UNIVERSIDAD DE MÁLAGA

E.T.S. INGENIERÍA INFORMÁTICA
UNIVERSIDAD DE MÁLAGA

GRADO EN INGENIERÍA DEL SOFTWARE

HERRAMIENTA PARA ANÁLISIS DE OPINIONES SOBRE POLÍTICA

A TOOL FOR POLITICAL OPINION ANALYSIS

Realizado por
PEDRO ÁVILA CASTRO

Tutorizado por
EDUARDO GUZMÁN DE LOS RISCOS

Departamento
DEPARTAMENTO DE LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA

MÁLAGA, FEBRERO 2020

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
Grado en Ingeniería del Software

Herramienta para análisis de opiniones sobre política

A tool for political opinion analysis

Realizado por
Pedro Ávila Castro

Tutorizado por
Eduardo Guzmán de los Riscos

Departamento
Departamento de Lenguajes y Ciencias de la Computación

UNIVERSIDAD DE MÁLAGA
MÁLAGA, DICIEMBRE DE 2019

Fecha defensa:
Fdo. El/la Secretario/a del Tribunal

D/Dª.:_____, con DNI _____, estudiante del Grado/Máster en _____ _____, de la Universidad de Málaga.

DECLARO QUE:

El Trabajo Fin de Grado/Máster denominado:

_____

_____

____

es de mi autoría, inédito (no ha sido difundido por ningún medio, incluyendo internet) y original (no es copia ni adaptación de otra), no habiendo sido presentado anteriormente por mí ni por ningún otro autor ni en parte ni en su totalidad. Así mismo, se ha desarrollado respetando los derechos intelectuales de terceros, para lo cual se han indicado las citas necesarias en las páginas donde se usan, y sus fuentes originales se han incorporado en la bibliografía. Igualmente se han respetado los derechos de propiedad industrial o intelectual que pudiesen afectar a cualquier institución o empresa.

Para que así conste, firmo la presente declaración en Málaga, a _____ de _____ de _____.

Fdo.: D/Dª _____

Resumen

Las redes sociales se han constituido como el escenario ideal donde verter opiniones sobre cualquier asunto gracias a su inmediatez y anonimato. Esto, sumado al agitado clima político, ha provocado que redes como Twitter, que aúna medios de comunicación, partidos políticos y ciudadanos de a pie, se establezca como una de las principales herramientas de comunicación política.

Sus usuarios, entre ellos los votantes y los propios partidos políticos, construyen tweet a tweet una base de datos de diferentes opiniones y valoraciones.
Sobre esta base de datos en continuo crecimiento, se pueden utilizar herramientas de análisis de opinión o sentimiento que permiten conocer si un texto expresa una valoración positiva o negativa, el grado de esta valoración e incluso detectar si existe ironía. Realizar estos análisis puede ser de utilidad para, por ejemplo, comprobar si una campaña electoral ha resultado exitosa o no en Twitter, o compararla con la del resto de partidos.

Este trabajo tiene como objetivo el desarrollo de una aplicación web que capture y filtre tweets que se ajusten a ciertos criterios, como puede ser contener una mención a un político o partido. Analizará y almacenará los tweets con su resultado, para permitir la consulta de datos del conjunto de los análisis, estadísticas y muestras de los análisis realizados en una interfaz de usuario.

Palabras clave:
twitter, análisis de opinión, análisis de sentimientos, política.

Abstract

Social networks have settled as the main place where people express opinions about any subject due to their immediacy and anonymity. This fact, along with the stressed political situation, has helped building social networks like Twitter to be the main tool for politics communication, as it gathers both social media, citizens and politics.

Their users feed, tweet by tweet, a constantly growing database with data that can be used for sentiment or opinion analysis. These techniques allow us to identify if a determinate tweet express a positive or negative opinion, how negative or positive and even if it express irony. Analysis like these could be useful to check if a campaign have succeed in Twitter or compare its results with the adversaries campaigns.

This project aims to develop a web application which captures and filters tweets according to specific criterion, like mentioning the name of a politician or a party. Besides it will store tweets along with its analysis result, and proportionate a user interface to display stats, analysis samples and more information to the user.

Keywords:

twitter, sentiment analysis, opinion analysis, politics.

Índice

RESUMEN 2

ABSTRACT 3

ÍNDICE 4

ÍNDICE DE FIGURAS 5

ÍNDICE DE TABLAS 6

INTRODUCCIÓN 7

1.1. ESTRUCTURA DE LA MEMORIA 7
1.2. METODOLOGÍA UTILIZADA 8
1.3. HERRAMIENTAS PRINCIPALES 8

2. EXTRACCIÓN DE TWEETS 15

2.1. API DE TWITTER 15
2.2. TWITTER4J 16

3. ANÁLISIS DE OPINIÓN. 21

3.1. HERRAMIENTAS Y SERVICIOS DISPONIBLES 21
3.2. SELECCIÓN DE HERRAMIENTAS 25

4. DESARROLLO DEL BACKEND 27

4.1. ESTRUCTURA DEL BACKEND 28
4.2. ANÁLISIS 29
4.3. MODELO 31
4.4. REPOSITORIOS 35
4.5. CONTROLADORES 36
4.6. MOTOR DE PLANTILLAS THYMELEAF 39
4.7. ARRANQUE DE LA APLICACIÓN 40

5. DESARROLLO DEL FRONTEND 43

5.1. VISTA PRINCIPAL 44
5.2. VISTA DE CONFIGURACIÓN. 46
5.3. VISTA DE TWEET Y ANÁLISIS. 46

6. CONCLUSIONES 49

6.1. LIMITACIONES Y PROBLEMAS ENCONTRADOS 49
6.2. DISCUSIÓN Y POSIBLES MEJORAS 50
6.3. CONCLUSIONES 50

BIBLIOGRAFÍA 53
ANEXO I. INSTALACIÓN Y ARRANQUE 55
ANEXO II. DATOS ELECCIONES ABRIL 2019 57

Índice de figuras

Figura 1: Logo De Spring Framework 9
Figura 2: Logo De Mongo Db 10
Figura 3: Logo De Twitter4j 10
Figura 4: Logo De Apache Maven 11
Figura 5: Logo De Bootstrap 11
Figura 6: Elementos De AdminLte 12
Figura 7: Logo De Chart.Js 13
Figura 8: Funcionamiento De La Api Streaming De Twitter 16
Figura 9: Logo De Microsoft Azure 21
Figura 10: Diagrama Google Cloud Natural Language Api 22
Figura 11: Logo De Monkeylear 23
Figura 12: Logo De Meaning Cloud 23
Figura 13: Logo De Stanford Core Nlp 24
Figura 14: Diagrama De Los Controladores De Spring Framweork 37
Figura 15: Logo De Thymeleaf 39
Figura 16: Vista Principal De La Aplicación 44
Figura 17: Vista De Configuración De La Aplicación. 46
Figura 18: Vista De Tweet Y Análisis De La Aplicación. 47
Figura 19: Ejemplo Análisis De Meaning Cloud 47
Figura 20: Aplicación Corriendo En Localhost:8080 55
Figura 21: Comandos Para Importar Tweets.csv 557

Índice de tablas

Tabla 1: Dependencia Twitter4j En Pom.Xml 17
Tabla 2: Twitter4j.Properties 17
Tabla 3: Obtención De Twittersteam 17
Tabla 4: Descripción Método Onstatus 18
Tabla 5: Creación Filtros Para Twitterstream 18
Tabla 6: Aplicación De Filtros A Twitterstream 18
Tabla 7: Dependencias Core Y Stream De Twitter4j 19
Tabla 8: Dependencias Stanford Core Nlp 26
Tabla 9: Dependencias Spring Framework 27
Tabla 10: Estructura Del Proyecto 28
Tabla 11: Stanfordcorenlp.Properties 29
Tabla 12: Sentimentanalyzer.Java 30
Tabla 13: Atributos De Status 32
Tabla 14: Atributos De Tag 33
Tabla 15: Atributos Info 34
Tabla 16: Repositorio Para Statusdocument 35
Tabla 17: Controlador Principal 38
Tabla 18: Dependencia Para Thymelaf 39
Tabla 19: Ejemplo Sintaxis De Thymeleaf 40
Tabla 20: Clase Twitteranalysistoolapplication 40
Tabla 21: Método Run() 41
Tabla 22: Método Restart() 41
Tabla 23: Método Startlistening() 42
Tabla 24: Script Para Gráficos Con Chart.Js 45
Tabla 25: Script Petición A Meaningcloud 48
Tabla 26: Comando Para Instalar La Aplicación 55
Tabla 27: Comando Para Ejecutar El Proyecto 55

Introducción

Este proyecto tiene como objetivo desarrollar una aplicación web que extraiga comentarios sobre política publicados en la red social Twitter y la aplicación sobre estos comentarios de herramientas de análisis de opinión.

La aplicación permitirá al usuario configurar la captura y filtrado de tweets a almacenar para su procesado, así como consultar el resultado de los análisis y las estadísticas obtenidas, presentando estos datos de una forma intuitiva e informativa.
Para muestra del funcionamiento, se usará la aplicación para obtener y analizar tweets relacionados con las elecciones españolas de abril de 2019

1.1. Estructura de la memoria

Para facilitar su consulta, la memoria se presenta en capítulos, siguiendo la siguiente estructura:

Capítulo 1: *Extracción de tweets.*

Se describe el proceso de captura de tweets para analizar en el proyecto, el acceso y consumo de la API de Twitter junto con sus características y limitaciones.

Capítulo 2: *Análisis de opinión.*

Estudio de las distintas herramientas y servicios para análisis de opinión, sus prestaciones, limitaciones y por último cuáles se utilizan y cómo se añaden al proyecto.

Capítulo 3: *Desarrollo del backend.*

Especificación de la estructura del backend, funcionamiento de cada uno de sus principales componentes y las diferentes herramientas que se han utilizado en ellos.

Capítulo 4: *Desarrollo del frontend.*

Descripción de las herramientas utilizadas para el frontend del proyecto, detalle de las distintas vistas y su funcionamiento.

Capítulo 5: *Conclusiones, observaciones y problemas.*

Se discuten los resultados finales del proyecto, posibles mejoras, errores que se han

cometido y cómo se han corregido.

1.2. Metodología utilizada

Previamente a la fase de desarrollo se plantea un periodo de investigación sobre tecnologías disponibles para el proyecto, entre ellas podemos diferenciar aquellas para extracción y almacenamiento de Tweets y tecnologías para análisis de opinión basada en texto.
Para el desarrollo se plantea una metodología que consta de tres fases, dentro de las que existirán diferentes iteraciones para añadir las diversas funcionalidades que se requieran en su fase. Las distintas iteraciones permitirán comprobar que se cumplen las expectativas y revisar el proyecto buscando mejoras en sus distintas funcionalidades antes de pasar a la siguiente fase.

La primera base vendrá marcada por la necesidad de obtener una base del backend del proyecto lo antes posible. Esta base debe permitir la búsqueda y filtrado de tweets y será puesta en funcionamiento durante el periodo de campaña de las elecciones españolas de abril de 2019.

Una vez la primera fase genere ese proyecto base, la segunda fase añadirá en el backend de la aplicación funcionalidades de análisis y almacenamiento de sus resultados.

En la última fase se desarrolla el frontend de la aplicación, una interfaz web que permitirá al usuario interactuar con la aplicación para iniciar o pausar la captura y análisis, modificar parámetros de la captura de tweets y consultar resultados y estadísticas.

1.3. Herramientas principales

En este proyecto se han usado diferentes herramientas, entre ellas, frameworks y librerías que nos facilitan el desarrollo de la aplicación.

El backend de la aplicación se ha desarrollado con Java EE, una tecnología que hemos usado en varias asignaturas de nuestro grado y que potenciaremos utilizando Spring Framework, un framework de código abierto, que supone una alternativa al modelo EJB.

Figura 1: Logo de Spring Framework

Spring Framework nos ayuda con la generación de código automático (endpoints, acceso a base de datos, etc.) y la configuración del proyecto. Es especialmente útil gracias a sus sistemas de anotaciones, que resuelve la inicialización de las instancias de clases y las dependencias entre ellas.
Spring es además modulable, pudiendo añadir distintas dependencias según nuestras necesidades. En la sección sobre desarrollo del backend se podrá encontrar más información acerca de los módulos de Spring Framework que hemos utilizado.

La aplicación desarrollada con Spring Framework se desplegará sobre un servidor Apache Tomcat.

El almacenamiento de datos es una de las partes fundamentales de nuestra aplicación, es la fuente de tweets que podremos analizar y que el usuario podrá consultar para observar análisis y estadísticas.

Dadas las características de nuestro proyecto tan solo necesitamos almacenar grandes cantidades de tweets y no tenemos interés en relacionarlos entre ellos, pues tan solo

requerimos del texto que contienen. Es por esto que se opta por una base de datos no relacional como MongoDB.

Figura 17: Logo de Mongo DB

MongoDB es un sistema de bases de datos NoSQL orientado a documentos. Se caracteriza por almacenar los documentos como estructuras de datos BSON, similar a JSON, lo que más adelante veremos, encaja a la perfección con nuestro proyecto.

Para alimentar nuestra base de datos el backend debe ejecutar la captura y filtrado de tweets, una funcionalidad esencial para nuestra aplicación. Más adelante se explicarán las utilidades y limitaciones que Twitter proporciona en sus APIs, pero ahora nos centraremos en qué herramienta vamos a utilizar para consumirlas.

Figura 33: Logo de Twitter4J

Twitter4J es una librería no oficial, desarrollada en java y de código abierto, que se integra fácilmente en cualquier proyecto Java con versión 5 o superior. Nos proporciona diferentes utilidades, entre ella el acceso a la API Streaming de Twitter y la configuración para capturar tweets cuyo texto atienda a determinados parámetros.

Se encarga además de la creación y gestión de hebras cuando se está consumiendo un stream de tweets.

Figura 49: Logo de Apache Maven

Para configurar y mantener las distintas dependencias que se utilizan en el backend de la aplicación utilizaremos Apache Maven, con el que fácilmente añadimos las dependencias y las versiones que necesitamos. Se pude encontrar información sobre el proyecto Apache Maven en https://maven.apache.org/guides/index.html

Para al frontend de la aplicación se han utilizado distintas herramientas típicas del desarrollo web como HTML, CSS y Javascript junto con Bootstrap y AdminLTE y Chart.js.

Figura 65: Logo de Bootstrap

Bootstrap es un framework CSS que proporciona todos los elementos necesarios para una interfaz de usuario tradicional: layouts, menús y barras de navegación, varios tipos de fuentes, botones, formularios.

Además podemos editar estos elementos y personalizarlos para que se adecuen mejor a

nuestro diseño.

Bootstrap fue inicialmente desarrollado por Twitter, lo que resulta en que ciertos elementos de nuestra interfaz sean similares a los de la red social, ayudando a dar una mejor sensación de integración a la aplicación.

En nuestro caso usaremos la versión 3 de Bootstrap debido a que la conocemos mejor y que usaremos una versión de AdminLTE que tan solo es compatible con Bootstrap 3.

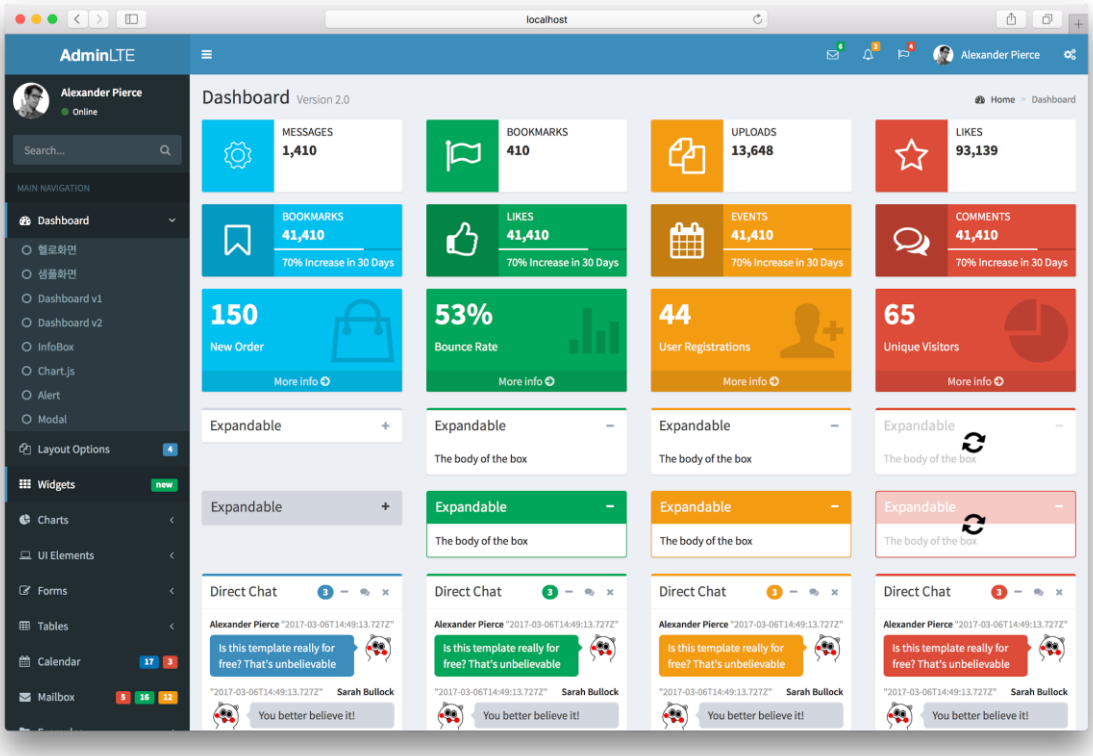

Figura 81: Elementos de AdminLTE

AdminLTE es una librería basada en Bootstrap para la construcción de los conocidos como dashboards de usuario, un tipo de interfaz que se adecua perfectamente al propósito de nuestro frontend.

Esta librería proporciona distintos widgets y elementos preconfigurados que podemos utilizar y personalizar.

Figura 97: Logo de Chart.js

Junto a los elementos gráficos que nos aportan AdminLTE y Bootstrap usaremos Chart.js, una librería en Javascript para la generación de gráficos HTML5.

Por último, el desarrollo del proyecto se ha realizado usando el sistema operativo Linux Mint 19.1 y el IDE Spring Tools Suite 3

2. Extracción de tweets

2.1. API de Twitter

La fuente principal de datos para nuestra aplicación es la red social Twitter, donde constantemente son publicados tweets, los cuales queremos capturar y almacenar atendiendo a distintos criterios.

Twitter dispone de APIs con variedad de puntos de conexión que permiten acceder a cuentas y usuarios, tweets y respuestas, mensajes directos, anuncios y distintas herramientas para desarrolladores.

En este proyecto necesitamos acceso a la parte de tweets y respuestas en tiempo real, a la que podemos acceder con la conocida como API Streaming de Twitter. Con ella podremos establecer unos filtros con palabras clave (*keywords*), localización o usuarios. Para poder usarla deberemos registrarnos como desarrollador y enviar una petición explicando el uso que vamos a hacer de su API, una vez aceptados nos concederán unas credenciales con la que podremos autenticarnos para el uso de la API. Como suele ocurrir tiene ciertas limitaciones, podemos establecer hasta:

- 400 keywords
- 5.000 Ids de usuario
- 25 localizaciones

No obstante, para nuestro proyecto no vamos a superar en ningún momento esas cifras. Filtrarlos por *keywords* como nombres de partidos político y nombres de presidentes de partidos y como localización configuraremos España.

El uso de la API *Streaming* requiere una conexión persistente y permanente entre nuestro backend y Twitter. Tan pronto como los tweets son publicados y se ajustan a nuestros filtros,

Twitter notifica a nuestro backend en tiempo real, lo que permite almacenarlos en nuestra base de datos sin el retraso respecto a una consulta de la API REST de Twitter.

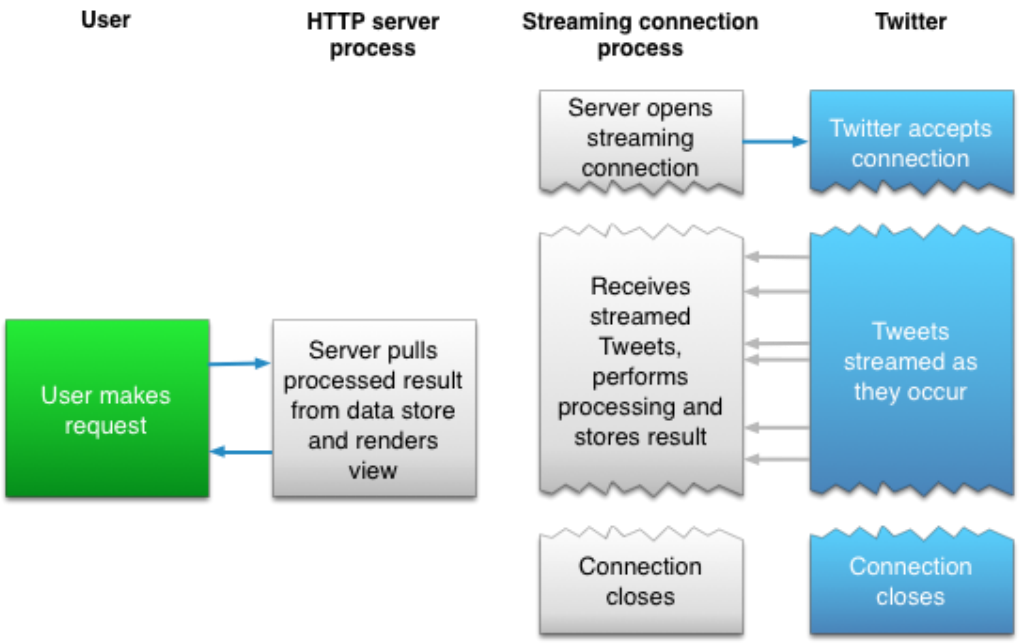

Figura 113: Funcionamiento de la API Streaming de Twitter

2.2. Twitter4J

Para manejar esto podríamos desarrollar nuestra propia solución, sin embargo, resulta más productivo usar alguna librería que ya implemente toda esta funcionalidad. Existen variedad de librerías para la API Streaming de Twitter para diversos lenguajes. Para Java la más conocida y usada es Twitter4J, desarrollada por Yusuke Yamamoto.

Podemos añadir Twitter4J fácilmente a nuestro proyecto incluyéndolo en el fichero pom.xml que Maven toma para manejar nuestras dependencias.

```
<dependencies>
     <dependency>
          <groupId>org.twitter4j</groupId>
          <artifactId>twitter4j-core</artifactId>
          <version>[4.0,)</version>
      </dependency>
      ...
  </dependencies>
```

Tabla 1: Dependencia Twitter4J en pom.xml

Como hemos visto, para consumir la API necesitamos configurar unas credenciales, para ello generamos un archivo *twitter4J.properties* que Twitter4J tomará para su configuración.

```
oauth.consumerKey = Vt8sr9I5EDBy1kWBW2mh6ct30
oauth.consumerSecret =  dBiItYAmXZcw0YEgsktPLNvjXLPknswunH6AVU2JnjK8zo4wQd
oauth.accessToken = 2273290248-k9Xy9rSq9Z7j8aKUsxu69J5sTtcCkRRHYtKT4um
oauth.accessTokenSecret = Ruxkd7oopSBg0DjMMiyRnHTtIcmwxe7b914thaiPrJ7QE
```

Tabla 2: Twitter4j.properties

Con esto ya podemos comenzar a integrar las funcionalidades de Twitter4J en nuestro backend. Más adelante se verá cómo están implementadas en nuestro backend pero ahora explicaremos el uso más básico. En primer lugar debemos obtener un stream de tweets (TwitterStream), esta tarea es simplificada por Twitter4J.

```
TwitterStream twitterStream = new TwitterStreamFactory().getInstance();
```

Tabla 3: Obtención de TwitterSteam

A este *twitterStream*, se le ha de configurar un *listener,* que ejecutará determinadas acciones cada vez que se capture un tweet, aunque también se puede configurar para otro tipo de eventos.

Twitter4J proporciona la interfaz StatusListener (Status es la clase que representa un tweet en este entorno), nosotros crearemos nuestra clase personalizada que implemente dicha interfaz. Para nuestras necesidades basta con implementar el método *onStatus.*

```
@Override
    public void onStatus(Status status) {
       // Extracción de información necesaria del objeto status
       // Análisis de opinión sobre el texto del tweet
     // Guardado del tweet y su análisis en base de datos
    }
```

Tabla 4: Descripción método onStatus

Por último, se configura el filtro a aplicar al stream de tweets mediante la clase FilterQuery de Twitter4J.

```
FilterQuery tweetFilterQuery = new FilterQuery();

// Añadir la lista de keywords para filtrar
tweetFilterQuery.track(keywords);

// Configurar el idioma
tweetFilterQuery.language(new String[] {"es"});
```

Tabla 5: Creación filtros para TwitterStream

Una vez configurado todo esto podemos comenzar la captura de tweets desde nuestro stream.

```
twitterStream.filter(tweetFilterQuery);
```

Tabla 6: Aplicación de filtros a TwitterStream

Con esto nuestro proyecto ya contaría con funcionalidad para capturar tweets ajustándose a las palabras clave que configuremos. Quedan aún pendientes su análisis, almacenamiento, control sobre esta funcionalidad, cómo pausar o activar la captura, configurar las palabras clave, y otras funcionalidades que se explican en las siguientes secciones.

Para añadir Twitter4J a nuestro proyecto añadimos a nuestro fichero pom.xml las siguientes dependencias:

```
<dependency>
    <groupId>org.twitter4j</groupId>
    <artifactId>twitter4j-core</artifactId>
    <version>[4.0,)</version>
</dependency>

<dependency>
    <groupId>org.twitter4j</groupId>
    <artifactId>twitter4j-stream</artifactId>
    <version>4.0.7</version>
</dependency>
```

Tabla 7: Dependencias core y stream de Twitter4J

3. Análisis de opinión.

Este proyecto pretende usar herramientas de análisis de opinión o sentimiento para sus objetivos. Desarrollar una herramienta propia podría ser otro proyecto adicional, sin embargo, al ser un conjunto de técnicas en auge encontramos una gran variedad de servicios y librerías que nos lo facilitan.
Nuestro proyecto requiere poder procesar gran cantidad de textos, de forma rápida y continua, que soporte lenguaje español y a ser posible, que el servicio tenga un coste gratuito.

A continuación analizaremos las distintas herramientas y servicios disponibles, centrándonos en sus limitaciones para encontrar las más adecuadas.

3.1. Herramientas y servicios disponibles

Figura 129: Logo de Microsoft Azure

Microsoft Text Analytics es un servicio en la nube que proporciona procesamiento de texto en lenguaje natural y funciones de análisis de sentimiento, extracción de frases clave, detección de lenguaje y reconocimiento de entidades.

Es parte de *Azure Cognitive Services*, una colección de algoritmos de aprendizaje automático y de inteligencia artificial.

Nos permite enviar texto plano y recibir una valoración positiva o negativa, entre 0 y 1, siendo 1 la más positiva.

- **Limitaciones** de la versión gratuita: límite de 100 peticiones por minuto, máximo de 5.000 peticiones por mes.

Google Cloud Natural Language API

Figura 145: Diagrama Google Cloud Natural Language API

Google Cloud dispone de diversidad de servicios en la nube dedicada al aprendizaje automático y el análisis de texto.

Entre estos servicios se encuentra *Natural Language*, que ofrece funciones de comprensión del lenguaje natural, análisis de opinión, análisis de entidades, análisis de opinión de entidades, clasificación de contenido y el análisis sintáctico.

El servicio de análisis de opinión es similar a los que encontramos en estas herramientas, permitiendo enviar texto y obtener una determinada valoración indicando si es positiva o negativa y una magnitud de ésta.

- **Limitaciones** de la versión gratuita: de nuevo encontramos un límite de 5.000 peticiones mensuales.

Figura 161: Logo de MonkeyLear

MonkeyLearn es una compañía especializada en el análisis de texto que ofrece una API y SDK para análisis de sentimiento destinada desarrolladores en una gran variedad de lenguajes.

- **Limitaciones** de la versión gratuita: las peticiones por mes son restringidas a 300 y la velocidad de respuesta es la menor disponible.

Figura 177: Logo de Meaning Cloud

Meaning Cloud proporciona entre sus diferentes APIs una dedicada al análisis de sentimiento para varios lenguajes, entre ellos el español, que permite añadir al usuario sus propios diccionarios y modelos.

Además de proporcionar la polaridad y su magnitud en los resultados cuenta también con un detector de ironía.

- **Limitaciones de la versión gratuita:** tan solo 2 peticiones por segundo, con un límite de 20.000 peticiones por mes.

Como se aprecia los servicios de grandes compañías especializadas están destinados a otras compañías dispuestas a pagar por estos, dejando su modelo gratuito destinado a pruebas o una muestra de su servicio.
Analicemos otras soluciones de código libre o con mejor servicio gratuito.

Text-processing.com

Esta web publica una sencilla API para procesamiento de lenguaje natural, no necesita de autenticación ni registro. Cuando se envía una petición con determinado texto nos responde con un parámetro indicando la valoración final del texto (pos: positiva, neg: negativa o neutral) y los valores numéricos indicando positividad, negatividad y neutralidad de la opinión del texto.

- **Limitaciones:** no soporta lenguaje español y su límite de peticiones es de 1.000 por día.

Figura 193: Logo de Stanford Core NLP

Stanford Core NLP es una librería en Java de herramientas para procesamiento del lenguaje natural desarrollada por la Universidad de Stanford. Dispone de un gran número de

funcionalidades como la detección de entidades, normalización de fechas, cantidades numéricas, extracción de estructuras, …

La herramienta para análisis de sentimiento funciona mediante lo que ellos denominan anotaciones, y es un módulo dentro de las diferentes herramientas de la librería.

Su desarrollo comenzó dando soporte al inglés, pero se han ido añadiendo capas para soporte de idiomas como el árabe, chino, francés, alemán y español.

Al tratarse de una librería que podemos añadir fácilmente a nuestro proyecto no nos supone ninguna limitación, salvo el conocimiento de su funcionamiento e integración en el proyecto.

3.2. Selección de herramientas

Analizadas las principales alternativas para nuestro proyecto escogemos dos de ellas, intentando cubrir las necesidades del proyecto pero también aportar cierta distinción.

Debemos de tener en cuenta que la cantidad de tweets que capturamos en tiempo real varía según las *keywords* que configuremos, la hora, si existe algún evento especial, etc. Es por ello por lo que necesitamos una herramienta que admita un gran número de peticiones continuas sin limitación.

La mejor alternativa es usar la librería Stanford Core NLP integrada en nuestro proyecto, evitando así las limitaciones de los otros servicios y además los tiempos de envío y respuesta en una API REST. Esta librería nos proporciona la polaridad de la opinión de un texto, mediante un valor numérico comprendido entre 0 y 4, siendo 0 la valoración más negativa y 4 la más positiva.

Stanford Core NLP será, por tanto, nuestra librería principal para el análisis de opinión pero la complementaremos con la API para análisis de sentimiento de Meaning Cloud, que tiene los límites de peticiones más razonables haciendo posible su uso en nuestro proyecto. Además de analizar la polaridad, este servicio detecta también si existe ironía en el texto analizado, una funcionalidad interesante y diferenciadora, que puede ser de utilidad en textos de ámbito político.

El sistema para análisis de opinión en nuestro proyecto quedaría de la siguiente forma:

- **Stanford Core NLP:** integrado en el proyecto, se utiliza para obtener la polaridad del texto de un tweet en cuanto este es capturado, almacenándose el tweet y el resultado del análisis obtenido por esta librería.
- **Meaning Cloud:** se utilizará tan sólo cuando el usuario consulte un tweet específico, evitando así superar el límite de peticiones. Junto al resultado de polaridad, se mostrará al usuario si se ha detectado ironía.

Para añadir Stanford Core NLP con las dependencias necesarias añadimos a nuestro fichero pom.xml lo siguiente:

```
<dependency>
   <groupId>edu.stanford.nlp</groupId>
   <artifactId>stanford-corenlp</artifactId>
   <version>3.9.2</version>
</dependency>
<dependency>
   <groupId>edu.stanford.nlp</groupId>
   <artifactId>stanford-corenlp</artifactId>
   <version>3.9.2</version>
   <classifier>models</classifier>
</dependency>
<dependency>
   <groupId>edu.stanford.nlp</groupId>
   <artifactId>stanford-corenlp</artifactId>
   <version>3.9.2</version>
   <classifier>models-spanish</classifier>
</dependency>
```

Tabla 8: Dependencias Stanford Core NLP

En la sección de backend veremos cómo utilizamos las herramientas que nos brinda la librería.

Para poder usar el servicio de análisis de sentimiento de Meaning Cloud deberemos registrarnos como desarrollador para obtener nuestras credenciales y poder realizar peticiones a su API. En este caso la petición a su API será manejada en el frontend de nuestra aplicación utilizando jQuery, en la sección de frontend se encuentran los detalles pertinentes.

4. Desarrollo del backend

El desarrollo del backend se realiza en Spring Framework como se ha comentado, utilizando su sistema de clases y anotaciones nos permite evitar las desventajas del sistema EJB de Java EE.

Spring Framework nos proporciona también las herramientas necesarias para conectarnos y operar en nuestra base de datos MongoDB. Las dependencias necesarias para el funcionamiento del backend quedan reflejadas en el fichero *pom.xml.*

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-mongodb</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-mongodb</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

Tabla 9: Dependencias Spring Framework

Estás dependencias de Spring junto con las que hemos ido anotando a lo largo de esta memoria son las necesarias para el correcto funcionamiento de nuestro proyecto.

4.1. Estructura del backend

Una vez contamos con las herramientas necesarias estructuramos el proyecto para facilitar su acceso, uso y mantenibilidad. Podemos diferenciar 4 paquetes en el proyecto:

- **Domain**: en este paquete se definen las clases de nuestro modelo.
- **Repositories:** donde creamos nuestros repositorios, extendiendo los que nos proporciona Spring para MongoDB, con consultas personalizadas a la base de datos.
- **Controllers:** paquete con los controladores de Spring Framework para manejar las peticiones del usuario a nuestro backend.
- **Analysis:** contendrá las clases y métodos relacionados con el análisis de opinión basada en texto.

```
│   ├── analysis/
│   │   ├── SentimentAnalyzer.java
│   ├── controllers/
│   │   ├── ConfigController.java
│   │   ├── MainController.java
│   │   ├──TweetsAndAnalysisController.java
│   ├── domain/
│   │   ├── Info.java
│   │   ├── Status.java
│   │   ├── Tag.java
│   ├── repositories/
│   │   ├── InfoRespository.java
│   │   ├── StatusDocumentRepository.java
│   │   ├── TagRepository.java
```

Tabla 10: Estructura del proyecto

En la raíz del proyecto se encuentran nuestro StatusListener personalizado y las clases para inicialización de la aplicación.

Para explicar el desarrollo y funcionamiento del backend de la aplicación iremos detallando cada una de las partes de la estructura.

4.2. Análisis

En este paquete se encuentra nuestra clase ***SentimentAnalyzer*** la cual se encarga de realizar los análisis de opinión utilizando la herramienta *Stanford Core NLP.*
Para utilizar esta librería se ha de configurar primero ciertos parámetros, la forma más sencilla es crear un fichero *StanfordCoreNLP.properties* como el siguiente:

```
annotators = tokenize, ssplit, parse, sentiment
tokenize.language = es
ssplit.language = es
parse.language = es
sentiment.language = es
```

Tabla 11: StanfordCoreNLP.properties

Con estas propiedades estamos configurando *Stanford Core NLP* para que utilice determinados *annotators* y el idioma para cada uno de ellos. Para más información sobre los *annotators* puede consultarse la documentación de *Stanford Core NLP:* https://stanfordnlp.github.io/CoreNLP/annotators.html

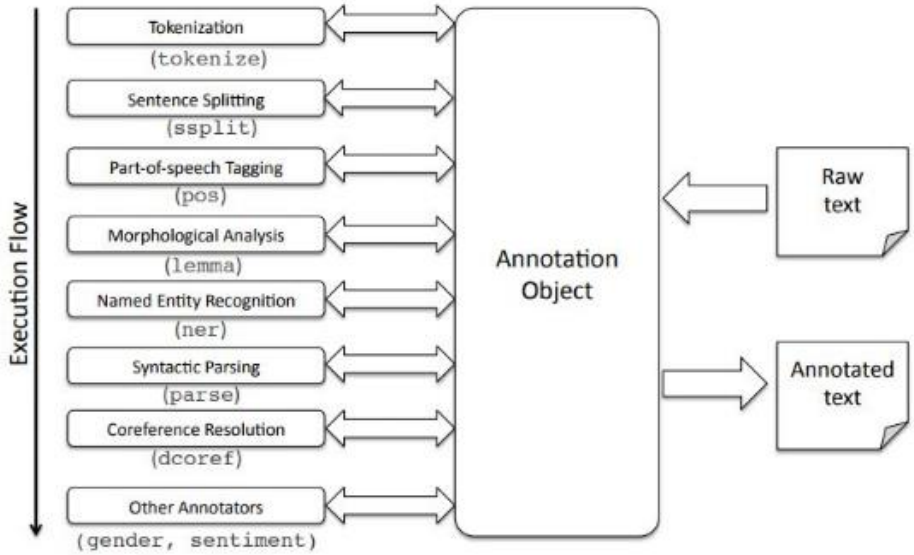

Figura 14. Diagrama anotaciones de Stanford Core NLP

Una vez configurada podemos utilizar las herramientas que esta librería nos aporta.

```
public class SentimentAnalyzer {
    private StanfordCoreNLP  pipeline;

    public SentimentAnalyzer() {
      pipeline = new StanfordCoreNLP();
    }

    public double stanfordScore(String text) {
      Annotation annotation = this.pipeline.process(text);
      // En caso de error o imposibilidad de análisis, por defecto valor
neutro 2
      double sentimentScore = 2;
      int sentenceMaxLength = 0;
      int currentSentenceLength;
      for (CoreMap sentence:
annotation.get(CoreAnnotations.SentencesAnnotation.class)) {
          currentSentenceLength = sentence.toString().length();
          if (currentSentenceLength > sentenceMaxLength) {
          Tree tree =
sentence.get(SentimentCoreAnnotations.SentimentAnnotatedTree.class);
          sentimentScore = RNNCoreAnnotations.getPredictedClass(tree);
          sentenceMaxLength = currentSentenceLength;
          }
      }
      return sentimentScore;
    }
}
```

Tabla 12: SentimentAnalyzer.java

Para construir esta clase nos basamos en la implementaciones de los tutoriales de la documentación: https://stanfordnlp.github.io/CoreNLP/tutorials.html.

El método *stanfordScore,* que tiene como argumento una cadena de caracteres, nos proporcionará como resultado del análisis de opinión de dicha cadena una valoración entre 0 y 4, siendo 0 la polaridad más negativa y 4 la más positiva.

4.3. Modelo

Nuestra aplicación cuenta con un modelo con 3 entidades básicas denominadas *Status*, *Tag* e *Info*. Estas entidades se encuentran son manejadas como POJOs mediante la capa de persistencia que nos aporta Spring Data MongoDB.

Estas entidades corresponden con los documentos almacenados en nuestra base de datos MongoDB. En la base de datos de la aplicación contaremos con una colección para cada tipo de documento, es decir, existirán las colecciones *tags, tweets* e *infos.*
Para indicar que estamos definiendo una identidad correspondiente a un documento en MongoDB basta con anotar las clases de nuestro paquete domain con:

| *@Document(collection = "nombreDeLaColección")*

Status

Es la entidad fundamental, correspondiente a un tweet, se denomina con el nombre Status porque es el utilizado por otras herramientas internas. Los documentos de este tipo son almacenados en la colección *tweets.*

Twitter4J es la herramienta que nos proporciona los datos necesarios para formar un *status*, sin embargo, el objeto que Twitter4J captura como status contiene muchísima información sobre éste. La librería recoge información sobre menciones a otros usuarios, geolocalización, multimedia asociada y una larga lista de atributos.

En un inicio, pensando que esta información podría ser útil para nuestra aplicación, se define nuestro modelo *status* de una forma idéntica al obtenido por Twitter4J. Sin embargo, conforme el número de documentos aumentaba el tamaño de su colección comenzaba a ser excesivo, impidiendo el correcto desarrollo del proyecto. Por ello se optó por redefinir nuestro modelo *status* conteniendo tan solo la información estrictamente necesaria e incluyendo tan sólo algunos atributos que podrían ser de interés en un futuro desarrollo.
Teniendo esto en cuenta nuestros documentos *status* vienen definidos por las siguientes propiedades:

Atributo	Descripción	Tipo
id	Identificador interno en MongoDB para el documento.	ObjectId
tweetId	Identificador del tweet en Twitter.	long
userId	Identificador en Twitter del usuario autor del tweet.	long
userScreenName	Nick del usuario en Twitter (@nick)	string
favoriteCount	Número de "me gustas" otorgados al tweet en el momento de la captura.	int
retweetCount	Número de retweets obtenidos por el tweet en el momento de la captura.	int
text	Texto del tweet sobre el que aplicaremos las herramientas de análisis.	string
stanfordScore	Valoración de la polaridad obtenida haciendo uso de la herramienta Stanford Core NLP	double

Tabla 13: Atributos de Status

Tag

Es la entidad correspondiente a un término de búsqueda. En un inicio se barajó tener varios tags distintos por partido político, sin embargo, es de mayor utilidad que los *tags* tengan a su vez una lista de subtérminos relacionados con su término padre. Por ejemplo, un *tag* podría ser el nombre del partido político y su lista de subtags contener sus siglas, nombre alternativo, el nombre de su presidente, alguna palabra relacionada con su orientación política, etc.

Atributo	Descripción	Tipo
id	Identificador interno en MongoDB para el documento.	ObjectId
term	Término principal o padre.	string
subtags	Lista de términos relacionados con *term.*	array[:string]
color	Código de color para elementos visuales del *tag.* Es un atributo reservado para el frontend.	string
pieColor	Código de color para los gráficos circulares. Es un atributo reservado para el frontend.	string
numTweets	Número de tweets almacenados que contengan en su texto *term* o alguno de los subtérrminos de *subtags.*	long
stanfordScore	Media del valor stanfordScore para los tweets asociados al tag.	double

Tabla 14: Atributos de Tag

Info

Por último, encontramos esta entidad que describe el estado de la aplicación y su configuración. Es imprescindible para mantener un control de la captura de tweets y almacenar cierta información para su rápido acceso.

Atributo	Descripción	Tipo
listening	Ajuste para la captura de tweets. Si se encuentra a *true* la captura está habilitado, en caso contrario, deshabilitada.	boolean
numTweetsTotal	Recuento del número de documentos status almacenados en base de datos.	long
lastUpdated	Fecha en la que la información correspondiente a datos que requieren largos tiempo de consulta fue actualizada.	Date

Tabla 15: Atributos Info

4.4. Repositorios

Spring Data MongoDB nos aporta los conocidos como Mongo Repositories, unas interfaces que permiten las operaciones tipo CRUD y consultas con nuestros documentos almacenados en MongoDB.

Los repositorios de Spring nos aportan ya de base las operaciones CRUD usuales que en la mayoría de casos son suficientes. Sin embargo, en algunas ocasiones esto no será suficiente y la interfaz que define el repositorio necesitará disponer de métodos no estándar, que podremos implementar extendiendo el repositorio mediante el sistema de anotaciones de Spring.

El caso que nos ocupa es el de los documentos Status, que requieren de una serie de consultas personalizadas. Para crear un repositorio correspondiente a un tipo de documento basta con indicar el tipo de entidad del documento al extender la interfaz MongoRepository.

```
public interface StatusDocumentRepository extends MongoRepository<Status,
String> {

    // Cuenta tweets que contengan cierta expresión en su texto
    @Query(value = "{text: {$regex :?0} }", count = true)
    long countByTextLike(String text);

    // Lista tweets que contengan cierta expresión en su texto
    @Query(value = "{text: {$regex :?0} }")
    List<Status> findByTextLike(String text);

    // Cuenta tweets que no sean un retweet
    @Query(value = "{text: {$regex : '^((?!RT).)*$'} }", count = true)
    long countByNoRT();

    // Lista tweets que no sean un retweet, paginando los resultados
    @Query(value = "{text: {$regex : '^((?!RT).)*$'} }")
    Page<Status> findByNoRT(PageRequest pageable);
}
```

Tabla 16: Repositorio para StatusDocument

Mediante la anotación *@Query(query, queryOptions)* configuramos la consulta, podemos añadir equals, not equals, expresiones regex y más opciones a la consulta atendiendo a las normas que podemos encontrar en la documentación de MongoRepository (https://docs.spring.io/spring-data/mongodb/docs/1.3.3.RELEASE/reference/html/mongo.repositories.html)

Con esto nuestro backend ya cuenta con las estructuras y herramientas necesarias para almacenar documentos y realizar consultas sobre ellos.

El uso de MongoRepository facilita y acelera el desarrollo ya que nos aporta mucho código que no necesitamos escribir pero, por otro lado, es un sistema muy rígido y aunque permite consultas personalizadas atendiendo a las normas de su sistema, su flexibilidad es escasa. En nuestro caso ha sido suficiente pero si necesitáramos consultas más detalladas la rigidez de este sistema podría suponer un obstáculo.

4.5. Controladores

En este paquete se encuentran los controladores de la aplicación. Son los encargados de recibir las peticiones HTTP que envíe el usuario a través del frontend en su navegador, procesarlas y devolver una respuesta adecuada.

Una vez más Spring nos proporciona mediante su sistema de anotaciones un sistema de creación de controladores muy sencillo y que nos ahorra gran cantidad de código frente a otros sistemas. Para crear un controlador basta con anotar una clase con *@Controller,* esto hará que a la clase se le aplique el estereotipo de bean controlador en Spring.

Al iniciarse la aplicación Spring escanea el proyecto en búsqueda de clases anotadas con *@Controller* y dentro de ellas detecta los métodos anotados con *@RequestMapping.*

Estos métodos mapean las distintas rutas para las peticiones que reciba la aplicación.

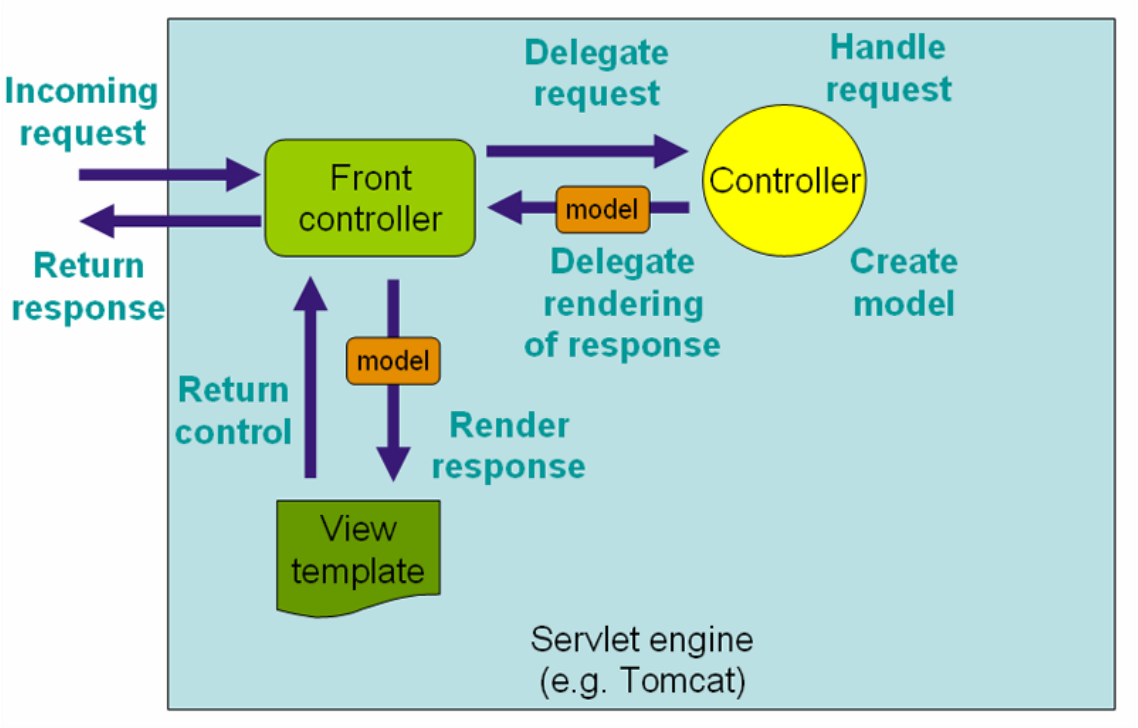

Figura 209: Diagrama de los controladores de Spring Framweork

Los métodos anotados con *@RequestMapping* cuentan con propiedades en su anotación para configurarlos:

- **Value**: indica la ruta que mapea el método, partiendo de la url base de la aplicación.
- **Method:** indica el tipo de petición HTTP a la que responde el método (GET, POST, PUT, DELETE…).

Existen más anotaciones para estos métodos, como *@RequestParam*, para indicar que se espera cierto parámetro en la petición.

Una vez se recibe la petición HTTP, el método configurado la procesará adecuadamente y enviará una respuesta. Para ello realizará las operaciones pertinentes asociadas a la petición, como extraer información de base de datos o modificar algún documento almacenado, y a continuación, configurar los datos oportunos en un ámbito al que se pueda acceder desde la vista que devolverá en la respuesta.

Los controladores están necesariamente ligados al funcionamiento y requisitos del frontend de la aplicación. Sus métodos responden a las peticiones que puedan llegar desde el frontend, por lo que existirán tantos métodos como funciones (peticiones) pueda realizar el usuario en el frontend de la aplicación.

Para indicar la vista los métodos devuelven un string en el que se especifica el nombre de la vista a mostrar. Spring hace accesibles los datos a las vistas mediante un modelo, al cual se le pueden añadir diferentes atributos desde nuestro controlador (se sigue el clásico patrón Modelo-Vista-Controlador).

```
@Controller
public class MainController{

    @Autowired InfoRepository infoRepo;
    @Autowired TagRepository tagRepo;

    @RequestMapping(value = "/", method = RequestMethod.GET )
    public String get_index(Model model) {

      Info info = infoRepo.findAll().get(0);
      List<Tag> tags = tagRepo.findAll();

      model.addAttribute("info", info);
      model.addAttribute("tags", tags);
      return "index";
    }
}
```

Tabla 17: Controlador Principal

4.6. Motor de plantillas Thymeleaf

En este punto cabe resaltar que para manejar las vistas se utiliza el motor de plantillas para *Thymeleaf* para Java.

Figura 225: Logo de Thymeleaf

Thymeleaf nos permite manejar plantillas *xhtml* pudiendo crearlas e incluirlas en nuestras vistas. AdminLTE, ya nos aporta diferentes elementos gráficos para el frontend de la aplicación por lo que no necesitamos muchas plantillas que reutilizar e incluir en las vistas.

Sin embargo, Thymeleaf nos proporciona un buen conjunto de herramientas para manejar los datos del modelo en las vistas. Con Thymeleaf podemos acceder sobre elementos del modelo, crear elementos html y modificar atributos de html

Para añadirlo a nuestras dependencias incluimos en nuestro *pom.xml:*

```
<dependency>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

Tabla 18: Dependencia para Thymelaf

Su uso es sencillo y se encuentra bien integrado con Spring, sin embargo, es bastante rígido y poco flexible frente a otros sistemas como EJS o ERB. En su documentación encontramos cómo funciona y su sintaxis (https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html)

```
<b><span class="info-box-text"
th:text="${tag.getTerm().toUpperCase()}"></span></b>
<span class="info-box-number"></span>
<b>Subtags: </b>
<div th:each="subtag : ${tag.getSubtags()}">
     <small th:text="${subtag}"></small>
</div>
```

Tabla 19: Ejemplo sintaxis de Thymeleaf

4.7. Arranque de la aplicación

Por último, observemos cómo los distintos componentes se orquestan al arrancar la aplicación. La clase principal de nuestra aplicación, ***TwitterAnalysisToolApplication*** se encarga de ello.

```
@SpringBootApplication
public class TwitterAnalysisToolApplication implements CommandLineRunner{

    @Autowired StatusDocumentRepository statusRepository;
    @Autowired InfoRepository infoRepository;
    @Autowired TagRepository tagRepository;

    private static ConfigurableApplicationContext context;

    public static void main(String[] args) {
      context = SpringApplication.run(TwitterAnalysisToolApplication.class,
args);
    }
}
```

Tabla 20: Clase TwitterAnalysisToolApplication

Como vemos la clase queda anotada con *@SpringBootApplication* que defina la clase como principal del proyecto y se encarga de la configuración y búsqueda de dependencias.
Spring se encarga de inyectar las dependencias necesarias como, por ejemplo, los repositorios, para eso tan solo tenemos que usar la anotación *@Autowired.*

A continuación redefinimos el método *run* para cargar la configuración de la aplicación, como si la escucha está activa y estadísticas guardadas para mostrar al usuario en la vista principal.

```
@Override
public void run(String... args) throws Exception {
    List<Info> config = infoRepository.findAll();
    boolean listen = config == null ? false :
config.get(0).isListening();

   if(listen) startListening();
}
```

Tabla 21: Método run()

Necesitamos también un método que permita reiniciar la aplicación en caso de que el usuario cambie ciertos parámetros importantes de la configuración.

```
public static void restart() {
    ApplicationArguments args = context.getBean(ApplicationArguments.class);

    Thread thread = new Thread(() -> {
        context.close();
        context = SpringApplication.run(TwitterAnalysisToolApplication.class,
args.getSourceArgs());
    });

    thread.setDaemon(false);
    thread.start();
}
```

Tabla 22: Método restart()

Y el método para iniciar la escucha, que en primer lugar recopila los documentos *Tag* almacenados junto con sus subtags. Una vez se cuenta con la lista de palabras clave para la captura de tweets, se configura nuestro *listener* y se inicia la captura.

```
private void startListening() {
    TwitterStream stream = new TwitterStreamFactory().getInstance();
    StatusListenerWithFilter listener = new StatusListenerWithFilter();

    listener.setRepository();
    stream.addListener(listener);

    FilterQuery tweetFilterQuery = new FilterQuery();


    String[] terms = loadTerms();
    tweetFilterQuery.track(terms);
    tweetFilterQuery.language(new String[] {"es"});

    stream.filter(tweetFilterQuery);
}
```

Tabla 23: Método startLIstening()

El método auxiliar *loadTerms()* simplemente carga los tags y subtgas en un array de strings.

5. Desarrollo del frontend

El frontend de la aplicación tendrá una estructura y estilo tipo dashboard de usuario, donde éste podrá navegar entre diferentes secciones para configurar la captura de tweets, consultar las estadísticas globales y resultados de análisis.

La principal tecnología usadas en el frontend de la aplicación, son la librería AdminLTE, que utiliza *Bootstrap,* junto con *Chart.js* para la generación de gráficos. AdminLTE nos proporciona cantidad de elementos listos para su uso en nuestra interfaz, pero permite también personalizarlos ya que podemos alterar sus ficheros fuentes.

Evidentemente junto a estas tecnologías usaremos el stack típico formado por HTML, teniendo en cuenta las anotaciones de *Thymeleaf,* CSS y Javascript, utilizando en ocasiones jQuery debido a nuestro conocimiento de la librería.

Como se explica en el capítulo 5, el frontend está necesariamente ligado a los controladores de nuestra aplicación, que completan con la información pertinente el modelo de nuestras distintas vistas. Nuestra interfaz contará con tres vistas principales que permiten al usuario el uso de la aplicación:

- **Vista principal.** Se muestra un resumen de las estadísticas total de tweets capturados, porcentaje de tweets por partido político y media de los análisis de Stanford Core NLP. Su controlador asociado es *MainController.*
- **Configuración.** Desde donde el usuario puede parar la captura de tweets y administrar los partidos políticos y los términos a observar. Su controlador asociado es *ConfigController.*
- **Tweet y análisis.** Muestra un tweet aleatorio y los resultados del análisis de *MeaningCloud* y *StanfordScore.* Su controlador asociado es *TweetsAndAnalysisController*.

En las vistas se usan una gran variedad de elementos de AdminLTE personalizados para nuestro frontend, como las *box, info-box, navbars, dropdowns, etc.*

A continuación nos centraremos más en el código Javascript que se ejecute en las vistas.

5.1. Vista principal

En ella el usuario accede al número total de tweets capturados hasta el momento, la lista de partidos con sus términos para la captura y 2 gráficos con estadísticas. Su ruta asociada es la raíz de la aplicación.

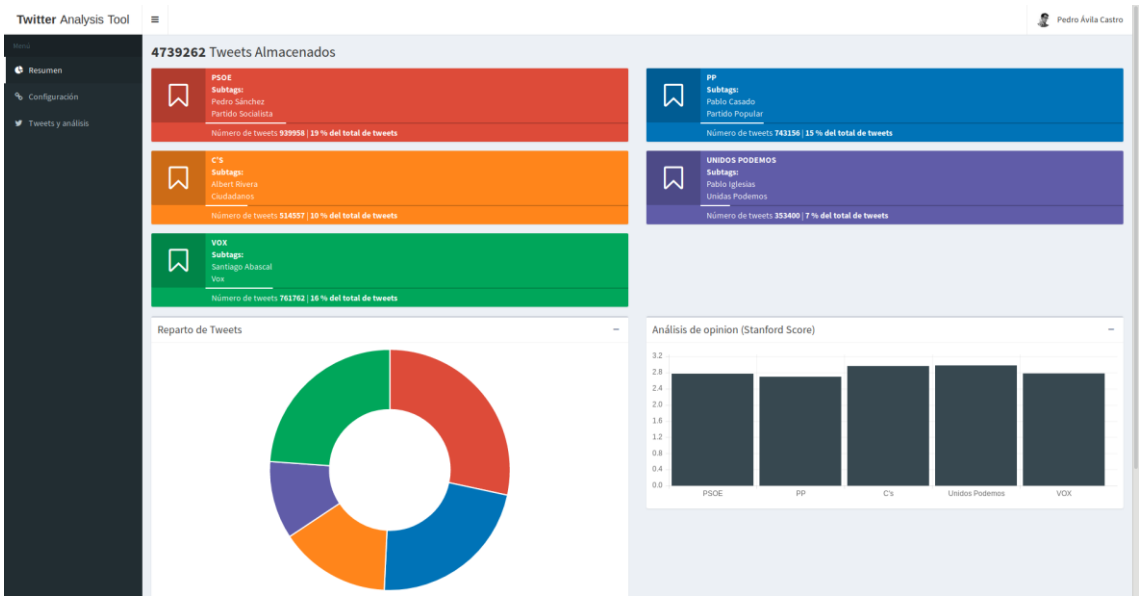

Figura 241: Vista principal de la aplicación

Cada tarjeta de un partido contiene su *tag* principal y sus *subtags,* se muestra el número de tweets capturados que contengan alguna de esas palabras clave y el porcentaje relativo al total de tweets capturados.

En el gráfico circular o "de tarta" se muestra el reparto porcentual de los tweets entre los diferentes partidos. En el gráfico de barras se observa la media de la polaridad de los tweets, entre 0 y 4, de cada uno de los partidos.

En la *sidebar* a la izquierda se encuentran las distintas secciones a las que el usuario puede acceder.

Para visualizar los gráficos cargamos previamente el modelo de la vista con los datos a mostrar en ellos. Con estos datos podemos cargar los gráficos con *Chart.js* de la siguiente forma.

```
var pieChartCanvas = $('#pieChart').get(0).getContext('2d');
var pieChart = new Chart(pieChartCanvas);
var PieData = [];
var dataThymeleaf = document.getElementById('dataThymeleaf');
var dataThymeleaf = dataThymeleaf.getElementsByTagName('small');

var length = dataThymeleaf.length;
for (var i = 0; i <= length - 1; i += 5) {
      var x = {
            value: dataThymeleaf[i].innerHTML,
            color: dataThymeleaf[i + 1].innerHTML,
            highlight: dataThymeleaf[i + 2].innerHTML,
            label: dataThymeleaf[i + 3].innerHTML
      };
      PieData.push(x);
}

var pieOptions = {
      //Boolean - Whether we should show a stroke on each segment
      segmentShowStroke: true,
      //String - The colour of each segment stroke
      segmentStrokeColor: '#fff',
      //Number - The width of each segment stroke
      segmentStrokeWidth: 2,
      //Number - The percentage of the chart that we cut out of the middle
      percentageInnerCutout: 50, // This is 0 for Pie charts
      //Number - Amount of animation steps
      animationSteps: 100,
      //String - Animation easing effect
      animationEasing: 'easeOutBounce',
      //Boolean - Whether we animate the rotation of the Doughnut
      animateRotate: true,
      //Boolean - Whether we animate scaling the Doughnut from the centre
      animateScale: false,
      //Boolean - whether to make the chart responsive to window resizing
      responsive: true,
      // Boolean - whether to maintain the starting aspect ratio or not when responsive, if set to
false, will take up entire container
      maintainAspectRatio: true,
      //String - A legend template
      legendTemplate: '<ul class="<%=name.toLowerCase()%>-legend"><% for (var i=0; i<segments.length;
i++){%><li><span style="background-
color:<%=segments[i].fillColor%>"></span><%if(segments[i].label){%><%=segments[i].label%><%}%></li><%
}%></ul>'
}
pieChart.Doughnut(PieData, pieOptions)
```

Tabla 24: Script para gráficos con Chart.js

Lo que este script está realizando es la recopilación de la información cargada por thymeleaf en unas variables determinadas; configurar el estilo del gráfico, los colores, leyenda, animación y cargando los datos en el gráfico. El código que genera el gráfico de barras es similar al mostrado para el gráfico circular.

5.2. Vista de configuración.

Aquí el usuario puede consultar información sobre la escucha y pausarla o reanudarla. Además puede añadir un *tag* para un nuevo partido político.

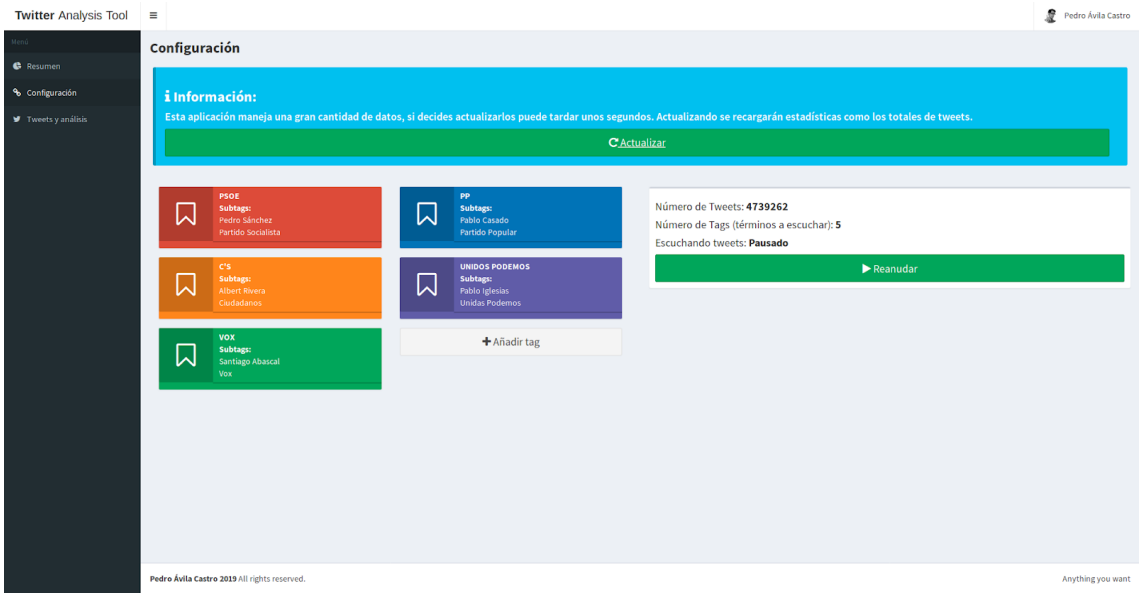

Figura 257: Vista de configuración de la aplicación.

Debido a la cantidad de datos que se manejan actualizar ciertas estadísticas toma tiempo, por ello se informa al usuario antes de actualizar la información de esta página.

5.3. Vista de tweet y análisis.

En esta vista se le muestra al usuario un tweet seleccionado aleatoriamente de entre los almacenados y los resultados de los análisis resultantes de nuestras dos herramientas, *MeaningCloud y Stanford Core NLP.*

Figura 273: Vista de tweet y análisis de la aplicación.

En esta vista debemos resaltar que el análisis de *Meaning Cloud* se realiza mediante una petición a su API desde la propia vista. Como se ha comentado *Meaning Cloud* ofrece un resultado con detalles más interesantes.

Figura 289: Ejemplo análisis de Meaning Cloud

Para realizar la petición al servicio de análisis de sentimiento de *Meaning Cloud* previamente debemos registrarnos y obtener nuestra API KEY; con ella ya podremos enviar una petición POST con el texto a analizar.

El servicio nos devolverá el resultado del análisis en el que podremos encontrar los siguientes parámetros:

- **Irony:** indica si se ha detectado ironía en el texto.
- **Agreement:** indica si existe acuerdo o no con lo que se enuncia en el texto.
- **Confidence:** valor de confianza del análisis entre 0 y 100.
- **ScoreTag:** indica la polaridad (P+ muy positiva, P positiva, NEU neutral, N negativa, N+ muy negativa, NONE no encontrada).

```
<script>
   var settings = {
     “async": true,
     "crossDomain": true,
     “url": "https://api.meaningcloud.com/sentiment-2.1",
     "method": "POST",
     "headers": {
          "content-type": "application/x-www-form-urlencoded"
     },
     "data": {
          "key": "07c8f88cde3f375e2f4623bddf9c5f43",
          "lang": "es",
          "txt": $('#tweet-text').text(),
          "txtf": "plain",
     }
   };

   $.ajax(settings).done(function (response) {
     $("#ironia").text(response.irony);
     $("#agreement").text(response.agreement);
     $("#confidence").text(response.confidence);
     $("#score_tag").text(response.score_tag);
   });
</script>
```

Tabla 25: Script petición a MeaningCloud

6. Conclusiones

Llegados a este punto hemos analizado los componentes del proyecto, que es ya usable para el usuario, permitiendo la captura y análisis de tweets y la consulta de estadísticas generales y resultados de los análisis.

Para llegar hasta aquí se han tomado determinadas decisiones durante el desarrollo que han determinado el funcionamiento y aspecto de la aplicación.

6.1. Limitaciones y Problemas Encontrados

Una de las principales limitaciones a la hora de desarrollar era la capacidad y potencia del equipo que hemos usado. El equipo contaba con apenas 120 GB de almacenamiento y 4 GB de memoria RAM.

Estas características técnicas no permiten el manejo de grandes cantidades de información, es por ello por lo que durante el desarrollo se tomó la decisión de almacenar la mínima información necesaria de un tweet. Así se pasó de una base de datos con unos 5 millones de tweets que ocupaba unos 25 GB a una base de datos de unos 5 GB, manteniendo el mismo número de tweets.

Otro de los principales obstáculos fue la rigidez de algunas de las tecnologías usadas. *Spring Framework* es tremendamente útil y potente pero se necesita tener muy claro que es el framework que se ajusta a tus necesidades y diseñar previamente el sistema con detalle.

En el ámbito laboral he descubierto otros frameworks como *Node.js* o *Ruby on Rails* que permiten un desarrollo más configurable, con una mayor cantidad de librerías y donde los cambios se integran más fácilmente.

Además para las librerías de Node.js o gemas Ruby la documentación es más accesible y existen más recursos para el aprendizaje.

6.2. Discusión y Posibles Mejoras

En cuanto al resultado de los análisis se observa que los proporcionados por *Meaning Cloud* son más correctos y aportan más y mejor información. Quizás para una futura extensión del proyecto se debería plantear el pago de una licencia para el uso de su servicio sin limitaciones.

Podemos destacar también que el uso de AdminLTE aporta un aspecto muy profesional a la aplicación al mismo tiempo que ahorra muchísimo tiempo en el desarrollo del frontend.

En nuestro caso el proyecto contiene ambos backend y frontend, lo que aporta ciertas ventajas, como no manejar dos proyectos con dependencias distintas y coordinar su comunicación. No obstante, considero que sería buena idea separar el frontend y backend de la aplicación en proyectos distintos.

Podría desarrollarse un backend que ofreciera una API tipo REST para la consulta de datos almacenados, estadísticas, etc. Y un frontend en alguna tecnología como React, que consuma dicha API. De este modo, la API podría ser consumida por otros proyectos, tanto propios como ajenos si se hiciera pública.

Por último, la aplicación no ha sido desplegada en ningún servidor externo, pero si se desea se puede ejecutar en local y utilizar alguna herramienta como *Ngrok* para construir un túnel hacia nuestro servidor local.

6.3. Conclusiones

El resultado del proyecto cumple con las metas que se propusieron en su inicio. Tanto el backend como el frontend de la aplicación funcionan correctamente, siendo usables para el usuario, que puede configurar la captura de tweets para partidos políticos y consultar el resultado de los análisis de este proceso.

Para llegar a ello ha sido necesaria la toma de decisiones que han mejorado el desarrollo y la funcionalidad de la aplicación.

En los aspectos de usabilidad de la aplicación se debe tener en cuenta que se manejan grandes cantidades de información, por lo que el servidor donde se ejecute de cubrir ciertos requisitos técnicos.

Por último, la calidad de los análisis depende del servicio que utilicemos para estos. Por ello la aplicación se puede extender y mejorar con el uso de otros servicios de análisis de opinión. Cuantos más tipos de análisis almacenemos el usuario podrá obtener un mejor resultado y la posibilidad de realizar comparativas, aunque como hemos visto los servicios que realmente merecen la pena suelen ser de pago

Bibliografía

Gautham Rajgopal. Twitter Sentiment Analysis. (2019). Obtenido en https://medium.com/@gauthambms/twitter- sentiment-analysis-62d72429608e

Saura, José & Reyes-Menendez, Ana & Palos-Sanchez, Pedro. (2018). Un Análisis de Sentimiento en Twitter con Machine Learning: Identificando el sentimiento sobre las ofertas de #BlackFriday.

Página web de la documentación de Spring Framework. (s.f.). Obtenido en https://spring.io/docs/reference

Página web de la documentación de MongoDB .(s.f.). Obtenido en https://docs.mongodb.com/

Página web de la documentación de Twitter4J . (s.f.). Obtenido en http://twitter4j.org/javadoc/index.html

Página web de la documentación de Bootstrap. (s.f.)

Obtenido en https://getbootstrap.com/docs/

Página web de la documentación de AdminLTE. (s.f.)

Obtenido en https://adminlte.io/docs

Página web de la documentación de Chart.js . (s.f.)

Obtenido en https://www.chartjs.org/docs/latest/

Página web de la documentación de la API de Twitter. (s.f.)

https://developer.twitter.com/en/docs

Página web de Microsoft Azure Text Analytics. (s.f.)

Obtenido en https://azure.microsoft.com/es-es/services/cognitive- services/text-analytics/

Página web de Google Cloud Natural Language. (s.f.)

Obtenido en https://cloud.google.com/natural-language/?hl=es

Página web de Monkey Learn. (s.f.)

Obtenido en https://monkeylearn.com/

Página web de la documentación de Meaning Cloud .(s.f.)

Obtenido en https://www.meaningcloud.com/developer/sentiment-analysis/doc/2.1

Página web de Text-Processing. (s.f.)

Obtenido en *http://www.text-processing.com*

Página web de la documentación de Stanford Core NLP. (s.f.)

Obtenido en https://stanfordnlp.github.io/CoreNLP/api.html

Página web de la documentación de Thymeleaf. (s.f.)

Obtenido en https://www.thymeleaf.org/documentation.html

Anexo I. Instalación y arranque

Para poder instalar la aplicación se necesita usar Maven y un entorno donde se puedan ejecutar aplicaciones Java, como se ha indicado durante esta memoria.

Se debe contar con acceso a internet puesto que es necesario para descargar las dependencias del proyecto.

Como nuestra aplicación hará uso de colecciones en MongoDB es necesario tener una instancia de Mongo DB ejecutándose. Para ello se puede ejecutar en la terminal el comando *mongod.*

Una vez hecho esto se puede ejecutar, en el directorio del proyecto, el siguiente comando:

> maven clean install

Tabla 26: Comando para instalar la aplicación

Y a continuación para ejecutar la aplicación:

> maven spring-boot:run

Tabla 27: Comando para ejecutar el proyecto

O bien desde Spring Tools Suite usando: *Run* > *As Spring Boot App*

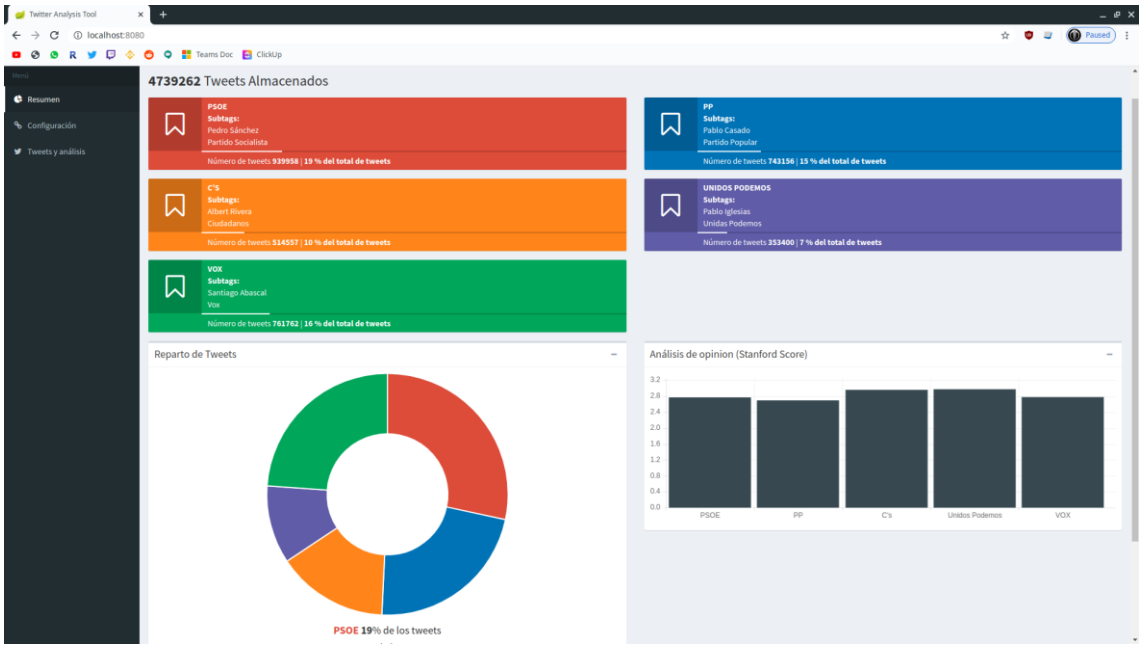

Figura 305: Aplicación corriendo en localhost:8080

Y se podría acceder a ella localmente desde *localhost:8080.*

Anexo II. Datos elecciones abril 2019

Si se desea se puede importar la base de datos resultante del análisis usando este proyecto. Para ello se adjunta el fichero, *tweets.csv*, que contiene una copia de la base de datos que se generó en ese periodo.

Se pueden utilizar herramientas para manejo de ficheros .csv o bien importar el fichero a una base de datos y utilizar la aplicación.

El proceso puede tomar bastante tiempo ya que contiene 4,739,252 tweets. Para importar la base de datos a MongoDB, se puede ejecutar la herramienta *mongoimport* con los siguientes parámetros:

```
> mongoimport –file tweets.csv –type vsv -d tweets -c tweets –headerline -
verbose
```

Figura 28: Comandos para importar tweets.csv

UNIVERSIDAD DE MÁLAGA | uma.es

E.T.S. DE INGENIERÍA INFORMÁTICA

E.T.S de Ingeniería Informática
Bulevar Louis Pasteur, 35
Campus de Teatinos
29071 Málaga