UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

HERRAMIENTA DE EXTRACCIÓN DE NEOLOGISMOS DEL ESPAÑOL EN REDES SOCIALES

Iñaki Cadalso Reymundo
Tutor: Pablo Haya Coll

JUNIO 2018

HERRAMIENTA DE EXTRACCIÓN DE NEOLOGISMOS DEL ESPAÑOL EN REDES SOCIALES

AUTOR: Iñaki Cadalso Reymundo
TUTOR: Pablo Haya Coll

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Junio de 2018

Resumen (castellano)

Este Trabajo Fin de Grado quiere dar unas herramientas para poder observar en las conversaciones que se realizan en la comunidad hispanohablante de Twitter nuevos neologismos y préstamos lingüísticos.

En un mundo en el que millones de hispanohablantes se comunican entre ellos y hacen evolucionar el castellano diariamente en esta red social.

Este uso constante del castellano por parte de diferentes gentes alrededor del mundo da pie a la invención de nuevas palabras, con un significado nuevo para los nuevos tiempos en los que vivimos.

Haciendo uso de composición de palabras ya existentes o adaptaciones de palabras extrajeras al castellano son creadas en conversaciones informales. A pesar de que algunos de estos términos tienen una vida efímera.

Ser conscientes de nuevas palabras y expresiones es lo que hace, y ha hecho, que el castellano sea un idioma tan rico y con tantas variantes.

Sin embargo, se ha de informar y recoger estos neologismos y préstamos lingüísticos para que todo el mundo este al tanto de las nuevas palabras que se están usando en la comunidad y de su significado.

Dentro de nuestra aplicación está el objetivo de encontrar nuevos neologismos analizando mensajes escritos en la red social Twitter. Analizar el valor de las palabras como nuevas y mostrarlas para todo el mundo, si han sido valoradas positivamente por gente cualificada.

Esta es una aplicación para navegadores web, sustentada en tecnologías actuales que son usadas por gran cantidad de empresas de primer nivel. Esta memoria describe la aplicación desarrollada en este proyecto, analizando las tecnologías explotadas, su uso en la aplicación y el diseño final de cara a los usuarios, así como todas las pruebas realizadas durante el proyecto para su correcto funcionamiento.

Palabras clave (castellano)

Anglicismo, Neologismo, Twitter, prestado, candidato

Abstract (English)

This Bachelor Thesis wants to give tools to see conversations that new neologisms and linguistic loans are made in the Hispanophone community of the social network Twitter.

In a world in which millions of Spanish speakers communicate with each other and make evolve the Spanish daily in this social network.

This constant use of Spanish by different people around the world gives rise to the invention of new words, with a new meaning for modern times in which we live.

Making use of composition of existing words or adaptations of words foreign to Spanish are created in informal conversations. While some of these terms have a short-lived.

Be aware of new words and expressions is what makes, and has made, that Spanish is a language so rich and so many variants.

However, should inform and collect these neologisms and linguistic loans for everyone to be aware of new words being used in the community and its meaning.

Within our application is the goal of finding new neologisms, analyzing messages posted on social networking site Twitter. Examine the value of the words as new and show them to everyone, if they have been valued positively by qualified people.

This is an application for web browsers, based on current technologies that are used by many leading companies. This report describes the application developed in this project, analyzing exploited technologies, their use in the application and the final design for users, as well as every test carried out during the project for its proper functioning.

Keywords (inglés)

Anglicism, Neologism, Twitter, borrowing, nominee

Agradecimientos

Aprovecho esta oportunidad para agradecer a todas las personas que me han ayudado a terminar la carrera y que han hecho posible este proyecto.

En primer lugar, quiero dar las gracias a mis padres, por darme la posibilidad de estudiar esta carrera, aguantarme durante estos años con los cambios de humor según el nivel de trabajo/estudio que tuviera.

A mis amigos y compañeros, que me ha ayudado en todo lo que han podido ha sacar todo esto adelante.

A todos los profesores de la Escuela Politécnica de la UAM que me han dado clase, salvo a los que me han suspendido. También a toda la gente de administración que hace posible que funcione esta Universidad.

Por ultimo y no menos importante, dar las gracias a Pablo, mi tutor de proyecto, que me a tutorizado perfectamente y gracias a él he conseguido terminar y entregar este proyecto sin contratiempos. Y a Antonio que me ayudado a entender mejor el mundo de la lingüística.

Seguro que falta agradecer a más gente, pero si no me acuerdo ahora no habrán sido tan importantes.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

Esta memoria de TFG responde al trabajo llevado a cabo para la construcción de una aplicación que responda a la necesidad del descubrir nuevas palabras usadas en nuestro idioma, el castellano, y recogerlas para poder estudiarlas.

La constante evolución de un idioma tan rico como es el castellano precisa monitorizar como se produce tal evolución, de manera que se pueda analizar las nuevas palabras prestadas de otros idiomas y compartirlas con el resto de hispanohablantes.

En la comunicación habitual entre distintas personas es donde se da tal evolución, y que mejor sitio que en Twitter donde el castellano es la segunda lengua más usada, y cuenta con millones de usuarios teniendo un alto porcentaje de conversaciones constantes cada día.

El sistema se encargará desde una fuente de tweets descubrir nuevas palabas, neologismos (predominantemente anglicismos). Posteriormente un usuario cualificado se encargará de aceptar los términos encontrados de manera que el público general pueda consultar que nuevas palabas se están usando en el castellano.

1.2 Objetivos

- **Analizador efectivo:** El programa encargado de analizar el texto fuente constara de distintos filtros que permitan configurar la búsqueda de los posibles neologismos.

- **Interfaz sencilla y de fácil uso:** La aplicación está dirigida a usuarios de corte no técnico. Por lo que las funcionalidades tienen que quedar claras y fáciles de ver. Tanto para el usuario cualificado para catalogar neologismos, como para el usuario común que consultara únicamente.

- **Diseñar una buena arquitectura de la aplicación:** Se quiere que este sea el programa desde el que se amplíen y extiendan funcionalidades, por lo que el diseño principal debe estar claro para posteriormente añadir módulos.

- **Dar la mejor información posible:** Para el usuario cualificado intentar dar el mejor posible contexto necesario para juzgar un posible candidato. Así durante la fase de análisis se recoge toda la información posible para su posterior visualización.

- **Despliegue del sistema:** Hacer una estructura exportable de la aplicación para su fácil instalación en el servidor final donde se asiente el programa.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- **Capítulo 1: Introducción**

  En el capítulo donde nos encontramos se da una breve instrucción del tema a tratar durante toda la memoria, los objetivos que cubre el trabajo realizado y se da una descripción de cómo está estructurada la memoria de aquí en adelante.

- **Capítulo 2: Estado del Arte**

  Este capítulo expone las ideas recogidas y trabajos recopilados entorno a la idea de la que parte este TFG.

- **Capítulo 3: Diseño**

  Este capítulo describe como se ha planteado el problema desde un punto de vista más técnico, la arquitectura que tiene la aplicación y las tecnologías usadas para afrontar el problema planteado.

- **Capítulo 4: Desarrollo**

  Este apartado recoge como se ha codificado y construido cada parte del programa desde el punto de vista del programador.

- **Capítulo 5: Pruebas**

  Este capítulo incluye las distintas pruebas realizadas para ver el correcto funcionamiento de los módulos que componen el programa.

- **Capítulo 6: Conclusiones y trabajo futuro**

  Este apartado contiene las conclusiones obtenidas después de la finalización del programa y las posibles mejoras/ampliaciones que podría tener en un futuro el programa.

2 Trabajos Previos

El estado de esta temática actualmente se encuentra sobre todo artículos/estudios de un corte más académico. Esta más desarrollada la metodología de cómo tratar la información encontrada que programas que analicen automáticamente la información

2.1 Estudios Académicos

Se nos ha dado un código previo sencillo por parte del *Laboratorio de Lingüística Informática* de la Universidad Autónoma de Madrid en colaboración con otros organismos como el *Instituto Cervantes en la Universidad de Harvard de* varios estudios.

2.1.1 Configuración lingüística de anglicismos procedentes de Twitter en el español estadounidense

Este estudio ofrece un análisis contextualizado de la relación de candidatos a anglicismos elaborada mediante la aplicación de un método de búsqueda en el corpus. El análisis permite tomar decisiones sobre qué anglicismos pueden incorporarse al diccionario y cuáles no, de acuerdo a los fines del repertorio lexicográfico. Asimismo, el análisis lingüístico de las formas seleccionadas finalmente para su incorporación al diccionario revela el perfil de los anglicismos que se están introduciendo en el español estadounidense, en su dimensión ortográfica, gramatical y léxico-semántica. [1]

2.1.2 Diccionario de anglicismos del español estadounidense

Este *Diccionario de anglicismos del español estadounidense* (DAEE), como indica su título, reúne anglicismos utilizados en el español de los Estados Unidos y aporta información descriptiva de su uso social, geográfico y estilístico. En realidad, se trata de un diccionario diferencial, descriptivo y de uso del español estadounidense, cualidades que delimitan sus objetivos y que responden a razones específicas. [2]

2.2 Aplicaciones de análisis

También nos suministraron unas series de *prototipos* de como trataban los textos. Como hacían un análisis metodológico para sacar posibles candidatos y su tratamiento oportuno.

2.2.1 Procedimiento para extraer candidatos a anglicismo

Consta de un listado de procedimientos de limpieza de texto hasta extraer la información deseada. Inicialmente eran una serie de comandos en bash de filtrado como: *cat, sed, gawk o grep.*

A destacar, el uso de un filtro propio de ellos denominado GRAMPAL, que se integrará en el programa creado y que se describirá más adelante en la memoria.

2.2.2 Banco de neologismos

La tecnología más cercana, a una parte del objetivo a desarrollar en este trabajo, es el banco de neologismo reconocido por el Centro Virtual Cervantes [3].

En este banco hay una recopilación de neologismos aceptado por el Instituto Cervantes en 2004, permite al usuario consultar un diccionario de palabras prestadas y aceptadas en castellano. Con información de cada neologismo, su contexto, fuente, etc.

3 Diseño

3.1 Descripción de las Tecnologías

A continuación, describiremos las tecnologías implicadas en la construcción del programa, porqué se han usado y su papel en el desarrollo del programa.

3.1.1 Elasticsearch

Es un motor de búsqueda basado en Lucene, que es una biblioteca de recuperación de información con todas las características, dando una gran cantidad de opciones a la hora de la búsqueda de texto. También actúa como una base de datos NoSQL haciendo de indexación de documentos como JSON con un gran rendimiento.

Actualmente, es usado por muchos desarrolladores por su gran escalabilidad y flexibilidad a la hora de extender los recursos y equilibrar carga entre nodos en un clúster.
Goza de una gran velocidad al ejecutar consultas extremadamente complejas y hacer uso de la cache para guardar conjuntos de resultados por cada tipo de filtro consultado anteriormente.

Ahondando en el uso de documentos JSON, intenta detectar la estructura de los datos y usarlo tanto para indexar, como para las búsquedas de texto.

Tiene una buena API accesible desde varios lenguajes de programación y también las acciones se pueden realizar utilizando una API Restful simple.

Se está adoptando o ha sido ya adoptado en marcas importantes, como: Tesco, Linkedin, Foursquare, Facebook, Netflix, Dell, Ebay, Wikipedia, The Guardian, New York Times, Salesforce, Docker, Orange, Groupon, Eventbrite y muchos otros dando más que notables resultados en muchos casos.

Viendo las grandes características que tiene Elasticsearch y su papel en la actualidad tecnológica de la informática pensamos en hacer uso de ella en nuestro programa.

Es una tecnología muy adecuada para el tratamiento de la información, porque los datos a tratar están en formato de texto tipo JSON, toda la información proporcionada por cada tweet esta codificada en ese formato.

Por lo que podemos almacenar los datos en una estructura conocida para su tratamiento tanto antes de indexarla, como tras consultarla.

La existencia de un cliente de bajo nivel para Python también es otro punto por lo que la hemos usado. Permitiendo una gran cantidad de formas de indexación y filtrado en las búsquedas de datos bajo la estructura del JSON establecida.

3.1.2 Django

Es un framework web de alto nivel de código abierto escrito en Python. Gracias a esto hereda todas las características y facilidades que nos da Python, entre ellas escribir código bastante fácil de entender, y sobre todo te permite desarrollar aplicaciones muy rápidas y potentes dentro del entorno de un sitio web complejo.

La rapidez y gran flexibilidad de la que hace gala Django encaja también con la filosofía que hemos visto anteriormente con Elasticsearch. Proporciona, por defecto, un sistema interno para configurar la administración del sitio como se quiera.

Como curiosidad Django nació en un ambiente periodístico, donde se subían noticias muy rápido, y como los desarrolladores no pudieron estar a ese ritmo, decidieron crear algo que sí lo hiciera, y así fue como nació Django. Conocidos sitios web Instagram, Mozilla, The Washington Times, Disqus o Bitbucket lo usan.

Por casi los mismos motivos que con Elasticsearch, Django nos encaja para la construcción tanto de un front-end para el usuario final, un sitio web, como parte de un back-end para hacer de nexo, en la manipulación de toda la información que maneje el programa, entre los distintos módulos del mismo. De cara a ampliaciones puntuales y futuras encaja su filosofía de no repetirse y la reutilización de códigos.

La codificación en Python también es un gran aliciente para que, nativamente, se pueda acoplar el código base en el que se sustenta el programa.

3.1.3 Docker

Es una herramienta para facilitar la creación y despliegue de aplicaciones o programas dentro de *contenedores.*

Proporciona características parecidas a las que una máquina virtual pueda tener, pero da muchas más opciones a la hora de empaquetar las aplicaciones en un *contenedor*, e independizarlo del resto del entorno en el que se vaya a desplegar.

Así el desarrollador se puede centrar en proporcionar, a su programa, todo lo necesario para su ejecución y empaquetado pudiéndose desplegar en un sistema ajeno, que soporte las tecnologías incluidas, conservando todas sus funcionalidades.

Tiene la ventaja de facilitar la distribución, pues se conserva tal como fue ideado y no importa el equipo o ambiente donde se despliegue. Las pruebas se pueden hacer en una copia del contenedor, en un ambiente exactamente igual y seguro.
Cuenta con un repositorio de imágenes con sistemas y configuraciones, las cuales se pueden usar de base para crear nuevos contenedores.

Actualmente están siendo probadas/usadas, en distinto grado, por compañías como Google, Spotify, Red Hat o Microsoft.

A nosotros, principalmente, nos interesa para facilitar el despliegue del programa, al contar con varios módulos, es muy sencillo configurar todo lo necesario para su ejecución, y poder empaquetar todo lo necesario en una imagen.

En el despliegue final en el servidor, con cargarlo estaría ejecutándose todo lo necesario para el correcto funcionamiento del programa.

3.1.4 Python

Es un lenguaje de programación conocido por su filosofía basada en un código limpio y una buena indentación *(ejemplo de neologismo-anglicismo aceptado).*

Por las razones expuestas, en las otras tecnologías, encaja perfectamente utilizar este lenguaje. Hemos elegido la versión 3 (actualmente la 3.6.5.), principalmente, porque esta versión trata mejor los textos codificado en UTF-8 y a nivel base de nuestro programa es fundamental. También por los paquetes que vamos a usar y que mencionaremos brevemente a continuación.

3.1.4.1 Elasticsearch-py

Como queda claro, por el nombre del paquete, es el cliente que se comunica con Elasticsearch haciendo posible todas las operaciones relacionadas con la manipulación de la información, en el programa, desde el código.

3.1.4.2 Json

Como ya hemos hablado, el uso del formato de JSON es necesario. Usamos este paquete específico para su tratamiento dentro de nuestro código.

3.1.4.3 Re

Es un paquete muy amplio, de los más amplios en Python, solo vamos a usar una parte. La referida al tratamiento de expresiones, y construcción de patrones, para el tratamiento de textos.

3.1.5 JSON

Es un formato ligero, de texto dirigido al intercambio de datos. Es fácil y entendible tanto para humanos, como para las máquinas. Está basado en JavaScript, pero es completamente independiente de otros lenguajes, aun usando convenciones de muchos de ellos para facilitar su uso. Por las propiedades mencionadas hacen este formato ideal para el intercambio de datos entre lenguajes.

Debido a sus propiedades, lo hemos elegido para hacerle, contenedor de los datos, cuando viajan entre módulos. Siendo perfecto, tanto para el almacenaje, como el tratamiento en Python ó Elasticsearch; y la fuente original de datos a usar también hace uso de este formato.

3.1.6 Bootstrap

Bootstrap, en su versión 3, es un framework de CSS que permite la rápida y fácil construcción de una interfaz amigable para front-end de un sitio web. Contiene plantillas de varios elementos clásicos de diseño que se basa en HTML, CSS y extensiones JavaScript.

Se ha usado para dar un mejor estilo, y funcionalidades, a la parte de front-end que construyamos con Django

3.1.7 Apache Kafka

Es un sistema basado en la publicación-suscripción de mensajes, en la que se intercambian datos entre: procesos, aplicaciones y servidores.
Se puede decir que en este sistema hay dos actores:

1. **Productores:** Crea un mensaje con el *Topic* que defina lo que quiere que se haga con el mensaje publicado en el sistema.

2. **Consumidores:** Espera, a poder procesar un mensaje de la forma que mande el *Topic.*

El *Topic* define la categoría que tiene el mensaje, el cual puede contener cualquier tipo de información.

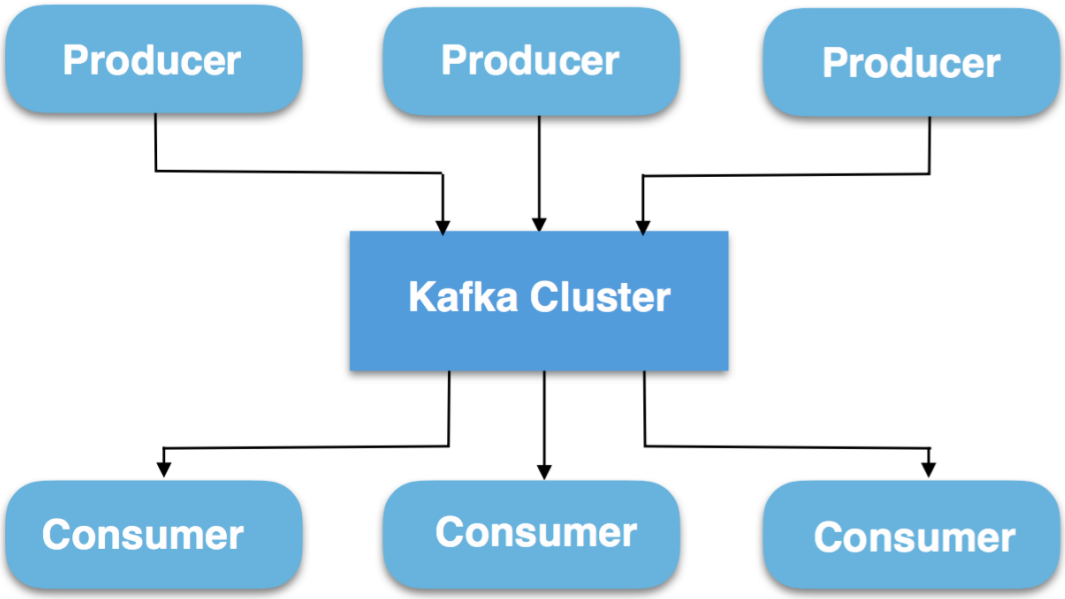


Figura 3-1: Diagrama Kafka Broker

El sistema de Kafka Broker se encarga de conectar a los Productores y a los Consumidores reteniendo los mensajes y distribuyéndolos. Haciendo posible la paralelización de los procesos repartiendo el trabajo en nodos de Consumidores según el *Topic*.

Permite muchas posibilidades al poder añadir más nodos de un tipo u otro, e incluso añadir más clústeres. Y repartir tareas, bien definidas, a distintos procesadores mediante suscripción.

En definitiva, es una forma de repartir trabajo más eficientemente, y hacer posible el uso de la paralelización; si se sabe los pasos que ha de seguir un proceso desde la entrada de datos.

Por todo lo expuesto anteriormente, esta tecnología se incluye a la hora de procesar cada tweet, de forma similar, podríamos repartir el trabajo en un clúster y no hacerlo todo linealmente al pasar cada filtrado de texto.

Finalmente, aunque fue probada su implementación, no representaba una mejora, al no disponer del tiempo/equipamiento. No justificaba su uso al no haber, realmente, un incremento en la eficiencia/rendimiento del procesamiento base.

3.2 Arquitectura del Sistema

La estructura de todo el sistema se compone de varios módulos.

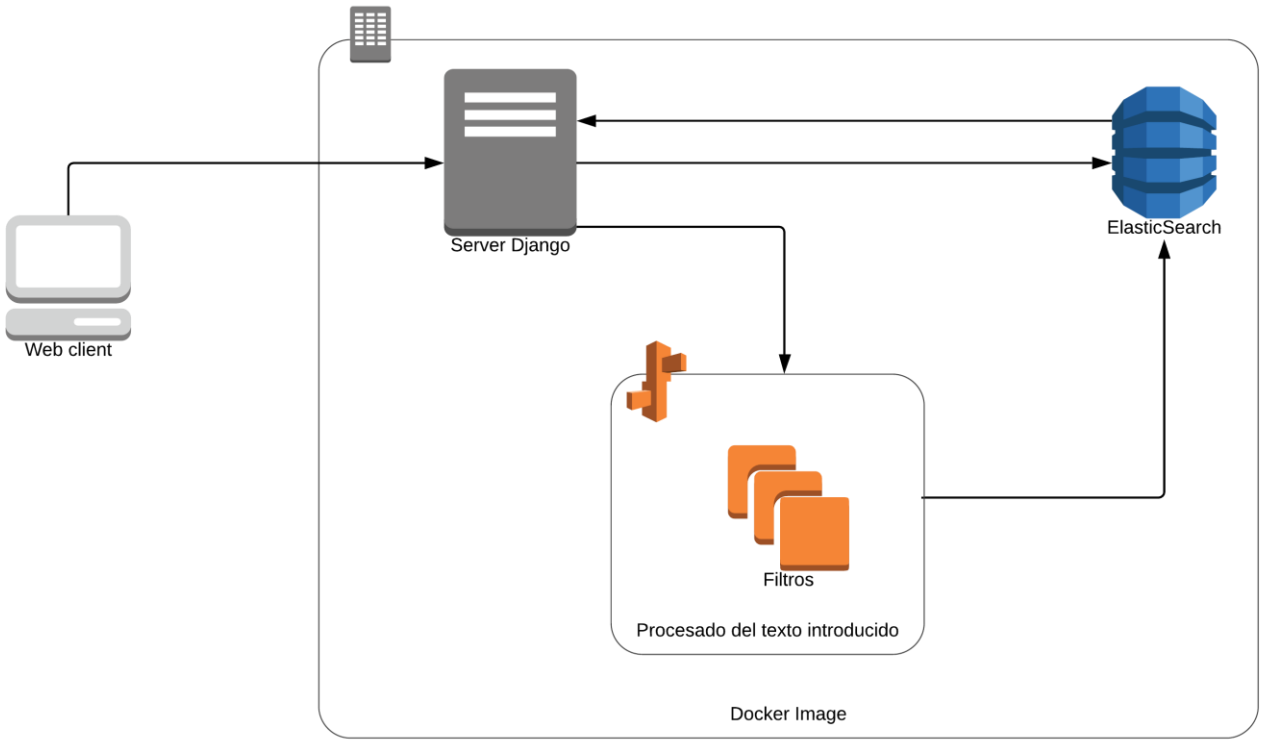


Figura 3-2: Diagrama de la Arquitectura

Como se puede ver en el diagrama tenemos varios módulos:

- **Web Client:** Es el front-end del programa, se accederá desde un navegador web con soporte.

- **Docker Imagen:** Recogerá toda la información necesaria del programa para funcionar como está previsto.

    - **Server Django:** Construido bajo Django, se encargará de hacer de nexo del back-end con el resto de módulos para las operaciones necesarias, y darle la información necesaria al front-end.

    - **Elasticsearch:** Encargado de actuar como base de datos y proporcionar la información almacenada en ella.

    - **Motor Procesamiento Lenguaje Natural:** El núcleo del programa. Se encargará de tratar la información suministrada por el usuario a través del front-end y pasando por todos los módulos necesarios hasta llegar aquí.

        - **Filtros:** la descomposición principal a la que llega el código. Cada uno de ellos se encargará de analizar de una forma, determinada, el texto en origen y dar un resultado que se ira almacenando.
          Unos ejemplos de cómo se aplican los filtros son:
            - Se limpiarán hashtags, usuarios, enlaces, emojis del texto, se corrigieran palabras a su forma correcta. Encontrar palabras bien formadas que no estén en un diccionario.

3.3 Diseño del Front-End

El sitio web, a mostrar al usuario final, estará compuesto por varias páginas con una funcionalidad asociada a cada una de ellas.

Aquí tenemos un mapa en forma de árbol del sitio web:

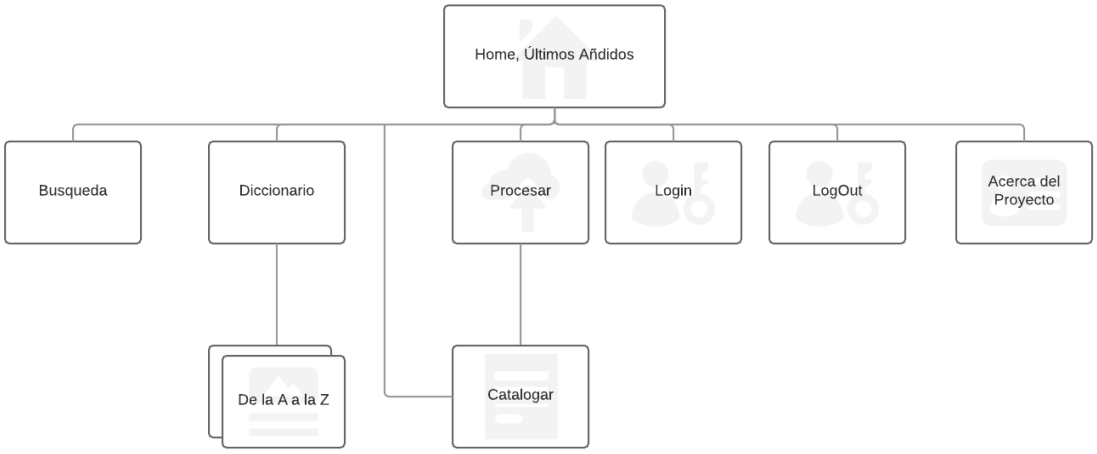
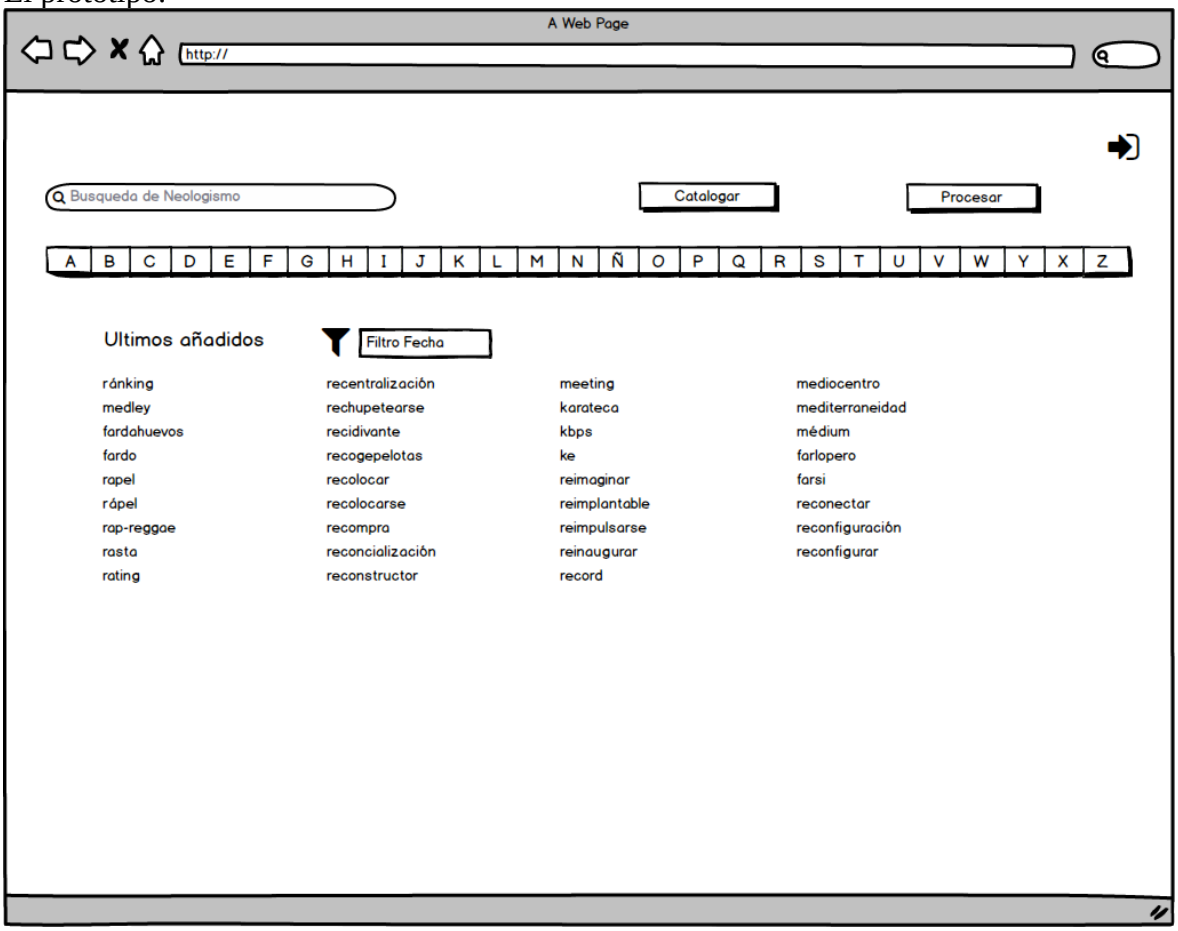
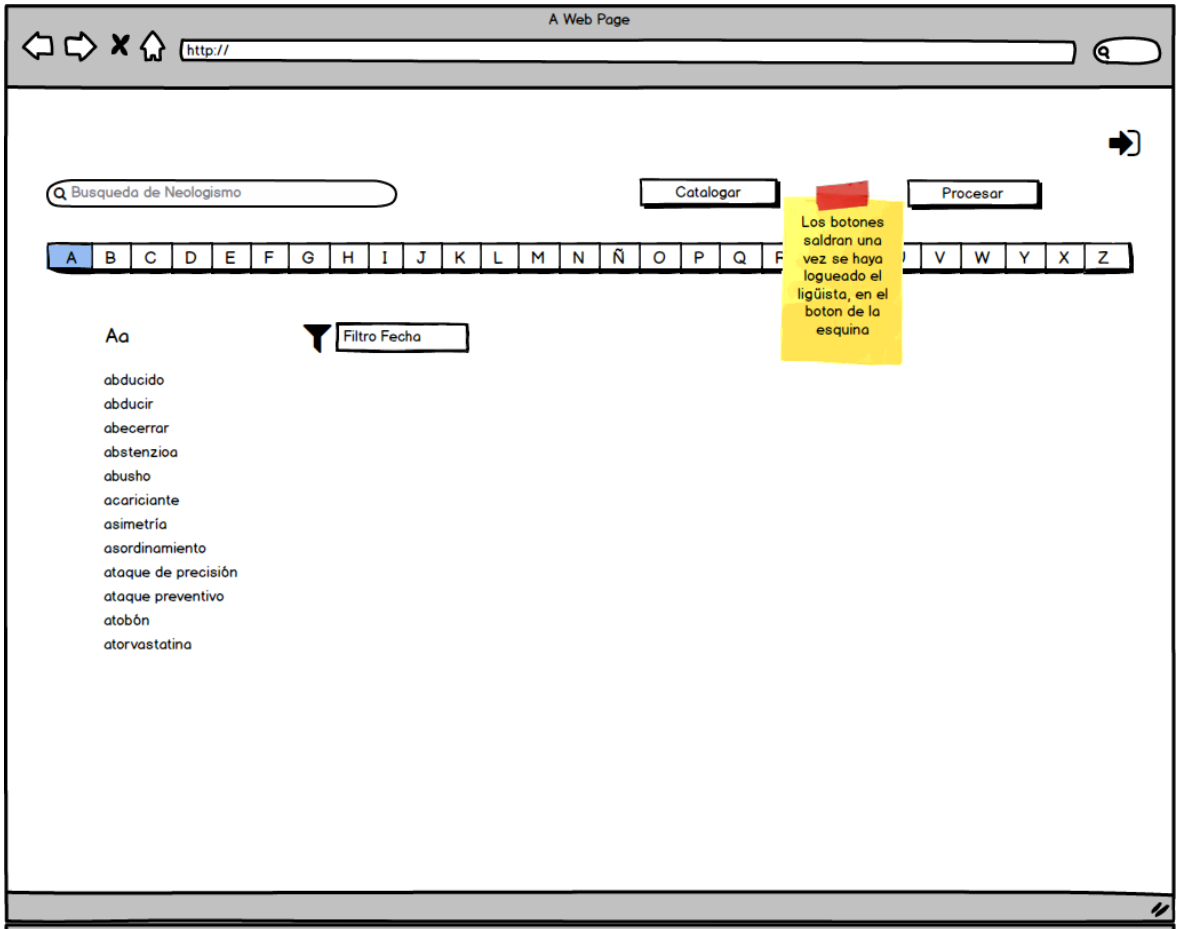
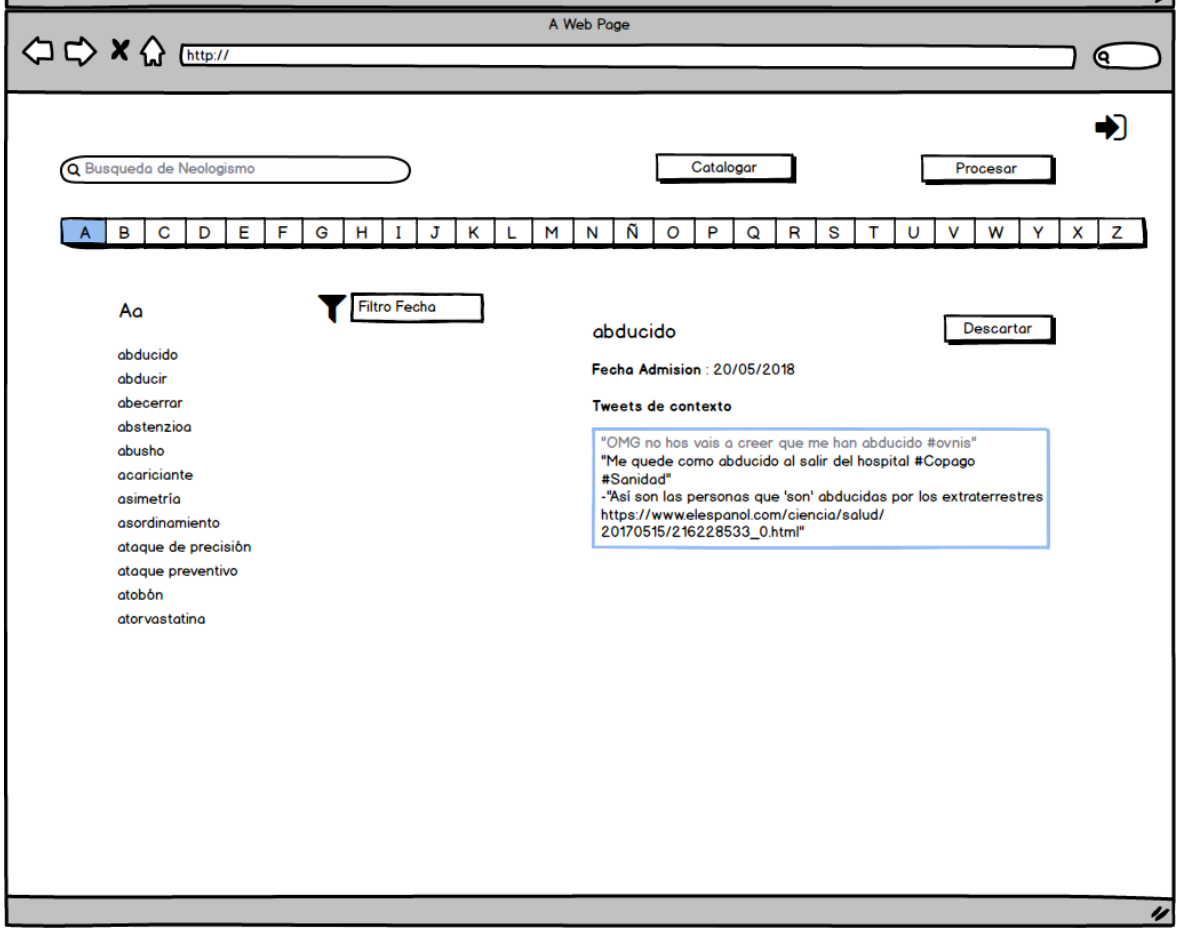


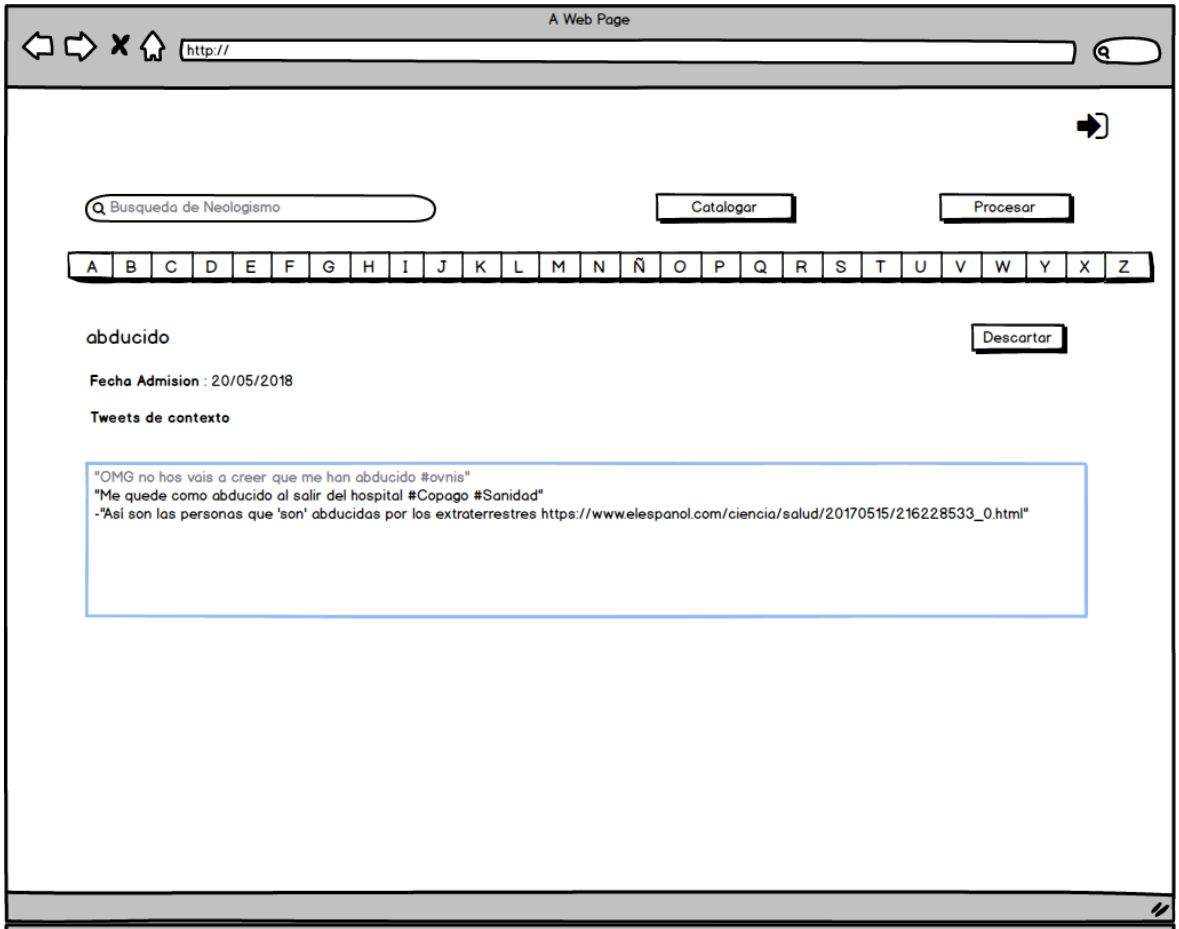
Figura 3-3: Árbol sitio Web

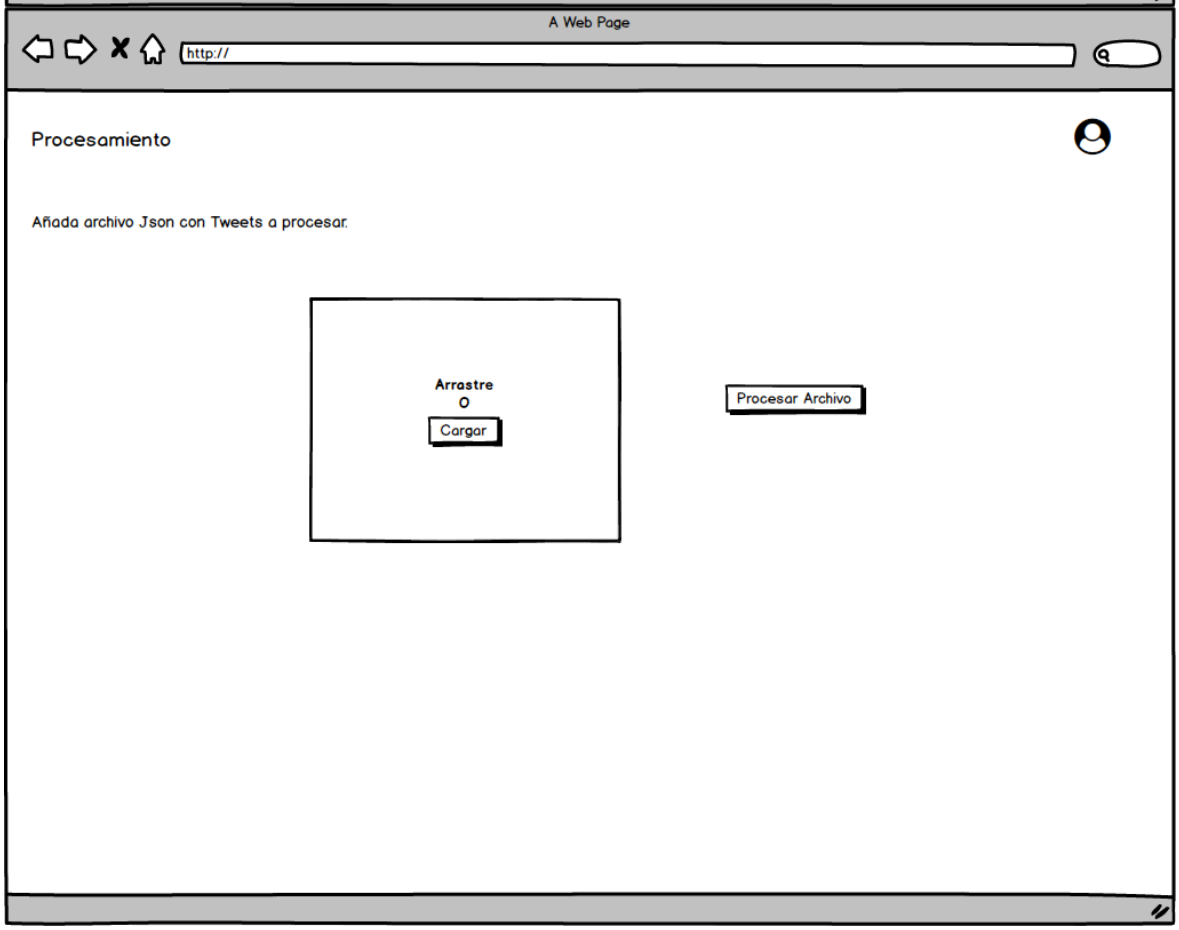
El prototipo:

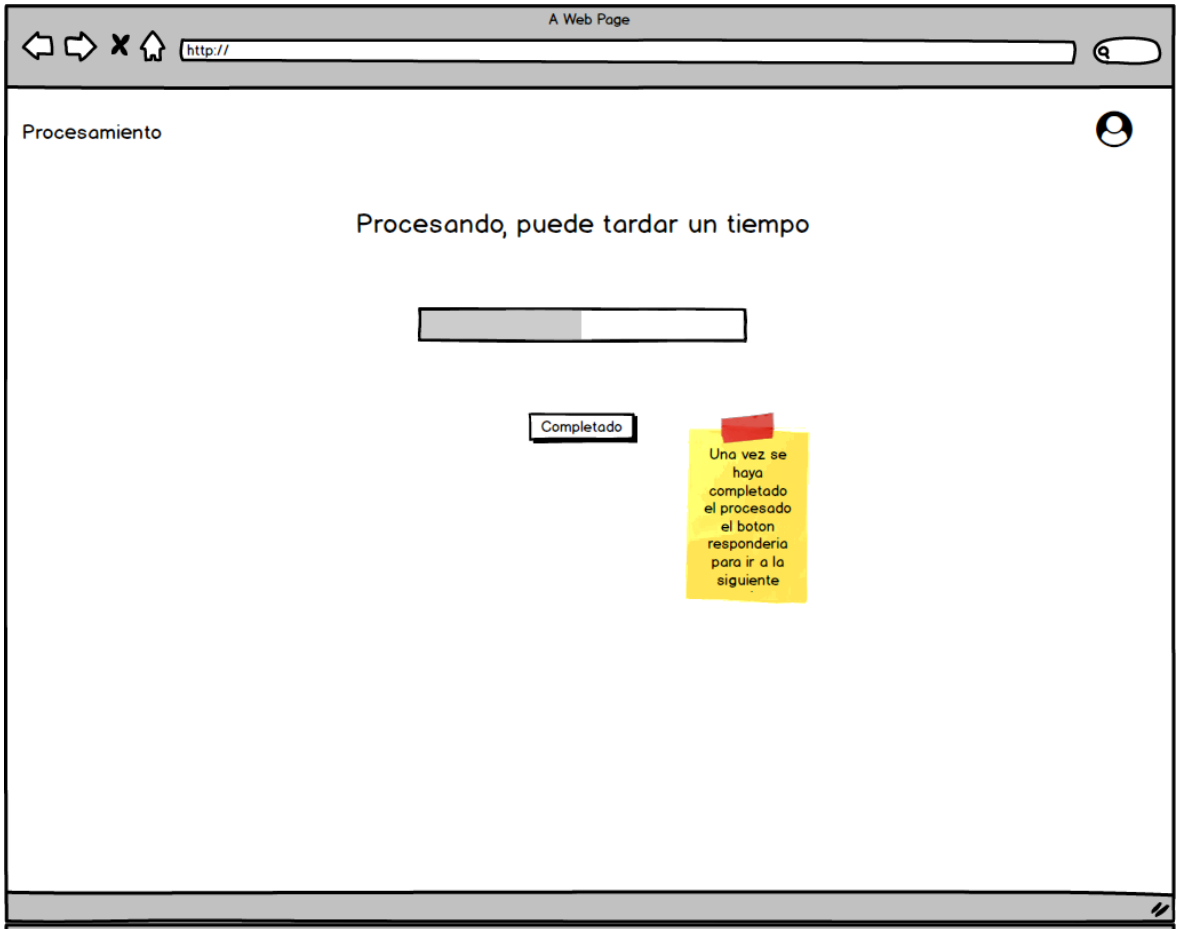


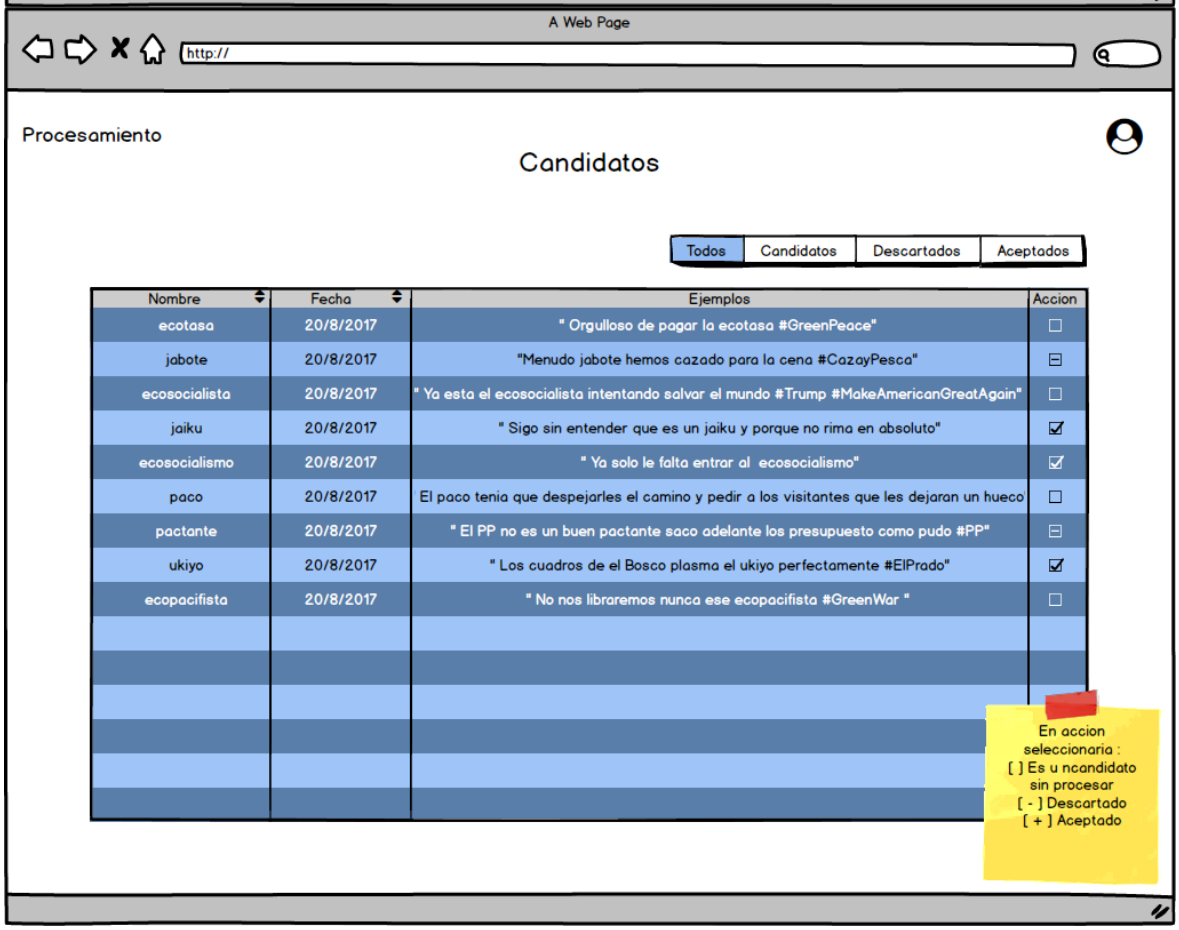

A Web Page
http://
Busqueda de Neologismo
Catalogar
Procesar
Los botones saldran una vez se haya logueado el ligüista, en el boton de la esquina
A B C D E F G H I J K L M N Ñ O P Q R V W Y X Z
Aa
Filtro Fecha
abducido
abducir
abecerrar
abstenzioa
abusho
acariciante
asimetría
asordinamiento
ataque de precisiôn
ataque preventivo
atobôn
atorvastatina


A Web Page
http://
Busqueda de Neologismo
Catalogar
Procesar
A B C D E F G H I J K L M N Ñ O P Q R S T U V W Y X Z
Aa
Filtro Fecha
abducido
abducir
abecerrar
abstenzioa
abusho
acariciante
asimetría
asordinamiento
ataque de precisiôn
ataque preventivo
atobôn
atorvastatina
abducido
Descartar
Fecha Admision : 20/05/2018
Tweets de contexto
"OMG no hos vais a creer que me han abducido #ovnis"
"Me quede como abducido al salir del hospital #Copago #Sanidad"
-"Así son las personas que 'son' abducidas por los extraterrestres https://www.elespanol.com/ciencia/salud/20170515/216228533_0.html"


A Web Page
http://
Busqueda de Neologismo
Catalogar
Procesar
A B C D E F G H I J K L M N Ñ O P Q R S T U V W Y X Z
abducido
Descartar
Fecha Admision : 20/05/2018
Tweets de contexto
"OMG no hos vais a creer que me han abducido #ovnis"
"Me quede como abducido al salir del hospital #Copago #Sanidad"
-"Así son las personas que 'son' abducidas por los extraterrestres https://www.elespanol.com/ciencia/salud/20170515/216228533_0.html"


A Web Page
http://
Procesamiento
Añada archivo Json con Tweets a procesar.
Arrastre
O
Cargar
Procesar Archivo


A Web Page
http://
Procesamiento
Procesando, puede tardar un tiempo
Completado
Una vez se haya completado el procesado el boton responderia para ir a la siguiente


A Web Page
http://
Procesamiento
Candidatos
Todos
Candidatos
Descartados
Aceptados
Nombre
Fecha
Ejemplos
Accion
ecotasa 20/8/2017 " Orgulloso de pagar la ecotasa #GreenPeace"
jabote 20/8/2017 "Menudo jabote hemos cazado para la cena #CazayPesca"
ecosocialista 20/8/2017 " Ya esta el ecosocialista intentando salvar el mundo #Trump #MakeAmericanGreatAgain"
jaiku 20/8/2017 " Sigo sin entender que es un jaiku y porque no rima en absoluto"
ecosocialismo 20/8/2017 " Ya solo le falta entrar al ecosocialismo"
paco 20/8/2017 El paco tenia que despejarles el camino y pedir a los visitantes que les dejaran un hueco
pactante 20/8/2017 " El PP no es un buen pactante saco adelante los presupuesto como pudo #PP"
ukiyo 20/8/2017 " Los cuadros de el Bosco plasma el ukiyo perfectamente #ElPrado"
ecopacifista 20/8/2017 " No nos libraremos nunca ese ecopacifista #GreenWar "
En accion seleccionaria :
[ ] Es u ncandidato sin procesar
[-] Descartado
[+] Aceptado

3.3.1 Inicio

La página principal recogerá todas las posibles navegaciones en el sitio web. Dando más opciones, una vez se haya identificado, como un usuario cualificado para la catalogación de neologismos.

Mostrará los últimos neologismos añadidos por orden cronológico.

3.3.2 Búsqueda

Mostrará la búsqueda, en concreto, del neologismo solicitado.

3.3.3 Diccionario por Letra

Mostrará, según la letra escogida del diccionario, todos los neologismos aceptados que comience por esa letra.

3.3.4 Login

Se introducirá un usuario valido en el sistema para poder acceder a las funcionalidades extras, que se comentaran a continuación.

3.3.5 Procesado

El usuario podrá subir un archivo valido, para procesar su información en el programa.

3.3.6 Catalogación

El usuario, tras haber procesado, uno o varios archivos válidos, vera la información de cada candidato introducido en una tabla; con la información pertinente para su posible aceptación o rechazo como candidato.

3.4 Conclusiones

Diseñar un programa desde cero puede resultar, en muchas ocasiones, difícil. Aun así, con unas ideas medianamente claras y con la ayuda de la tecnología seleccionada. Se puede llevar a buen puerto el proyecto, permitiendo dejar una buena base desde la que construir lo que se propone.

Con las ideas plasmadas, en un diseño eficiente, pasaremos a detallar a continuación como se ha utilizado lo descrito, anteriormente, para llegar al objetivo propuesto.

4 Desarrollo

En este punto describiremos, los pasos más cercanos, en el desarrollo del programa. Son, de una forma resumida, los pasos dados para la codificación de la parte que se ha desarrollado desde cero en el programa.

4.1 Codificación Procesador

En este apartado veremos cómo se han codificados los principales módulos implicados en la manipulación de los datos, a más bajo nivel, dentro del programa.

Un descriptivo diagrama de clases, de las que a continuación describiremos:

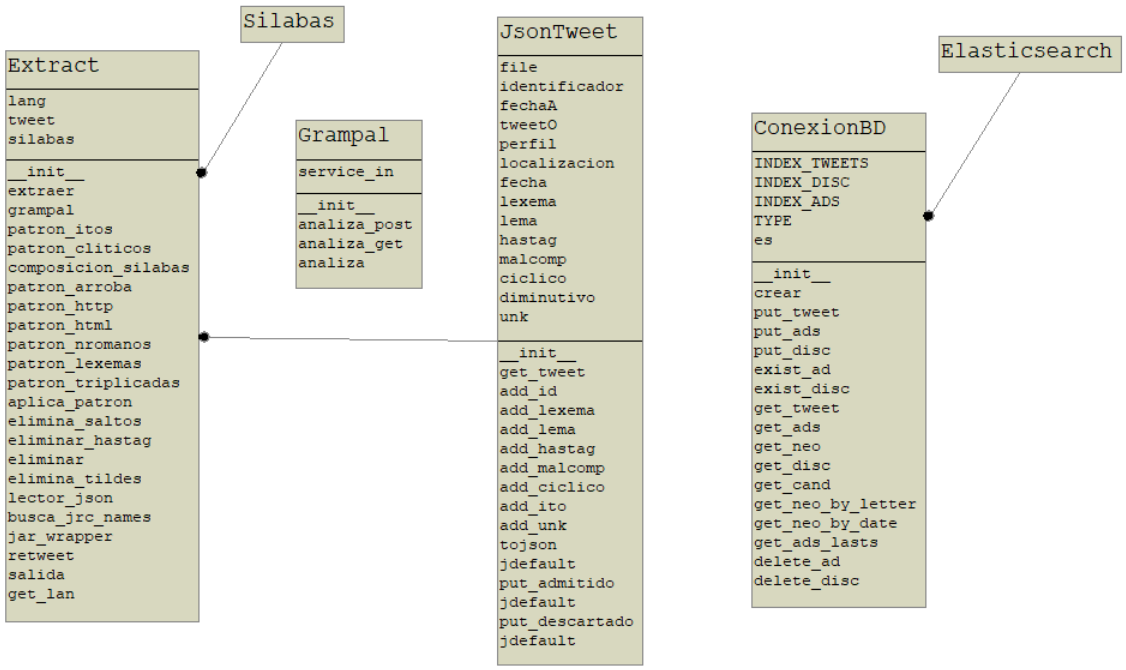


Figura 4-4: Diagrama de Clases Procesador

4.1.1 Clase Extrac

Esta clase escrita en Python es de los principales módulos del programa. En ella se trata el procesado de un archivo con datos originales, tratándolos de las diferentes formas que explicaremos a continuación obteniendo nueva información útil para los análisis posteriores en otros módulos.

Trata cada línea del archivo original, el JSON, contiene un tweet con cierta información, de la que sabemos la estructura y, por lo tanto, podemos acceder a la información necesaria con ayuda del paquete json de Python, ya mencionado en el anterior capitulo. Cada uno se

trata individualmente, de la misma forma. Filtrando en buscan de patrones y expresiones que se van almacenando.

Las formas de filtrado y lo que obtenemos lo veremos en el siguiente apartado.

4.1.2 Filtros

Disponemos de varios filtros para tratar el texto en cuestión, van desde los más especiales a otros más sencillos que "limpian" el texto, cada salida de un filtro alimenta al siguiente.

Hablemos de los sencillos, estos se dedican a eliminar nombres de usuarios de texto del tweet, ya que no nos interesa y podría entrar dentro de la protección de datos.

La limpieza de códigos HTML como emojis, limpieza de links http, números romanos y otros símbolos que no nos aporten información.

La eliminación de hashtags, texto seguido de #, aunque pueden aportar información y son guardados. Intentamos corregir palabras mal escritas; como la simplificación de palabras, y la repetición de más de dos letras, pues en nuestro idioma eso no es posible.

El turno de los más especiales estos, una vez ha quedado el texto lo más limpio posible pueden hacer su trabajo de buscar patrones y expresiones poco habituales, que son:

- **Lexemas:** con ayuda del paquete ya comentado, y usado en los filtros anteriores, a un nivel más sencillo, creamos un patrón (((#[a-zñáéíóúü]+|[a-zñáéíóúü]+)[a-zñáéíóúü]+)') para reconocer palabras validas sueltas. Las palabras encontradas se almacenan y se pasan al siguiente filtro.

- **Grampal:** con las palabras encontradas hacemos uso de la herramienta GRAMPAL, que nos permite saber, sintácticamente, que función desempeña la palabra, si esta herramienta no encuentra una función devuelve UNK por lo que no la ha reconocido y es un fuerte candidato.

- **Diminutivos:** las palabras derivabas pueden no ser reconocidas como válidas. Por lo que con la lista de posibles sufijos ("ito$", "ita$", "ico$", "ica$", "illo$", "illa$", "ico$", "ica$", "ucho$", "ucha$", "ín$","ina$", "uelo$", "uela$", "ete$", "eta$", "uco$", "uca$") que componen un diminutivo, los buscamos en las palabras dadas por si son encontrados eliminarlos, y dejar la palabra original antes del diminutivo.

- **Clíticos:** pasa algo similar que con los diminutivos. Con dos listas de posibles sufijos, (["lo", "la", "le", "les"] y ["me", "te", "se"]) hacemos las combinaciones posibles y las buscamos en el texto, si son encontradas se eliminan y devuelta la palabra original supuesta.

- **Composición:** a partir de un listado con todas las posibles silabas [4] en nuestro idioma cribamos nuestra palabra en busca de cada posible silaba, y se va consumiendo hasta que no queden silabas, o haya una no reconocida. Este filtro solo confirma, o no, si es una palabra es correcta, hablando en composición de silabas.

- **JRC Names:** este filtro comprueba si de alguna forma una palabra, o combinación de palabras, se refiere a un lugar, marca o persona conocido, por el [5],
  Por lo que este filtro hace necesario de todas las palabras no reconocidas en el tweet de texto.
  JRC proporciona un archivo JAR para poder hacer consultas de información. Por lo que hemos incluido este JAR dentro del código haciendo la consulta de las palabras encontradas y comprobamos su resultado
  En la salida del JAR dada, buscamos si ha habido palabras encontradas en su sistema y las descartamos de las que teníamos, originalmente, como candidatas.

4.1.3 Clase JsonTweet

Esta clase se encarga de almacenar la información relevante de los tweets a analizar, de alguna forma se tenía que almacenar toda la información que íbamos encontrando en los filtros expuestos anteriormente.

Dispone de varias listas sin repeticiones, en las que se guardan la siguiente información:

- Identificador del tweet
- Tweet original
- Perfil del usuario creador del tweet
- Localización del usuario creador del tweet
- Fecha del tweet
- Las palabras dadas por cada filtro, cada una en lista correspondiente al filtro
- Fecha de creación de esta estructura

A la hora de haber reunido toda la información, después del filtrado, se convierte a un formato tipo JSON. Es, en modo debug guardado en un archivo de texto, conectado con el siguiente modulo que trataremos para su almacenaje en la base de datos.

4.1.4 Clase ConexionBD

Esta clase se encarga de hacer de intermediario, con ayuda del paquete elasticsearch-py, con Elasticsearch para las tareas de indexación y búsquedas.

Entre sus funciones se encuentran las siguientes:

- La de crear los índices que usaremos: Tweets con candidatos aceptados y descartados. Cada uno contendrá su propio índice y alojaran el mismo tipo.

- Indexar el texto dando con un id o no, dentro de cada índice creando el archivo JSON dado.

- Buscar información de varias formas, según las necesidades, por distintas querys creadas. Desde ver si existe un neologismo, a devolver los últimos neologismos admitidos o buscar neologismos que empiecen por una letra.

- Borra archivos de los índices creados.

Por este módulo pasa toda la interacción hacia Elasticsearch tanto de los módulos explicados antes, como de los siguientes que expliquemos, que hagan uso de la información referida a los tweets.

4.2 Codificación Django

Como ya sabemos Django provee de mucho código ya hecho para dar facilidades a la hora de la construcción de aplicaciones web.
Por lo que, solo era necesario, de la configuración apropiada, para nuestro programa, creación de usuarios, los caminos url para navegar por la aplicación web, así como distintas rutas de archivos propios que entraban en uso en el sitio web.
El núcleo del trabajo desarrollado se encuentra en las *Views* y su asociación con las *Templates* que, ambas, explicaremos a continuación.

4.2.1 Views

La mayoría de funciones implican: dada una información se consulta a través de ConexionBD la información asociada, y se adapta a los modelos que usamos dentro de Django para mostrar la información.
La información recopilada se le pasa a la *Template* para que la muestre, y se espera alguna otra interacción relacionada con la información dada.
Esta información es, un listado de neologismos o candidatos, y su información asociada, tweet de contexto, bio y localidad del usuario, fecha de tweet y admisión del neologismo si procede. Esta información también se usa para el caso de crear un nuevo JsonTweet, y admitir el candidato o descartarlo.
Otras funciones son las propias del proceso de login, subida del archivo JSON a procesar y la ejecución de la clase Extrac para el procesado base con el archivo subido al sitio web.

4.2.2 Templates

En las *Templates* seguimos el principio que rige la filosofía de Django, reutilización de código, por lo que una *template* sirve de base de todas en la que se incluyen las cabeceras, pies de páginas y enlaces que se encuentran en todas las paginas (Loguin, Acerca de, etc..).

Luego se puede decir que hay tres tipos de *Templates*:

1. La consulta de información, a las que tienen acceso un usuario normal, la muestra de varios neologismos con su información, búsqueda de uno o filtrado por letra. Comparte la forma de creación al ser muy similares, pero cambiando la información pedida por el usuario.

2. La *template* de procesado que requiere las comprobaciones de subir un archivo verificarlo y luego procesarlo dentro del programa.

3. La última, y quizás la más importante, desde el punto de vista del usuario objetivo inicial, es la de catalogación. En ella con la ayuda de una tabla de Bootstrap que proporciona muchas funcionalidades por defecto, mostramos toda la información

de cada candidato encontrado en el procesado y sus posibles acciones dentro del programa, aceptar o descartar.

4.3 Codificación Imagen Docker

La creación de una imagen de Docker en la cual contendrá todo el programa ya funcional puede parecer complicado, pero con la ayuda de plantillas ya hechas para todo tipo de programas se puede adaptar fácilmente.

Lo primero es ver si hacer, uso o no de un Sistema Operativo (SO), pues esa es una de la ventaja de Docker puede compartir los recursos con el SO sobre el que se ejecute. Finalmente decidimos tenerlo mejor encapsulado añadiendo el SO por defecto, una versión ligera del ultimo Ubuntu estable.

Identificamos los paquetes de las tecnologías que vamos a necesitar y los añadimos como si se instalaran sobre Ubuntu.

Creamos un usuario y una carpeta en la que tenga los permisos con todo el código desarrollado y ya configurado para su ejecución. Esto es la aplicación de Django, el código y los datos de Elasticsearch.

Con esto ya tendríamos la configuración para una imagen de Docker hecha con el lenguaje propio de Docker en un Dockerfile. Dentro del anexo, en un manual, se explicará detalladamente como se ha realizado y los comandos de Docker para su creación y ejecución.

5 Integración, pruebas y resultados

5.1 Integración

De cara a la integración entre módulos distintos en tecnología no ha habido problemas ni contratiempos que no fueran graves por qué se ha construido teniendo en mente “a priori” su comunicación. Integramos Django, el código creado y Elasticsearch.

La adaptación/instalación en un servidor remoto ha añadido otra capa al desarrollo por adaptar el programa de pruebas en local a aceptar desde remoto. Una vez dados los pasos de configuración en la aplicación de Django y conocida la arquitectura del servidor, se podía manejar como si fuera en local, con la salvedad de ejecutar los comandos vía SSH y acceder al sitio web con una dirección IP de Internet facilitada por el servidor.

5.2 Pruebas

Probamos que información contenía el archivo original con toda la información de cada tweet, que responde a una estructura de esta forma.

```
{ ⊟
  "id":"tag:search.twitter.com,2005:936388635838033920",
  "objectType":"activity",
  "actor":{ ⊞ },
  "verb":"post",
  "postedTime":"2017-12-01T00:17:02.000Z",
  "generator":{ ⊞ },
  "provider":{ ⊞ },
  "link":"http://twitter.com/yakusokus/statuses/936388635838033920",
  "body":"@Lady_Guitarist @cloudytempo_ AJÁ. SABÍA QUE ZUMITO ERA TO POSTU",
  "object":{ ⊞ },
  "inReplyTo":{ ⊞ },
  "favoritesCount":0,
  "twitter_entities":{ ⊟
    "hashtags":[ ⊞ ],
    "user_mentions":[ ⊞ ],
    "symbols":[ ⊞ ],
    "urls":[ ⊞ ]
  },
  "twitter_filter_level":"low",
  "twitter_lang":"es",
  "display_text_range":[ ⊞ ],
  "retweetCount":0,
  "gnip":{ ⊞ }
}
```


Sacamos la información que nos interesaba como si fuera un diccionario, archivo[id], etc. Comprobábamos efectivamente que esa era la información deseada desde un test del módulo, que luego iría a los distintos procesamientos que tenemos.
De una cantidad aproximada de 100.000 tweets obteníamos unos 50.000 candidatos posibles.

De cada filtro, existente dentro del núcleo del código, hemos diseñado un test que prueba el objetivo para el que ha sido creado. Con distintas entradas de texto, una que contenga lo que se está buscando y otra sin ello. Cada test imprime el resultado del antes y después de pasar el filtro, viendo según el filtro si ha sido el modificado en el texto, encontrado las palabras objetivo o una respuesta binaria de Si o No.

Por otra parte, a la hora de la creación de archivos JSON en la clase JsonTweet cuando han de ser completados y preparados para su indexado, es guardado en un archivo de texto como si se tratara de un modo debug para seguir la pista de lo que se ha procesado.

De la conexión a Elasticsearch, se ha preparado un pequeño test para probar la conexión y distintos indexados y querys para probar la validez de estas construcciones de cara al tránsito que ocurriría en el sitio web.

Las anotaciones de las pruebas del front-end de Django han sido más propias de usabilidad de un usuario probando las distintas páginas, viendo donde podía fallar y arreglándolo posteriormente.

Se recoge su feedback, tanto de errores encontrados, como apreciaciones de como deberían comportarse y verse la aplicación web.

Por último, las imágenes compiladas de Docker han sido probadas localmente, comportándose igual que si no estuvieran dentro de un conteiner de Docker, lanzando individuamente cada módulo dentro de Linux.

A pesar de que, una de las características de Docker es precisamente su encapsulamiento al no importar donde se ejecuta, pues solo se accedería desde un navegador web. Hemos probado en distintos ordenadores, con Docker instalado, tanto en local como en remoto y las reacciones del sitio web a los testeos han sido similares por no decir iguales.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Hemos creado un sistema que cumple en gran parte los puntos dados por el solicitante, según su idea preconcebida del programa que se necesitaba para el departamento.

El programa queda alojado en un sitio web cómodo tanto para el usuario especializado como para le normal. Uno puede hacer su trabajo, de considerar si un candidato es un posible neologismo con la información suministrada, y el otro usuario puede consultar nuevas palabras que están siendo usadas en nuestro hablar cotidiano, para aumentar nuestro rico idioma en cuanto a palabras y significados.

En la parte técnica, he disfrutado aprendiendo nuevas tecnologías y puliendo las que ya conocía. Diseñar un proyecto de este tamaño con un programa desde cero, y ver todas las etapas hasta que es funcional para su uso, es una agradable experiencia en informática.

Siempre queda la duda de poder haberle dedicado más tiempo o haber encarado de otra forma algunas decisiones, pero en general estoy satisfecho con el trabajo realizado

6.2 Trabajo futuro

Este programa quedará en manos de los integrantes del Departamento de Ingeniería Informática, y del laboratorio de *Lingüística Informática* para poder seguir desarrollándola.

Durante el proyecto se han visto posibles actualizaciones, que por falta de tiempo y equipo no se han podido acometer, algunas de estas podrían ser:

- **Implementar completamente Apache Kafka**, si la información tratada se aumenta y se dispone de más equipamiento sería recomendable la implementación de esta tecnología para mejorar la eficiencia del programa.

- **Afinar los filtros del extractor,** con la gran cantidad de datos generados se podría hacer un análisis serio de que puede faltar o que filtros mejorar para afinar el procesado de los filtros.

- **Mejorar el diseño estético,** aun con la inclusión de elementos de Bootstrap no me destaco por mi gran sentido de la estética en un sitio web, me centro más en lo funcional. Sin embargo, una aplicación quizás necesite de una mejor presencia visual.

- **Usar y añadir información,** dentro de los archivos suministrados hay mucha más información por tweet. Se podría hacer uso de ella de alguna forma. También se podría facilitar al usuario en el sitio web más información sí la precisara, incluso darle la opción de bajarse toda la información recabada.

- **Optimizar,** dependiendo de la cantidad de datos que se termine tratando dentro de la web, se debería pensar en mejorar las peticiones a Elasticsearch y la cantidad de información mostrada.

Referencias

[1] Diccionario de anglicismos del español estadounidense, por
Francisco Moreno-Fernández,
Informes del Observatorio / Observatorio Reports. 037-01/2018SP
ISBN: 978-0-692-04726-2 doi: 10.15427/OR037-01/2018SP

[2] Configuración lingüística de anglicismos procedentes de Twitter en el español estadounidense, por
Francisco Moreno Fernández
Universidad de Alcalá – Instituto Cervantes en la Universidad de Harvard
Antonio Moreno Sandoval
Universidad Autónoma de Madrid – Instituto de Ingeniería del Conocimiento

[3] Web del banco de neologismos creado por el Instituto Cervantes virtual
https://cvc.cervantes.es/lengua/banco_neologismos/listado_neologismos.asp

[4] Pagina web el castellano.org con Las sílabas del castellano
http://www.elcastellano.org/ns/edicion/2014/abril/silabas.html

[5] EU Science Hub
The European Commission's science and knowledge service
https://ec.europa.eu/jrc/en/language-technologies/jrc-names

Glosario

API	Application Programming Interface
API Restful	Es un API que usa de HTTP requests para GET, PUT, POST y DELETE data.
TFG	Trabajo de Fin de Grado
GRAMPAL	Es un analizador sintáctico creado por el *Departamento de Lingüística del Instituto de Ingeniería del Conocimiento*
NoSQL	Es un tipo de base de datos que no responde al esquema SQL, es decir sus tablas y búsquedas.
JSON	JavaScript Object Notation
UTF-8	Formato de Transformación UCS/Unicode de 8 bits
CSS	*Cascading Style Sheets*
HTML	HyperText Markup Language
Clítico	Es un elemento gramatical que se escribe como una palabra o partícula átona independiente, pero que en realidad se pronuncia como parte de la palabra anterior o siguiente.
JAR	Java Archive
SO	Sistema Operativo
SSH	Secure Shell
IP	Internet protocol

Anexos

A Manual de instalación

Describiremos como hacer uso de la aplicación creada desde una imagen de Docker ya creada (para saber cómo crear una imagen propia se explica en el Anexo B).

Lo primero es tener instalado Docker en nuestro sistema y abrir una terminal. En esta terminal ejecutamos, desde la carpeta en donde este la imagen o introducimos su ruta en el comando en vez de solo el nombre del a imagen.

docker load -i <ruta a la imagen o solo su nombre>

Docker cargara los módulos necesarios en el sistema. Se puede comprobar con

docker images -a

si la imagen ha sido cargada con éxito y el nombre que tiene.
Para desplegar la imagen en un *conteiner*

docker run -p 8000:8000 -it --name < nombre del conteiner> < nombre de la imagen>

8000:8000 es la conexión de puertos, el 8000 de dentro del a aplicación y 8000 del sistema.

El container se ejecutar automáticamente, entrando con el usuario creado y en su carpeta.
Se ejecutará el script de inicio.

Sh init.sh

Se iniciarán todos los sistemas necesarios y estará operativa la aplicación en la dirección de red asignada por el sistema huésped.

B Manual del programador

Como crear una imagen con el código que hemos creado. Tenemos que diseñar un archivo dockerfile como este:

```
# Docker file for a slim Ubuntu-based Python3 image

FROM ubuntu:latest
FROM openjdk:8

RUN apt-get update \
  && apt-get install -y python3-pip python3-dev \
  && ln -s /usr/bin/python3 python \
  && pip3 install --upgrade pip \
  && pip install -U requests \
  && pip install elasticsearch \
  && pip install Django

RUN mkdir /home/appuser
RUN chmod -R 777 /home/appuser

COPY ./django /home/appuser/django
RUN chmod -R 777 /home/appuser/django

RUN groupadd -g 999 appuser && \
    useradd -r -u 999 -g appuser appuser
USER appuser

RUN cd /home/appuser/django/neoweb \
  && curl -L -O https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-6.2.4.tar.gz \
  && tar -xvf elasticsearch-6.2.4.tar.gz

COPY init.sh /home/appuser/django/neoweb

EXPOSE 8000 9300
WORKDIR /home/appuser/django/neoweb
```

En el cogemos lo necesario la última imagen de Ubuntu, openkdk 8 e instalamos python3 con los paquetes necesarios. Creamos una carpeta con permisos y copiamos nuestro código. Creamos el usuario que pueda manipular los datos añadidos.
Descargamos la aplicación de Elasticsearch, copiamos el script de inicio de todo, abrimos los puertos necesarios hacia fuera y ponemos pro defecto iniciar en la carpeta del usuario.

Con el Dockerfile definido podemos generar ya la imagen. Abrimos una terminal y ejecutamos

docker build -t <nombre de la imagen> . (el punto final es importante)

Docker pasaría a crear la imagen con lo que la hemos definido. Se crearía dentro de Docker y para poder llevar a otro sistema bastaría con

docker save -o <ruta donde generar el archivo> <nombre de la imagen>