



UNIVERSITAT  
POLITÈCNICA  
DE VALÈNCIA



Escola Tècnica  
Superior d'Enginyeria  
Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica  
Universitat Politècnica de València

Seguimiento de eventos en la red social Twitter

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor:** Ángel Igualada Moraga

**Tutores:** Lluís Felip Hurtado Oliver

Ferran Pla Santamaría

2018-2019

# Resumen

---

Twitter se ha convertido en una herramienta popular de comunicación en la que los usuarios expresan opiniones sobre una gran cantidad de temas. La recolección y análisis de esta información ha despertado gran interés en la comunidad científica con el objetivo de extraer información relevante sobre un gran número de temas.

En este TFG se propone la creación de una aplicación que realice la captura, gestión, análisis y almacenamiento de mensajes capturados desde la API de Twitter.

La aplicación deberá permitir realizar consultas de diferente naturaleza, por temas, *hashtags*, usuarios, etc., así como buscar una manera adecuada de estructurar toda esta información.

Esto conlleva solucionar diferentes problemas impuestos por el API como el límite de consultas, número de mensajes a recuperar, control de errores, control de tweets duplicados, etc. Por otra parte, obtener los tweets en un determinado intervalo de tiempo también supone un reto para garantizar la obtención del mayor número de tweets posibles.

Respecto al almacenamiento de los tweets se propone el uso de un sistema de bases de datos no relacional como MongoDB debido a que se adapta perfectamente al formato interno de los tweets devueltos por la API, así como poder definir diferentes colecciones asociadas a los temas de interés.

La herramienta también debe permitir realizar un seguimiento histórico de las consultas que se hayan realizado, tanto utilizando la API como la web de Twitter.

Con todo esto, se podrá visualizar y monitorizar los temas de interés mediante una interfaz gráfica que presente diferente información estadística interesante.

Como caso práctico se presentará el uso de la herramienta para hacer un seguimiento de las opiniones sobre los líderes políticos atendiendo al número de ‘likes’ obtenidos de Twitter.

**Palabras clave:** Twitter , API, *tweets*, análisis, redes sociales, MongoDB.

# Abstract

---

Twitter has become a popular communication tool in which users express opinions on a large number of topics. The collection and analysis of this information has aroused great interest in the scientific community with the aim of extracting relevant information on a large number of subjects.

In this TFG we propose the creation of an application that captures, manages, analyzes and stores messages captured from the Twitter API.

The application should allow queries of different kinds, by topic, hashtags, users, etc., as well as find an appropriate way to structure all this information.

This involves solving different problems imposed by the API such as the limit of queries, number of messages to recover, error control, control of duplicate tweets, etc. Moreover, getting as many tweets as possible in a certain time interval is also a challenge.

Regarding the storage of the tweets, the use of a non-relational database system such as MongoDB is proposed because it adapts perfectly to the internal format of the tweets returned by the API, as well as being able to define different collections associated with the topics of interest.

The tool should also allow a historical tracking of the queries that have been made, both using the API and the Twitter web.

With all this, you can visualize and monitor the topics of interest through a graphical interface that presents different interesting statistical information.

As a practical case, the use of the tool will be presented to follow up the opinions on the political leaders attending to the number of '*likes*' obtained from Twitter.

**Keywords :** Twitter , API, *tweets*, analysis, social networks, MongoDB.

# Prefacio

---

Por la cercanía de las elecciones generales de 2019, como valor añadido se han desarrollado opciones y módulos para el análisis específico de usuarios de un partido así como un módulo para obtener datos que no nos provee el API de Twitter, en concreto, se ha realizado un sistema automático que captura quien ha dado *like* a cierto *tweet*, debido a que en la web solo podemos ver los 25 últimos usuarios que dieron *like* a una publicación, este proceso se ejecutará en bucle intentando obtener el máximo número de *likes*. Por último, también se ha añadido una sección de visualización de datos únicamente políticos.

# Tabla de contenidos

---

1.	Introducción .....	7
2.	Objetivos.....	7
3.	Tecnología utilizada.....	8
4.	Arquitectura del sistema .....	10
	Arquitectura de la aplicación principal.....	10
	Arquitectura de la aplicación de visualización .....	11
5.	Diseño detallado .....	11
	Diseño detallado aplicación principal.....	11
	Diseño detallado aplicación de visualización .....	12
6.	Desarrollo de la solución .....	16
	Desarrollo de la aplicación principal. ....	16
	Desarrollo de la aplicación visualización.....	32
7.	Caso de estudio.....	39
8.	Conclusiones .....	46
	Recopilación de <i>tweets</i> .....	46
	Adición de información no proveniente del API. ....	46
	Gestión del almacenamiento .....	46
	Ejecuciones no acotadas con recuperación tras error .....	46
	Visualización.....	46
9.	Relación del trabajo desarrollado con el grado cursado .....	47
10.	Referencias .....	48
11.	Anexos .....	49
	Anexo 1: Volcado de variables al fichero de estadísticas. ....	49
	Anexo 2: Carga del fichero de estadísticas.....	50
	Anexo 3: Modos y opciones de la aplicación principal. ....	51
	Opciones adicionales línea de comandos .....	52
12.	Glosario.....	54

# Índice de imágenes

---

Figura 1: Arquitectura simplificada de la aplicación principal .....	10
Figura 2 : Arquitectura simplificada de la aplicación de visualización .....	11
Figura 3 : Diseño detallado de la aplicación principal .....	12
Figura 4 : Estructura de directorios de la aplicación principal .....	12
Figura 5 : Diseño detallado de la aplicación de visualización 1.....	13
Figura 6 : Diseño detallado de la aplicación de visualización 2 .....	14
Figura 7 : Estructura de directorios de la aplicación de visualización .....	15
Figura 8: Tratamiento previo a la inserción en MongoDB 1 .....	16
Figura 9 : Ejemplo de mensaje recuperado del API de Twitter.....	17
Figura 10 : Ejemplo de mensaje de error del API de Twitter .....	17
Figura 11 : Tratamiento previo a la inserción en MongoDB 2 .....	18
Figura 12 : Ejemplo de registro del fichero de búsquedas por consulta.....	19
Figura 13 : Ejemplo de registro de fichero de búsquedas por streaming.....	19
Figura 14 : Ejemplo de registro de fichero de búsquedas por usuario .....	20
Figura 15 : Ejemplo de registro de fichero de likes .....	21
Figura 16 : Ejemplo de registro de fichero de conteo de likes .....	22
Figura 17 : Ejemplo de URL de un tweet .....	22
Figura 18 : Ejemplo de tweet .....	22
Figura 19 : Sección de like de Twitter.....	23
Figura 20 : Recuperación de los 20 mensajes más recientes de un usuario desde MongoDB.....	23
Figura 21 : Detalle del proceso de likes (versión 1) .....	24
Figura 22 : Estructura de datos del proceso de likes.....	25
Figura 23 : Detalle del proceso de likes (versión 2) .....	26
Figura 24 : Detalle proceso de likes (versión 3) .....	27
Figura 25 : Detalle del proceso de likes (versión 4) .....	28
Figura 26 : Detalle del proceso de like (versión 5) .....	29
Figura 27 : Tratamiento previo a inserción en MongoDB 3 .....	30
Figura 28: Cambio en el registro del fichero de usuarios .....	31
Figura 29 : Funcionamiento de la aplicación de visualización básica.....	32
Figura 30 : Menú de elección de la colección .....	33
Figura 31 : Ciclo de paso de variables.....	33
Figura 32 : Gráficos generados por la aplicación de visualización.....	34
Figura 33 : Ejemplo de tabla de la aplicación visualización.....	34
Figura 34 : Obtención del código de embebido desde la web de Twitter .....	35
Figura 35 : Tag HTML de obtención de la foto perfil Twitter .....	35
Figura 36 : Ejemplo ranking de tweets.....	36
Figura 37 : Ejemplo ranking de tweets activo .....	36
Figura 38 : Ejemplo de la visualización del ranking de usuarios .....	36
Figura 39 : Visualización de la sección documentos especiales .....	37
Figura 40 : Tabla de monitoreo de la recopilación de likes .....	37
Figura 41 : Visualización de los resultados de la búsqueda.....	38
Figura 42: Visualización de los ficheros de conteo de likes .....	38
Figura 43 : Comando para la obtención de los procesos activos.....	39
Figura 44: Ejemplo de cabecera de la sección de usuario .....	40

Figura 45: Simulador de votos .....	41
Ilustración 46: Distribución de los resultados de las elecciones generales de 2019 .....	41
Ilustración 47: Distribución de votantes en Simulaciones sin filtros.....	42
Ilustración 48: Distribución de los likes capturados por partido .....	43
Ilustración 49: Distribución de tweets capturados por partido .....	43
Ilustración 50: Ratio likes partido tweets .....	43
Ilustración 51: Distribución de los votantes antiguos .....	44
Ilustración 52: Distribución de los votantes antiguos verificados .....	44
Ilustración 53: Distribución votantes con al menos 10 likes capturados.....	45

# 1. Introducción

---

En este trabajo, se describe el proceso llevado a cabo para el desarrollo de una aplicación que aúna la captura, gestión, análisis y almacenamiento de mensajes capturados utilizando el API que provee Twitter.

En primer lugar, se explicará porque Twitter es una red social idónea para la obtención de datos a través de su API, cómo podemos interactuar con la misma y cómo serán las respuestas que obtengamos.

Teniendo esa primera aproximación, podremos valorar por qué se ha elegido MongoDB como sistema de persistencia y en qué nos facilitará la creación de la aplicación.

Partiendo de esto, se detallará cómo funcionan los distintos procesos de la aplicación y cómo se realiza la visualización en la aplicación web de los datos que almacenamos en MongoDB.

## 2. Objetivos

---

Este trabajo tiene cinco objetivos principales: recopilación de tweets, adición de información no proveniente del API, gestión del almacenamiento, ejecuciones no acotadas con recuperación ante error y visualización de los datos almacenados.

En cuanto a la recopilación de tweets, debe ser posible la captura de *tweets* dada una consulta, dado uno o más usuarios y la captura de *tweets* en *streaming* dado un conjunto de palabras clave. Se deben controlar los límites establecidos por el API y se debe controlar la llegada de mensajes de error del API.

En lo referente a la adición de información no proveniente del API, se debe permitir la captura de los usuarios que dieron *like* a un mensaje. Esta información no está disponible desde el API y debe encontrarse la forma de obtenerla y almacenarla.

La aplicación debe gestionar el almacenamiento de los datos; debe garantizar un control de duplicados, permitir el almacenamiento en distintos conjuntos o colecciones, y debe brindar la posibilidad de guardar en fichero. También debe permitir actualizar los mensajes recuperados y proporcionar para cada colección:

- Almacenamiento de consultas lanzadas por colección.
- Almacenamiento de búsquedas lanzadas por *streaming* por colección.
- Almacenamiento de usuarios buscados por colección.
- Posibilidad de relanzar cualquiera de los conjuntos de búsqueda anteriores con un límite prefijado con lo ya capturado.

La aplicación permitirá realizar ejecuciones no acotadas (sin límite de tiempo) con recuperación tras error. Además, se hace necesaria la ejecución sin interacción del usuario durante largos periodos de tiempo (semanas) y ante posibles errores, el proceso debe ser capaz de continuar o reiniciarse por sí mismo.

En cuanto a la visualización, se debe proporcionar una visualización de estadísticas por colección, así como la posibilidad de visualizar todos los documentos de una colección con opción de búsqueda y los datos genéricos de los usuarios.

### 3. Tecnología utilizada

---

En primer lugar, se ha elegido el lenguaje de programación en el que se va a realizar la aplicación. Se ha elegido Python por su claridad, su gran número de módulos implementados y para garantizar la integración con otras aplicaciones existentes en el grupo de investigación.

Habiendo elegido el lenguaje de programación, se ha elegido tweepy como módulo de conexión con el API de Twitter desde Python entre las múltiples opciones existentes para este lenguaje. Tweepy será el nexo de unión entre nuestro programa Python y el API de Twitter permitiéndonos recopilar información de esta red social.

Tras el análisis de la interacción con el API de Twitter mediante Python, se ha elegido una base de datos no relacional, en concreto MongoDB como sistema de persistencia ya que las respuestas del API son documentos json y este es el tipo de documento que MongoDB usa internamente para almacenar información. Además, MongoDB usa un id interno para cada documento para identificarlo, con fijar ese id con el id de cada *tweet* garantizaremos el control de duplicados de forma transparente. El módulo pymongo será el encargado de conectar Python con MongoDB.

Con estas tecnologías, se puede implementar la interacción con el API de Twitter y su almacenamiento, pero para poder obtener quien dio *like* a un determinado *tweet* necesitaremos interactuar directamente con la web de Twitter con otros módulos:

- Selenium: Este módulo Python nos permitirá realizar un sistema automático que utilice un navegador para acceder a las URLs de los usuarios y consultar quien dio *like* a un *tweet*.
- BeautifulSoup: Este módulo Python nos permitirá obtener los html de las webs que deseemos, pero no podremos interactuar con ellas, es más rápido que Selenium pero no nos proporciona toda la información.

Para realizar la parte gráfica se ha elegido realizar una aplicación web.



La parte del servidor se implementará en Python, usando el módulo Flask que nos servirá para asignar las URL que queremos utilizar y que queremos que nos muestre.

Además, Flask nos permitirá pasar variables a HTML.

En cuanto a la parte web, usaremos HTML5, CSS3 y Javascript.

Cabe destacar que excepto Python, se ha tenido que estudiar el uso de todas las tecnologías.

## 4. Arquitectura del sistema

Se ha decidido realizar dos aplicaciones independientes para lograr los objetivos propuestos, la aplicación principal alberga toda la lógica de la obtención, gestión, análisis y almacenamiento de la información mientras que la segunda se centra en la visualización de los datos obtenidos.

### Arquitectura de la aplicación principal

La aplicación principal deberá interactuar con la web de Twitter para obtener información de los usuarios que dieron *like* a cierto *tweet*, con el API de Twitter para obtener *tweets* y además, con la base de datos para insertar modificar o eliminar documentos. Por tanto, como se muestra en la Figura 1, necesitaremos un consumidor de la web de Twitter, un consumidor del API y un módulo que interactúe con la base de datos.

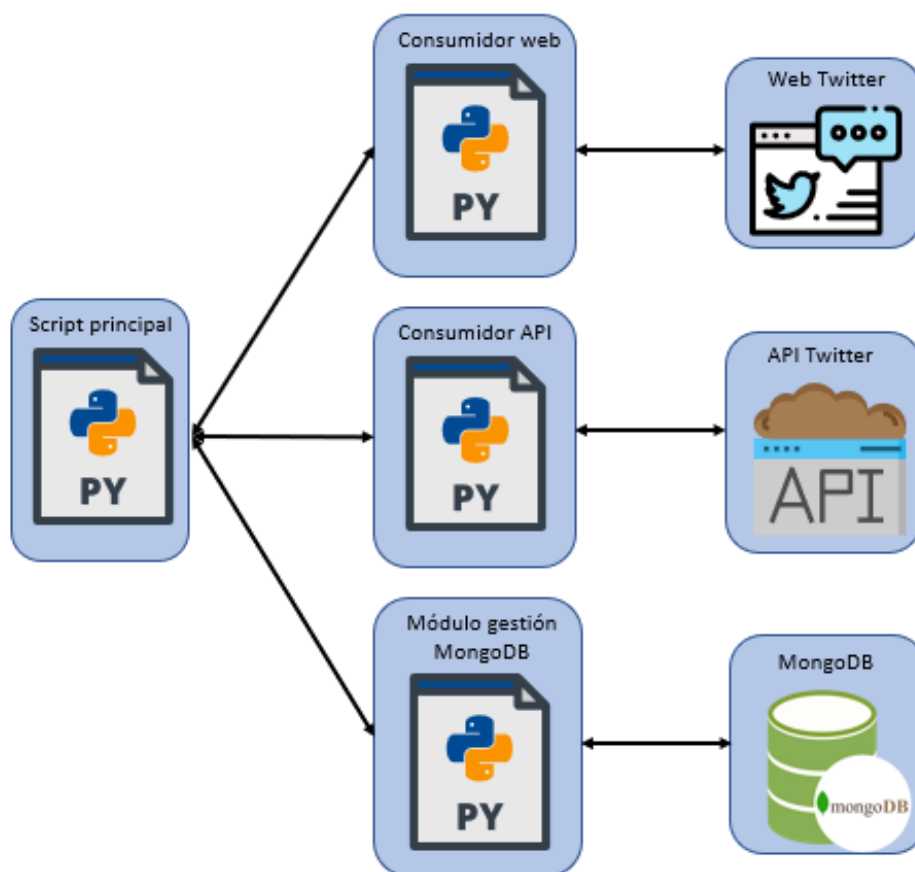


Figura 1: Arquitectura simplificada de la aplicación principal

## Arquitectura de la aplicación de visualización

La aplicación de visualización deberá hacer consultas a MongoDB en función de lo que el usuario seleccione en la aplicación web.

Como podemos observar en la Figura 2, el programa Python se comunicará con la parte web mediante Flask y la parte web se comunicará con el programa Python mediante peticiones HTTP GET (obtener una URL) o POST (enviar la colección en la que se están realizando las acciones).

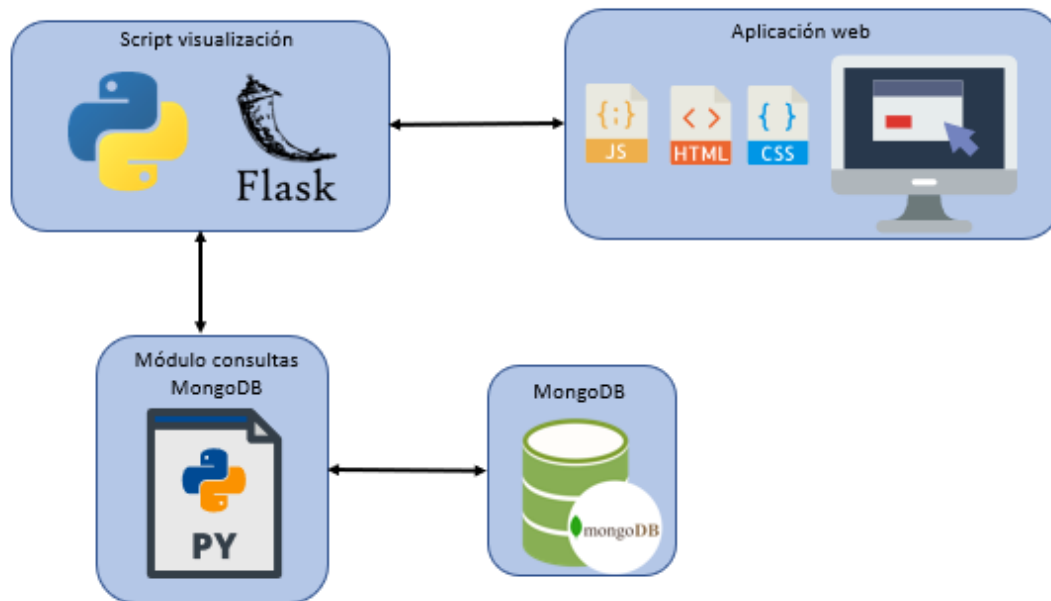


Figura 2 : Arquitectura simplificada de la aplicación de visualización

## 5. Diseño detallado

### Diseño detallado aplicación principal

La aplicación principal cuenta con 6 módulos Python:

- `main_script.py` : Es el script principal, cuenta con múltiples opciones por línea de comandos para poder ejecutar los distintos subprocesos que forman la aplicación según se requiera.
- `global_variables.py` : Es el módulo que almacena las variables globales, con este módulo se realiza la carga y descarga de las estadísticas de una colección.
- `global_functions.py` : Es el módulo que almacena funciones comunes a todos los demás módulos.
- `twitter_web_consumer.py` : Es el módulo que interactúa con la web de Twitter. Tiene métodos que interactúan con ella mediante un navegador (mediante el módulo Selenium) y otros que lo hacen sin necesidad de este (mediante el módulo BeautifulSoup). Nos ofrece funcionalidad para obtener los

identificadores de los últimos *tweets* o retweets de un usuario hasta una fecha o un *tweet\_id*, la obtención de los últimos usuarios que dieron *like* a un *tweet* desde distintas aproximaciones entre otros.

- `twitter_api_consumer.py` : Es el módulo que interactúa con el API de Twitter, para ello utiliza el módulo `tweepy`. Nos proporciona funcionalidad para la captura de *tweets* por distintos medios o tipos de búsqueda entre otras.
- `mongo_conector.py` : Nos proporciona la funcionalidad necesaria para interactuar con MongoDB (mediante `pymongo`) y gestionar las distintas colecciones así como sus ficheros especiales ( ficheros de estadísticas, de consultas lanzadas ...).

En la Figura 3 se puede observar la arquitectura detallada de la aplicación principal.

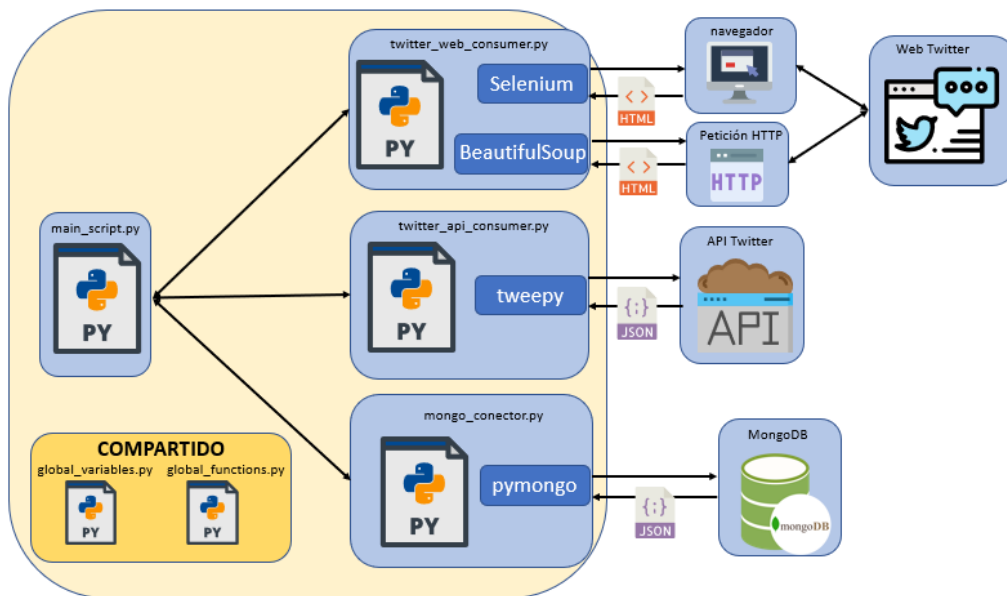


Figura 3 : Diseño detallado de la aplicación principal

La estructura de directorios de esta aplicación será la que se muestra en la Figura 4:

```

main_code
├── __pycache__
├── global_functions.py
├── global_variables.py
├── main_script.py
├── mongo_conector.py
├── twitter_api_consumer.py
└── twitter_web_consumer.py

```

Figura 4 : Estructura de directorios de la aplicación principal

## Diseño detallado aplicación de visualización

La aplicación de visualización contará con 2 módulos Python:

- `__init__.py` : Este módulo permite levantar el servidor web mediante Flask. Será en este módulo donde fijemos que URLs podrán ser utilizadas y que HTML

estará ligado a cada una de ellas. A estos ficheros HTML, como se observa en la Figura 5, les podremos pasar variables Python (que más tarde podremos pasar a sus scripts en JavaScript), y será así como se enviará la información recuperada de MongoDB a la parte web. Por otra parte, estableceremos varios *endpoints*, es decir, rutas en las que recibiremos información desde la parte web pero en las que no devolveremos nada. Estos *endpoints* recibirán información como cuál es la colección para la que el usuario está visualizando datos ahora y en función de esto obtendremos los datos requeridos de MongoDB.

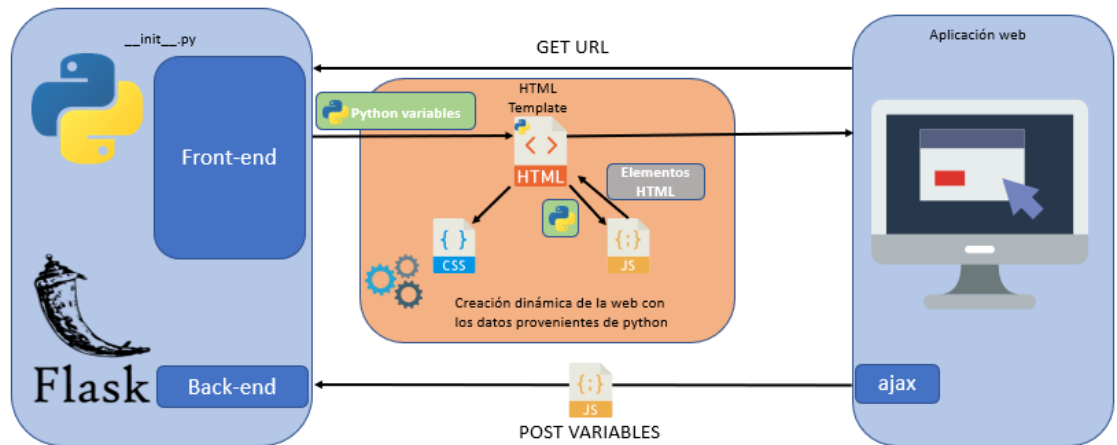


Figura 5 : Diseño detallado de la aplicación de visualización 1

- `mongo_conector_for_flask.py` : Este módulo es una versión reducida del módulo de la aplicación principal ya que en este solo están los métodos de consulta y no de inserción, modificación o borrado. Como se observa en la Figura 6, será el nexo de unión entre el programa y la base de datos.

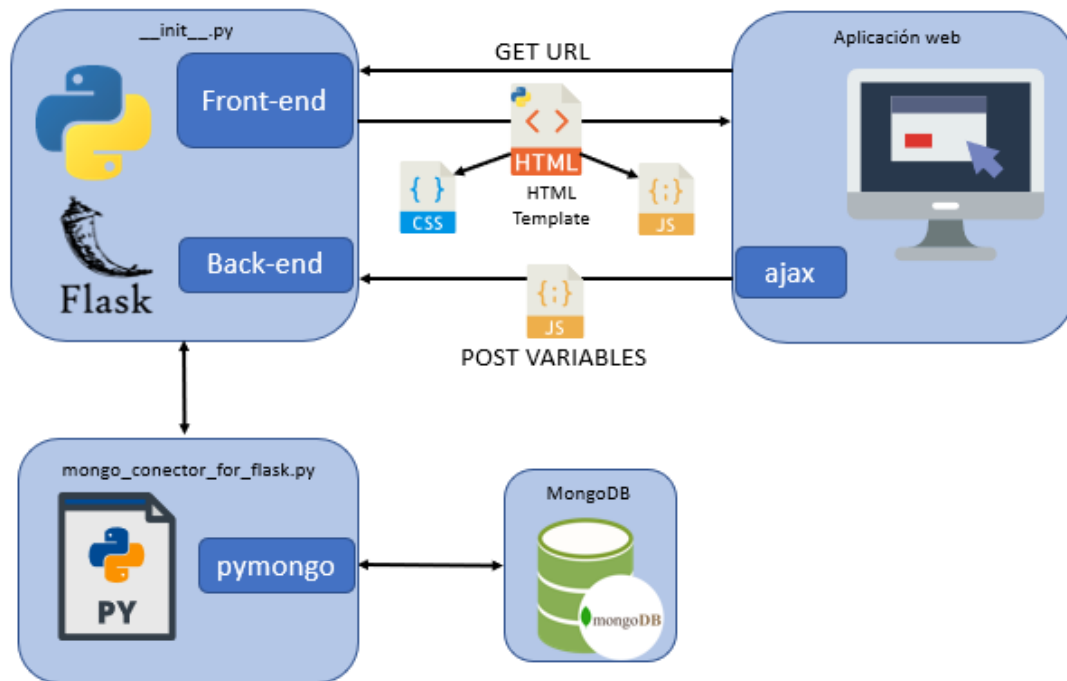
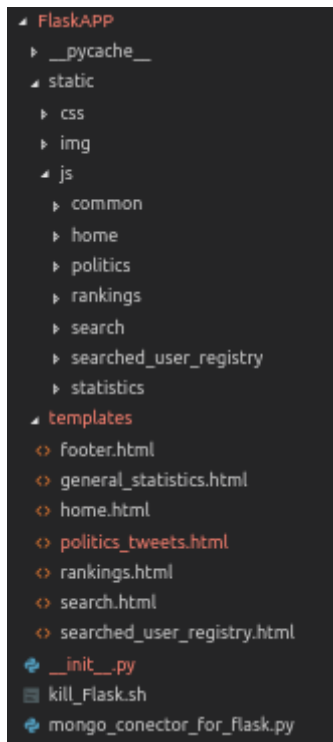


Figura 6 : Diseño detallado de la aplicación de visualización 2

Además del código Python tendremos 6 ficheros HTML (uno para cada sección de la página web), múltiples CSS y múltiples *scripts* :

- `home.html` : Este fichero, muestra qué ficheros existen para la colección actual y nos permitirá visualizarlos. Los ficheros existirán o no en función de si se ha ejecutado alguna vez su proceso en esa colección (fichero de consultas por usuario, fichero de búsquedas por *streaming* etc).
- `general_statistics.html` : Este fichero contiene la vista en la que se visualizarán las estadísticas de una colección.
- `rankings.html` : Dado que en el proceso de análisis se generarán adicionalmente algunos rankings, como los *tweets* con más *likes*, se ha decidido que estos se recojan en una única sección y este fichero será el que la contenga.
- `search.html` : Este fichero contiene la búsqueda por *tweet* id mediante expresión regular. Así se podrá visualizar el contenido de los ficheros json de cada *tweet* desde la web.
- `politics_tweets.html` : Por el caso de estudio de este trabajo, se ha añadido una sección específica para la visualización de *tweets* de políticos en este fichero.
- `searched_user_registry.html`: Muestra información de cada usuario para el cual se han recuperado tweets por consulta, muestra como son los usuarios que dieron *like* a sus tweets y cuantos *likes* dieron.

La Figura 7 muestra la estructura de directorios de la aplicación de visualización:



*Figura 7 : Estructura de directorios de la aplicación de visualización*

## 6. Desarrollo de la solución

Debido a la complejidad tanto de la aplicación principal como de la aplicación de visualización, se realizó un desarrollo iterativo con múltiples versiones que se detalla a continuación.

### Desarrollo de la aplicación principal.

El primer paso en el desarrollo fue la petición de uso al API de Twitter para obtener las claves necesarias para el uso de esta.

#### Versión 1

Habiendo obtenido las claves, se implementó una versión simple de consumidor del API por consulta que guardaba los *tweets* en ficheros json y analizaba los *tweets* recuperados generando unas estadísticas. En esta simple versión ya se detectó alguna casuística a tener en cuenta:

- Respuestas del API con códigos de error.
- Obtención de *tweets* duplicados por distintos motivos.
- Necesidad de almacenar las estadísticas.

#### Versión 2

En la siguiente versión, se implementó la posibilidad de guardar los *tweets* en MongoDB o en un fichero. Para guardar los *tweets* en MongoDB, simplemente se realizaba un tratamiento sencillo que consistía en añadir un nuevo campo al diccionario Python del *tweet*. Este campo es el que MongoDB usará para identificar el documento (`_id`).

A este campo le fijaremos el valor del id del *tweet* devuelto por el API y así garantiremos que no hay duplicados. Así pues, en la Figura 8 se muestran el campo añadido al tweet antes de su almacenamiento.

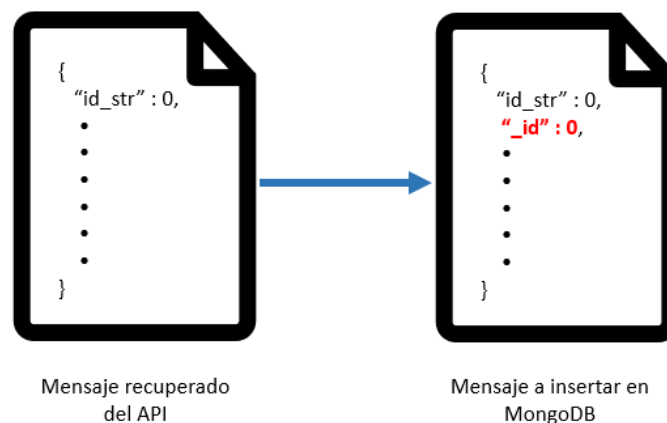


Figura 8: Tratamiento previo a la inserción en MongoDB 1

Para no insertar *tweets* erróneos, mensajes de error o mensajes informando de que se alcanzó el límite de consultas, se añadió un tratamiento basado en la existencia del



campo “id\_str” devuelto por el API, ya que solo los mensajes correctos lo tienen y es el usado para identificar un *tweet*, así el consumidor del API solo nos devolverá mensajes correctos. La Figura 9 y 10 nos muestran un ejemplo de mensaje recuperado del API correctamente y un ejemplo de mensaje de error.

```
{
  "id_str": "54691802283900928",
  "text": "RT @PostGradProblem: In preparation for the NFL lockout [...]",
  "truncated": true,
  "in_reply_to_user_id": null,
  "in_reply_to_status_id": null,
  "favorited": false,
  "source": "<a href=\"http://twitter.com/\" rel=\"nofollow\">Twitter for iPhone</a>",
  "in_reply_to_screen_name": null,
  "in_reply_to_status_id_str": null,
  "entities": {
  },
  "contributors": null,
  "retweeted": false,
  "in_reply_to_user_id_str": null,
  "place": null,
  "retweet_count": 4,
  "created_at": "Sun Apr 03 23:48:36 +0000 2011",
  "retweeted_status": {
  },
  "user": {
  },
  "id": 54691802283900930,
  "coordinates": null,
  "geo": null
}
```

Figura 9 : Ejemplo de mensaje recuperado del API de Twitter

```
{
  "limit": {
    "timestamp_ms": "1556655179774",
    "track": 133
  }
}
```

Figura 10 : Ejemplo de mensaje de error del API de Twitter

También, se añadieron las opciones para poder analizar ficheros, directorios o directorios de directorios y generar sus estadísticas.

En esta nueva versión se detectaron los siguientes problemas:

- Errores de escritura en MongoDB al intentar insertar un documento con un “\_id” que ya estaba en la colección.
- Necesidad de almacenar las estadísticas.

## **Versión 3**

En la versión 3 de la aplicación:

- Se implementó la captura de *tweets* por usuario y mediante *streaming* (sobreescribiendo la clase StreamListener proporcionada por tweepy ).
- Se separó el proceso de análisis de *tweets* de los de recopilación y se cambió el proceso de análisis para tratar de forma diferente los *tweets* actualizados pero ya analizados.

- Se añadió la posibilidad de ejecutar todas las consultas que se hayan realizado en una colección.
- Se añadieron tres nuevos campos adicionales en el tratamiento de los mensajes antes de su inserción:
  - “*first\_capture*” : fecha y hora de la primera recopilación.
  - “*last\_update*” : fecha y hora de la última actualización.
  - “*analyzed*” : Campo que informa de si un *tweet* ha sido analizado, puede tener 3 valores:
    - True : Ha sido analizado y no hay cambios.
    - False: No ha sido analizado.
    - “*to update*” : Fue analizado pero ha sido actualizado por lo que hay que actualizar algunas estadísticas.

En la Figura 11, se muestran los campos añadidos al tweet recuperado del API.

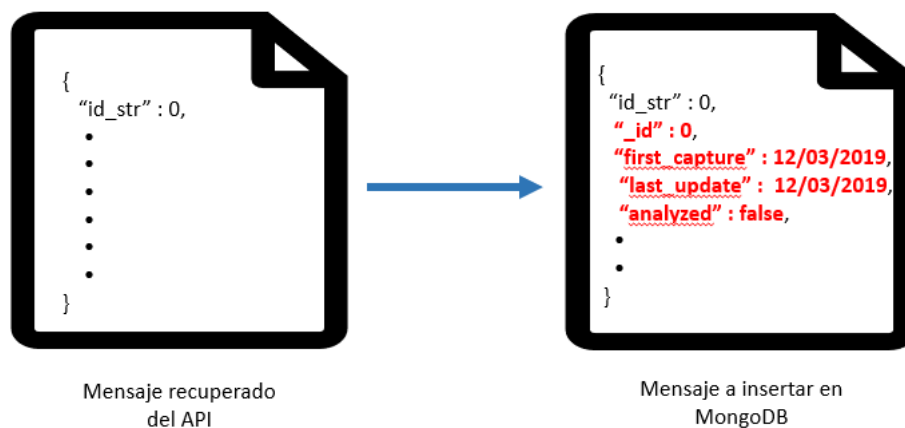


Figura 11 : Tratamiento previo a la inserción en MongoDB 2

- Se cambió la inserción en MongoDB, pasó de ser una inserción a nivel de fichero a ser una inserción múltiple. En esta inserción en primer lugar se obtienen los identificadores de la colección y con ellos se comprueba si los documentos que se intentan insertar están ya en la colección, si es así, no serán insertados. Por último, se devuelve la lista de los que sí han sido insertados.
- Se añadió la funcionalidad de actualización de *tweets* en una colección, para ello se implementó la recopilación de *tweets* del api dado su *tweet\_id*.

Además, se añadió la funcionalidad con los 4 primeros ficheros especiales:

- Fichero de consultas: Su identificador es : “*query\_file\_id*”. Almacena la información referente a las búsquedas de *tweets* por consulta. Para cada consulta se almacenan el máximo y mínimo *tweet\_id* recopilado y la máxima y mínima fecha de creación de los *tweets* recopilados para esa *query* con el fin de poder reejecutar la consulta a partir de los *tweets* ya capturados. En la Figura 12, se muestra un ejemplo de registro de este fichero.

```
{
  "_id": "query_file_id",
  "#coche": {
    "max_tweet_id": "1108410413283246085",
    "query": "#coche",
    "max_creation_date": "Wed Mar 20 16:50:10 +0000 2019",
    "min_creation_date": "Wed Mar 20 16:50:10 +0000 2019",
    "captured_tweets": 1,
    "min_tweet_id": "1108410413283246085",
    "last_execution": "2019-03-20 18:36:18.096564",
    "search_type": "tweets captured by query"
  }
}
```

Figura 12 : Ejemplo de registro del fichero de búsquedas por consulta

- Fichero de consultas por *streamming*. Su identificador es: "*streamming\_file\_id*".  
Almacena la información referente a las búsquedas por usuario.  
Debido a que la búsqueda por *streamming* se hace para un conjunto de palabras, para cada lista de palabras en minúsculas y ordenadas alfabéticamente se almacena información referente al primer y último *tweet* capturado.  
En la Figura 13, se muestra un ejemplo de registro de este fichero.

```
{
  "_id": "streamming_file_id",
  "coche,futbol,juego": {
    "captured_tweets": 6,
    "last_execution": "2019-05-02 21:54:39.245991",
    "max_creation_date": "Thu May 02 19:54:34 +0000 2019",
    "max_tweet_id": "1124039495626764292",
    "min_creation_date": "Thu May 02 19:54:32 +0000 2019",
    "min_tweet_id": "1124039489830162432",
    "search_type": "tweets captured by streamming",
    "words": [
      "coche",
      "futbol",
      "juego"
    ],
    "words_comprobatation": "coche,futbol,juego"
  },
  "total_captured_tweets": 6
}
```

Figura 13 : Ejemplo de registro de fichero de búsquedas por *streamming*

- Fichero de consultas por usuario. Su identificador es: "*searched\_users\_file\_id*".  
Almacena la información de los usuarios para los cuales se han capturado *tweets* a partir de su nombre de usuario.  
En este documento, para cada usuario, también podremos obtener si pertenece a algún partido político si en su primera búsqueda se etiquetó como miembro.  
En la Figura 14, se muestra un ejemplo de registro de este fichero.

```
{
  "_id": "searched_users_file_id",
  "albert_rivera": {
    "captured_tweets": 2000,
    "last_execution": "2019-04-24 20:18:44.445842",
    "max_creation_date": "Wed Apr 24 17:52:20 +0000 2019",
    "max_tweet_id": "1121109632628535296",
    "min_creation_date": "Wed Apr 24 14:43:48 +0000 2019",
    "min_tweet_id": "1022007618020298753",
    "partido": "CIUDADANOS",
    "search_type": "tweets captured by user",
    "user": "albert_rivera",
    "user_id": "108994652"
  },
  "total_captured_tweets": 2000
}
```

Figura 14 : Ejemplo de registro de fichero de búsquedas por usuario

- Fichero de estadísticas. Su identificador es: "statistics\_file\_id". Guarda la información de las estadísticas de los *tweets* analizados. En el proceso de análisis el primer paso es recuperar este documento de la colección si existe y volcar sus valores a las variables de `global_variables.py` mediante reflexión para continuar con el análisis con esas variables. Cuando el análisis o el *trunk* especificado de este termina se crea un diccionario con las variables y se guarda en MongoDB. La carga y descarga de variables se explica con más detalle en los Anexos 1 y 2.

En esta versión se detectaron los siguientes problemas:

- Errores al guardar el fichero de estadísticas en MongoDB. Estos errores se debían a que se guarda un diccionario con las formas de envío de los mensajes cuyas claves pueden contener puntos y MongoDB no permite claves con puntos ya que en su sintaxis esto implica que se referencia a un elemento interno. Este error se corrigió cambiando los puntos por “*bullets*” antes de la inserción y deshaciendo el cambio en la carga.
- Alcance del límite de peticiones en *streamming*. Este error se corrigió de forma que cuando detecta estos límites se mata el proceso. Además, se añadió la posibilidad de establecer un número máximo de mensajes a recopilar o un tiempo máximo de ejecución.
- En el proceso de actualización se detectaron resultados extraños cuando el *tweet* había sido eliminado de Twitter y se mantuvo la versión de la colección.

#### Versión 4

En esta versión se añadieron las primeras pruebas de consumo de la web de Twitter con Selenium.

Se realizó el primer sistema automático de consumo que abría un navegador e iniciaba sesión en Twitter, para esto, se analizó la estructura del html de la url de inicio de sesión de Twitter.

En este punto se detectó un problema aislado, ya que al cargar la url de inicio de sesión de Twitter, en ocasiones se cargaba la versión móvil por lo que el html no tenía los mismos elementos y el sistema automático no encontraba los campos esperados y fallaba. Esto se resolvió añadiéndole la posibilidad de reintentar hasta 5 veces.

Tras esto, se analizó la estructura html de las páginas de los usuarios y se implementó el recorrido de las urls de los usuarios buscados de una colección.

Se implementaron varios métodos que permitían obtener n mensajes de un usuario o recorrer su perfil hasta una fecha o un *tweet\_id*, eso sería la base del futuro proceso de recopilación de *likes*.

En esta versión se detectaron los siguientes problemas:

- Problemas de interacción con la web: Esto era debido a que gran parte de los elementos que contenían información útil no eran directamente accionables y se debía interactuar con elementos superiores. Debieron ser insertadas algunas esperas ya que el intentar realizar acciones antes de que la web cargará llevaba a errores.

### Versión 5

En esta versión se implementó el primer proceso de *likes* y se añadió la posibilidad de la ejecución de las consultas de una colección en bucle (ininterrumpidamente), esto sería de especial utilidad para intentar capturar nuevos *tweets* de los usuarios buscados de una colección.

Para ello se crearon dos nuevos ficheros especiales:

- Fichero de *likes* capturados por *tweet*. Su identificador es: "*like\_list\_file\_id*". Contiene información de los *tweets* para los que se ha capturado algún *like*, contiene un diccionario con los usuarios que han dado *like*, el número de *likes* que tenía el *tweet* cuando se consultó y el número de *likes* capturados. En la Figura 15 se observa un ejemplo de registro de este fichero.

```
{
  "1039563032815386624": {
    "last_like_registered": "2019-04-16 23:03:23",
    "num_likes": "139",
    "num_likes_capturados": 9,
    "tweet_id": "1039563032815386624",
    "user_id": "1358849804",
    "user_screen_name": "maribelmorag",
    "users_who_liked": {
      "1056236993275985920": {
        "user_id": "1056236993275985920",
        "user_name": "QueenLatifah /\u2764",
        "user_screen_name": "akkari_latifa"
      },
      "1087384719648526337": {
        "user_id": "1087384719648526337",
        "user_name": "@terehm",
        "user_screen_name": "terehm3"
      }
    }
  }
}
```

Figura 15 : Ejemplo de registro de fichero de likes

- Fichero de conteo de *likes* por partido. Su identificador es: "*likes\_list\_file\_id*". Contiene todos los usuarios que han dado algún *like* y cuantos *likes* han dado a cada uno de los partidos contemplados en la aplicación (PP, PSOE, Podemos,

Ciudadanos, Compromís y Vox). La Figura 16 muestra un ejemplo de fichero de conteo de *likes*.

```
{
  "1011533705863663616": {
    "last_like_registered": "2019-04-24 17:23:16.166044",
    "likes_to_CIUADANOS": 1,
    "likes_to_COMPROMIS": 0,
    "likes_to_PODEMOS": 0,
    "likes_to_PP": 0,
    "likes_to_PSOE": 0,
    "likes_to_VOX": 0,
    "tweet_ids_liked_list": [
      "1119184693063077889"
    ],
    "user_id": "1011533705863663616",
    "user_screen_name": "JuanitoBlanco7"
  }
}
```

Figura 16 : Ejemplo de registro de fichero de conteo de likes

Para obtener información de un determinado *tweet*, se carga la url con su usuario y el *tweet\_id*. Dicha URL tiene la estructura mostrada en la Figura 17.

<https://twitter.com/usuario/status/1123514993536917504>

tweet id

Figura 17 : Ejemplo de URL de un tweet

Twitter limita la información relacionada con quién dio *like* a que mensaje. Hay dos formas de visualizar los últimos usuarios que dieron *like* a un *tweet*, directamente en la URL del *tweet* podemos obtener los últimos 9 usuarios que dieron *like* a ese *tweet*. En la Figura 18 se señala la posición de esta información en el *tweet*.



Figura 18 : Ejemplo de tweet

Pero accionando un botón podemos obtener los últimos 25 *likes* (es necesario accionar el botón ya que no hay otra forma de acceder ya que sigue siendo la misma URL). Se puede observar cómo obtener esta información en la Figura 19.



Figura 19 : Sección de like de Twitter

En esta versión se decidió seguir utilizando Selenium y mediante el navegador cargar las páginas de los *tweets* que teníamos en nuestra colección y tras accionar el botón obtener los últimos 25 usuarios que dieron *like* a ese *tweet*, aunque eso conllevara un aumento del tiempo.

Dada la importancia del tiempo en este proceso, ya que se podría considerar un proceso en semi tiempo real, se buscó con esta aproximación maximizar el número de *likes* recopilados ya que con la llegada de nuevos *likes*, los anteriores no se podrían obtener.

Como muestra la Figura 20, el proceso de *likes* en este punto obtenía el fichero de la colección si existía y el fichero de usuarios buscados y para cada usuario recuperaba sus 20 últimos mensajes (mayor identificador).

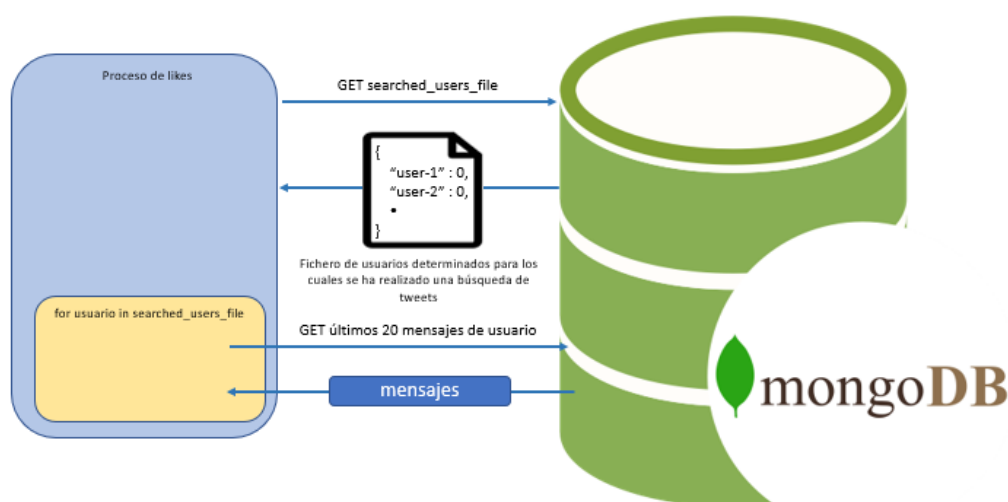


Figura 20 : Recuperación de los 20 mensajes más recientes de un usuario desde MongoDB

Para cada mensaje recuperado de cada usuario debe cargar su url y capturar los últimos 25 usuarios que dieron *like*. Tras esto, se comprobará si el *tweet* ya está en el fichero de *likes*, en caso afirmativo comprobará si los usuarios que dieron *like* ya están en él y en caso contrario se añadirían todos. Para los nuevos *likes*, si el usuario propietario del



*tweet* para el cual hemos capturado *likes* tiene un partido asignado deberían aumentarse los contadores de *likes* por partido y actualizar el fichero de *likes* por partido en MongoDB. Se puede observar un detalle de este proceso en la Figura 21.

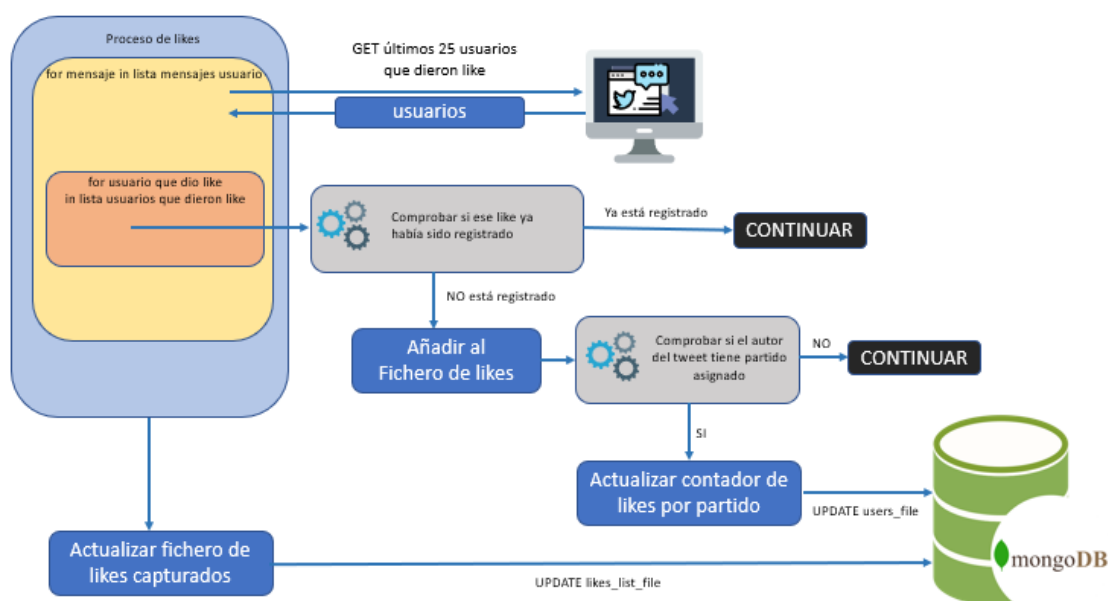


Figura 21 : Detalle del proceso de likes (versión 1)

También se creó una colección nueva en la que se capturaron los últimos 20 *tweets* de 10 políticos de los 6 partidos elegidos (PP, PSOE, Podemos, Ciudadanos, Compromís y Vox) y se dejó en ejecución la captura de *tweets* de estos usuarios para obtener nuevos *tweets*.

En esta versión se detectaron los siguientes problemas:

- Se detectó que se tardaba varias horas en recorrer todos los *tweets* por lo que se perdía una gran cantidad de *likes*.
- Se detectó que se recorrían *tweets* aunque no se hubieran obtenido nuevos *likes* en la última iteración.
- Se detectó que no había garantía de que si los usuarios hacían nuevos *tweets*, se añadieran a la colección.
- Se detectó que se hacían demasiadas escrituras en Mongo para actualizar los contadores de *likes* por partido.

### Versión 6

En esta versión se implementó la primera versión de la aplicación de visualización y se modificó el proceso de obtención de *likes*.

El principal objetivo de esta versión fue aumentar la cantidad de *likes* capturados intentando comprobar los *tweets* más relevantes.

El criterio para determinar si un *tweet* es relevante o no se estableció de la forma siguiente, se decidió que un *tweet* sería relevante cuando reportara al menos 30 nuevos *likes* en 30 minutos. Así pues, para cada *tweet* se comprobaba si habían transcurrido



30 minutos desde la última comprobación y en caso afirmativo si se habían alcanzado los 30 *like* se resetearía el contador y el temporizador. Si no se hubiera alcanzado la meta de 30 *like* el *tweet* sería eliminado.

Partiendo de esta idea básica, se creó una estructura de datos específica para este proceso y esta tarea.

Esta estructura, como podemos observar en la Figura 22, se basa en un diccionario con tantas entradas como usuarios para los que se habían capturado *tweets*. Cada entrada del diccionario tenía como valor otro diccionario, que tenía un número indefinido de entradas dependiendo de la ejecución, tenía tantas entradas como *tweets* se hubieran estado comprobando durante la ejecución y una entrada fija que contenía la cola de *tweets* a comprobar y su estado actual. Cada entrada de cada *tweet*, contenía el contador de *likes* de ese *tweet* y su timeout en el que se cumplían los 30 minutos hasta revisar resultados.

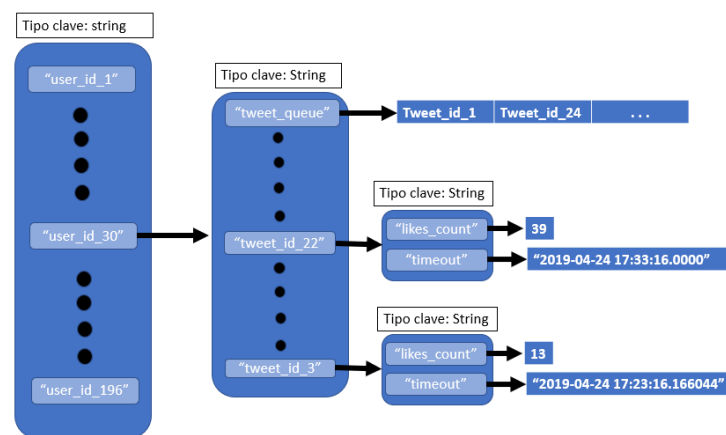


Figura 22 : Estructura de datos del proceso de likes

De esta forma, al inicio del proceso de *likes*, se comprueban todos los *tweets* una vez, se añadirán a la estructura de datos y se registrará la hora a la que se habían capturado *like* por primera vez, los primeros *likes* no se tienen en cuenta y en cada pasada por ese *tweet* se comprueba si han transcurrido los 30 minutos y en caso afirmativo si se han alcanzado los 30 *likes* se reseteará el contador y el temporizador. Si no se ha alcanzado la meta de 30 *likes* el *tweet* es eliminado.

Para garantizar que se detectan los nuevos *tweets* creados, por un lado en cada recorrido por los usuarios, se comprueba si hay algún nuevo *tweet* del usuario actual ( la comprobación consiste en ver si existe algún *tweet* en la colección con identificador mayor que el último de la cola del usuario y que no tenga entrada todavía en el registro del usuario).

Por otro lado, al recorrer todos los usuarios se lanzaba un hilo que abría un navegador, iterando por las URLS de los usuarios obteniendo los nuevos *tweets* ids hasta el máximo *tweet* id recuperado para ese usuario, en caso de haberlos crea un nuevo fichero especial que contenía estos nuevos ids. El proceso los añade a las colas de los usuarios y les fija sus tiempos límite.

El proceso termina cuando todas las colas están vacías a no ser que se hubiera lanzado con la opción cíclica, en tal caso se reinician las colas con los 20 últimos *tweets* y se reinicia el proceso. La Figura 23 representa este proceso.

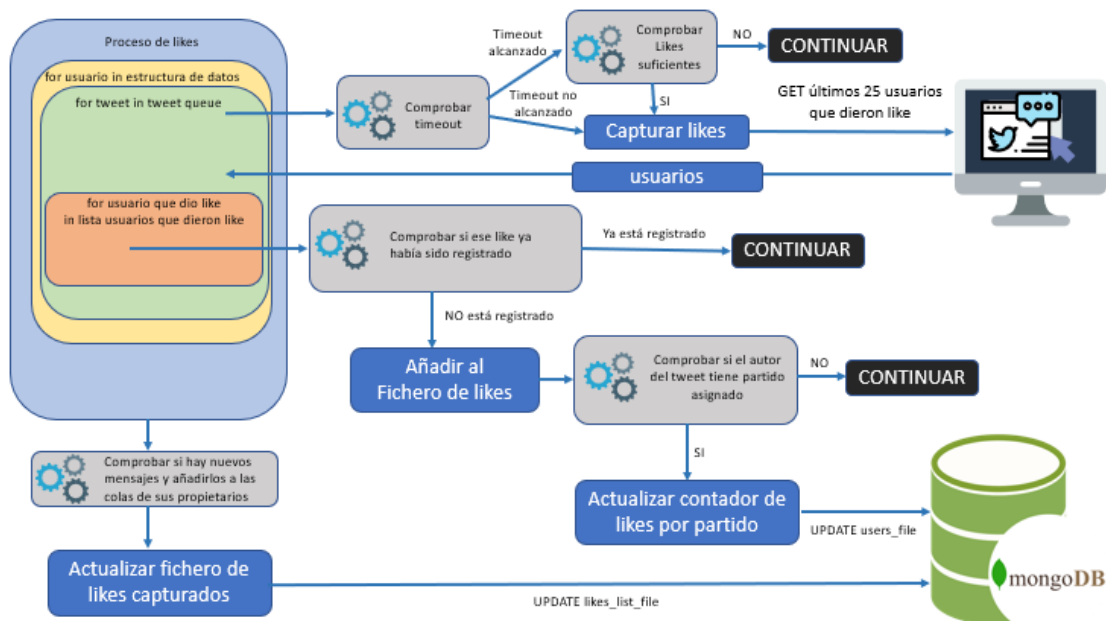


Figura 23 : Detalle del proceso de likes (versión 2)

En esta versión se detectaron los siguientes problemas:

- Se detectó que se hacían demasiadas escrituras en MongoDB para actualizar los contadores de *likes* por partido.
- Se detectó que, aunque el ratio de *likes* aumentó, se tardaba demasiado en volver a recorrer un mensaje.
- Se debería cambiar el criterio para considerar a un *tweet* relevante dependiendo del momento del día porque un criterio estático no se consideraba la mejor opción.

### Versión 7

En esta versión se intentó aumentar el número de *tweets* recorridos por minuto, pruebas anteriores de medición de tiempo, parecían señalar que la carga de cada url era el cuello de botella por lo que en esta versión se intentó cambiar la aproximación.

En esta versión el proceso de *likes* se hizo más sencillo y se intentó eludir las cargas de páginas web completas. Para ello, se descartó el modelo de mensajes a revisar por usuario y se volvió a los últimos n mensajes de un usuario.

Esta vez, el proceso de *likes*, carga la URL del usuario en lugar de cada URL de cada *tweet*, y desde esa página, *clicka* uno por uno en los n primeros *tweets* (garantizando así que siempre detecta nuevos *tweets*) para poder acceder al *tweet* en cuestión y tras esto, siguiendo con los modelos anteriores, accionaría el botón para poder obtener los últimos 25 usuarios que dieron *like* a ese *tweet*. La Figura 24 detalla este proceso.

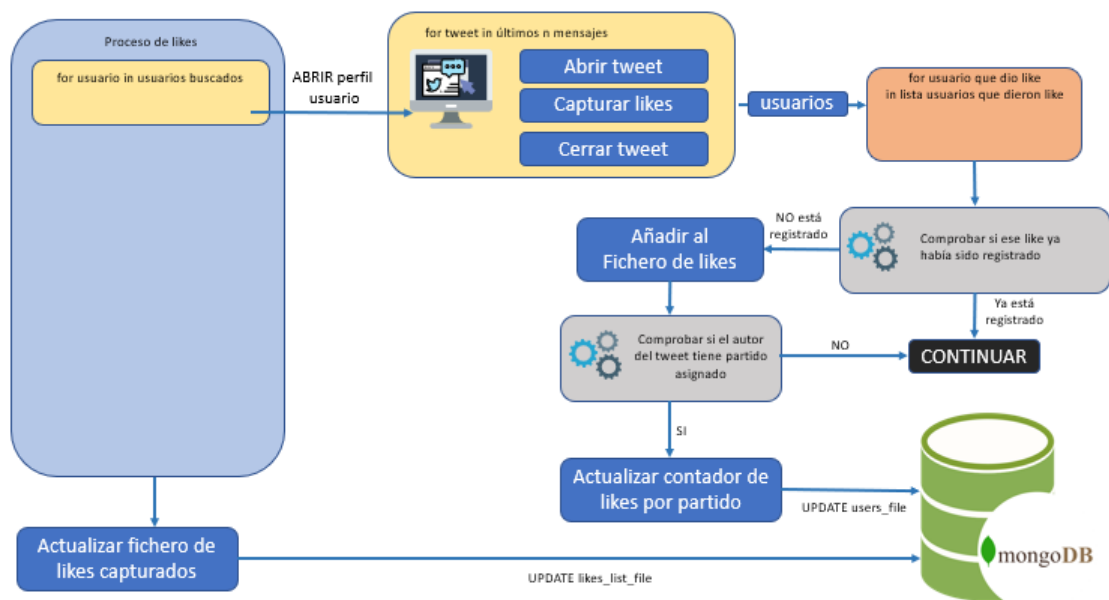


Figura 24 : Detalle proceso de likes (versión 3)

Este sencillo modelo tampoco funcionó como se esperaba ya que los tiempos seguían siendo muy superiores a lo necesitado y se perdían la inmensa mayoría de *likes* en nuevas publicaciones pero si mejoró el sistema de detección de nuevos *tweets* recién creados.

En esta versión se detectaron los siguientes problemas:

- Se detectó que se hacían demasiadas escrituras en MongoDB para actualizar los contadores de *likes* por partido.

### Versión 8

En esta versión se cambió de nuevo el proceso de captura de *likes*.

Se decidió renunciar a la captura de los 25 últimos usuarios que dieron *like* a un *tweet* y en su lugar obtener los 9 últimos usuarios que dieron *like* pero pudiendo así prescindir casi completamente de la interacción con el navegador y Selenium.

Se usaba Selenium una única vez por usuario para obtener los ids de los últimos n mensajes ya que esa funcionalidad ya estaba implementada y con estos ids y la ayuda de BeautifulSoup podíamos obtener los html de las urls de los *tweets* requeridos y analizándolas, podíamos obtener los últimos 9 *likes* de cada *tweet*. La Figura 25 detalla este proceso.

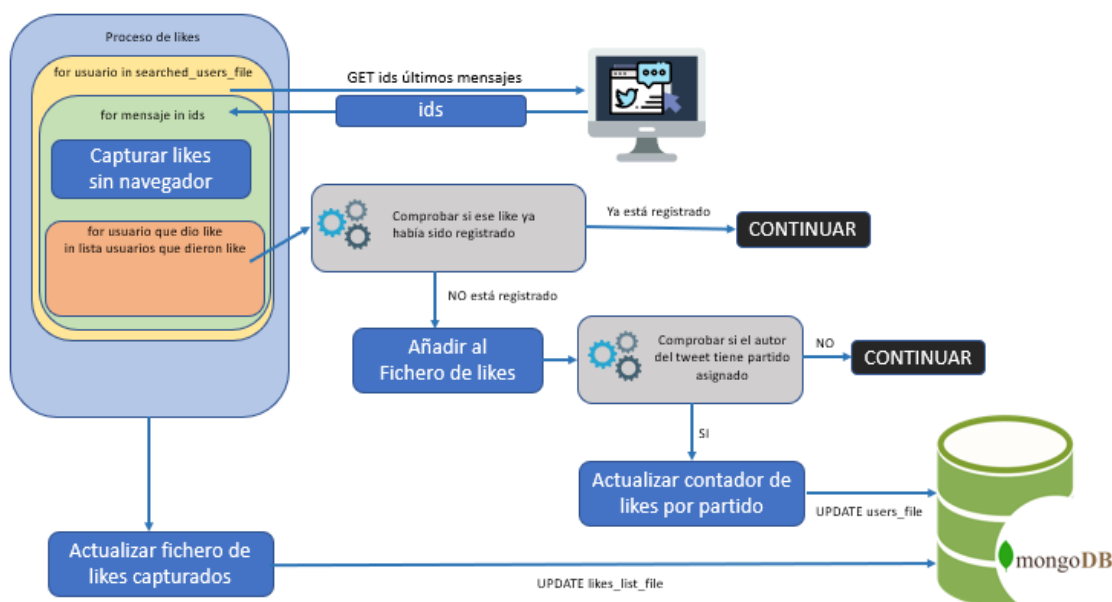


Figura 25 : Detalle del proceso de likes (versión 4)

Con este cambio, se obtuvieron mejoras pero no al nivel esperado. Esta versión fue la primera en bajar de 1 h (42 minutos) en recorrer los 20 últimos *tweets* de los 60 políticos seleccionados.

En esta versión se detectaron los siguientes problemas:

- Se detectó que se hacían demasiadas escrituras en MongoDB para actualizar los contadores de *likes* por partido

### Versión 9

En esta versión se separó el proceso de conteo de *likes* por partido del proceso de obtención de *likes*, se modificó el proceso de *likes* para ser ejecutado con 6 hilos y se añadió la generación de un fichero de log en cada ejecución para monitorear los tiempos de cada hilo y el número de *likes* capturados de una forma más simple. También se cambió el cálculo del conteo de *likes* haciendo menos escrituras en Mongo.

En esta versión, el proceso principal divide los usuarios en 6 listas que pasa como parámetro a sus seis hilos hijos para que realicen el proceso por si mismos. Cada hilo recorrerá la lista de usuarios y para cada usuario abrirá un navegador para obtener los *n* últimos identificadores de mensajes y con ellos, capturará los 9 últimos *likes* de cada mensaje.

Cuando acaba con todos los mensajes de todos sus usuarios, actualiza el fichero de *likes* en MongoDB. Debido a que cada hilo tiene sus usuarios, no se deberían dar estados inconsistentes. En primera instancia, la actualización se realizaba con un reemplazo del fichero completo, pero esto podía llevar a errores por actualización simultanea por lo que se sustituyó por actualizaciones a nivel de *tweet* con el operador “\$set” que garantiza que la operación es atómica. La Figura 26 representa este proceso.

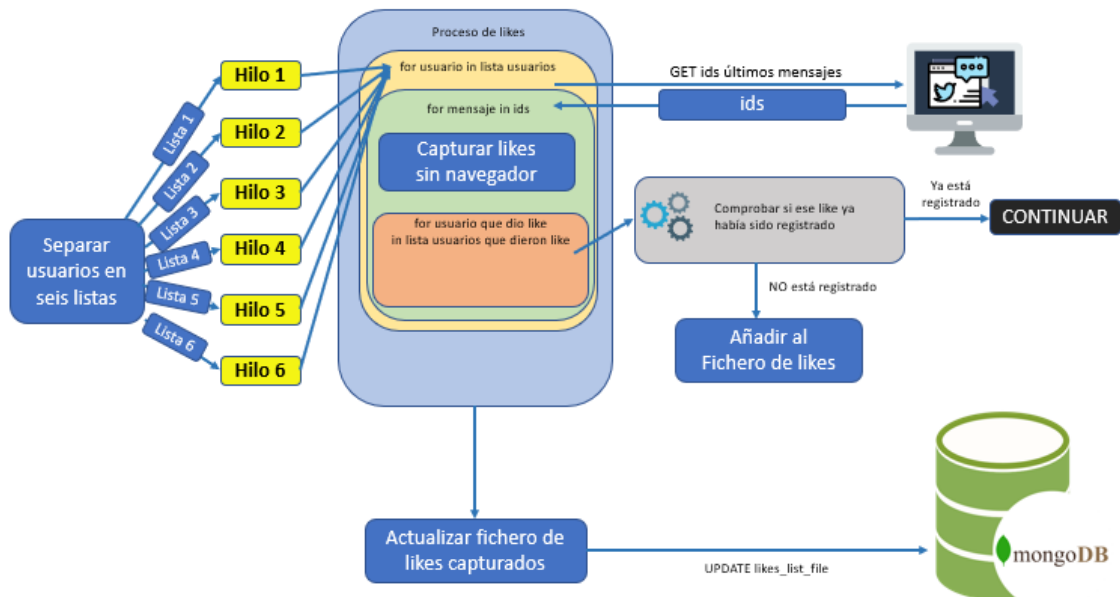


Figura 26 : Detalle del proceso de like (versión 5)

En esta versión se consiguió una gran mejora hasta el punto de que se revisitaba un *tweet* cada 5 minutos (comprobando alrededor de 1200 *tweets*) lo que supone comprobar 4 *tweets* por segundo aproximadamente.

Debido a estos resultados, se realizó un script “orquestador” para realizar una primera ejecución no acotada en una máquina del DSIC. Este script, lanzaba los procesos de *likes*, análisis, captura de *likes* y inicio del servidor web y en caso de que cualquier proceso muriera (fallase) era capaz de relanzarlo.

Esta versión funcionó correctamente durante casi tres días hasta que habiendo recopilado más de 140.000 *likes*, el proceso falló durante horas. La causa de este fallo fue que el fichero de *likes* alcanzó el tamaño máximo permitido para un único fichero.

El error era el siguiente:

`pymongo.errors.WriteError: Resulting document after update is larger than 16777216`

En este punto fue necesario cambiar gran parte de la organización de la aplicación principal así como de la aplicación de visualización.

En esta versión se detectaron los siguientes problemas:

- El proceso de *likes* fallaba en todas las escrituras al haber superado el espacio máximo permitido para un único fichero.
- Era necesario cambiar la forma de almacenar los *likes*, lo que conllevaría particionar ficheros, crear colecciones adicionales u otros cambios muy importantes.
- El cambio del almacenamiento de *likes* afectaría al proceso de conteo por partido y a la aplicación de visualización.

**Versión 10**

En esta versión se modificó el almacenamiento de los *likes*, ahora cada *tweet* almacena la información referente a sus *likes*.

Para ello, en la inserción en MongoDB, a cada documento se le añadía un nuevo campo “has\_likes\_info” que señalaba si se han capturado *likes* para dicho mensaje o no. El tratamiento previo a la inserción se puede observar en la Figura 27.

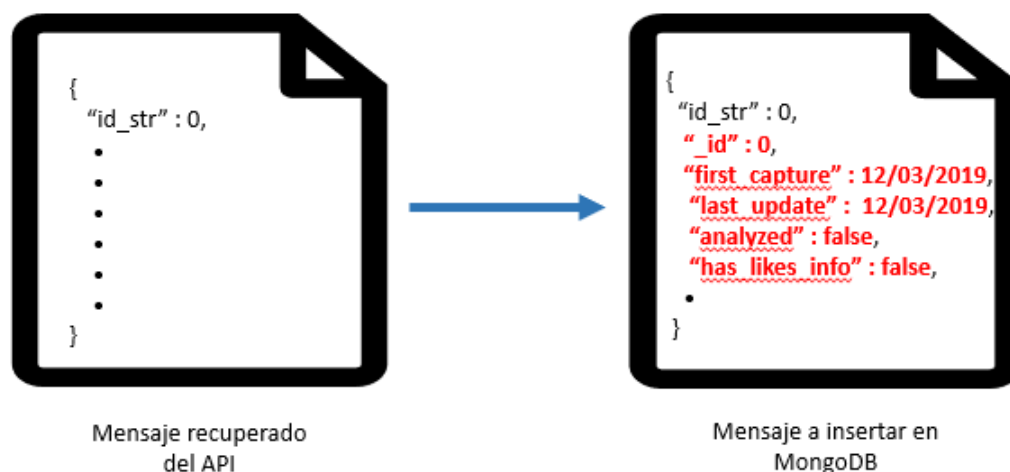


Figura 27 : Tratamiento previo a inserción en MongoDB 3

Si hubiera información referente a los *likes*, estaría en el campo “likes\_info”.

Esta decisión destapaba otro problema, ¿Cómo se gestionarán los *tweets* para los que se capturen *likes* pero no estén en la colección?

Se decidió guardar la información con *likes* de *tweets* que no estuvieran en la colección en documentos cuyo identificador fuera el identificador del *tweet* concatenado con “\_tmp”. Con estos documentos se controlaron los diferentes casos posibles:

- Captura de *likes* de un *tweet* que no está en la colección y cuyo “\_tmp” no ha sido creado.
- Captura de *likes* de un *tweet* que no está en la colección y cuyo “\_tmp” ha sido creado.
- Captura de *likes* de un *tweet* que está en la colección.
- Captura de *tweet* para el cual existe “\_tmp” en la colección.

Esta versión estuvo en ejecución durante 8 días antes de las elecciones generales y capturó 500.000 *likes* en ese periodo de tiempo.

**Versión 11**

En esta versión se hicieron pequeños cambios y optimizaciones y se añadieron algunas opciones:

- Posibilidad de elegir si ejecutar el proceso de *likes* con hilos o con el esquema de cola dinámica.

- Posibilidad de elegir si ejecutar el proceso de *likes* usando el navegador o sin él.
- Posibilidad de hacer un análisis forzoso (analizar todos los *tweets* aunque haya algunos que ya se hayan analizado).
- Posibilidad de hacer un conteo de *likes* forzoso (contar todos los *likes* de todos los *tweets* aunque ya se hayan analizado).
- Mejor parametrización de todos los métodos.

## Versión 12

En esta versión se particionó el fichero de conteo de *likes*, de forma que cada fichero albergaría información de 40000 usuarios.

Como se puede observar en la Figura 28, se añadió información adicional a los ficheros de usuarios buscados y a la información referente a los *likes* de cada fichero para poder agrupar los usuarios por antigüedad, número de tweets publicados y tipo de cuenta (verificada o no verificada).

```
{
  "_id": "searched_users_file_id",
  "abalosmeco": {
    "captured_tweets": 1,
    "favourites": 6728,
    "followers": 39929,
    "following": 1850,
    "joined": "2010/10/13 15:20",
    "last_execution": "2019-05-23 16:51:51.897412",
    "max_creation_date": "Thu May 23 12:19:22 +0000 2019",
    "max_tweet_id": "1131535089358299136",
    "min_creation_date": "Thu May 23 12:19:22 +0000 2019",
    "min_tweet_id": "1131535089358299136",
    "partido": "PSOE",
    "search_type": "tweets captured by user",
    "tweets": 16401,
    "user": "abalosmeco",
    "user_id": "202372417",
    "user_name": "Jos\u00e9 Luis \u00c1balos",
    "verified": true
  }
}
```

Figura 28: Cambio en el registro del fichero de usuarios

Con todo esto, se ha creado una aplicación customizada que contaba con 10 procesos principales : análisis de *tweets* en el sistema de archivos, recopilación de *tweets* por consulta, recopilación de *tweets* por *streaming*, recopilación de *tweets* por usuario, ejecución de las búsquedas por consulta realizadas para una colección, ejecución de las búsquedas por usuario realizadas para una colección, ejecución de las búsquedas por *streaming* realizadas para una colección, recopilación de *likes* de los mensajes de los usuarios buscados de una colección , conteo de los *likes* recopilados por partido y actualización de los mensajes de la colección.

Las distintas opciones de cada modo se detallan en el Anexo 3.

## Desarrollo de la aplicación visualización.

### Versión 1

La primera versión de la aplicación de visualización constaba del servidor web Flask y únicamente tenía una URL que mostraba el fichero de estadísticas de la colección por defecto. Su funcionamiento, se muestra en la Figura 29.

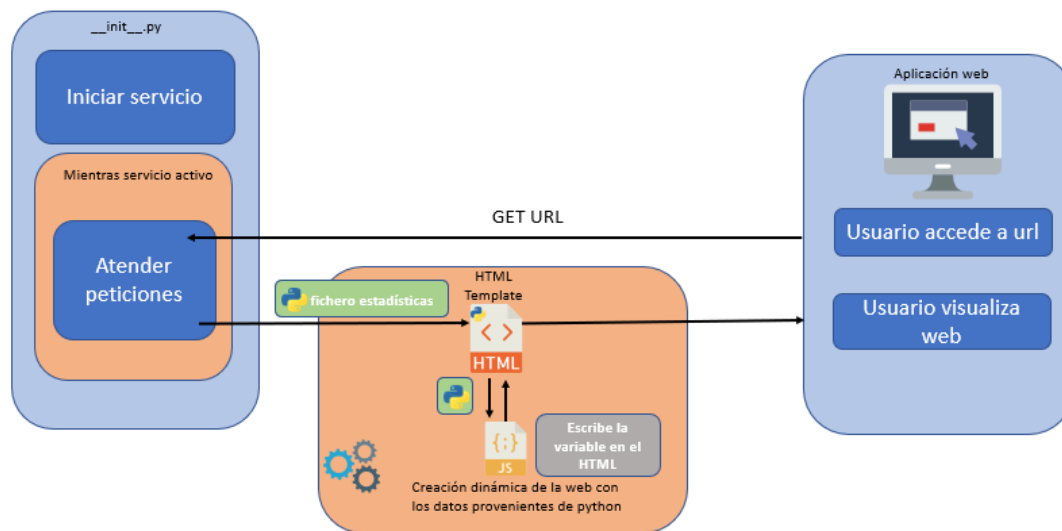


Figura 29 : Funcionamiento de la aplicación de visualización básica

En esta primera versión se encontró un problema en el paso de variables de Python a HTML y de este a JavaScript. Este problema venía dado por la incompatibilidad de los diccionarios Python con los “diccionarios” de Javascript.

Mientras que en Python los diccionarios o maps son una estructura básica, en javascript son objetos similares a un map pero con grandes diferencias.

Los objetos JavaScript tienen un prototipo, por lo que hay clases por defecto en el map. Las claves de un objeto deben ser de tipo String obligatoriamente al contrario que en python que pueden ser de cualquier tipo. Los diccionarios python almacenan su tamaño y puede ser obtenido directamente.

La solución fue convertir los diccionarios python a json antes de pasarlos a JavaScript. De esta forma, utilizando este formato común pudimos comunicar python y javascript a través de HTML con ayuda de Flask.

### Versión 2

En esta versión se crearon los 5 html necesarios, así como la estructura general que compartirían. Se implementó por completo la sección de estadísticas de una colección, la sección de rankings y la elección de colección de MongoDB.

Para poder elegir la colección de MongoDB que se quería visualizar fue necesario implementar comunicación en sentido contrario, es decir, desde JavaScript a Python.



Para ello, en primer lugar es necesario mostrar las opciones al usuario, esto se hará generando las opciones dinámicamente con la información que recibamos de Python ( cada vez que accedamos a una URL, Python consultará que colecciones existen y se las enviará al HTML para que este genere su menú). En la Figura 30, podemos ver una captura de este menú.

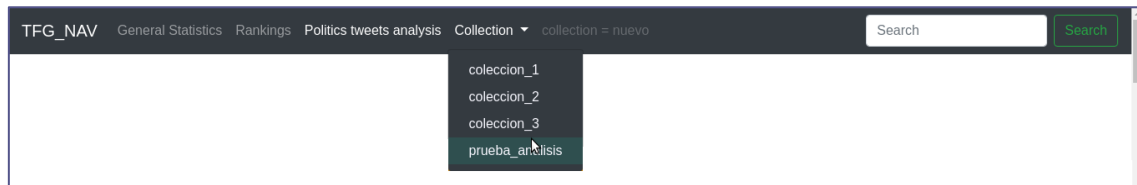


Figura 30 : Menú de elección de la colección

En Flask deberemos habilitar una ruta que en lugar de devolver una web, escuchará peticiones de tipo POST (envío de datos al servidor), esta ruta se llama *endpoint*.

Cuando el usuario elija una colección, se disparará un evento que enviará una consulta POST al *endpoint* de Flask con el valor de la colección que se haya elegido y con ese valor fijaremos la colección actual para la cual se realizarán consultas.

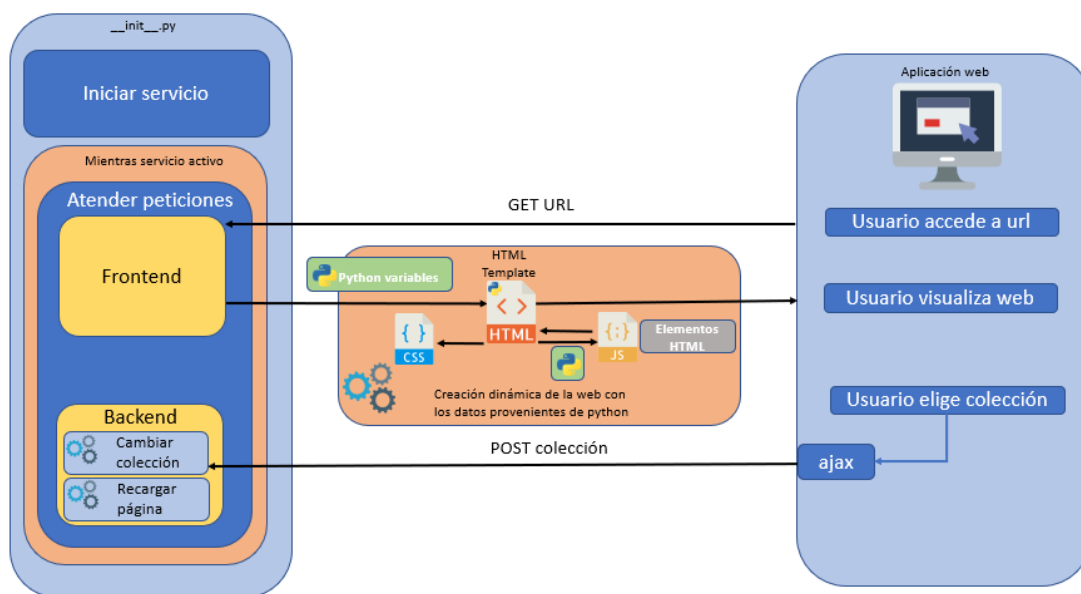


Figura 31 : Ciclo de paso de variables

Cuando se cambia la colección la página se recarga y se genera con los datos de la nueva colección como se puede observar en la Figura 31.

En la sección de estadísticas generales (solamente tendrá contenido cuando se haya lanzado el proceso de análisis de la colección al menos una vez) se incluyeron múltiples gráficos de distintos tipos creados con highcharts.js que recogen información como número de mensajes recuperados, número de *tweets* y retweets, distribución por tipo de envío etc. La generación dinámica de estos gráficos requiere un tratamiento de la variables y preparación de los datos distinto por cada tipo de gráfico y distribución.

En la Figura 32 se nos presenta un ejemplo de gráficos con la distribución de los mensajes por forma de envío y de los usuarios según si la cuenta es o no verificada.

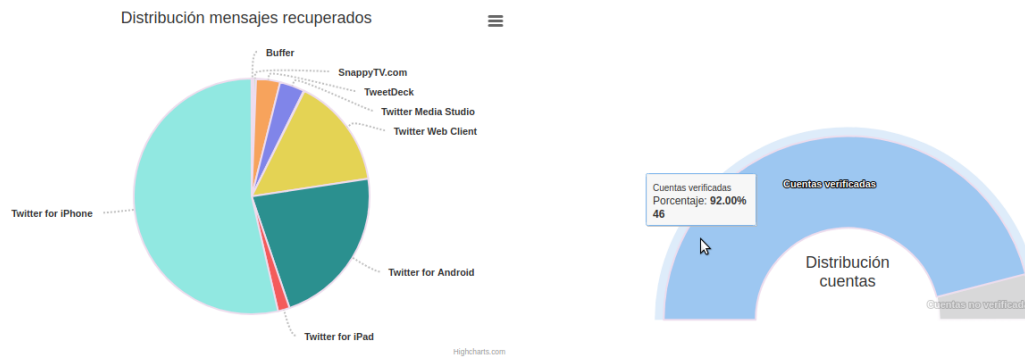


Figura 32 : Gráficos generados por la aplicación de visualización

También se añadieron en esta sección tablas con la información de los usuarios (usuarios verificados y no verificados) consultas, usuarios, búsquedas por *streamming* realizadas. Cabe destacar que todas estas tablas cuentan con paginación (las tablas tienen un número de filas a mostrar), filtrado (tienen barra de búsqueda incluida para poder mostrar solo los resultados que nos interesan) y ordenación (se pueden ordenar por cualquier campo). Se puede observar una captura de una tabla en la Figura 33.

#### BÚSQUEDAS POR USUARIO DE LA COLECCIÓN

Search		
user	captured tweets	last execution
abalosmeco	20	2019-05-02 22:08:04.618995
adrilastra	20	2019-05-02 22:08:00.388332
ahorapodemos	20	2019-05-02 22:08:07.218712
albert_rivera	20	2019-05-02 22:08:21.137687
albiol_xg	20	2019-05-02 22:07:51.751773
alevysoler	20	2019-05-02 22:07:52.768686
ander_gil	20	2019-05-02 22:08:01.324803
begonavillaci	20	2019-05-02 22:08:25.742163
ciudadanoscs	20	2019-05-02 22:08:19.180931
ciudadanoville	20	2019-05-02 22:08:22.182733

Figura 33 : Ejemplo de tabla de la aplicación visualización

En la sección de rankings de 10 miembros (únicamente tendrá contenido cuando se haya lanzado el proceso de análisis de la colección al menos una vez) se muestran rankings tanto de usuarios como de mensajes.

Debido a que el número de mensajes de estos rankings es reducido, se modificó el proceso de análisis para que añadiera al fichero de estadísticas un diccionario con dos nuevos campos por cada mensaje que formara parte de un ranking para poder visualizarlos tal y como lo hace la web de Twitter.

Para conseguir esto, Twitter nos ofrece un fragmento de código html, que podemos obtener desde su web y añadirlo a nuestra aplicación como se nos muestra en la Figura 34.



Figura 34 : Obtención del código de embebido desde la web de Twitter

Con el fin de obtener este fragmento para embeber un *tweet* en la página web, se creó un sistema automático con Selenium similar al de recopilación de *tweets* que accionase los botones necesarios y cuando el fragmento fuera visible, analizara el html para obtenerlo. Se hizo con Selenium ya que son necesarias acciones y accionar varios botones para obtener el código de embebido.

Llegado este punto se detectó que el fragmento para embeber un *tweet* no aparecía en el html en ningún momento. Ante esto, la solución fue simular el copiado del código con el sistema automático y el mayor problema fue conseguir pasar el contenido del portapapeles a una variable Python durante la ejecución.

Para ello se investigó y se decidió usar el módulo pyperclip que era el único módulo que parecía proporcionar esta característica, pero pyperclip no podía encontrar el mecanismo de copiado y pegado del sistema (Ubuntu). Esto se resolvió instalando una utilidad llamada xsel.

Tras las primeras pruebas se detectó que en algunos *tweets* existían dos códigos de embebido, uno con multimedia y otro sin ella, se modificó el proceso de análisis para obtener ambos y se modificó la web para dar la opción de elegir el tipo de visualización de los *tweets* de los rankings.

Los rankings de usuarios y de *tweets* seguían la misma estructura con la salvedad de que en los rankings de *tweets*, se mostraba el *tweet* al estar 2 segundos sobre él y que en los rankings de usuario aparecía en todo momento su foto de perfil.

Para obtener la foto de perfil de un usuario, simplemente se usa un tag html con la url del usuario como se muestra en la Figura 35:

```
<img src='https://avatars.io/twitter/usuario' />
```

Figura 35 : Tag HTML de obtención de la foto perfil Twitter

A continuación, la Figura 36 muestra una captura de un *ranking* de *tweets* no seleccionado y la Figura 37 lo muestra seleccionado.

Top 10 de mensajes con más likes	
1	Last update: 2019-05-02 22:06:30
user: Pablo_Iglesias_	tweet_id: 1121896483954069504
45641 likes	
2	Last update: 2019-05-02 22:06:20
user: sanchezcastejon	tweet_id: 1122631314824421376
32553 likes	

Figura 36 : Ejemplo ranking de tweets


Top 10 de mensajes con más likes	
1	Last update: 2019-05-02 22:06:30
user: Pablo_Iglesias_	tweet_id: 1121896483954069504
45641 likes	
	
2	Last update: 2019-05-02 22:06:20
user: sanchezcastejon	tweet_id: 1122631314824421376
32553 likes	

Figura 37 : Ejemplo ranking de tweets activo

### Versión 3

En esta versión se cambió la visualización de los rankings de usuarios a una versión más compacta como se puede observar en la Figura 38.

Top 10 de usuarios con más mensajes publicados:










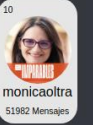
1	2	3	4	5
				
CiudadanosCs	ahorapodemos	PSOE	populares	compromis
130884 Mensajes	100351 Mensajes	96713 Mensajes	82307 Mensajes	72686 Mensajes
6	7	8	9	10
				
CompromisVLC	Tonicanto1	CristinaNarbona	Albert_Rivera	monicaoltra
65066 Mensajes	56132 Mensajes	55938 Mensajes	54993 Mensajes	51982 Mensajes

Figura 38 : Ejemplo de la visualización del ranking de usuarios

Se implementaron las secciones de visualización de ficheros especiales y de información relacionada con *likes* y política.

En la sección de ficheros especiales (véase la Figura 39) se muestra si existe o no el fichero, su última actualización y un icono para visualizarlo.

TFG_NAV	General Statistics	Rankings	Politics tweets analysis	Collection	collection = test2	Search	Search
<div> </div> <div>Last analysis = 2019-05-02 22:24:37.214188</div>							
DOCUMENTOS ESPECIALES DE LA COLECCIÓN							
Documento	Id	Estado					
Statistics File	"statistics_file_id"						
Query File	"query_file"						
Streaming File	"streaming_file_id"						
Searched Users File	"searched_users_file_id"						
Likes List File	"likes_list_file_id"						
Users File	"users_file_id"						

Figura 39 : Visualización de la sección documentos especiales

En la sección de políticos, se incluyeron distintas gráficas y tablas. Cabe destacar la generación de una tabla sin paginación que permite ver cuantos *likes* se han recuperado de cada *tweet* de una forma muy visual con una barra de progreso como podemos apreciar en la Figura 40.

TFG_NAV	General Statistics	Rankings	Politics tweets analysis	Collection	collection = test2	Search	Search
1122491412744945670	perefuset	9 (1 veces)	12	75.00%	<div><div></div></div>	2019-05-02 22:47:21.694072	
1123194482491559936	giuseppgrezzi	9 (1 veces)	12	75.00%	<div><div></div></div>	2019-05-02 22:43:50.076446	
1123894310070562816	podem_	9 (1 veces)	12	75.00%	<div><div></div></div>	2019-05-02 22:43:45.082087	
1123907569507082240	ppcv	9 (1 veces)	12	75.00%	<div><div></div></div>	2019-05-02 22:49:00.140193	
1123133879567757312	monicaoltra	9 (1 veces)	13	69.23%	<div><div></div></div>	2019-05-02 22:43:00.978132	
1123182028684505090	teresarodr_	9 (1 veces)	13	69.23%	<div><div></div></div>	2019-05-02 22:44:54.300363	
1123884906927546368	zaidacantera	9 (1 veces)	13	69.23%	<div><div></div></div>	2019-05-02 22:46:40.740878	
1121755859112157185	alevisoler	9 (1 veces)	14	64.29%	<div><div></div></div>	2019-05-02 22:44:04.557364	
1123953449782120451	teresarodr_	9 (1 veces)	14	64.29%	<div><div></div></div>	2019-05-02 22:44:34.429723	
1123977994488438785	monicaoltra	9 (1 veces)	14	64.29%	<div><div></div></div>	2019-05-02 22:42:48.279963	
1123997158045364227	girautaoicial	9 (1 veces)	14	64.29%	<div><div></div></div>	2019-05-02 22:46:01.044446	
1122943713217478664	enricomorera	9 (1 veces)	15	60.00%	<div><div></div></div>	2019-05-02 22:42:56.729161	
1123637189433278464	teresarodr_	9 (1 veces)	17	52.94%	<div><div></div></div>	2019-05-02 22:44:41.720611	
1123655519179296768	jordi_canyas	9 (1 veces)	17	52.94%	<div><div></div></div>	2019-05-02 22:47:38.974852	
1123868287400644608	ppcv	9 (1 veces)	17	52.94%	<div><div></div></div>	2019-05-02 22:49:14.923100	
1123903899193233408	compromis	9 (1 veces)	17	52.94%	<div><div></div></div>	2019-05-02 22:48:38.024996	
1123984463195725825	ander_gil	9 (1 veces)	17	52.94%	<div><div></div></div>	2019-05-02 22:45:15.783168	

Figura 40 : Tabla de monitoreo de la recopilación de likes

Con la finalidad de poder monitorear si los parámetros elegidos para la recopilación de *like* eran los adecuados se añadió un campo a la información de los *likes* de cada *tweet* en la que se almacena cuantas veces ha sido comprobado ese *tweet* para lograr los *likes* obtenidos.

#### Versión 4

En esta versión se realizó la sección de búsqueda por identificador de *tweet*. La búsqueda se realiza con una expresión regular por lo que se mostrarán todos los documentos cuyo id tenga como prefijo, sufijo o contenga el valor buscado.

Los resultados de esta búsqueda pueden ser filtrados y ordenados por todos los campos.

En esta sección al igual que en la de documentos especiales se introdujeron iconos que permitían la visualización del *tweet* como se observa en la Figura 41.

Se han encontrado 1375 Coincidencias








Search				
id str	user.screen name	created at	last update	Show file
1088368436747612160	pbustinduy	Thu Jan 24 09:30:30 +0000 2019	2019-05-02 22:06:37	
1089125250925821952	pbustinduy	Sat Jan 26 11:37:49 +0000 2019	2019-05-02 22:06:37	
1089125255384313856	pbustinduy	Sat Jan 26 11:37:50 +0000 2019	2019-05-02 22:06:37	
1089125257875730433	pbustinduy	Sat Jan 26 11:37:51 +0000 2019	2019-05-02 22:06:37	
1089958639421530112	pbustinduy	Mon Jan 28 18:49:24 +0000 2019	2019-05-02 22:06:37	
1093479411112988672	pbustinduy	Thu Feb 07 11:59:42 +0000 2019	2019-05-02 22:06:37	
1097471261029163008_tmp	pbustinduy	--	--	

Figura 41 : Visualización de los resultados de la búsqueda

## Versión 5

En esta versión, debido al cambio de almacenamiento de los *likes* se tuvo que modificar gran parte de la sección de *likes* y política y además, se cambiaron algunos elementos meramente estéticos.

Llegado este punto se ha implementado una interfaz gráfica que permite monitorear las colecciones de MongoDB de una forma sencilla e intuitiva.

Esta interfaz nos permite controlar las fechas de las ejecuciones, así como cada uno de los documentos y nos muestra estadísticas de cada colección.

## Versión 6

Se modificó la forma de mostrar el fichero de conteo de *likes* ya que en adelante sería un fichero particionado, de esta forma, en la interfaz gráfica, se separó de los demás ficheros especiales como se aprecia en la Figura 42.


DOCUMENTOS DE CONTEO DE LIKES DE LA COLECCIÓN (PARTICIONADO)		
Id	Número usuarios	Visualizar
likes_count_file_id_0	40000	
likes_count_file_id_1	40000	
likes_count_file_id_2	15668	

Figura 42: Visualización de los ficheros de conteo de likes

Además, se añadieron visualizaciones de los tipos de usuarios que dieron like a un partido o usuario en concreto y se implementó un simulador de votos en función de los *likes*. Todo esto, se detalla en el caso de estudio.

## 7. Caso de estudio

---

Aprovechando la cercanía de las elecciones al parlamento español, se decidió realizar un caso de estudio de la aplicación. Con este objetivo, se instaló la aplicación en una máquina del grupo de investigación ELIRF (Enginyeria del Llenguatge Natural i Reconeiximent de Formes) el lunes 15 de abril de 2019 para intentar recopilar el máximo número de *likes* de usuarios a políticos con la intención de hacer una comparativa de los resultados reales en las elecciones generales del 28 de abril y los resultados obtenidos por la aplicación.

Para este propósito se eligieron en primer lugar las 10 cuentas oficiales más importantes de políticos pertenecientes a cada uno de los partidos que se pretendía monitorizar con la aplicación (PP, PSOE, Podemos, Ciudadanos, Compromís y Vox).

Se creó una nueva colección y se capturaron los últimos 20 *tweets* de cada usuario etiquetándolos como miembro de su partido.

Tras esto, se creó un “orquestador” con el objetivo de que este lanzase 4 procesos en paralelo:

- Proceso de captura de *tweets* de los usuarios buscados de la colección.
- Proceso de recopilación de *likes* de los 20 últimos *tweets* de los usuarios buscados de la colección.
- Proceso de análisis de los mensajes de la colección (este proceso se lanzaría cada 15 minutos).
- Servidor web.

Además de lanzar estos 4 procesos, cada 5 minutos el orquestador debe comprobar si siguen activos y en caso contrario relanzarlos, así se podrían corregir errores no controlados.

Este orquestador se hizo en python. La gestión de la espera de 5 minutos hasta la nueva comprobación del estado de los procesos se realizó con timers.

```
angel@angel-Lenovo-G50-80:~$ ps -ef | grep python3
angel 1887 1618 0 22:39 ?        00:00:00 /usr/bin/python3 /usr/share/system-config-printer/applet.py
angel 1899 1618 0 22:39 ?        00:00:00 /usr/bin/python3 /usr/bin/blueman-applet
angel 2682 2671 0 22:40 pts/1    00:00:00 python3 FlaskAPP/ __init__.py
angel 2695 2682 0 22:40 pts/1    00:00:01 /usr/bin/python3 /home/angel/Escritorio/TFG/TFG_UNI/TFG_UNI_3/FlaskAPP/ __init__.py
angel 3487 3465 0 22:43 pts/2    00:00:00 python3 controler_v1.py
angel 3538 3519 0 22:44 pts/3    00:00:00 grep --color=auto python3
```

Figura 43 : Comando para la obtención de los procesos activos

La comprobación consistía en ejecutar una serie de comandos Linux (mediante la librería subprocess) para poder ver que procesos Python estaban activos, localizar si

alguno no existía y relanzarlo (también con la librería subprocess). En la Figura 45 se puede observar el comando de obtención de los procesos Python activos.

El proceso funcionó correctamente durante 3 días hasta que el fichero en el que se almacenaban los *likes* alcanzó el tamaño máximo permitido por MongoDB por lo que se tuvo que cambiar el almacenamiento de *likes* como se detalló en la versión 10 de la aplicación principal.

Tras el cambio de almacenamiento, el proceso se volvió a poner en ejecución y hasta el día de las elecciones obtuvo 500.000 *likes*.

Por el interés en intentar clasificar a los usuarios de los cuales se capturan *likes*, detectar posibles *bots* y comprobar si la información relacionada con los *likes* tiene alguna similitud con los resultados electorales, se añadió una nueva sección que recoge para cada usuario buscado estadísticas del usuario así como de los *likes* que se capturaron para sus mensajes. A esta sección se le añadió la funcionalidad para conectarse a la Wikipedia y en caso de que el usuario tuviera un artículo recuperar su resumen. Podemos observar una captura de una parte de esta sección en la Figura 43.



Figura 44: Ejemplo de cabecera de la sección de usuario

Además, se añadieron en la sección de cada usuario gráficos representando las distintas clasificaciones de los usuarios que dieron *like*.

Estas clasificaciones, separaban los usuarios por tipo de cuenta, (verificados y no verificados), por antigüedad (menos de 3 meses, entre 3 y 6 meses, entre 6 y 12 meses, entre 1 y 2 años, entre 2 y 3 años y más de 3 años) y por número de tweets publicados (menos de 50 mensajes, entre 50 y 99 mensajes, entre 100 y 249 mensajes, entre 250 y 499 mensajes, entre 500 y 999 mensajes y con 1000 o más mensajes).

En la sección de tweets políticos, se añadieron gráficos con las clasificaciones de los usuarios que dieron *like* a cada partido.

Por último, se añadió un simulador de votaciones, que partiendo de los *likes* que cada usuario dio a cada partido, asignaba un partido (al que más *likes* hubiera dado) a un usuario, con esta base, se desarrolló un simulador dinámico en el cuál podemos elegir que tipos de usuarios contabilizar según antigüedad, tipo de cuenta y número de mensajes publicados con la finalidad de eludir cuentas falsas.



Además, es posible fijar el número de *likes* dados a mensajes de un partido como para considerar a ese usuario como un posible votante. En la Figura 44 se puede observar el aspecto final del simulador de votos realizado.



Figura 45: Simulador de votos

Con el objetivo de encontrar relación entre el impacto en Twitter y los resultados electorales, se han realizado múltiples simulaciones.

En las simulaciones, cada usuario se contabiliza como potencial votante una única vez.

Así, si únicamente tuviéramos en cuenta los partidos contemplados por la aplicación, los resultados de las elecciones generales serían los siguientes:



Ilustración 46: Distribución de los resultados de las elecciones generales de 2019

PSOE contaría con un 39,35 % de los votos, PP contaría con un 21,15 % de los votos, Ciudadanos contaría con un 18,26% de los votos, Podemos contaría con un 13,48% de los votos, Compromís contaría con un 0,004 % de los votos y Vox con 7,72%

Las simulaciones, con los datos recogidos y sin ningún filtro, nos proporcionan los resultados que podemos observar en la Figura 47:



*Ilustración 47: Distribución de votantes en Simulaciones sin filtros*

Como podemos observar, en nuestras simulaciones VOX es el gran beneficiado frente a los resultados reales.

Además, Compromís a pesar de ser un partido de ámbito valenciano, tiene un impacto notorio en Twitter.

La comparativa se muestra en la siguiente tabla:

Partido	% elecciones	% simulaciones sin filtros
<b>PP</b>	21,15 %	8,00%
<b>PSOE</b>	39,35 %	28,68%
<b>Ciudadanos</b>	18,26%	9,80%
<b>Podemos</b>	13,48%	17,87%
<b>Compromís</b>	0,004%	8,03%
<b>VOX</b>	7,72%	27,62%

Así pues observamos que los partidos clásicos (PP y PSOE) y Ciudadanos disminuyen en gran medida su masa de votantes en las simulaciones. Mientras, los partidos más recientes obtienen una mayor implicación en las redes sociales.

Vox es el partido que más potenciales votantes tiene según su número de *likes* en Twitter como muestra la Figura 48:

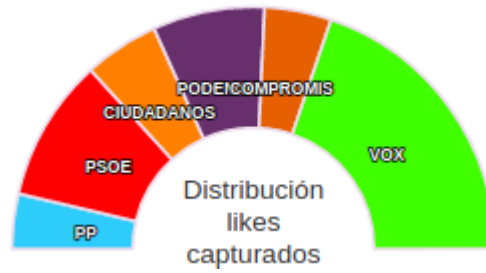


Ilustración 48: Distribución de los likes capturados por partido

Como podemos observar comparando la Figura 48 y 49, Vox no obtiene más *likes* por publicar más tweets que el resto de partidos, sino que es uno de los partidos menos activos en Twitter.



Ilustración 49: Distribución de tweets capturados por partido

Tanto Ciudadanos como PP obtienen un ratio de *likes* inferior a la mitad del ratio del resto de partidos como nos muestra la Figura 50:

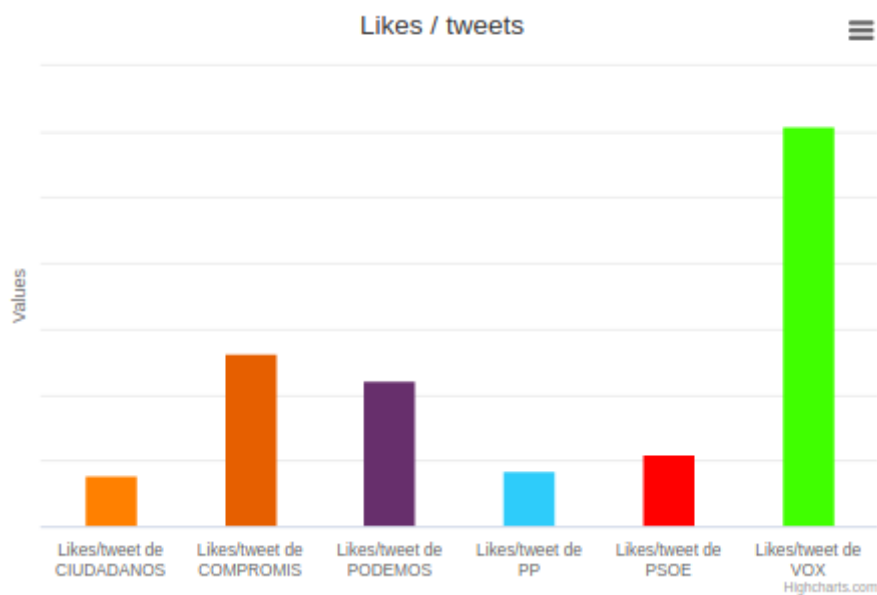


Ilustración 50: Ratio likes partido tweets

Si únicamente tenemos en cuenta aquellos usuarios (verificados y no verificados) que han publicado al menos 1000 tweets y que llevan más de 3 años en la plataforma, obtenemos la siguiente distribución:



*Ilustración 51: Distribución de los votantes antiguos*

Partido	% elecciones	% simulaciones sin filtros	% simulaciones usuarios antiguos
PP	21,15 %	8,00%	8,16%
PSOE	39,35 %	28,68%	30,19%
Ciudadanos	18,26%	9,80%	9,99%
Podemos	13,48%	17,87%	19,53%
Compromís	0,004%	8,03%	8,21%
VOX	7,72%	27,62%	23,92%

Si únicamente tenemos en cuenta los usuarios verificados, con una antigüedad superior a 3 años con al menos 1000 mensajes publicados obtenemos lo siguiente:



*Ilustración 52: Distribución de los votantes antiguos verificados*

Partido	% elecciones	% simulaciones sin filtros	% simulaciones usuarios antiguos	% simulaciones usuarios antiguos verificados
<b>PP</b>	21,15 %	8,00%	8,16%	15,78%
<b>PSOE</b>	39,35 %	28,68%	30,19%	27,75%
<b>Ciudadanos</b>	18,26%	9,80%	9,99%	12,67%
<b>Podemos</b>	13,48%	17,87%	19,53%	17,25%
<b>Compromís</b>	0,004%	8,03%	8,21%	7,42%
<b>VOX</b>	7,72%	27,62%	23,92%	19,83%

Si consideramos como potenciales votantes a los usuarios que han dado *likes* a al menos 10 tweets de un partido la distribución es la siguiente:

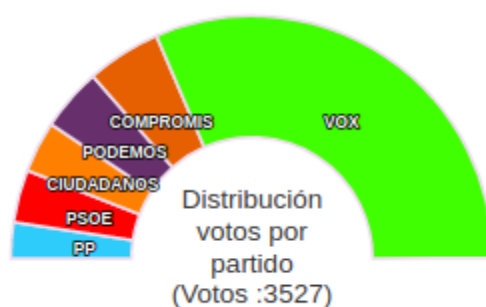


Ilustración 53: Distribución votantes con al menos 10 likes capturados

Partido	% elecciones	% simulaciones sin filtros	% simulaciones usuarios antiguos	% simulaciones usuarios antiguos verificados	% simulaciones usuarios con al menos 10 <i>likes</i>
<b>PP</b>	21,15 %	8,00%	8,16%	15,78%	4,96%
<b>PSOE</b>	39,35 %	28,68%	30,19%	27,75%	7,50%
<b>Ciudadanos</b>	18,26%	9,80%	9,99%	12,67%	7,30%
<b>Podemos</b>	13,48%	17,87%	19,53%	17,25%	8,74%
<b>Compromís</b>	0,004%	8,03%	8,21%	7,42%	10,62%
<b>VOX</b>	7,72%	27,62%	23,92%	19,83%	60,88%

Cabe destacar que VOX cuenta con muchos usuarios que dan *like* a un gran número de sus tweets.

Como conclusión, podemos señalar que los partidos clásicos tienen una menor repercusión en Twitter que los partidos más recientes a excepción de Ciudadanos que tiene el peor ratio de *likes* / tweet.

Vox es el partido con más aparente influencia en Twitter debido a que aunque no es el que más usuarios seguidores tiene, tiene una gran cantidad de seguidores constantes que dan like a muchos de sus tweets.

## 8. Conclusiones

---

Se ha llevado a cabo el desarrollo de una aplicación para la recopilación de información de Twitter desde dos fuentes, el API de Twitter y la web de Twitter.

Esta aplicación ha añadido funcionalidad adicional a las disponibles en el API y proporciona gestión para el almacenamiento, la actualización y el análisis de los datos.

Se han cumplido los objetos añadiendo en algunos casos funcionalidad adicional:

### Recopilación de *tweets*

- Es posible la captura de *tweets* dada una consulta, dado uno o más usuarios y la captura de *tweets* en *streaming* dado un conjunto de palabras clave.
- Se controlan los límites establecidos por el API y la llegada de mensajes de error del API.

### Adición de información no proveniente del API.

- Se proporcionan dos formas de capturar los usuarios que dieron *like* a un mensaje.

### Gestión del almacenamiento

- El uso de MongoDB garantiza la ausencia de duplicados.
- Se pueden gestionar varias colecciones.
- Para cada colección se ofrecen ficheros históricos con las consultas de cada tipo y es posible reejecutarlas automáticamente. Además, se ofrece información sobre la última ejecución.
- Es posible almacenar los mensajes recopilados en ficheros json.
- Existe un proceso de actualización de mensajes de la colección.

### Ejecuciones no acotadas con recuperación tras error

- La aplicación cubre gran cantidad de errores y además se implementó un proceso auxiliar capaz de relanzar procesos ante fallos.

### Visualización

- Se proporciona una visualización de estadísticas por colección, así como la posibilidad de visualizar todos los documentos de una colección con una opción de búsqueda.

## 9. Relación del trabajo desarrollado con el grado cursado

---

En la realización del presente trabajo de fin de grado se han aplicado conocimientos aprendidos durante el grado (algorítmica, estructuras de datos, programación concurrente, buenas prácticas ...), tecnologías utilizadas durante el grado (python3) y se han ampliado estos conocimientos con:

- Programación web (HTML, CSS3 y JavaScript).
- Uso de bases de datos no relacionales (MongoDB).
- Uso y creación de peticiones HTTP.
- Servidores web.
- Interacción con APIs.
- Creación de sistemas automáticos.

## 10. Referencias

---

Interactive JavaScript charts for your webpage | Highcharts. (s.f.). Recuperado 3 mayo, 2019, de <https://www.highcharts.com/>

Jonny Strömberg (s.f.). List.js. Recuperado 3 mayo, 2019, de <https://listjs.com/>

Flask (A Python Microframework). (s.f.). Recuperado 3 mayo, 2019, de <http://flask.pocoo.org/>

Miguel Grinberg (s.f.). The Flask Mega-Tutorial Part I: Hello, World! - miguelgrinberg.com. Recuperado 3 mayo, 2019, de <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>

Stack Overflow en español (s.f.). Stack Overflow en español. Recuperado 3 mayo, 2019, de <https://es.stackoverflow.com/>

JSON. (s.f.). Recuperado 3 mayo, 2019, de <https://www.json.org/>

MongoDB Manual Contents — MongoDB Manual. (s.f.). Recuperado 3 mayo, 2019, de <https://docs.mongodb.com/manual/contents/>

Selenium Documentation — Selenium Documentation. (s.f.). Recuperado 3 mayo, 2019, de <https://www.seleniumhq.org/docs/>

Beautiful Soup Documentation — Beautiful Soup 4.4.0 documentation. (s.f.). Recuperado 3 mayo, 2019, de <https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

Twitter docs. (s.f.). Recuperado 3 mayo, 2019, de <https://developer.twitter.com/en/docs.html>

Mark Otto, Jacob Thornton, and Bootstrap contributors (s.f.). Introduction. Recuperado 3 mayo, 2019, de <https://getbootstrap.com/docs/4.3/getting-started/introduction/>

JS Foundation - js.foundation (s.f.). jQuery API Documentation. Recuperado 3 mayo, 2019, de <https://api.jquery.com/>



# 11. Anexos

---

## Anexo 1: Volcado de variables al fichero de estadísticas.

### Creación del diccionario con las variables de las estadísticas

La creación del diccionario con las variables de las estadísticas se desarrolla con el uso de la reflexión del módulo de “global\_variables.py”, es decir, en primer lugar “se pregunta al propio módulo que variables tiene”, para esto, usaremos el método `globals()` que nos devolverá todas las variables globales del módulo en el que se ejecute (`global_variables`) pero también contendrá funciones, variables internas y especiales por lo que deberemos “filtrarlas”.

Así pues, obtendremos un diccionario con las variables globales en el cuál las claves son los nombres de las variables (strings) y los valores, los valores que estas variables toman actualmente. Cabe destacar, que modificar los valores de este diccionario, conlleva cambiar el valor de las variables.

Se ha creado una función `check_variable_conditions` que dado una clave y un valor, devuelve `true` si es una variable que queremos almacenar en el fichero de estadísticas o `false` en caso contrario.

Para esto, hemos fijados que la clave cumpla unas condiciones:

- No empezar por ‘\_’
- No estar entre las siguientes variables puesto que no queremos almacenarlas, "tmp", "In", "Out", "ID", "AMOUNT".
- No estar en la lista de variables que queremos excluir.

El valor debe cumplir:

- No tener método “`__call__`” (no ser una función)

La siguiente Figura nos muestra la implementación:

```
def check_variable_conditions(k,v):
    accepted_dicts = ["users_dict","tweets_owner_dict","tweets_embed_html_dict",
                      "verified_account_dict_tweets","not_verified_account_dict_tweets"]
    key_conditions = not k.startswith('_') and k not in ["tmp","In","Out","ID","AMOUNT"]
                      and (not "dict" in k or (k in accepted_dicts))

    # si no es una funcion
    value_condition = not hasattr(v, '__call__')
    return key_conditions and value_condition
```

Con el método `get_statistics_dict()` obtendremos un diccionario con los nombres de las variables que nos interesan como clave y los valores de estas como valores como podemos observar en la siguiente Figura.

```
def get_statistics_dict():
    tmp = globals().copy()
    return { k : v for k,v in tmp.items() if check_variable_conditions(k,v)}
```

### Preproceso e inserción del fichero de estadísticas

Llamando a *mongo\_conector.insert\_statistics\_file\_in\_collection* dándole como parámetros el diccionario con las variables y la colección en la que se debe insertar, en primer lugar añadirá un nuevo campo “\_id”: “statistics\_file\_id” al diccionario para que mongo lo use como id y el campo ultima\_modificación con la fecha y hora actual.

Por último, dado que mongo no soporta claves con puntos, recorreremos el campo “way\_of\_send\_counter” que contiene un diccionario con las distintas proveniencias de los *tweets* y su número de *tweets* cambiando los puntos (‘.’) dentro de sus claves por bullets (‘•’). Esto se revertirá en la carga del fichero de estadísticas. Tras esto, lo insertará reemplazando el anterior si lo hubiera.

### Anexo 2: Carga del fichero de estadísticas.

La funcionalidad está implementada en el método *set\_statistics\_from\_statistics\_dict* del módulo **global\_variables.py**.

En primer lugar, se comprueba si el fichero de estadísticas contiene todas las variables que debería, sino es así, se nos da la opción de continuar pulsando “c” o abortar de otra manera.

Tras esto, por cada par (key, value) del fichero de estadísticas, modificamos su valor en el *globals()* de *global\_variables*.

Con este sencillo proceso las variables ya tendrán los valores del fichero de estadísticas.

## Anexo 3: Modos y opciones de la aplicación principal.

### Opciones principales línea de comandos

Opción	Explicación	Opciones adicionales
"-f", "--file"	Analiza los <i>tweets</i> de un fichero json dado.	
"-D", "-d", "--directory", "-DD", "-dd", "--directory_of_directories"	Analiza los <i>tweets</i> de los ficheros json de un directorio dado.  Analiza los <i>tweets</i> de los ficheros json de los subdirectorios un directorio dado.  <b>IMPORTANTE:</b> Si hay ficheros json en el directorio de directorios no serán analizados	
"-qf", "-QF", "--query_file"	Consume mensajes del api que cumplan la <i>query</i> escrita. Guarda los <i>tweets</i> en un fichero.	-o -mm
"-q", "-Q", "--query"	Consume mensajes del api que cumplan la <i>query</i> escrita. Guarda los mensajes en MongoDB	-mm -c
"-qu", "-QU", "--query_user"	Recupera <i>tweets</i> del api de uno o varios screen_name dados. Guarda los mensajes en MongoDB Añade o actualiza el registro en el fichero de usuarios buscados de la colección en la que se use.	-mm -p -c
"-s", "-S", "--streamming"	Consume <i>tweets</i> por <i>streamming</i> y los guarda en mongoDB.	-w -mt -mm -c
"-cq", "-CQ", "--collection_query"	Recupera el fichero de <i>queries</i> de la colección con la que se ejecute. Recorre las <i>queries</i> consumiendo <i>tweets</i> del api que las cumplan. Almacena los nuevos <i>tweets</i> en MongoDB.	-c -mm -l
"-cu", "-CU", "--collection_user"	Recupera el fichero de usuarios buscados de la colección con la que se ejecute. Recorre los usuarios consumiendo <i>tweets</i> del api que las cumplan. Almacena los nuevos <i>tweets</i> en MongoDB.	-c -mm -l
"-a", "--analyze"	Analiza los <i>tweets</i> no analizados de la colección.	-c -t -l --forced
"--likes"	Captura <i>likes</i> de mensajes de los usuarios buscados de la colección	-c -im -l -lr -likes_method -likes_parser

--likes_count	Realiza el conteo de los <i>likes</i> recuperados de la colección por partido.	-c --forced
MODO POR DEFECTO (AUSENCIA OPCIONES ANTERIORES)	Actualiza los <i>tweets</i> de una colección	-up -c

## Opciones adicionales línea de comandos

Opción	Explicación
"-up", "-UP", "--update"	Actualiza los json de mongoDB (solo disponible en el modo por defecto)
"-mm", "--max_messages"	Especifica el número de mensajes máximo a consumir.
"-mt", "-MT", "--max_time"	Especifica el número de minutos consumiendo con la opción "-s"
"-w", "-W", "--words"	Especifica palabras a partir de las cuales recuperar <i>tweets</i> . Solo es compatible con la opción "-s"
"-l", "-L", "--loop"	Especifica que se realizará la opción en bucle.
"p", "P", "--partido"	Especifica el partido de uno o varios usuarios. Solo está disponible con la opción "-qu"
"-e", "-E", "--examples"	Muestra ejemplos.
"-o", "--output_file"	Guarda los resultados en un fichero.
"-c", "-C", "--collection"	Establece la colección a usar (por defecto <i>tweets</i> ).
"-t", "-T", "--analysis_trunk"	Especifica el numero máximo de mensajes a analizar antes de cada actualización del fichero de estadísticas en el proceso de análisis.
"-lr", "--like_ratio"	Especifica el número de <i>likes</i> que un <i>tweet</i> debe obtener en 30 minutos para seguir visitándolo.
"-im", "--initial_messages"	Número de mensajes iniciales a comprobar en el proceso de <i>likes</i> .

--forced	En el proceso de análisis y en el proceso de conteo de <i>like</i> obliga a recalcular todo desde cero.
-likes_method	Selecciona el método de captura de <i>like</i> (puede ser tomando los n últimos mensajes de un usuario o con una cola dinámica).
-likes_parser	Especifica si se usará el navegador para capturar mensajes o no.

## 12. Glosario

---

- **Twitter** : Red social basada en el envío de mensajes de texto plano de corta longitud, con un máximo de 280 caracteres (originalmente 140), llamados tuits o *tweets*. Los usuarios pueden suscribirse a los *tweets* de otros usuarios (a esto se le llama "seguir" y a los usuarios que siguen a otra cuenta se les llama seguidores o *followers*)
- **API** : Una API es una interfaz de programación de aplicaciones (del inglés *Application Programming Interface*). Es un conjunto de rutinas que provee acceso a funciones de un determinado software. Utilizando estas rutinas se pueden obtener funcionalidades o datos sin saber como funciona internamente.
- **Streaming** : La retransmisión (en inglés *streaming* o transmisión, transmisión por secuencias, lectura en continuo, difusión en continuo o descarga continua) es la distribución digital de contenido multimedia a través de una red de computadoras, de manera que el usuario utiliza el producto a la vez que se descarga.
- **Like** : En el ámbito de las redes sociales representa que un usuario señaló que una publicación le gustó.
- **Python** : Python es un lenguaje de programación o cuya filosofía hace hincapié en una sintaxis que favorezca un código legible.  
Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.
- **Json** : JSON (acrónimo de *JavaScript Object Notation*, «notación de objeto de JavaScript») es un formato de texto sencillo para el intercambio de datos. Se trata de un subconjunto de la notación literal de objetos de JavaScript, aunque, debido a su amplia adopción como alternativa a XML, se considera un formato de lenguaje independiente.
- **HTTP** : El Protocolo de transferencia de hipertexto (en inglés: *Hypertext Transfer Protocol* ) es el protocolo de comunicación que permite las transferencias de información en la *World Wide Web*. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. El cliente realiza una petición enviando un mensaje, con cierto formato al servidor. El servidor le envía un mensaje de respuesta.
- **Petición HTTP GET** : El método GET solicita una representación del recurso especificado. Las solicitudes que usan GET solo deben recuperar datos y no deben tener ningún otro efecto.
- **Petición HTTP POST** : Envía los datos para que sean procesados por el recurso identificado. Los datos se incluirán en el cuerpo de la petición. Esto puede resultar en la creación de un nuevo recurso o de las actualizaciones de los recursos existentes o ambas cosas.
- **Frontend y backend** : Son términos que se refieren a la separación de intereses entre una capa de presentación y una capa de acceso a datos, respectivamente. Pueden traducirse al español el primero como interfaz, frontal final o frontal y el segundo como motor, dorsal final o zaga, aunque es común dejar estos por separado.

- HTML : HTML, siglas en inglés de *HyperText Markup Language* (lenguaje de marcas de hipertexto), hace referencia al lenguaje de marcado para la elaboración de páginas web. Es un estándar que sirve de referencia del software que conecta con la elaboración de páginas web en sus diferentes versiones, define una estructura básica y un código (denominado código HTML) para la definición de contenido de una página web, como texto, imágenes, videos, juegos, entre otros.