediante un cambio de arquitectura.
2. Extender el análisis de la calidad de los datos de Twitter incorporando la noción de contexto

El primer objetivo se ha logrado luego de varias iteraciones, llegando a una arquitectura integral que contempla nuevas herramientas (como base de datos tanto SQL como NoSQL) y otros protocolos posteriormente detallados. Además, como parte del cambio de arquitectura, se ha iterado por distintas soluciones hasta coincidir con una solución idónea en tanto que la lógica de la interfaz y cálculos no sea implementada desde el lado del cliente que, como ya hemos mencionado, satura el navegador del cliente y hace inviable la navegación en el sitio web. Por consiguiente, se ha decidido migrar todo al lado del servidor ya que en caso de una alta demanda por parte de los usuarios, podría escalar horizontalmente y exponer una API Rest.


HTTP
Java
kafkaProducer v1
Apache
Zookeeper
kafka
onEvent
Proxy
API Rest
ElasticSearch 6.4.0
dataquality-itba

La API REST está desarrollada en NodeJS v10.16.3[12] utilizando el framework de ExpressJS. Además se agregan nuevos módulos externos para la comunicación con las bases de datos PostgreSQL y ElasticSearch[8]; y un framework que utiliza el protocolo de comunicación SMTP para comunicarse con el servidor de mail de Gmail, con la cuenta dataquality-itba@gmail.com[3].

Un nuevo servicio sigue cumpliento la funcionalidad de proxy (desencolar de Kafka y enviar ante un evento un tweet), pero ahora en lugar de hacerlo al cliente por un WebSocket, lo hace a la base de ElasticSearch[8] al puerto 9200, en la que por cada búsqueda se genera un nuevo índice de igual nombre que el tópico de Kafka.

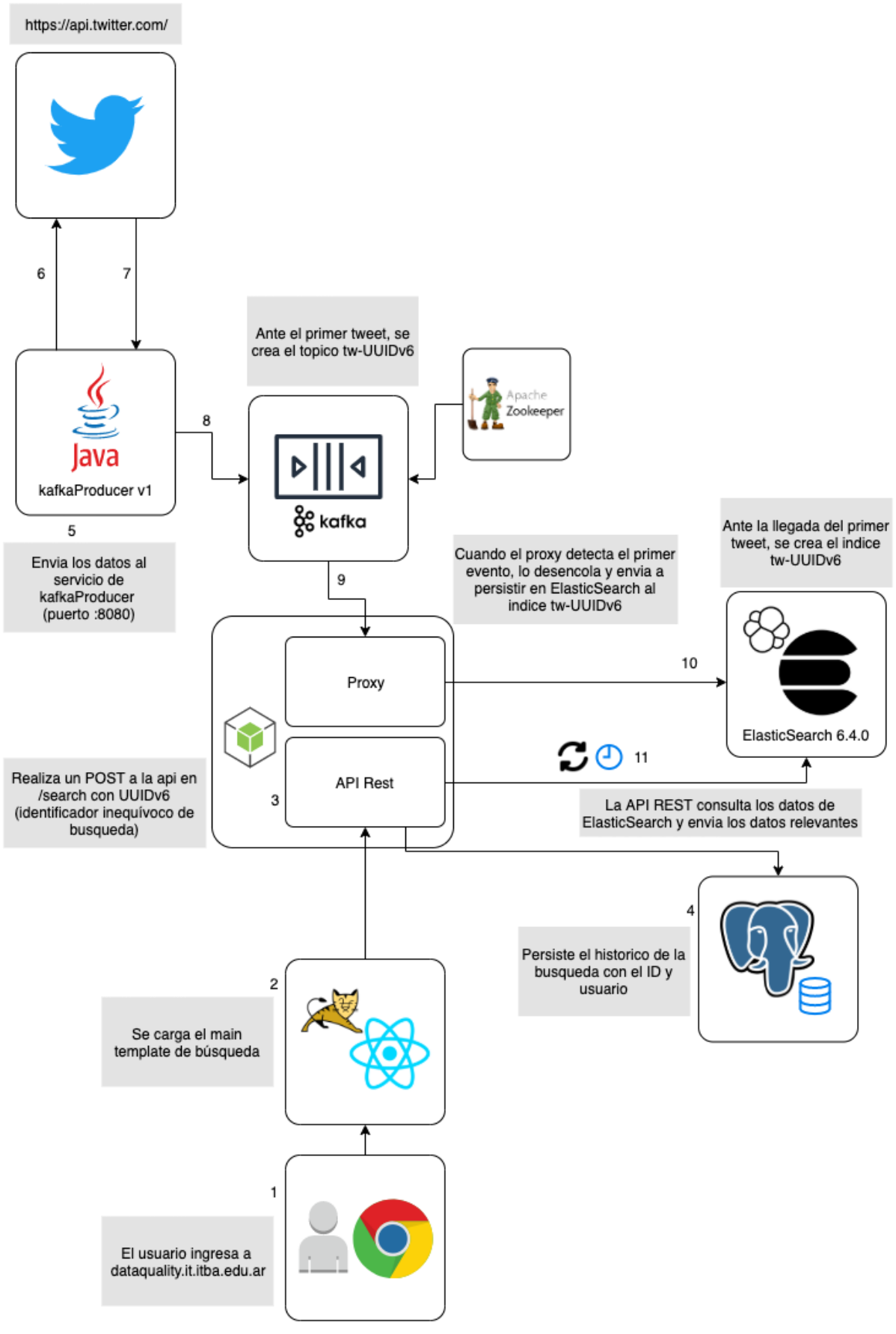
Una vez que el usuario ingresa al sitio mediante su usuario y contraseña, podrá realizar las búsquedas que considere necesario. Inicialmente, el usuario carga los datos para comenzar la búsqueda en Twitter y desde la interfaz UI hará un request POST a la API expuesta. Luego, el proxy atenderá su pedido y redirigirá el mismo al servicio de kafkaProducer, encargado de establecer la sesión con Twitter y persistir en un tópico particular de Kafka los mensajes que vaya encontrando. El nombre del tópico es un identificador inequívoco universal de 32 bytes.

A medida que Twitter encuentre resultados de búsqueda, enviará los datos en tiempo real al servicio de *kafkaProducer*, encargado de aplicar los valores de calidad de datos ya mencionados en el informe. Posteriormente, el microservicio que atiende al consumer (*PF-Consumer*) recibirá los eventos que vayan encolándose al tópico de Kafka y persiste los datos a un nuevo índice de ElasticSearch[9]. En paralelo la API Rest (*PF-NewProxy*) puede agregar datos adicionales basados en las preferencias del usuario, como por ejemplo hallar keywords en el mensaje del tweet que le resulte interesante [dicha configuración la veremos en un apartado siguiente].

Con el lapso del tiempo, el usuario final puede monitorear el estado de la búsqueda en tiempo real y frenar la búsqueda.

Para ejemplificar, a continuación está detallado paso a paso es el flujo de envio y recepcion de datos desde que el cliente hace un pedido desde la interfaz Web (1), llega a Twitter (6, 7), hasta persistir en el motor de ElasticSearch (11).

[3] Se ha decidido implementar que la lógica de envío de mails sea mediante el servidor de mail SMTP de Google considerando eventualmente en el futuro se use un mail institucional de la Universidad.


https://api.twitter.com/
6
7
kafkaProducer v1
5
Envia los datos al servicio de kafkaProducer (puerto :8080)
8
Ante el primer tweet, se crea el topico tw-UUIDv6
Apache Zookeeper
kafka
9
Cuando el proxy detecta el primer evento, lo desencola y envia a persistir en ElasticSearch al indice tw-UUIDv6
Ante la llegada del primer tweet, se crea el indice tw-UUIDv6
Proxy
10
ElasticSearch 6.4.0
11
API Rest
3
Realiza un POST a la api en /search con UUIDv6 (identificador inequívoco de busqueda)
La API REST consulta los datos de ElasticSearch y envia los datos relevantes
4
Persiste el historico de la busqueda con el ID y usuario
2
Se carga el main template de búsqueda
1
El usuario ingresa a dataquality.it.itba.edu.ar

IMPLEMENTACIÓN DE API REST

Tal lo comentado anteriormente, la API REST está desarrollada en Node JS, utilizando distintos frameworks, como ExpressJS para exponer las rutas de acceso vía HTTP, el módulo de Elasticsearch, express-jwt (JSON Web Tokens), gmail-send para el envío de mails mediante SMTP al servidor de Gmail, google-trends-api, jwt-decode y sequelize para el manejo de los datos de la base de datos PostgreSQL.

El patrón utilizado es el de Model-View-Controller, es decir que cada ruta o contexto web HTTP mappea con un Controller particular y este mismo está relacionado con uno o más modelos, dependiendo el caso. El diagrama de clases, detallado en las próximas páginas, explica a grandes rasgos la comunicación entre componentes.

El punto de acceso de los usuarios es mediante la interfaz web deployada en *dataquality.it.itba.edu.ar*. Los usuarios no pueden acceder por medio de otro acceso que no sea ese ya que no está habilitado CORS[4] para todos los dominios. Este es un mecanismo que permite que se puedan solicitar recursos restringidos en una página web desde un dominio fuera del dominio que sirvió el primer recurso.

El punto de entrada al usuario es mediante las “routes”, donde cada ruta puede ejecutar uno o más controllers. Cada controller recibe como parámetros obligatorios dos callbacks, uno de salida con errores y otro de salida sin errores. Esto se debió hacer ya que la mayoría de las comunicaciones con ElasticSearch o mismo el motor de mensajería de mail son asincrónicos por lo que no se puede garantizar el momento de finalización de su ejecución. Cada Controller habla con su correspondiente modelo, que es un simple mapeo objeto-relacional con la base PostgreSQL. Para poder lograrlo, se utilizó el framework “Sequelize”, el soporte ORM para base de datos relacionales en NodeJS. Dicho framework tiene una sintaxis basada en objetos de JSON que lo traduce a una query SQL. La ventaja de usar un ORM como este es la capacidad y facilidad de migrar de distintos motores como PostgreSQL, SQL Server o MySQL sin tener que hacer eventuales modificaciones en el código.

Para la autenticación, la aplicación usa JSON Web Tokens. Un usuario hace login, y posteriormente el servidor retorna un token válido como respuesta si los parámetros de ingreso fueron correctos. El usuario debe enviar dicho token en las siguientes peticiones para poder acceder a los recursos del servicio como header de cada request enviado a la

[4] El Intercambio de Recursos de Origen Cruzado (CORS) es un mecanismo que utiliza cabeceras HTTP adicionales para permitir que un user agent obtenga permiso para acceder a recursos seleccionados desde un servidor, en un origen distinto (dominio) al que pertenece.

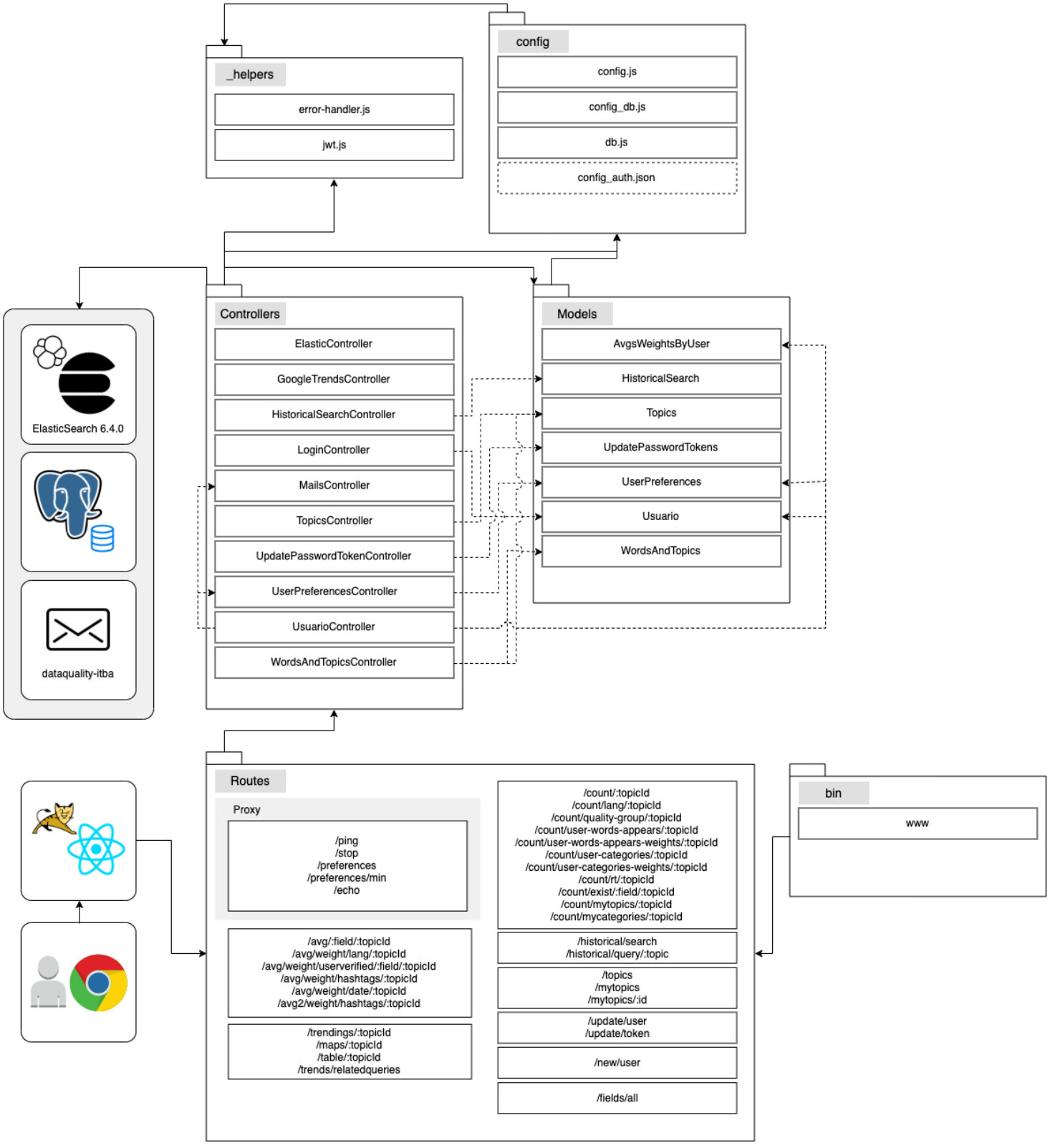
API Rest. En cada petición el servidor debe comprobar el token proporcionado por el usuario y si es correcto podrá acceder a los recursos solicitados, de otra forma deberá denegar la petición.
El formato de un JWT está compuesto por 3 strings separados por un punto, en la que cada string de la cadena representan datos distintos:

1. Header: La primera parte es la cabecera del token, que a su vez tiene otras dos partes, el tipo y la codificación utilizada (HMAC SHA256).
2. Payload: Está compuesto por los llamados JWT Claims donde irán colocados la atributos que definen nuestro token.
    a. sub: Identifica al usuario por medio de un ID.
    b. iat: Identifica la fecha de creación del token.
    c. exp: Identifica a la fecha de expiración del token.
3. Signature: Tercera y última parte del JSON Web Token. Está formada por los anteriores componentes (Header y Payload) cifrados en Base64 con una clave secreta (almacenada en nuestro backend). Así sirve de Hash para comprobar que todo está bien.

Cuando la API recibe por parámetro en el header el Authorization Bearer (token codificado con JWT), verifica junto con jwt-decode y la clave privada los datos del mismo. Si son válidos, ejecuta el request y devuelve la respuesta solicitada. Caso contrario, envía una respuesta *"401 Unauthorized"*.

Las rutas, o punto de acceso al cliente, están validadas por tokens. Es decir, exceptuando el login, todos los pedidos deben recibir en el header de Authorization Bearer. Las rutas están divididas por funcionalidad. Por una parte contamos con las rutas heredadas de la versión anterior del proxy, y otras representan los datos de las búsquedas que luego el front-end, diseñado en React, visualizará mediante distintos tipos de charts. La mayor parte de estos requests consultan a *ElasticController*, encargado de la comunicación con ElasticSearch.

Dentro de este Controller, existen distintos tipos llamados a ElasticSearch: los que retornan solo un valor numérico, o mismo inclusive los que devuelven una lista de valores. A pesar de que el dato a retornar sea distinto, los requests son prácticamente muy similares, aunque con particularidades. El body para la busqueda esta en formato "Lucene". El centro de la arquitectura lógica de Lucene se encuentra el concepto de documento que contiene campos de texto. Esta flexibilidad permite a Lucene ser independiente del formato del fichero.


_helpers
error-handler.js
jwt.js
config
config.js
config_db.js
db.js
config_auth.json
ElasticSearch 6.4.0
dataquality-itba
Controllers
ElasticController
GoogleTrendsController
HistoricalSearchController
LoginController
MailsController
TopicsController
UpdatePasswordTokenController
UserPreferencesController
UsuarioController
WordsAndTopicsController
Models
AvgsWeightsByUser
HistoricalSearch
Topics
UpdatePasswordTokens
UserPreferences
Usuario
WordsAndTopics
Routes
Proxy
/ping
/stop
/preferences
/preferences/min
/echo
/avg/:field/:topicId
/avg/weight/lang/:topicId
/avg/weight/userverified/:field/:topicId
/avg/weight/hashtags/:topicId
/avg/weight/date/:topicId
/avg2/weight/hashtags/:topicId
/trendings/:topicId
/maps/:topicId
/table/:topicId
/trends/relatedqueries
/count/:topicId
/count/lang/:topicId
/count/quality-group/:topicId
/count/user-words-appears/:topicId
/count/user-words-appears-weights/:topicId
/count/user-categories/:topicId
/count/user-categories-weights/:topicId
/count/rt/:topicId
/count/exist/:field/:topicId
/count/mytopics/:topicId
/count/mycategories/:topicId
/historical/search
/historical/query/:topic
/topics
/mytopics
/mytopics/:id
/update/user
/update/token
/new/user
/fields/all
bin
www

Tal como mencionamos en el apartado anterior, el siguiente gráfico representa el esquema de modelos y clases utilizados para la API Rest.

El proyecto se deploya al servidor mediante *forever*, una herramienta por comando de línea simple para garantizar que el script (archivo *www* situado en el directorio *bin)* se ejecute continuamente. El archivo a deployar contiene, entre otras funcionalidades, el puerto en el que el usuario realizará los requests y las rutas expuestas públicamente. Por defecto, el puerto es el 9001.

Las rutas están divididas en varios segmentos, principalmente los que hacen requests llamados a *ElasticController*, los que hacen llamada a los Controllers de Base de Datos SQL o al *PF-Consumer* (Proxy). Siguiendo el patrón Model-View-Controller, todas las rutas no poseen lógica de negocio, sino que llaman a uno o más controladores, dependiendo el caso.

Todos los controladores tienen obligatoriamente dos parámetros de entrada:

1. Topic ID: UUIDv6 del tópico de la búsqueda. Utilizado para hacer las queries con el índice correcto a ElasticSearch.
2. CallbackOK: Función de ejecución de salida en caso que la respuesta sea válida y completa.

También hay un tercer parámetro, pero no es obligatorio en la mayor parte de los casos, y es el CallbackERR, cuya función es similar a la de CallbackOK pero en casos donde la respuesta no sea válida.

Los controladores de Base de Datos Relacionales hacen llamadas a los modelos, que mediante Sequelize, mapean uno o más datos de una tabla de la base de datos a un modelo JSON independientemente del motor de base de datos. Los datos de acceso y configuración a la Base de Datos se encuentran parametrizados en dos archivo: *config_db.js* y *db.js.*

Finalmente, el directorio *helpers* contiene dos archivos que corresponden a configuraciones varias en cuanto a mensajes de error en caso de no estar autenticado o de rutas que no lo requieren; es decir, cuyos requests llegan sin el header Authorization Bearer y son aceptadas por la API. Este último caso se da en ocasiones de login o cuando el usuario quiere restaurar su password.

BASE DE DATOS RELACIONAL

En la base de datos PostgreSQL no se almacenan datos de los tweets específicamente, sino que se guardan otro tipo de datos, como preferencias del usuario, historial de búsqueda o datos de acceso (usuario, contraseña y mail).

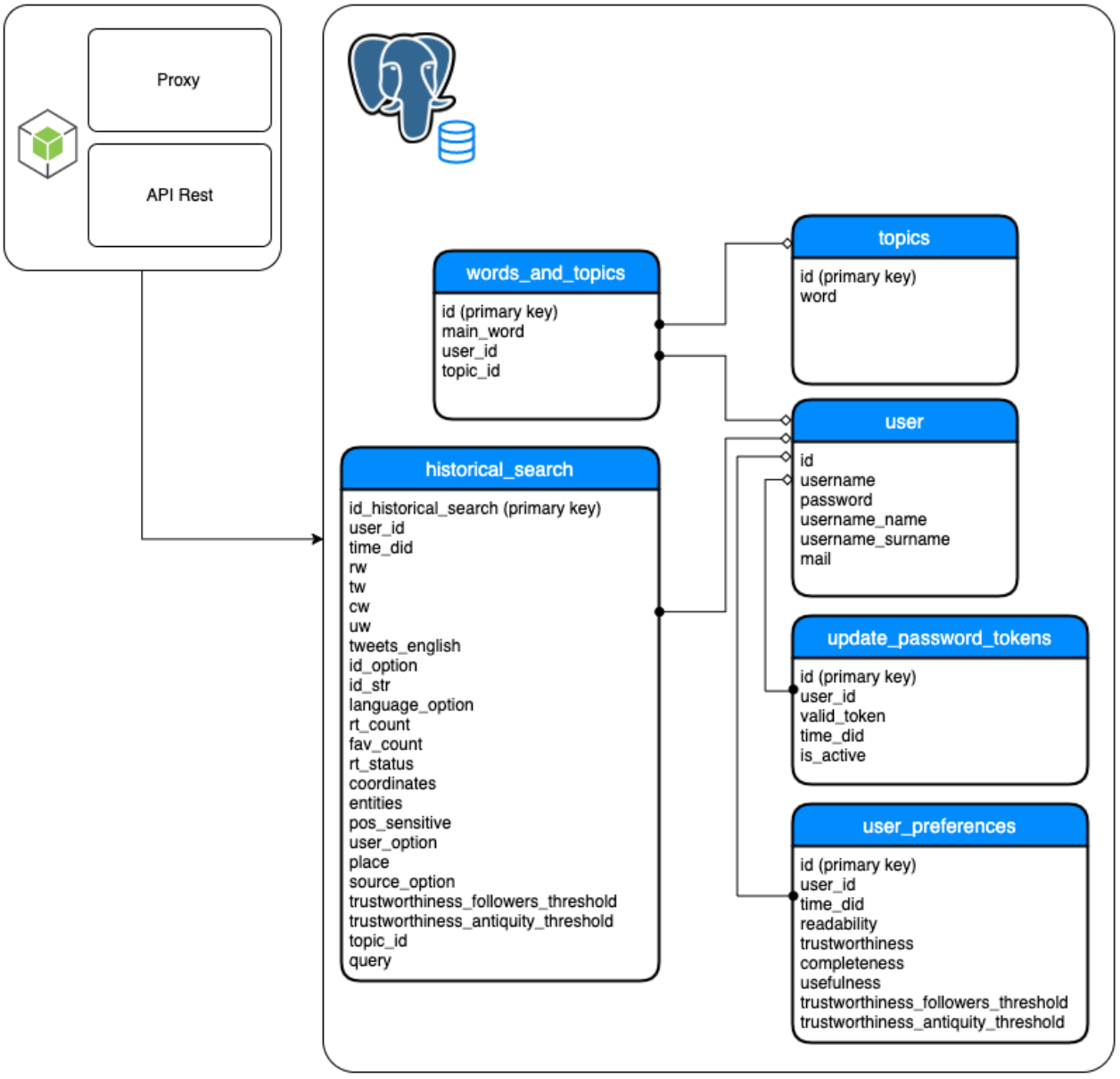
La tabla principal es la de *users* ya que toda la información persistida en todas las tablas debe tener una referencia a quien la ejecutó. Adicionalmente, es la tabla que persiste la información básica del usuario para la autenticación al sistema. La API Rest validará si la información obtenida en el login para un usuario coincide con la persistida en la base de datos y de allí genera el token.

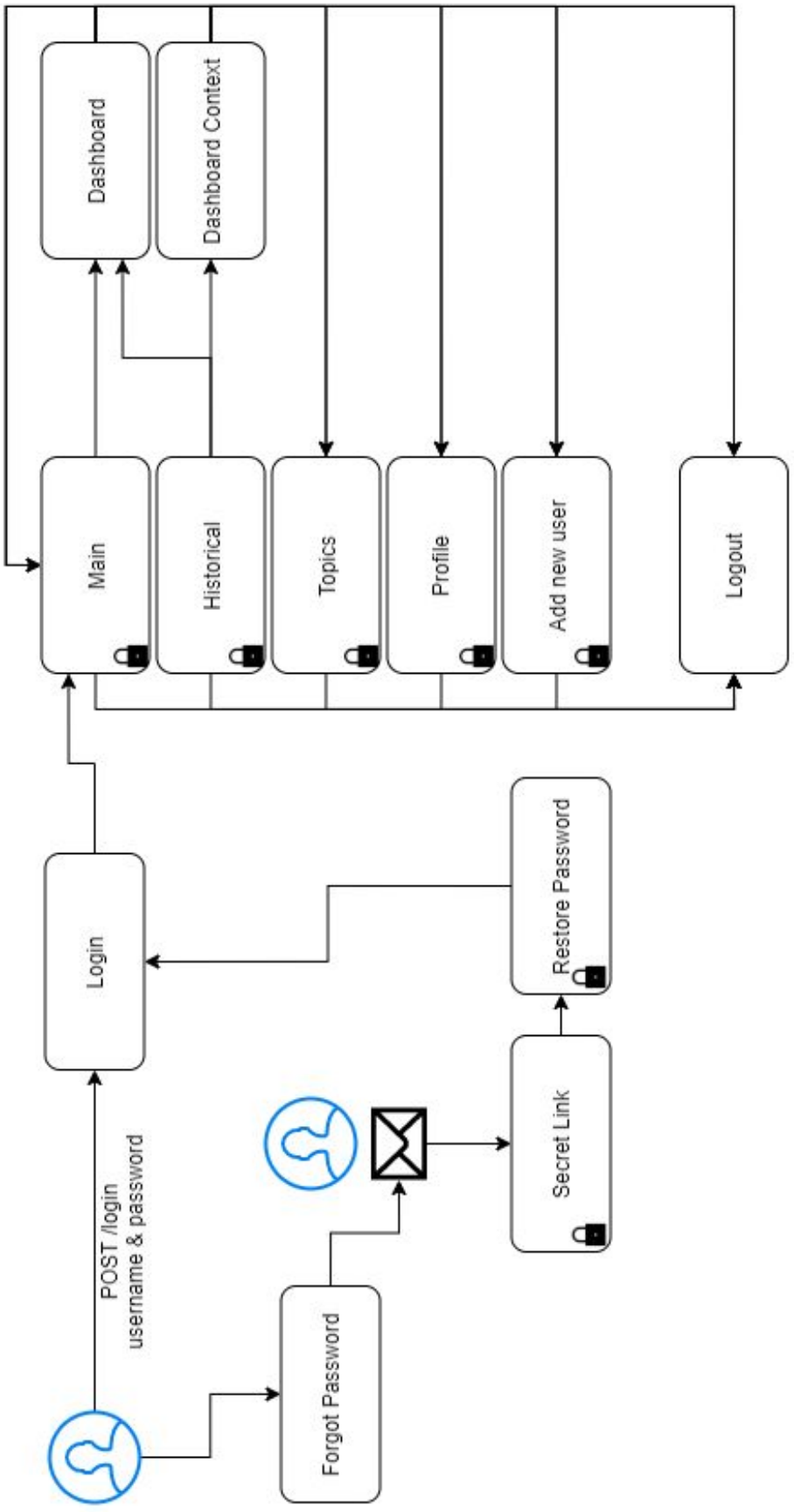
La tabla *words_and_topics* es la que contiene la información de keywords relevantes para el usuario, en la que buscará en todos los mensajes alguna de las palabras en la lista del usuario. El usuario puede almacenar palabras claves y hacerlas corresponder con ciertos tópicos de interés, como por ejemplo: *"todos los tweets que contengan la palabra clave Messi hacerlos vincular con el tópico de Deportes"*.

Por otra parte, está la tabla de *user_preferences* en la que se almacenan los datos de preferencia del usuario cuando realice una búsqueda en Twitter desde la interfaz principal de búsqueda. Una vez que la interfaz al usuario cargue, los componentes se actualizan a los valores por defecto que tiene la tabla. Los valores a guardar corresponden a los umbrales en porcentaje de la calidad de datos (Trustworthiness, Usefulness, Completeness, Readability).

Otra tabla no menos importante es la de *update_password_tokens*, que genera y persiste los tokens para la actualización de la contraseña en caso de olvidarla. Vale aclarar que el token generado no es del mismo tipo cuando el usuario ingresa al sistema, y además tiene fecha de caducidad. Cuando el usuario recibe el mail para cambiar el password, inicialmente irá a dicha tabla a buscar si existe el token. En caso de que así sea, inclusive valida que el campo is_active este en "true". Caso contrario, significa que ya se hizo la actualización con ese mismo token.

Finalmente, la tabla *historical_search* contiene las búsquedas realizadas por el usuario anteriormente. Dicha tabla no contiene información de la búsqueda en sí ya que ese dato lo recolecta mediante el *topic_id* a ElasticSearch. La mayoría de los campos son booleans, ya que cada uno de ellos representa el toolbox de selección en el sitio principal de búsqueda.


Proxy
API Rest
words_and_topics
id (primary key)
main_word
user_id
topic_id
topics
id (primary key)
word
user
id
username
password
username_name
username_surname
mail
historical_search
id_historical_search (primary key)
user_id
time_did
rw
tw
cw
uw
tweets_english
id_option
id_str
language_option
rt_count
fav_count
rt_status
coordinates
entities
pos_sensitive
user_option
place
source_option
trustworthiness_followers_threshold
trustworthiness_antiquity_threshold
topic_id
query
update_password_tokens
id (primary key)
user_id
valid_token
time_did
is_active
user_preferences
id (primary key)
user_id
time_did
readability
trustworthiness
completeness
usefulness
trustworthiness_followers_threshold
trustworthiness_antiquity_threshold


Dashboard
Dashboard Context
Main
Historical
Topics
Profile
Add new user
Logout
Login
Restore Password
Secret Link
POST /login
username & password
Forgot Password

El flujo para el usuario inicia desde que hace login con su usuario y contraseña al sitio. Para ello, otra persona con acceso al sitio deberá registrarlo con su nombre, apellido, una contraseña por defecto y un mail válido. Cuando el usuario es creado, se le envía una notificación al usuario confirmando la apertura de la cuenta. Una vez que cuenta con dicho usuario, el mismo podrá cambiar la contraseña haciendo clic en "Forgot Password" en el sitio principal. En caso de suceder, el usuario deberá ingresar su nombre y esperar la notificación de mail. Dicho mensaje contendrá un link válido por única vez para cambiar la contraseña. Una vez cambiado los accesos, el link no volverá a tener validez.

Una vez que el usuario ha ingresado satisfactoriamente al sitio, la pantalla inicial será la de búsqueda por default.

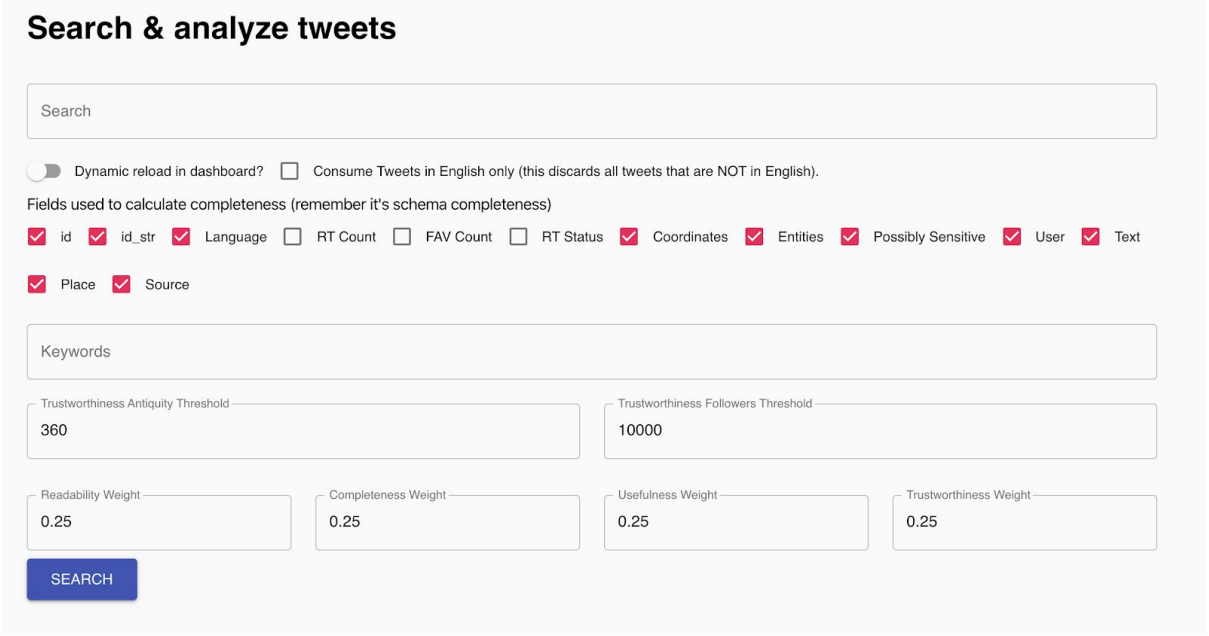
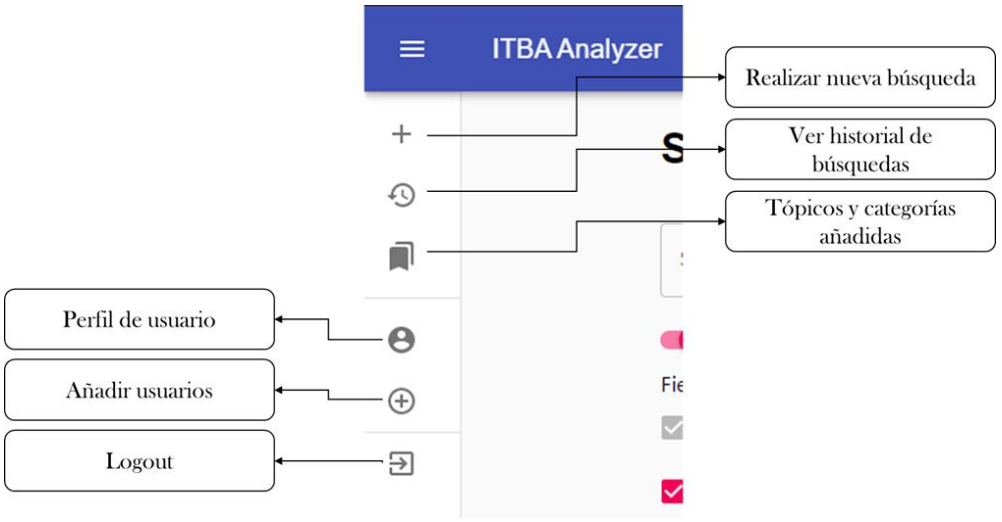


Imagen - Sitio principal de búsqueda una vez ingresado.

El sitio contempla la opción de que el usuario pueda filtrar qué campos desea obtener de la API de Twitter. A su vez, puede hacer búsquedas exactas por palabras, o mismo por Keywords. Por otra parte, también puede configurar los valores numéricos tanto para los umbrales, como para el porcentaje de la calidad de dato (readability, completeness, usefullness y trustworthiness).

Del lado izquierdo hay una serie de opciones en las que el usuario pueda navegar por el sitio.



Haciendo clic en el icono de "Ver historial de búsquedas", el usuario puede acceder al listado de búsquedas hechas en el pasado, con el ID de búsqueda y la query realizada. De allí, puede luego visualizar los distintos tipos de dashboards:

1. Analyze Dashboard: Ingresar al dashboard con todas las métricas
2. Analyze Dashboard Context: Ingresa al dashboard pero con un panel para cambiar el contexto de la calidad de los datos.
3. Compare: Compara dos dashboards seleccionados.

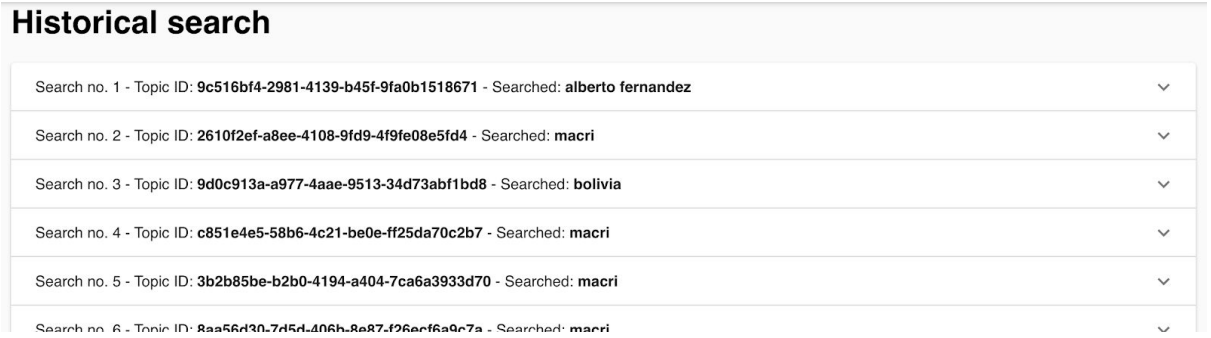


Imagen - Ejemplo de listado ordenados por fecha de búsqueda.

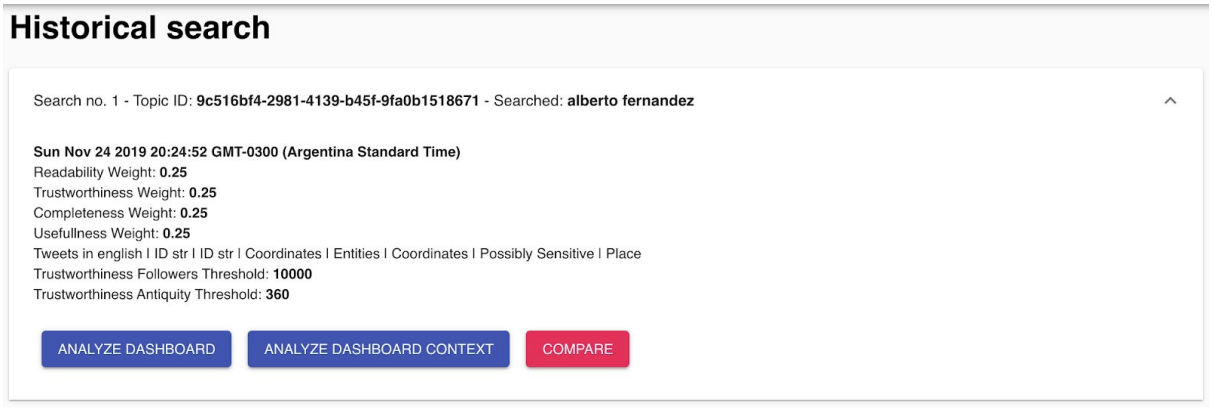


Imagen - Clic con detalle en una de las búsquedas históricas

NUEVAS FUNCIONALIDADES

En esta nueva versión, el usuario podrá subir una lista de “palabras claves” que referencian a un tipo de clase en particular. Por ejemplo, si el usuario quiere saber con respecto a la calidad de los datos que contengan cierta palabra clave, como puede ser el nombre de un político, un país o un tema de interés, puede agregarlo a la lista de “Topics” (a la izquierda en la navegación). La lista es completamente dinámica, por lo que el usuario y administrador podrá eliminar los tópicos cuando lo deseen.

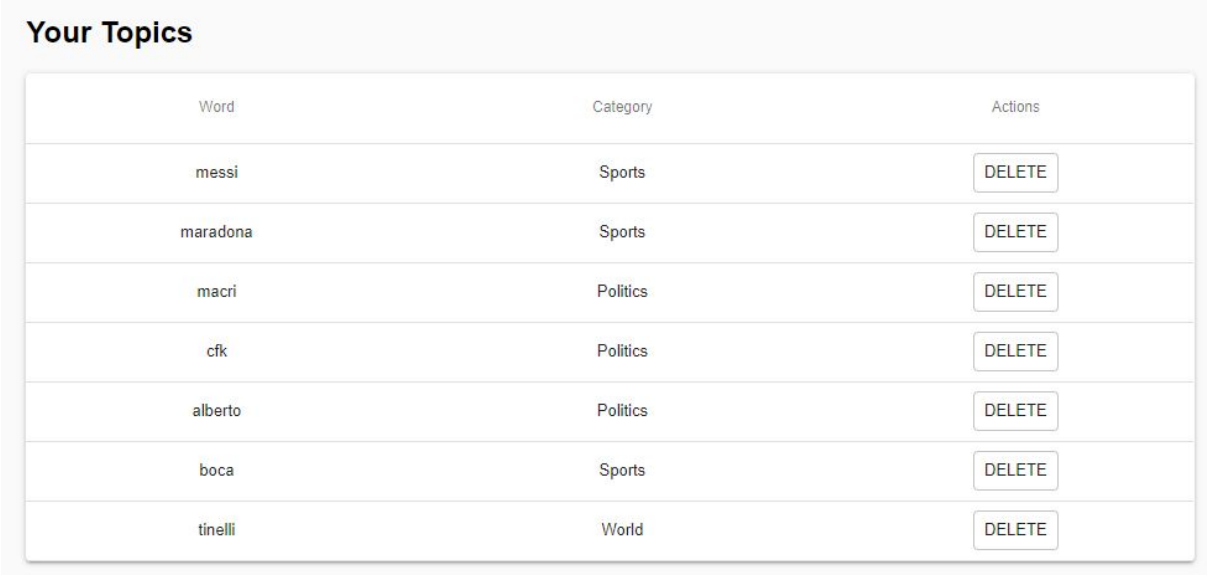


Imagen - Listado de tópicos y categorías almacenadas por el usuario. Nótese que también pueden ser eliminados en cualquier momento

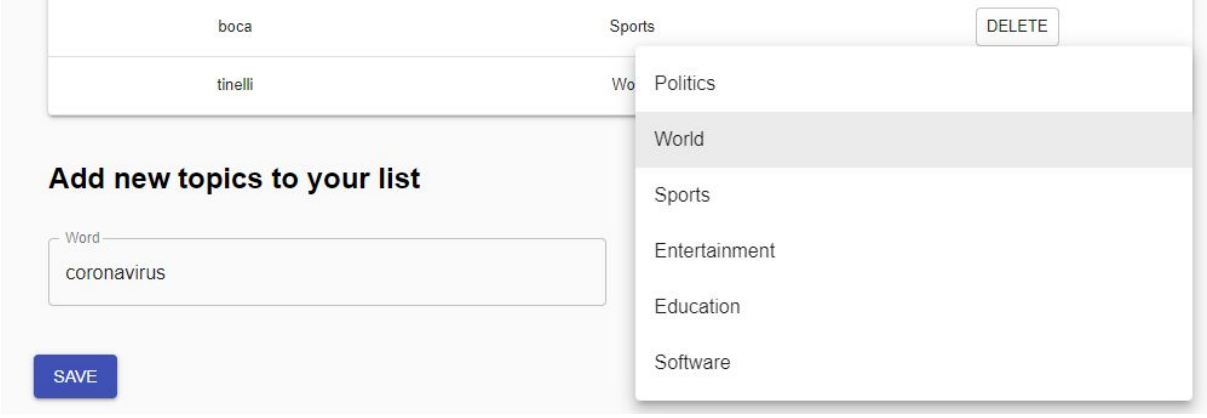


Imagen - El usuario puede agregar palabras que resulten de interés y marcarlas dentro de una categoría en particular. Luego, dicho campo se verá reflejado en la lista de la imagen anterior

Por otra parte, las preferencias para hacer búsquedas también son dinámicas. Es decir, en el ítem de “Profile” a la izquierda de la navegación, el usuario puede saber cual es el promedio de los datos con los que suele hacer la búsqueda y ponerlos como por defecto para próximas búsquedas. Por el contrario, también puede especificar los valores de los campos por defecto.

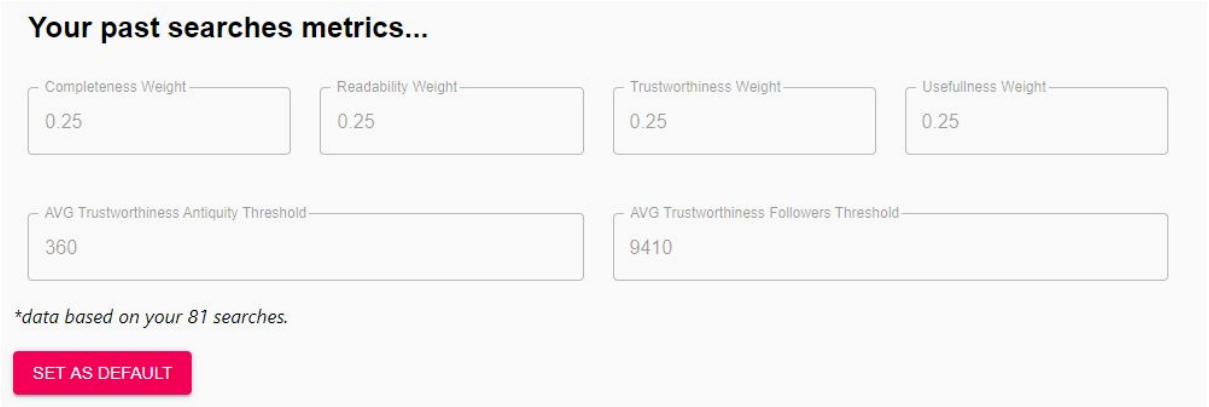


Imagen - El usuario puede ver el promedio general de los campos de las búsquedas que hizo anteriormente

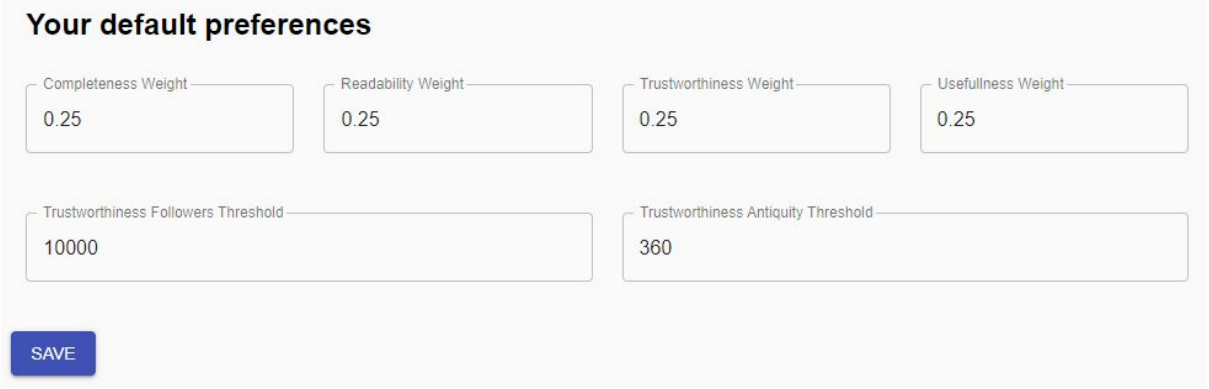


Imagen - El usuario puede ver el promedio general de los campos de las búsquedas que hizo anteriormente

TIPOS DE DASHBOARDS

Para la búsqueda, existe 3 formas de visualizar los campos. A pesar de ello, los campos que se visualizan en todos son los mismos; es decir, en todas se puede ver los trending hashtags, total de tweets, promedio de calidad de datos en sus 4 variables, total de tweets por lenguaje, etc...

Analyze Dashboard

Ingresar al dashboard con todas las métricas

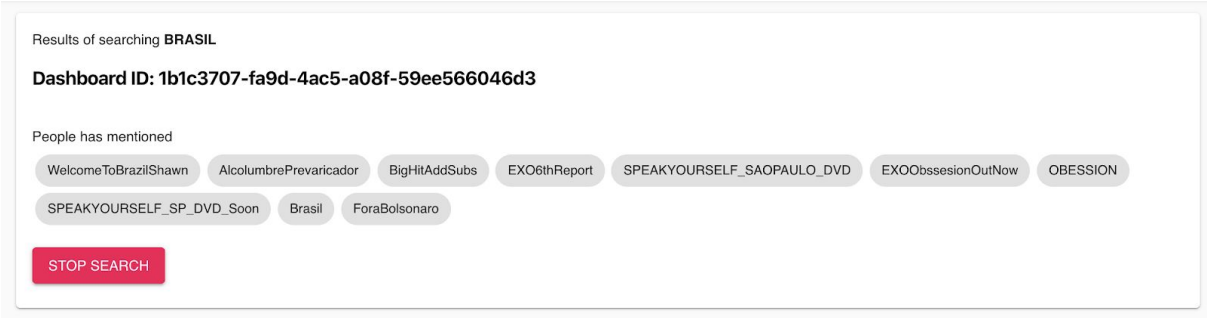


Imagen - Top Hashtags de la busqueda de "Brasil"

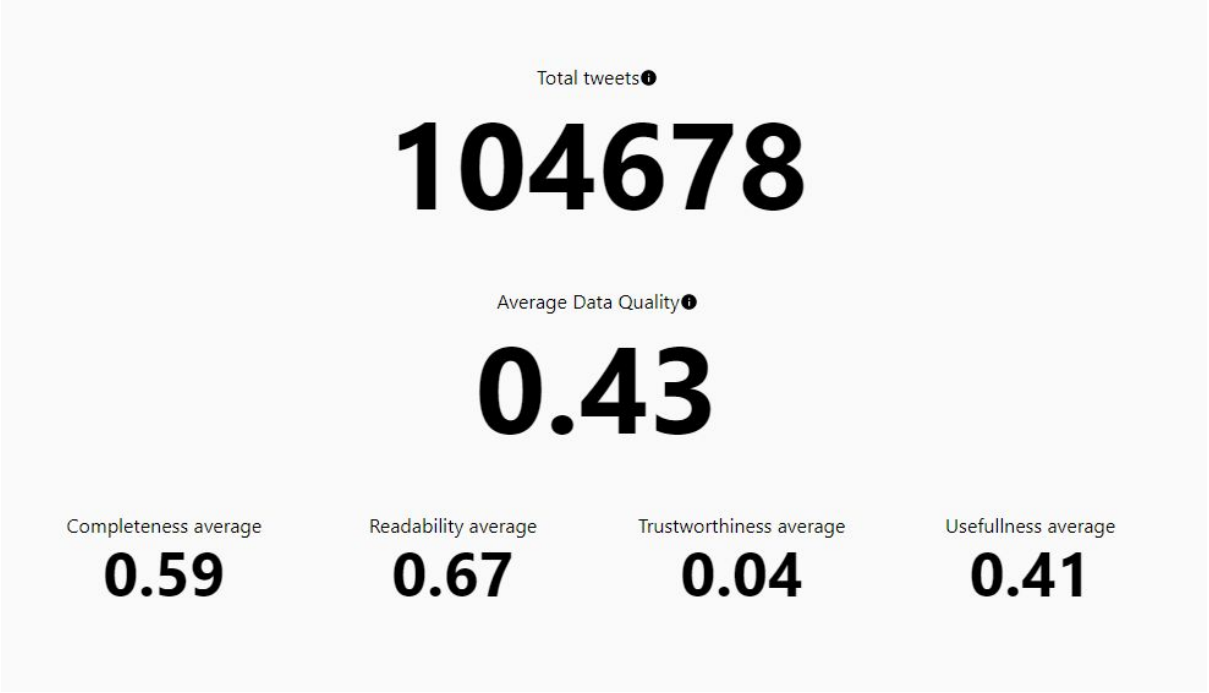


Imagen - Seccion 2 del dashboard. Se visualiza la cantidad de tweets, el promedio general de la calidad de los datos y el promedio de cada una de las distintas calidades calculadas del dato.

Los dashboards presentan distintos tipos de gráficos: promedios y cantidades representadas como número, pie charts, line charts y mapas.
Dentro de los pie charts, podemos encontrar datos que representan la cantidad de tweets divididos por lenguaje, la cantidad de tweets en las que figuran los tópicos agregados previamente por el usuario; o mismo inclusive por la categoría de los tópicos.

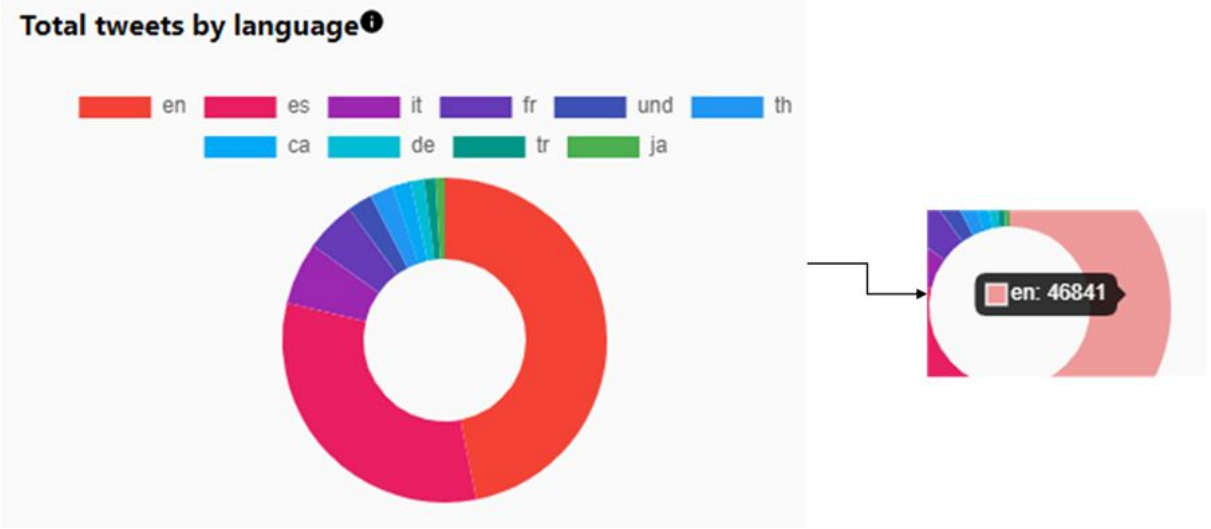


Imagen - Representación del pie chart de cantidad de tweets por idioma. Al pasar el cursor por encima, el usuario puede distinguir el idioma y la cantidad.

Por otra parte, cabe notar que todas las visualizaciones, sean de cualquier tipo, cuentan con un label [] que describe brevemente la información visualizada

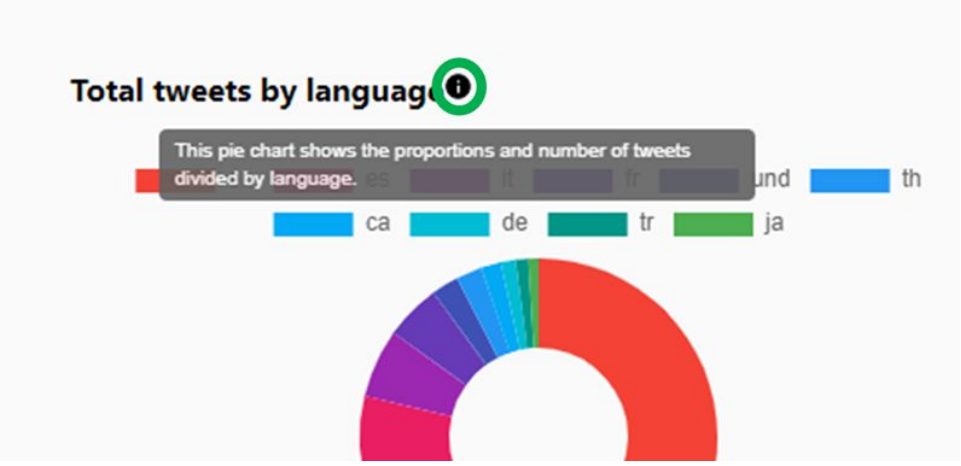


Imagen - Todas las visualizaciones, sean de cualquier tipo, cuentan con un label [] que describe brevemente la información visualizada.

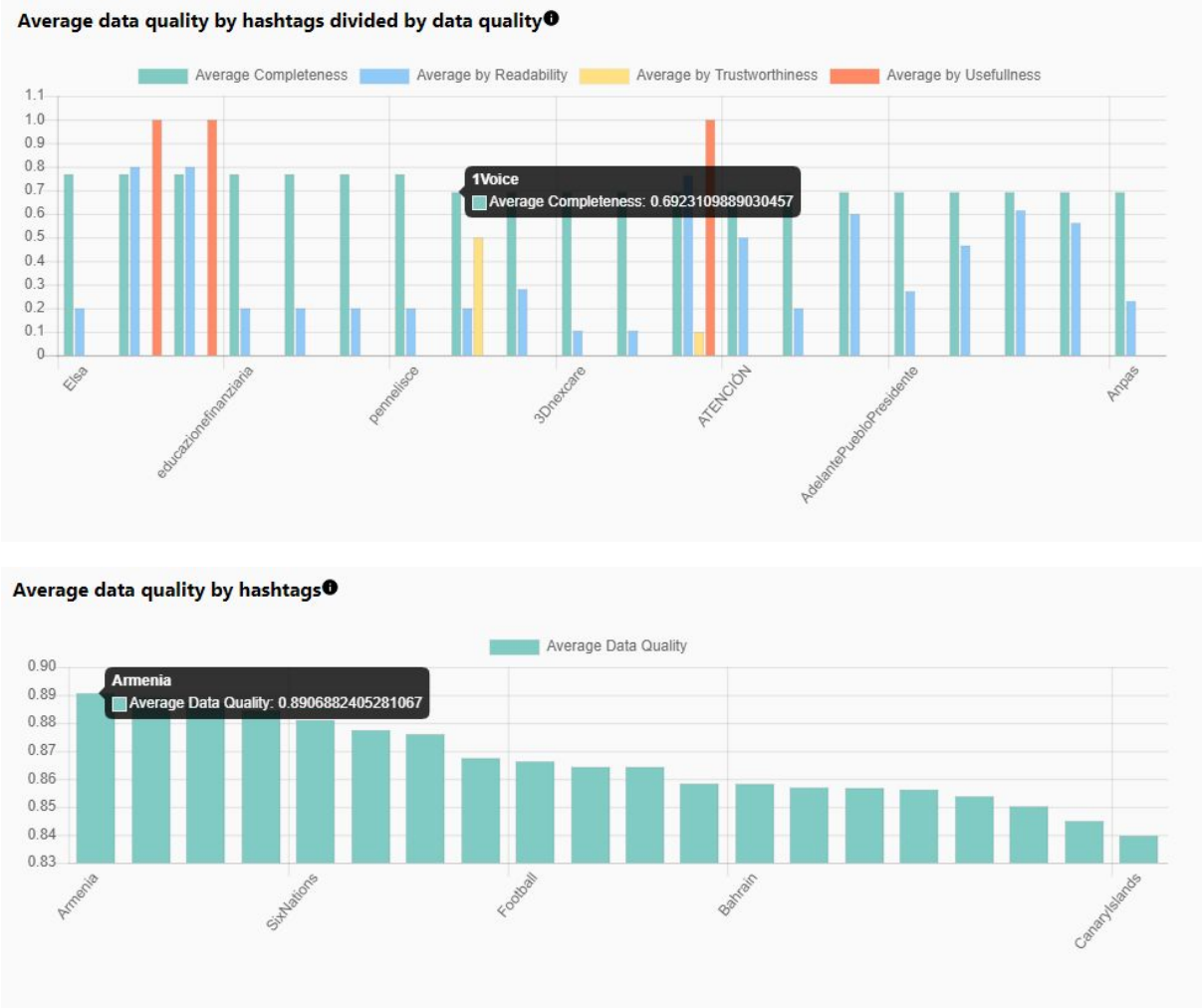


Imagen - Dos tipos de visualizaciones de bar charts. El primero muestra el promedio de la calidad por trending hashtag, es decir, los hashtags más mencionados. El segundo muestra el promedio ponderado de la calidad ordenado

Imagen - Mapa parcial con los puntos donde se han localizado tweets. Haciendo click en ellos se puede ver la cantidad y el promedio de la calidad.

Key	Count	Completeness	Readability	Trustworthiness	Usefullness	Average
Barcelona, España	15	0.626	0.518	0.033	0.000	0.294

Key	Count	Completeness	Readability	Trustworthiness	Usefullness	Average
RT @realDonaldTrump: Cryin' Chuck Schumer is complaining, for publicity purposes only, that I should be asking for more money than $2.5 Bil…	2234	0.583	0.864	0.037	1.000	0.621
RT @SenBlumenthal: This morning's classified coronavirus briefing should have been made fully open to the American people—they would be as…	1526	0.596	0.900	0.029	1.000	0.631
RT @inpattwetrust: Todo el mundo preocupado por el #Coronavirus 😷 Murciano: https://t.co/DixFfuk3Q2	867	0.657	0.417	0.000	0.000	0.268
RT @thidakarn: ชาวอเมริกันที่เดินทางกลับจากประเทศเสี่ยงติด #COVID เลือกที่จะ Self-Quarantine อยู่บ้าน หยุดงาน แยกจากคนในครอบครัว แม้จะลำบาก…	819	0.586	0.200	0.005	0.000	0.198
RT @homeralef: Mucho coronavirus pero a nadie le preocupa que en pleno febrero estemos a 20°. Eso sí que es para decir que	735	0.575	0.630	0.001	0.000	0.302

Imagen - Tabla parcial con los tweets más mencionados / retwitteados. Cada uno de ellos muestra la cantidad de apariciones, el promedio de las calidades de dato y el promedio ponderado.

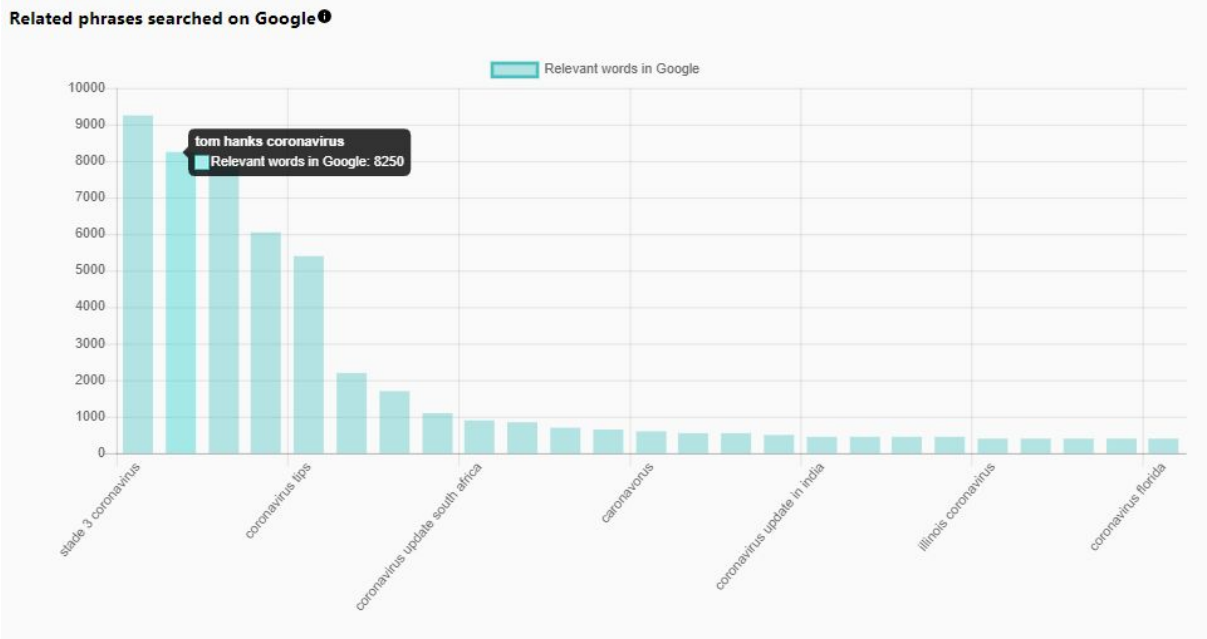


Imagen - Finalmente, hay otras visualizaciones que no tratan los datos de Twitter, sino de las búsquedas de los usuarios desde Google relacionados con la búsqueda original en Twitter.

Analyze Dashboard Context

Ingresa al dashboard pero con un panel para cambiar el contexto de la calidad de los datos.

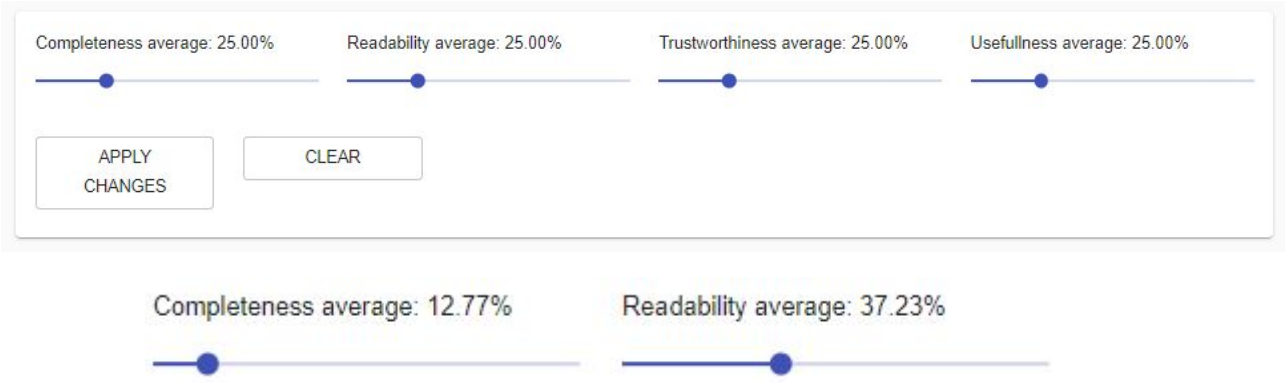


Imagen - Panel de contexto. Cada valor o slider representa en porcentaje a la calidad del dato.

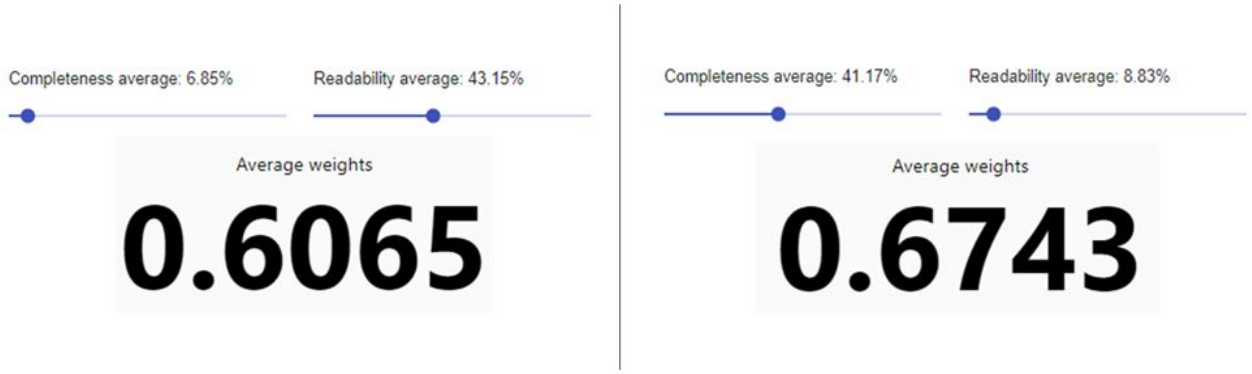
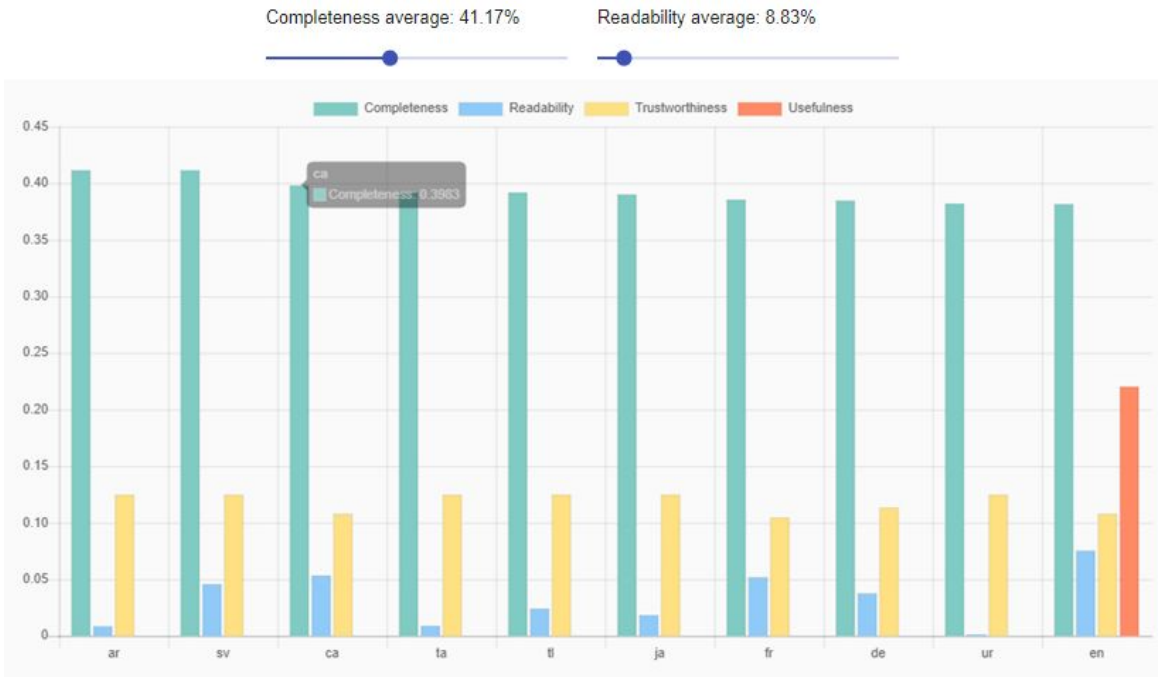


Imagen - Comparación de cómo varía el promedio ponderado si variamos las calidades de "Completeness" y "Readability"



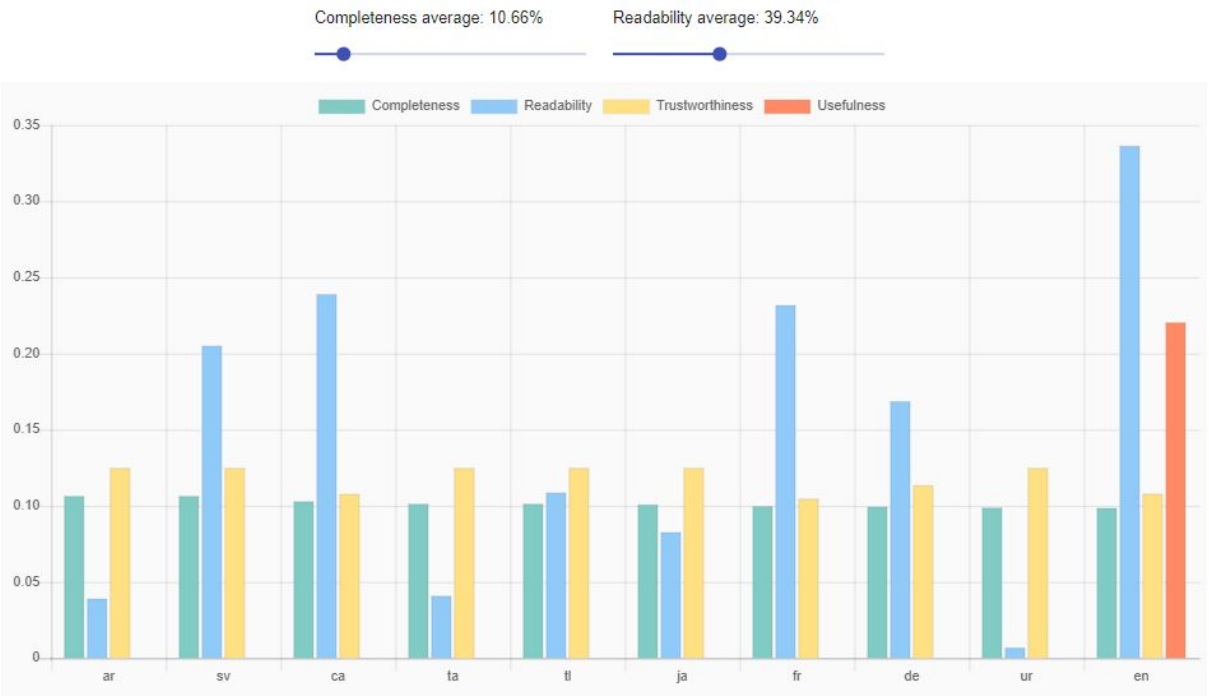


Imagen - Ejemplo de dos bar charts iguales pero variando uno de los datos. Cuando el usuario hace click en el botón "apply changes", todas las visualizaciones del dashboard varían. Cuando la búsqueda se hace en tiempo real no es necesario que el usuario haga click en dicho botón.

Compare

Como parte de las mejoras en la experiencia al usuario a implementar, se ha agregado la opción de que el mismo pueda comparar dos dashboards de dos consultas, distintas o iguales que fueron realizadas en el pasado.

De ésta manera, el usuario puede sacar conclusiones en lo que respecta a distintos momentos temporales de búsqueda.

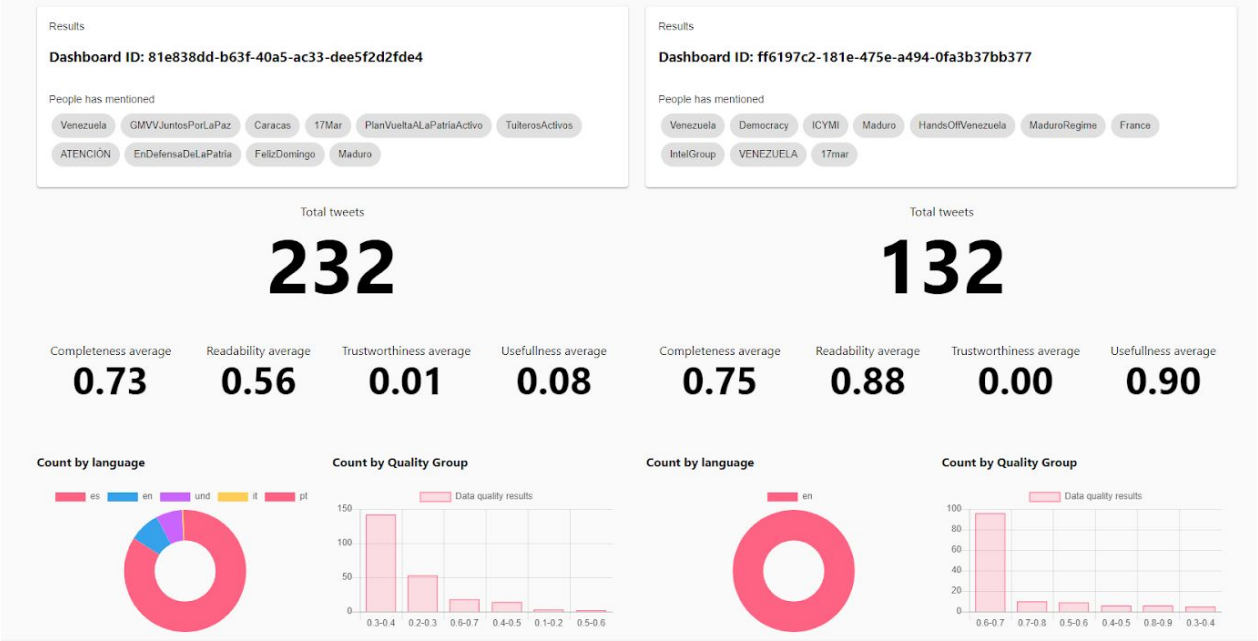


Imagen - Captura de dos dashboards comparados. Ambos presentan los mismos datos, pero con distintos valores.

CONCLUSIONES, LIMITACIONES Y MEJORAS

Este informe estudió las particularidades de evaluar la calidad de los datos en un contexto de Big Data, y presentó un sistema que permite analizar dicha calidad a través de las transmisiones de Twitter en tiempo real, continuando el legado del Ingeniero Arolfo en su entrega "*Data quality in a big data context: about Twitter's data quality*".

Los experimentos realizados en diferentes búsquedas mostraron cómo los conceptos presentados y las herramientas desarrolladas podrían aplicarse en un escenario de Big Data del mundo real. Al realizar una búsqueda mediante una consulta política, como "Trump", los servicios de Twitter proporcionan buena alimentación en términos de las dimensiones de legibilidad (*readability*) y calidad de utilidad (*usefulness)*. Pero en nivel general, ocurren problemas en la dimensión calidad de integridad en las coordenadas de los campos y lugar. Los campos de coordenadas geográficas no están presentes en todos los tweets ya que para visualizarlos, el usuario tiene que prestar conformidad de compartir su ubicación. La cantidad de tweets que contienen dicho campo no representa ni el 5% sobre el total.

Además, los re-tweets tienden a tener una mejor calidad que los feeds no retuiteados, y estos cubren más del 70% de la muestra total en una búsqueda política. Esto puede explicar por la razón de un buen valor de calidad de la muestra total. Ésto nos da la pauta que los usuarios tienden a compartir contenido con un buen valor de calidad.

En cuanto a las mejoras implementadas en ésta nueva versión, la interfaz y la experiencia al usuario ha mejorado notablemente. En particular la usabilidad ya que se diseñó una nueva interfaz teniendo en cuenta que el principal uso sería la de mostrar un dashboard integral que contemple todas las visualizaciones. En contraparte con la anterior versión, en este nuevo entregable se evita el conflicto de que el navegador se cuelgue cuando se supera el umbral de los 10K tweets. Parte de esa mejora se debe a que ahora la calidad de los datos se persisten en una base de datos orientada a grandes volúmenes de datos como ElasticSearch y no se envíen mediante un socket TCP.

Cada componente de la UI rediseñada hace un request GET a la API Rest (también desarrollada para ésta nueva versión) y la misma redirecciona la petición mediante una query a ElasticSearch con sintaxis Lucene. De esta manera, toda la lógica de cómputo pasa del lado del servidor y ganando los siguientes beneficios:

1. Se pueden hacer búsquedas y cerrar el navegador

2. Las búsquedas quedan persistidas en un historial para luego hacer comparaciones con otras cronológicamente
3. Consultar por búsquedas previamente realizadas
4. No saturar al navegador del cliente

Otra de las ventajas en ésta nueva versión es que el usuario puede registrarse e ingresar al sistema mediante clave y contraseña, revisar su historial de búsqueda y parametrizar el dashboard con los campos de porcentajes de la calidad de datos.

Aún así, hay mucho espacio para más trabajo. Una línea de investigación podría orientarse a definir más dimensiones y métricas de DQ para esta u otras configuraciones, dado que, como se explicó, esta es una DQ típica dependiente del contexto. Además, se pueden aplicar herramientas aún más sofisticadas de visualización para ampliar y mejorar el marco implementado.

En cuanto a arquitectura, se podrían adaptar todos los microservicios a Docker e instanciarse en un ambiente clusterizado, tolerante a fallas como Kubernetes. Kubernetes ofrece un entorno de administración centrado en contenedores y orquesta la infraestructura de cómputo, redes y almacenamiento para que las cargas de trabajo de los usuarios no tengan que hacerlo. Esto ofrece la simplicidad de las Plataformas como Servicio (PaaS) con la flexibilidad de la Infraestructura como Servicio (IaaS) y permite la portabilidad entre proveedores de infraestructura.

LIMITACIONES

Como todo sistema, ha sido diseñado con la idea de que sea tolerante a fallos. Sin embargo, existen otras cuestiones externas que hacen que el sistema actual presente algunas limitaciones.

En primer lugar, utilizar una red Social como Twitter presenta sus desventajas a la hora de términos y condiciones, ya que dicha red social podría implementar nuevas políticas para que no se pueda hacer streaming en tiempo real de los datos, que no se pueda consultar por las coordenadas geográficas o mismo también que cambien la parametrización de búsqueda.

En segundo lugar, siempre contamos con la desventaja de utilizar muchos recursos de hardware (ya sea memoria o CPU) ya que éste sistema requiere procesamiento de una gran cantidad de datos utilizando varios microservicios sincronizados y en tiempo real.

Muchos de estos problemas hoy se solucionan escalando a una infraestructura como Kubernetes o Apache Mesos, que saben administrar los recursos de la infraestructura inteligentemente y escalando a un ambiente más fácil de administrar como una nube pública, ya sea AWS, Azure o GCP. Dichos problemas no se han observado en varias pruebas realizadas en el nodo 4 del cluster, aunque sí se ha visto que algunas consultas que traen mucha cantidad de tweets por segundo (a razón de 30 tweets/s) el nivel del procesamiento del cluster llegó a un pico máximo.

BIBLIOGRAFÍA

[1] Arolfo F., Vaisman A. (2018). Data quality in a big data context: about Twitter's data quality ADBIS 2018 Budapest, Hungría, 2018.
DOI: https://ri.itba.edu.ar/handle/123456789/1172

[2] Diane M. Strong, Yang W. Lee, Richard Y. Wang. (2002). Data Quality in Context. Communications of the ACM. 40. 10.1145/253769.253804.
DOI: https://www.researchgate.net/publication/2527556_Data_Quality_in_Context

[3] Batini, Carlo & Rula, Anisa & Scannapieco, Monica & Viscusi, Gianluigi. (2015). From Data Quality to Big Data Quality. Journal of Database Management. 26. 60-82. 10.4018/JDM.2015010103. DOI:
https://www.researchgate.net/publication/283681085_From_Data_Quality_to_Big_Data_Quality

[4] Ciaccia P., Torlone R. (2011) Modeling the Propagation of User Preferences. In: Jeusfeld M., Delcambre L., Ling TW. (eds) Conceptual Modeling – ER 2011. ER 2011. Lecture Notes in Computer Science, vol 6998. Springer, Berlin, Heidelberg. DOI:
https://doi.org/10.1007/978-3-642-24606-7_23

[5] Miele, Antonio & Quintarelli, Elisa & Rabosio, Emanuele & Tanca, Letizia. (2013). A data-mining approach to preference-based data ranking founded on contextual information. Information Systems. 38. 524–544. 10.1016/j.is.2012.12.002.
DOI:
https://www.researchgate.net/publication/257158125_A_data-mining_approach_to_preference-based_data_ranking_founded_on_contextual_information

[6] Ntsikayezwe Yahya Fakude. (2019).The Importance of understanding Context in communication. DOI:
https://medium.com/@ntsikayezwefakude/the-importance-of-understanding-context-in-communication-3f921f1b5b24

[7] Sascha Depold. Sequelize ORM
DOI: https://sequelize.org/v5/

[8] Elasticsearch - Configuration. Elasticsearch B.V.
DOI: https://www.elastic.co/guide/en/elasticsearch/reference/current/settings.html
https://marutitech.com/elasticsearch-big-data-analytics/

[9] ReactJS. Facebook Open Source. Facebook Inc.
DOI: https://es.reactjs.org/

[10] MaterialUI for a ReactJS environment
DOI: https://material-ui.com/

[11] ChartJS
DOI: https://github.com/chartjs/Chart.js

[12] NodeJS & Express. OpenJS Foundation. Joyent, Inc. Node.js Foundation.
DOI: https://nodejs.org/docs/latest-v9.x/api/

https://expressjs.com/es/

[13] Apache Software Foundation. Apache Kafka
DOI: https://kafka.apache.org/

[14] Apache Software Foundation. Apache Zookeeper
DOI: https://zookeeper.apache.org/

[15] Java Spring Boot. Pivotal Software, inc.
DOI: https://spring.io/projects/spring-boot

[16] Java HTTP client for consuming Twitter's realtime Streaming API. Twitter
DOI: https://github.com/twitter/hbc