

# Trabajo fin de grado

Herramienta de descarga y análisis de datos de Twitter sobre participación ciudadana



José Antonio García del Saz

Escuela Politécnica Superior  
Universidad Autónoma de Madrid  
C\Francisco Tomás y Valiente nº 11



**UNIVERSIDAD AUTÓNOMA DE MADRID**  
**ESCUELA POLITÉCNICA SUPERIOR**



**Grado en Ingeniería Informática y Matemáticas**

**TRABAJO FIN DE GRADO**

**Herramienta de descarga y análisis de datos de  
Twitter sobre participación ciudadana**

**Autor: José Antonio García del Saz**

**Tutor: Iván Cantador Gutiérrez**

**junio 2019**

**Todos los derechos reservados.**

Queda prohibida, salvo excepción prevista en la Ley, cualquier forma de reproducción, distribución comunicación pública y transformación de esta obra sin contar con la autorización de los titulares de la propiedad intelectual.

La infracción de los derechos mencionados puede ser constitutiva de delito contra la propiedad intelectual (*arts. 270 y sgts. del Código Penal*).

DERECHOS RESERVADOS

© 20 de Junio de 2019 por UNIVERSIDAD AUTÓNOMA DE MADRID  
Francisco Tomás y Valiente, nº 1  
Madrid, 28049  
Spain

**José Antonio García del Saz**

*Herramienta de descarga y análisis de datos de Twitter sobre participación ciudadana*

**José Antonio García del Saz**

C\Las Torcas N°1 Portal 3 2ºB

IMPRESO EN ESPAÑA – PRINTED IN SPAIN

*A mi familia, mis amigos, mis compañeros y mis maestros*

*Si quieres aprender, enseña.*

*Cicerón*



# PREFACIO

---

Este estilo de  $\text{\LaTeX} 2_{\varepsilon}$  ha sido diseñado con dos propósitos. El primer propósito es el de facilitar en lo posible la escritura de trabajos de fin de grado y de máster y de tesis doctorales. En ese sentido se han diseñado un conjunto de comandos que simplifican la escritura y diseño de estos trabajos pero que reducen en cierta forma las capacidades de los paquetes de  $\text{\LaTeX}$  utilizados. Sin embargo, dado que los paquetes están incluidos en esta clase, pueden utilizarse directamente y hacer diseños más complejos pero si se hace esto se recomienda mantener una estética coherente con el resto del documento.

El segundo de los propósitos es que estos documentos mantengan una estética uniforme en la Universidad Autónoma de Madrid y fomentar una imagen corporativa en documentos tan relevantes como los trabajos de fin de grado o de máster y las tesis doctorales. Por ese motivo se recomienda mantener una coherencia estética en todo momento. El diseño facilita esa coherencia pero es posible salirse del diseño si se mantiene dicha coherencia.

Como creador de este estilo espero fervientemente que al usar este estilo te sientas cómodo y te facilite la escritura de un documento que es muy relevante en esta etapa de tu vida. Para facilitártela aún más, el código fuente de este documento también está disponible en tu ordenador o en overleaf para que te sirva a modo de ejemplo.

José Antonio García del Saz



# AGRADECIMIENTOS

---

En primer lugar me gustaría agradecer a la Escuela Politécnica Superior por su apoyo para la creación de esta clase y que sea el formato básico para la creación de tesis, trabajos fin de grado y trabajos fin de master.

En particular quiero destacar el trabajo realizado por Fernando López-Colino por su apoyo en la comisión de imagen institucional y por sus comentarios para mejorar este estilo.

También quiero tener un recuerdo para Carmen Navarrete Navarrete dado que este estilo comencé a crearlo a partir de sus necesidades a la hora de escribir la tesis. Y por supuesto a no quiero olvidarme de mi esposa e hijos que han servido de conejillos de indias en sis correspondientes trabajos fin de master y de grado. No quiero olvidar a todos los estudiantes que me pidieron este estilo y lo han usado para presentar sus trabajos pero son muchos y podría olvidarme de alguno, por tanto, mi agradecimiento en general a todos ellos.



# RESUMEN

---

En nuestra Escuela se producen un número considerable de documentos, tanto docentes como investigadores. Nuestros alumnos también contribuyen a esta producción a través de sus trabajos de fin de grado, máster y tesis. El objetivo de este material es facilitar la edición de todos estos documentos y a la vez fomentar nuestra imagen corporativa, facilitando la visibilidad y el reconocimiento de nuestro Centro.

En este sentido se ha intentado diseñar un estilo de  $\text{\LaTeX} 2_{\varepsilon}$  que mantenga una imagen corporativa y con comandos simples que permitan mantener la imagen corporativa con la calidad necesaria sin olvidar las necesidades del autor. Para ello se han creado un conjunto de comandos simples en torno a paquetes complejos. Estos comandos permiten realizar la mayoría de las operaciones que un documento de este tipo pueda necesitar.

Así mismo se puede controlar un poco el diseño del documento a través de las opciones del estilo pero siempre manteniendo la imagen institucional.

## PALABRAS CLAVE

---

Diseño de documento,  $\text{\LaTeX} 2_{\varepsilon}$ , thesis, trabajo fin de grado, trabajo fin de master



# ABSTRACT

---

In our School a considerable number of documents are produced, as many educational as research. Our students also contribute to this production through his final degree, master and thesis projects. The objective of this material is to facilitate the editing of all these documents and at the same time to promote our corporate image, facilitating the visibility and recognition of our center.

In this sense we have tried to design a style of  $\text{\LaTeX}2_{\varepsilon}$  that maintains a corporate image and with simple commands that allow to maintain the corporate image with the necessary quality without forgetting the needs of the author. For this, a set of simple commands have been created around complex packages. These commands allow you to perform most of the operations that a document of this type may need.

Likewise, you can control a little the design of the document through the options of the style but always maintaining the institutional image.

# KEYWORDS

---

Document design,  $\text{\LaTeX}2_{\varepsilon}$ , thesis, final degree project, final master project



# ÍNDICE

---

<b>1 Introducción</b>	<b>1</b>
1.1 Motivación del proyecto .....	1
1.2 Objetivos y enfoque .....	2
<b>2 Participación ciudadana, tecnología y redes sociales</b>	<b>3</b>
2.1 Estado del Arte .....	3
2.2 Tecnologías utilizadas .....	4
<b>3 Análisis de requisitos</b>	<b>5</b>
3.1 Requisitos funcionales .....	5
3.2 Requisitos no funcionales .....	7
<b>4 Concepción y diseño</b>	<b>9</b>
4.1 Modelo de datos .....	11
<b>5 Desarrollo y mantenimiento</b>	<b>19</b>
5.1 Mantenimiento .....	27
<b>6 Resultados y ejemplos</b>	<b>29</b>
<b>7 Conclusiones y trabajo futuro</b>	<b>31</b>
<b>Bibliografía</b>	<b>33</b>
<b>Definiciones</b>	<b>35</b>
<b>Acrónimos</b>	<b>37</b>
<b>Apéndices</b>	<b>39</b>
<b>A Diagramas</b>	<b>41</b>
<b>B Tablas</b>	<b>45</b>
<b>C Códigos</b>	<b>47</b>
<b>D Imágenes de la interfaz gráfica</b>	<b>63</b>



# LISTAS

---

## **Lista de algoritmos**

## **Lista de códigos**

5.1	Clase GenericService .....	21
5.2	Consulta para analizar el volumen de tweets diario .....	26
C.1	Clase GenericService completa (I) .....	48
C.2	Clase GenericService completa (II) .....	49
C.3	Clase AbstractGenericDAO completa (I) .....	50
C.4	Clase AbstractGenericDAO completa (II) .....	51
C.5	Anotaciones en la clase Extraction .....	52
C.6	Anotaciones de polimorfismo en la clase abstracta AnalyticsReport .....	53
C.7	Anotaciones de polimorfismo en la clase TrendingWordsReport .....	54
C.8	Anotaciones de polimorfismo en la clase AnalyticsReportRegister .....	55
C.9	Inicialización de la GUI .....	56
C.10	Carga de un diálogo modal .....	57
C.11	Clase TweetExtractorFXTask .....	58
C.12	Tarea concurrente que carga los tweets de una extracción .....	59
C.13	Lanzamos una tarea concurrente desde la GUI .....	60
C.14	Método de traducción Filtros <->Consulta .....	60
C.15	Método de consulta a la API de Twitter .....	61
C.16	Generando un Word Cloud con Kumo .....	62

## **Lista de cuadros**

## **Lista de ecuaciones**

## **Lista de figuras**

2.1	CONSUL en España y el mundo .....	3
-----	-----------------------------------	---

3.1	Matriz de compatibilidad reporte-gráfico .....	7
4.1	Diseño del entorno .....	10
4.2	Diagrama UML Principal .....	13
4.3	Ciclo de vida de una tarea asíncrona .....	14
4.4	Funcionamiento general de JFreeChart.....	15
4.5	Diagrama UML Preferencias Gráficos .....	16
4.6	Diagrama UML Análisis Semántico .....	18
5.1	Diagrama de funcionamiento Spring + Hibernate + JPA .....	20
5.2	Estructura de una aplicación JavaFX .....	22
5.3	Diagrama de funcionamiento de JavaFX Task .....	23
5.4	Diagrama conexión SOAP+HTTPS .....	24
A.1	Diagrama UML Tareas Asíncronas .....	42
A.2	Diagrama UML Reportes Análisis .....	43
D.1	Pantalla inicial .....	63
D.2	Registro e inicio de sesión .....	63
D.3	Pantalla de bienvenida .....	64
D.4	Constructor de consultas .....	64
D.5	Edición de credenciales .....	65
D.6	Tarea en segundo plano .....	65

## **Lista de tablas**

2.1	Tecnologías utilizadas .....	4
4.1	Operadores de Twitter API .....	11
5.1	Referencia de Filtros Disponibles .....	25
B.1	Idiomas soportados por la API de Twitter .....	45
B.2	Referencia de servicios web SOAP de TweetExtractorServer .....	46

## **Lista de cuadros**

# INTRODUCCIÓN

---

En este Trabajo de Fin de Grado se plantea el desarrollo de una herramienta software que permita la descarga automática desde Twitter de datos sobre participación ciudadana, y el posterior análisis de estos. La herramienta permitirá la configuración de parámetros de entrada para acotar el dominio, temáticas y alcance de los datos a descargar, así como el cálculo y visualización de una serie de gráficas, estadísticas y métricas a partir de los datos descargados. Como caso de uso, se propone evaluar la herramienta con tweets sobre problemáticas, propuestas y discusiones acerca de la ciudad de Madrid y su plataforma electrónica de presupuestos participativos ‘Decide Madrid’.

## 1.1. Motivación del proyecto

Tras los primeros encuentros con el tutor, éste fue introduciéndonos y desarrollándonos una idea sobre la cual él había estado pensando. Consistía en una plataforma o interfaz donde existiese la posibilidad de seleccionar, extraer, analizar y tratar datos (desde distintas fuentes) sobre la participación ciudadana en las ciudades.

La idea sería que cada “módulo” de dicho sistema se dedicase a una de las posibles fuentes y que fuese el propio sistema el que se ocupase de la recopilación, organización y visualización de los resultados.

Entre las fuentes susceptibles de contener información objeto de análisis se encuentran las tan populares redes sociales. Concretamente, en la ciudad de Madrid existe la plataforma Decide Madrid, donde se lanzan y votan propuestas e incluso se vota a qué proyectos se destina el dinero de los presupuestos municipales. Dicha plataforma tiene una cierta integración con Twitter, plataforma en la que los usuarios proponen iniciativas y también opinan y votan.

## 1.2. Objetivos y enfoque

### Lista de objetivos principales

- Desarrollo de herramienta para extraer datos acotables desde Twitter
  - Se usará el lenguaje de programación Java y la API REST de Twitter.
  - Se dotará al sistema de una interfaz gráfica sencilla desde la cual configurar fácilmente los datos a extraer, para después llevar a cabo las extracciones y poder visualizar los datos.
- Tratamiento de los datos
  - Se analizarán los datos extraídos para obtener reportes del volumen de datos, la naturaleza de los datos (analizando hashtags, usuarios, menciones, etc.), la semántica del texto, etc.
  - Se obtendrán representaciones de los reportes obtenidos por medio de gráficos y tablas.
- Extracción de conclusiones
  - Una vez terminado el tratamiento de los datos, se intentarán extraer conclusiones a partir de los resultados.
  - Se analizarán los potenciales resultados que tendría nuestra herramienta en otros escenarios o casos de uso.

# PARTICIPACIÓN CIUDADANA, TECONOLOGÍA Y REDES SOCIALES

---

## 2.1. Estado del Arte

Con la revolución tecnológica ha cambiado de forma sustancial la forma en que los ciudadanos interactuamos con nuestras ciudades y con nuestros vecinos y gobernantes. La explosión de las redes sociales (como Twitter) ha provocado que aparezca la posibilidad de interesar de forma directa (y también pública) a empresas, entidades públicas, individuos, etc, con lo que la diversidad de opiniones públicas y la participación ciudadana abogan por transformar nuestras democracias representativas en democracias más directas como las que ya existieron en Atenas o la República Romana, o como las existen hoy en Suiza.

Es por esto que han nacido proyectos como CONSUL, con los cuales se implementan interfaces que facilitan la participación ciudadana online a través de foros de opinión, de votaciones de propuestas on-line o de herramientas que se adaptan a cada ciudad (a sus distritos, barrios, comunidades, etc.)



(a) Ayuntamientos en España



(b) Ayuntamientos en el mundo

**Figura 2.1:** Ayuntamientos que utilizan el proyecto CONSUL para participación ciudadana.

Este tipo de herramienta ya está siendo utilizada por 33 países, 100 instituciones y 90 millones de ciudadanos en todo el mundo. En concreto en la ciudad de Madrid funciona desde 2015 bajo el nombre de Decide Madrid y está notablemente integrada con la red social Twitter (pueden compartirse a través de ella propuestas, apoyos, opiniones...).

Este tipo de herramientas, junto con las versátiles API de Twitter proporcionan la posibilidad de navegar en un universo de datos de los cuales se pueden formular hipótesis sobre temas tales como qué preocupa a los ciudadanos, cuáles son las medidas más o menos populares, qué usuarios son más activos y cómo se relacionan entre ellos, etc. Y todo esto en tiempo real.

## 2.2. Tecnologías utilizadas

Nombre	Versión	Objetivos
Twitter API	v4.0	Esta API REST nos la proporciona Twitter para acceder a sus datos. Con ella realizaremos consultas que nos permitirán obtener los datos que son sujeto de nuestro análisis
OpenJDK	v12.0.1	La versión OpenSource del más que conocido Java Developpement Kit. Nuestra aplicación se desarrollará en Java y el entorno gráfico lo gestionará JavaFX. El servidor es un proyecto Java EE
Twitter4J	v4.0.7	Una librería Java que ofrece métodos y clases para la explotación de la API de Twitter en el código Java
Spring Framework	v5.1.7	Framework de código abierto para el desarrollo de aplicaciones y contenedor de inversión de control. Nos permitirá compartir recursos entre los diferentes puntos de la aplicación a través de contextos.
Hibernate	v5.4.3	Herramienta de mapeo objeto-relacional que se apoyará sobre el driver jdbc para conectar los módulos de nuestra aplicación al servidor de bases de datos.
PostgreSQL Server	v11.3	Servidor de bases de datos que guardará y gestionará todos nuestros datos.
Apache Tomcat	v8.5.41	Servidor de aplicaciones Java donde estará desplegado el módulo servidor de nuestra aplicación.
Kumo API	v1.17	Librería Java que nos permite crear Word Clouds muy configurables a partir de palabras.
JFreeChart	v1.0.19	Librería para generar gráficos de distintos tipos en código Java. Se usará para mostrar resultados de los análisis.
Apache Lucene	v8.0.0	Librería para la recuperación de información desde texto y web para el código Java. Se usará para la tokenización y tratamiento de textos.
SSL/TLS	v1.2	Protocolo criptográfico que garantiza las conexiones seguras en la red. Se implementará en todas y cada una de las comunicaciones que se realicen entre cada módulo de nuestra aplicación.

**Tabla 2.1:** En esta tabla se citan las tecnologías utilizadas en el proyecto.

# ANÁLISIS DE REQUISITOS

Con el fin de facilitar la concepción y el desarrollo del sistema, se ha procedido a enumerar y clasificar los diferentes requisitos (tanto funcionales como no funcionales) que nuestro sistema debe satisfacer para que la configuración, el funcionamiento y los resultados se desarrollem según lo esperado.

## 3.1. Requisitos funcionales

### Autentificación

**RF-1.**– El acceso al sistema está restringido para usuarios registrados.

**RF-1.1.**– El registro es libre, pudiendo hacerse desde el propio sistema.

**RF-1.2.**– Las credenciales se componen de un nombre de usuario (único) y contraseña

**RF-2.**– Las contraseñas tendrán entre 6-16 caracteres y contendrá al menos una mayúscula, una minúscula y un número.

**RF-3.**– Un usuario que ha iniciado sesión puede cambiar su contraseña desde la GUI para los accesos posteriores.

**RF-4.**– Un usuario que ha iniciado sesión puede eliminar su cuenta, eliminando también todos los datos que hubiere almacenado en la base de datos (extracciones, reportes, gráficos, etc.).

### Extracciones

**RF-5.**– Un usuario puede crear extracciones desde la GUI que serán de su propiedad.

**RF-6.**– El perímetro de una extracción se delimita (durante la creación) a través de filtros, necesarios para la creación de la extracción (cada extracción contiene al menos un filtro).

**RF-7.**– Un usuario puede alimentar una extracción en primer plano (desde la GUI) o en segundo plano (con una tarea asíncrona del servidor).

**RF-8.**– Una extracción podrá alimentarse de forma indefinida en segundo plano.

**RF-9.**– Los tweets extraídos contenidos en una extracción pertenecen a ésta: si se elimina la extracción se eliminan los tweets.

**RF-10.**– Los tweets de las extracciones son consultables en crudo desde la GUI. También pueden eliminarse de forma individual desde la GUI.

**RF-11.**– Los tweets de una extracción se pueden exportar en un fichero XML para su integración externa.

**RF-12.**– Una misma extracción no podrá contener dos veces el mismo tweet.

## Credenciales de la API de Twitter

- RF-13.**– Un usuario puede añadir, modificar y eliminar credenciales para la API de Twitter en su cuenta.
- RF-14.**– El usuario debe tener al menos unos credenciales añadidos para poder comenzar a crear extracciones, tareas y reportes analíticos.

## Tareas asíncronas

- RF-15.**– Un módulo servidor web existirá para gestionar la ejecución de tareas asíncronas en segundo plano.
- RF-16.**– La conexión entre la GUI y el servidor es configurable desde la GUI, y esta configuración se guarda en el registro del sistema operativo.
- RF-17.**– La GUI se comunicará con el módulo servidor a través de servicios web (SOAP sobre SSL/TLS).
- RF-18.**– Las tareas asíncronas tienen un ciclo de vida específico que depende del tipo de tarea. (Ver diagrama 4.3)
- RF-19.**– Con el arranque del servidor, las tareas existentes son cargadas desde la base de datos.
- RF-20.**– Existen tipos de tareas que se ejecutan de forma indefinida si nunca son detenidas
- RF-21.**– El usuario puede crear, eliminar, preparar, lanzar y parar tareas asíncronas desde la GUI.
- RF-22.**– Una tarea puede ser programada para ejecutarse en un momento dado del futuro.
- RF-23.**– Tras un reinicio del servidor, las tareas programadas aún sin caducar deben ser reprogramadas automáticamente, las tareas que se estaban ejecutando deberán volver a ejecutarse automáticamente y las tareas que hayan caducado pasarán a dicho estado.
- RF-24.**– La ejecución de una tarea programada puede ser cancelada pasando dicha tarea al estado "preparada".

## Análisis

- RF-25.**– Un usuario puede crear, modificar y eliminar diversos tipos de reportes analíticos sobre los datos extraídos.
- RF-26.**– Los contenidos (registros) de un reporte pueden ser actualizados para no quedar obsoletos con el tiempo.
- RF-27.**– Un reporte puede ser actualizado en segundo plano con una tarea asíncrona de servidor.
- RF-28.**– Los datos de un reporte se pueden consultar en crudo en la GUI y exportarse en un archivo .csv para su integración externa.
- RF-29.**– Se guardarán en base de datos tanto el instante en que se creó el reporte como el instante en el que se actualizó por última vez.
- RF-30.**– Existen reportes sobre la evolución del volumen de tweets en el tiempo.
- RF-31.**– Existen reportes sobre las tendencias en las extracciones (hashtags, usuarios, menciones, términos...).
- RF-32.**– Existen reportes sobre la clasificación semántica de los tweets.

## Gráficos

- RF-33.**– Desde la GUI, un usuario puede crear, configurar, modificar y eliminar distintos tipos de gráficos que muestran los datos de los reportes disponibles.
- RF-34.**– Cada tipo de gráfico tiene unas configuraciones específicas (tipos de línea, colores, tamaño y estilo de fuentes, etc.) que serán parametrizables desde la GUI durante la creación del gráfico.
- RF-35.**– Las configuraciones de cada tipo de gráfico se guardan en base de datos para ser reutilizadas posteriormente.
- RF-36.**– Los gráficos pueden tanto verse desde la GUI como exportarse a un archivo JPEG.
- RF-37.**– Se podrán generar Word Clouds desde la GUI.
- RF-38.**– Existe una matriz de compatibilidad entre los tipos de reportes y los tipos de gráficos que son compatibles

entre sí:

Este diagrama es una matriz que muestra la compatibilidad entre diferentes tipos de reportes analíticos en las filas y diferentes tipos de gráficos en las columnas. Una diagonal negra de celdas marcadas con un 'X' indica que cada tipo de reporte es compatible consigo mismo. Las filas representan los tipos de reporte y las columnas representan los tipos de gráfico.

	Time Series Chart	XY Bar Chart	Category 3D Bar Chart	3D Pie Chart	Pie Chart	Word Cloud
Timeline Tweet Volume Report	X					
Timeline Top N Hashtags Report	X	X				
Trending Hashtags Reports	X	X	X			
Trending User Mentions Report	X	X	X	X		
Trending Users Report	X	X	X	X	X	
Trending Words Report	X	X	X	X	X	
Tweet Volume by Topics Report	X	X	X	X	X	
Tweet Volume by Named Entities Report	X	X	X	X	X	

**Figura 3.1:** Matriz de compatibilidad entre los tipos de reporte analítico y los tipos de gráficos.

### Procesamiento del lenguaje natural (NLP)

- RF-39.**– Para cada idioma, un usuario puede crear listas de palabras ignorables en ese idioma. De esta manera se permite personalizar las palabras irreverentes en cada contexto y en cada idioma.
- RF-40.**– Las palabras ignorables no se tendrán en cuenta para la medición de frecuencias o la elaboración de reportes relacionados con dichas frecuencias.
- RF-41.**– Para cada idioma, el usuario podrá crear, modificar y eliminar diferentes preferencias para el procesamiento del lenguaje natural desde la GUI. Dichas preferencias contienen una serie de categorías y subcategorías (temas) a través de las cuales luego se podrán clasificar los tweets.
- RF-42.**– Dado un conjunto de extracciones, un usuario puede separar todos sus tweets en palabras para obtener un conjunto de términos presentes en los tweets.
- RF-43.**– Los conjuntos de términos pueden crearse y eliminarse desde la GUI y se almacenan en la base de datos.
- RF-44.**– Dado un conjunto de términos y unas preferencias NLP, el usuario puede clasificar los términos en las diferentes categorías y subcategorías desde la GUI. De este modo se entrena la clasificación semántica de tweets en cada contexto.
- RF-45.**– La clasificación de los términos se puede hacer en paralelo (varios humanos usando la aplicación en varias máquinas) sin que se clasifique la misma palabra dos veces ni en serie ni en paralelo.

## 3.2. Requisitos no funcionales

### Entorno

- RNF-1.**– El sistema es distribuido, siendo sus módulos aislables y orientables.
- RNF-2.**– Las conexiones con la base de datos no se realizan en texto plano. Debe cifrarse punto a punto mediante SSL/TLS.
- RNF-3.**– Los diferentes módulos generan ficheros de log para el mejor trazado de las actividades.
- RNF-4.**– Los logs históricos se comprimen para optimizar el uso de almacenamiento.
- RNF-5.**– El sistema debe proporcionar mensajes de error que sean informativos y orientados a usuario final.
- RNF-6.**– Las tareas largas (acceso a datos, análisis, generación de reportes...) no bloquean la GUI completamente: se ejecutan en otro hilo mientras se muestra una ventana modal que informa de las acciones que ocurren en paralelo.

## Autentificación

**RNF-7.**– Las contraseñas de los usuarios no se pueden guardar en texto plano en la base de datos.

## Extracciones

**RNF-8.**– Una extracción no puede ser alimentada desde dos lugares distintos al mismo tiempo, ni siquiera desde la misma máquina.

**RNF-9.**– Se tendrán en cuenta las restricciones para las credenciales gratuitas que tiene la API de Twitter.

## Tareas asíncronas

**RNF-10.**– Una tarea no puede ejecutarse varias veces en paralelo, sólo se concibe una ejecución a la vez por tarea.

**RNF-11.**– Ningún intercambio de datos entre la GUI y el servidor se puede hacer en texto plano. Siempre se debe usar el cifrado punto a punto sobre SSL/TLS.

# CONCEPCIÓN Y DISEÑO

Dados los requisitos anteriores, llega el momento de decidir la forma en que vamos a satisfacerlos. En primer lugar se enumeraran los distintos módulos que se desarrollarán, la concepción del modelo de datos y las infraestructuras sobre las que se distribuirá el sistema.

## Aplicaciones ejecutables

Como nombre para nuestro sistema se ha elegido **TweetExtractor**, y contendrá dos aplicaciones:

- **TweetExtractorFX:** Es la interfaz gráfica de usuario. Comprende un conjunto de ventanas a través de las cuales se pueden realizar diferentes actos de gestión de la aplicación como gestión de usuarios, extracciones y preferencias; visualización de resultados, reportes y gráficos... Se ha elegido JavaFX (se desarrolla en Java+XML) como librería para la creación de la interfaz gráfica debido a que ya viene integrada con el JDK y también a su sencillez, versatilidad e integración con el back-end en Java. Gracias a Java, podremos ejecutarla en todos los sistemas operativos que lo soporten.
- **TweetExtractor Server:** Es una aplicación web que funciona a modo de servidor. Sigue ejecutándose aunque la interfaz gráfica esté cerrada y se encarga de gestionar las tareas asíncronas (las crea, las modifica, controla su ejecución, las elimina...). Se desarrollará en Java y se desplegará en un servidor de aplicaciones Apache Tomcat. Se comunicará con TweetExtractorFX a través de servicios web Java, que funcionarán sobre la infraestructura de los protocolos SOAP + HTTP + SSL/TLS.

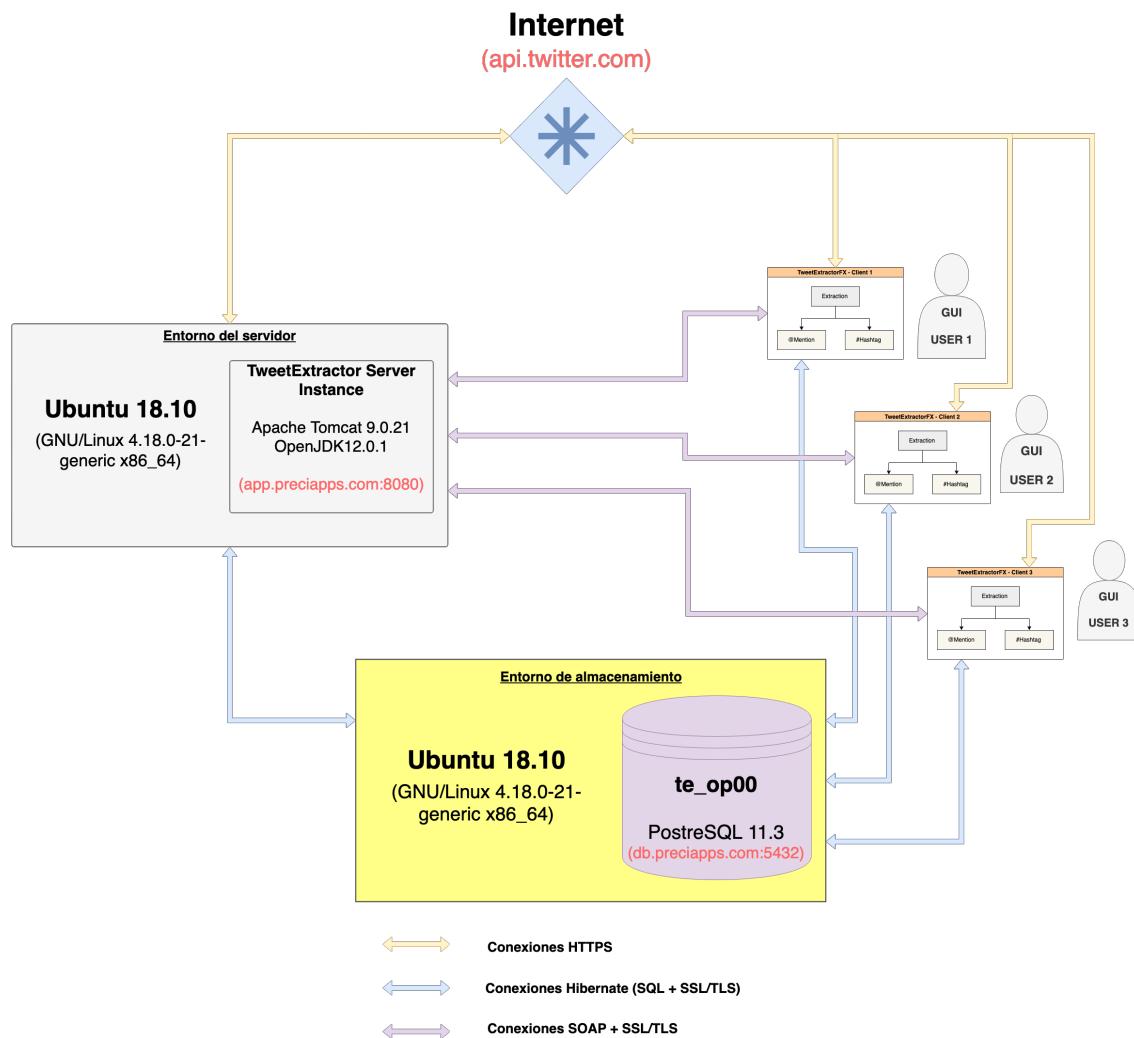
## Entornos

Como podemos observar en el diagrama inferior (4.1), podemos separar los entornos en 3:

- **Entornos de usuario:** Son los entornos (que pueden ser distintos) donde se ejecutaría la interfaz gráfica. Debe tener conexión a Internet para conectarse a Twitter. Se limitarán los entornos compatibles para acotar la fase de pruebas: sólo se probará en Ubuntu (GNU/Linux), macOS y Microsoft Windows con una versión de Java superior a 1.8.
- **Entorno del servidor:** Es el entorno que más carga de memoria soportará. Debe tener conexión a Internet para conectarse a Twitter. Puede ser cualquier entorno donde se pueda desplegar un contenedor de aplicaciones web de Java. En nuestro caso se ha elegido un servidor Linux con una instancia de Apache Tomcat 9.0.21. Se podrá acceder desde Internet a nuestra instancia a través del nombre de dominio app.preciapps.com en el puerto 8080.

- **Entorno de almacenamiento:** Es el entorno que más carga de almacenamiento y conexiones de red soportará. En él se ejecutará una instancia de servidor de base de datos PostgreSQL. En nuestro caso, hemos optado también por un servidor Linux con una instancia PostgreSQL v11.3. Se podrá acceder desde Internet a nuestra instancia a través del nombre de dominio db.preciapps.com en el puerto 5432.

El diagrama siguiente trata de desglosar y hacer más fácilmente comprensible la configuración de los entornos y las interconexiones entre ellos. Nótese que los distintos entornos pueden también fusionarse (incluso funcionar en una única máquina), pero optaremos por no hacerlo por motivos de rendimiento (con los entornos separados la optimización de cada uno de ellos es más sencilla).



**Figura 4.1:** Diagrama que muestra el diseño del entorno, con los diferentes módulos y las conexiones entre ellos.

## 4.1. Modelo de datos

### Extracciones y filtros

Esta es una parte muy importante de la construcción del modelo de datos. Comenzaremos creando la clase Extracción, que pertenecerá al usuario que la creó y contendrá una lista de tweets extraídos desde Twitter.

Siguiendo los requisitos funcionales, un usuario debe poder configurar qué condiciones deben cumplir los tweets que desea extraer. En este sentido tenemos que tomar como referencia la API de Twitter.

Con ella, podemos ejecutar consultas utilizando operadores, de los cuales algunos se enumeran en la documentación de la siguiente manera:

Operador	Encuentra Tweets...
viendo ahora	que contienen ambos términos “viendo” y “ahora”. Este es el operador principal.
“estado civil”	que contienen la frase exacta “estado civil”
amor OR odio	que contienen la palabra “amor” o bien la palabra “odio” (o ambas).
cerveza -raíz	que contienen la palabra “cerveza”, pero no contienen la palabra “raíz”
#haiku	que contienen el hashtag #haiku
from:interior	enviados desde la cuenta de Twitter “interior”
list:NASA/astronauts-in-space-now	enviado desde una cuenta de twitter en la lista “astronauts-in-space-now” de NASA
to:NASA	enviados en respuesta a la cuenta de Twitter “NASA”
@NASA	mentionando a la cuenta de Twitter “NASA”
política -filter:safe	que contienen “política” pero no están marcados como potencialmente sensibles
año -filter:retweets	que contienen “año” pero no son retweets.
madrid filter:images	que contienen “madrid” y tienen una imagen adjunta.
batería url:amazon	que contienen la palabra “batería” y una URL con la palabra “amazon” en cualquier lugar de ella.
as since:2015-12-21	que contienen “as” y fueron enviados desde el 21 de diciembre de 2015.
onu until:2015-12-21	que contienen “onu” y fueron enviados hasta el 21 de diciembre de 2015.
película :)	que contienen “película” y tienen una actitud positiva.
vuelo :(	que contienen “vuelo” y tienen una actitud negativa.
tráfico ?	que contienen “tráfico” y hacen una pregunta.

**Tabla 4.1:** En esta tabla se muestran algunos operadores que soportan las consultas a la API de Twitter.

Para modelar estos operadores se ha creado la clase Filtro. Una extracción contiene una lista de filtros (que serán configurables por el usuario desde la GUI), y cada uno de estos filtros se podrá traducir en una cadena de caracteres que se corresponderá con el texto de su operador análogo. Por tanto, al realizar la extracción podremos encadenar la lista de operadores extraída desde la lista de filtros y construir la consulta. Tras un análisis de los operadores distinguimos dos grandes tipos de filtros:

- **Filtros lógicos:** Se corresponden con los operadores lógicos “OR” y “NOT” (el operador “AND” ya equivale a concatenar dos operadores). Son filtros que actúan como operadores entre filtros (“OR” es un operador que actúa sobre dos filtros y “NOT” sobre un filtro).
- **Filtros no lógicos:** Son el resto de filtros que no actúan como operadores sobre otros filtros sino que actúan (metafóricamente) como constantes. Una extracción debe contener al menos uno de estos filtros para poder lanzar una consulta.

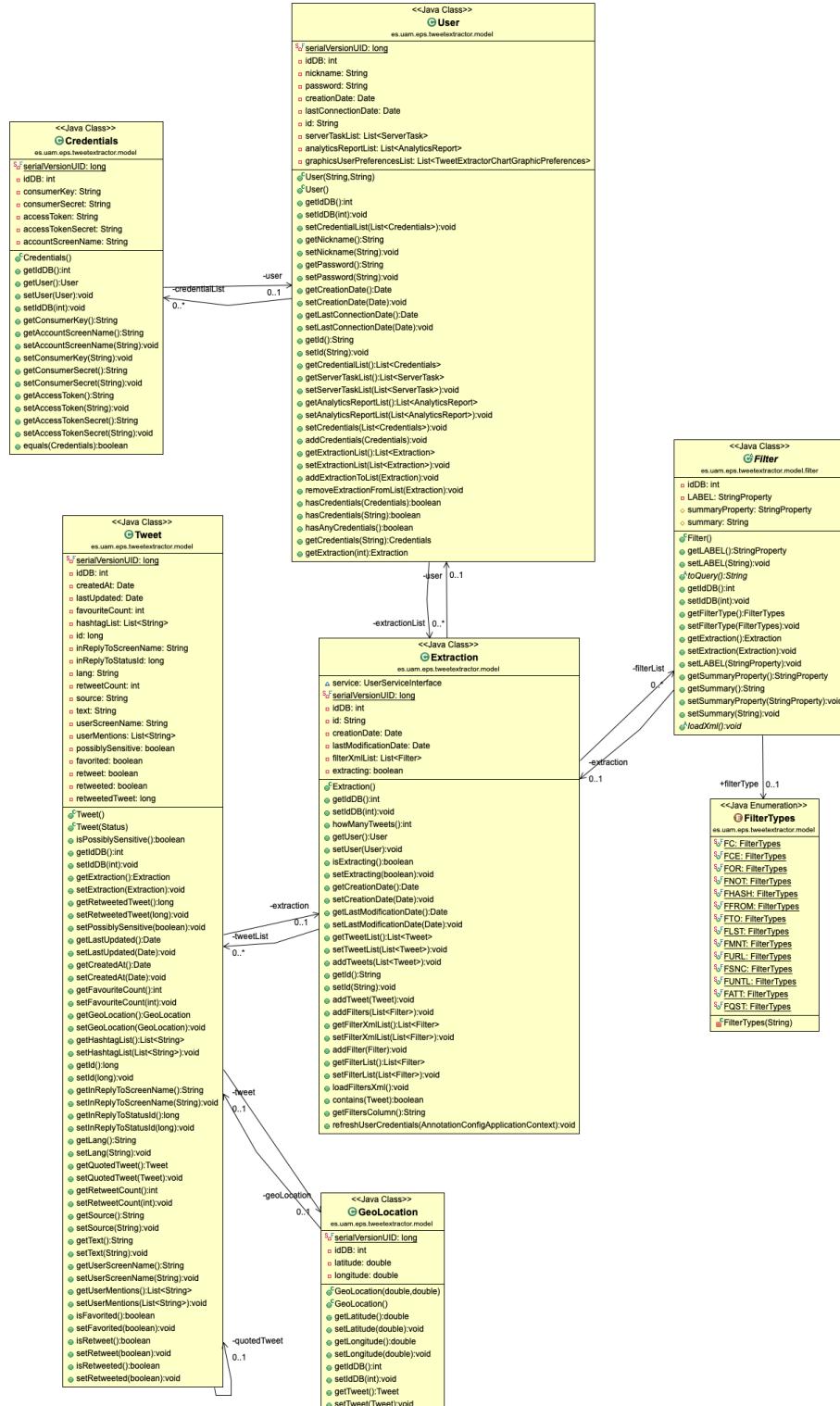
## Usuarios y credenciales (locales)

Para cumplir con los requisitos funcionales de autentificación, crearemos clases Java que representen a los distintos usuarios que se pueden registrar y autenticar en nuestro sistema. Cada usuario estará identificado por un nombre único y una contraseña que usará para acceder a la GUI.

## Tweets y Geolocalizaciones

Para modelar un tweet hemos creado la clase Java Tweet. Esta clase contendrá como atributos los datos y metadatos que nos parece interesante guardar para cada tweet extraído (nos hemos basado en la clase Status de la librería Twitter4J, que también encapsula los tweets):

- Fecha de creación
- Recuento de veces marcado como favorito
- Recuento de veces retuiteado
- Lista de hashtags contenidos
- ID de Twitter: es un número entero que identifica inequívocamente un Tweet.
- Cuenta a la que responde el Tweet (si procede)
- Tweet original al que responde el tweet (si procede).
- Idioma del tweet (si se reconoce).
- Origen del tweet (iOS, Android, Twitter Web, etc.)
- Cuerpo del tweet (texto)
- Cuenta desde la que se envía el tweet
- Lista de menciones a otras cuentas
- Marcador de tweets potencialmente sensibles
- Geolocalización desde la cual se envió el tweet (si está disponible).
- Añadiremos un identificador numérico para nuestra base de datos aparte del que nos proporciona Twitter (por que nosotros podemos guardar el mismo tweet dos veces en dos extracciones diferentes)



**Figura 4.2:** Diagrama UML de las clases principales del modelo de datos. Contiene las clases Usuario, Extracción, Tweet, Filtro, Credenciales, Geolocalización y Filtro

## Credenciales de la API de Twitter

Para modelar unos credenciales de la API de Twitter hemos creado la clase Credenciales, la cual contendrá las cuatro cadenas de caracteres o “tokens” que nos autentifican en el servicio, así como el nombre público de la cuenta.

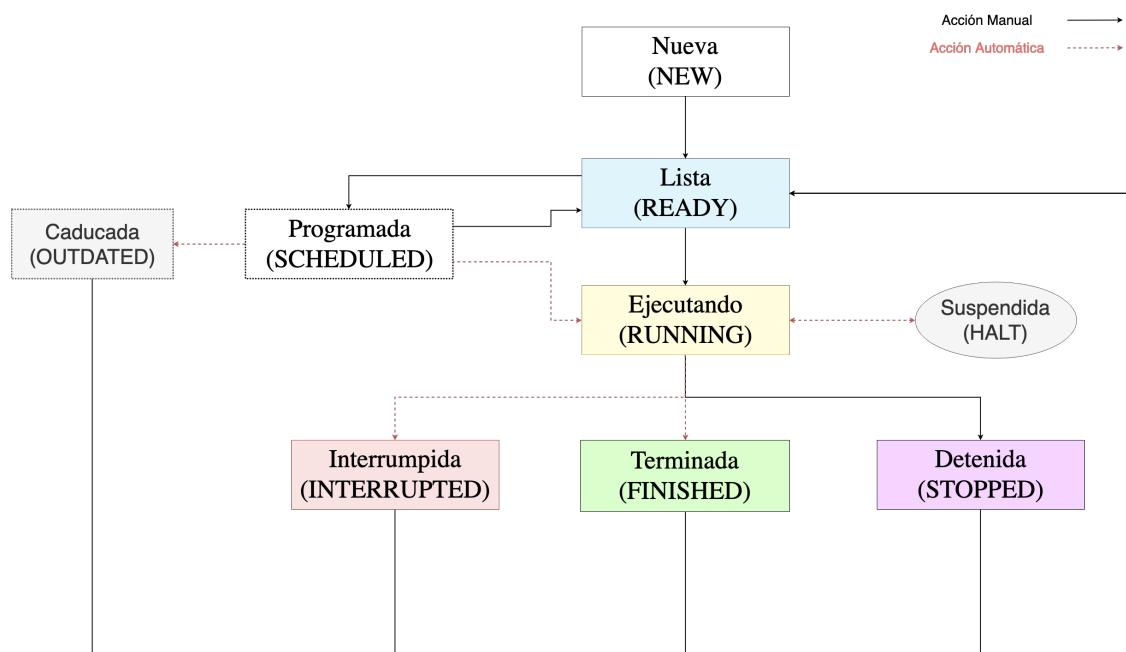
## Tareas asíncronas del servidor

Para modelar las tareas asíncronas y sus diferentes tipos se creará la clase TareaServidor y todas las clases que heredarán de ella. Según su funcionalidad, se modelarán dos grandes tipos de tareas:

- **Tareas de extracción:** Tareas que realizarán alguna acción concreta sobre una extracción, por ejemplo alimentar una extracción indefinidamente (ver RF-8).
- **Tareas de análisis:** Se encargarán de generar y/o actualizar reportes analíticos sobre los tweets ya extraídos.

El diagrama UML completo conteniendo los tipos de tareas está disponible en el anexo (ver A.1)

Las tareas tendrán un ciclo de vida muy concreto que se modelará mediante unos estados en los que puede estar la tarea dependiendo de su tipo, de las acciones manuales de los usuarios y de los eventos que puedan ocurrir durante sus ejecuciones. Este ciclo de vida se muestra en el siguiente diagrama:



**Figura 4.3:** Ciclo de vida de una tarea asíncrona en el servidor. Las líneas continuas marcan acciones manuales (del usuario). Las líneas discontinuas con color marcan acciones automáticas (del servidor)

Una tarea tendrá el estado “Nueva” cuando sea creada y podrá ser eliminada en cualquier estado. Los estados rodeados por línea discontinua son específicos de algún tipo de tarea (el estado

“Programada” sólo sera alcanzable para las tareas programables).

## Reportes Analíticos y Registros

Siguiendo los requisitos funcionales, una vez extraídos y guardados los tweets se podrán crear, modificar y eliminar diferentes tipos de reportes que contendrán los resultados de los análisis que realicemos sobre los datos.

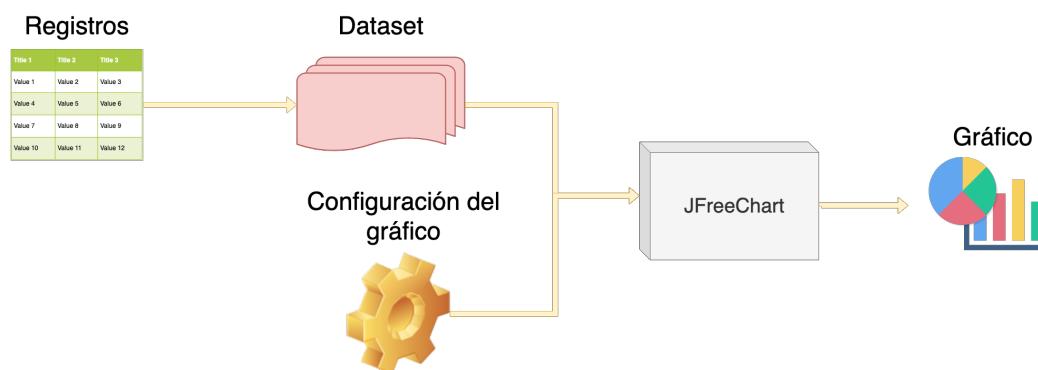
Para modelar todos los tipos de reportes se creará la clase ReporteAnalítico y todas las clases que heredarán de ella. Cada distinto tipo de reporte tendrá unos registros que contendrán los resultados, siendo estos registros también de tipos diversos dependiendo del tipo de reporte al que pertenezcan.

El diagrama UML que representa las clases e interfaces más relevantes que modelan estos reportes y registros se encuentra en los apéndices (ver A.2)

## Gráficos

Para la elaboración de gráficos se usará la librería externa JFreeChart. Esta librería nos permitirá, en nuestro entorno Java, configurar y generar gráficos sencillos que mostrarán los resultados de los reportes analíticos que hayamos generado. Soporta distintos tipos de gráficas (barras, línea, circular,etc).

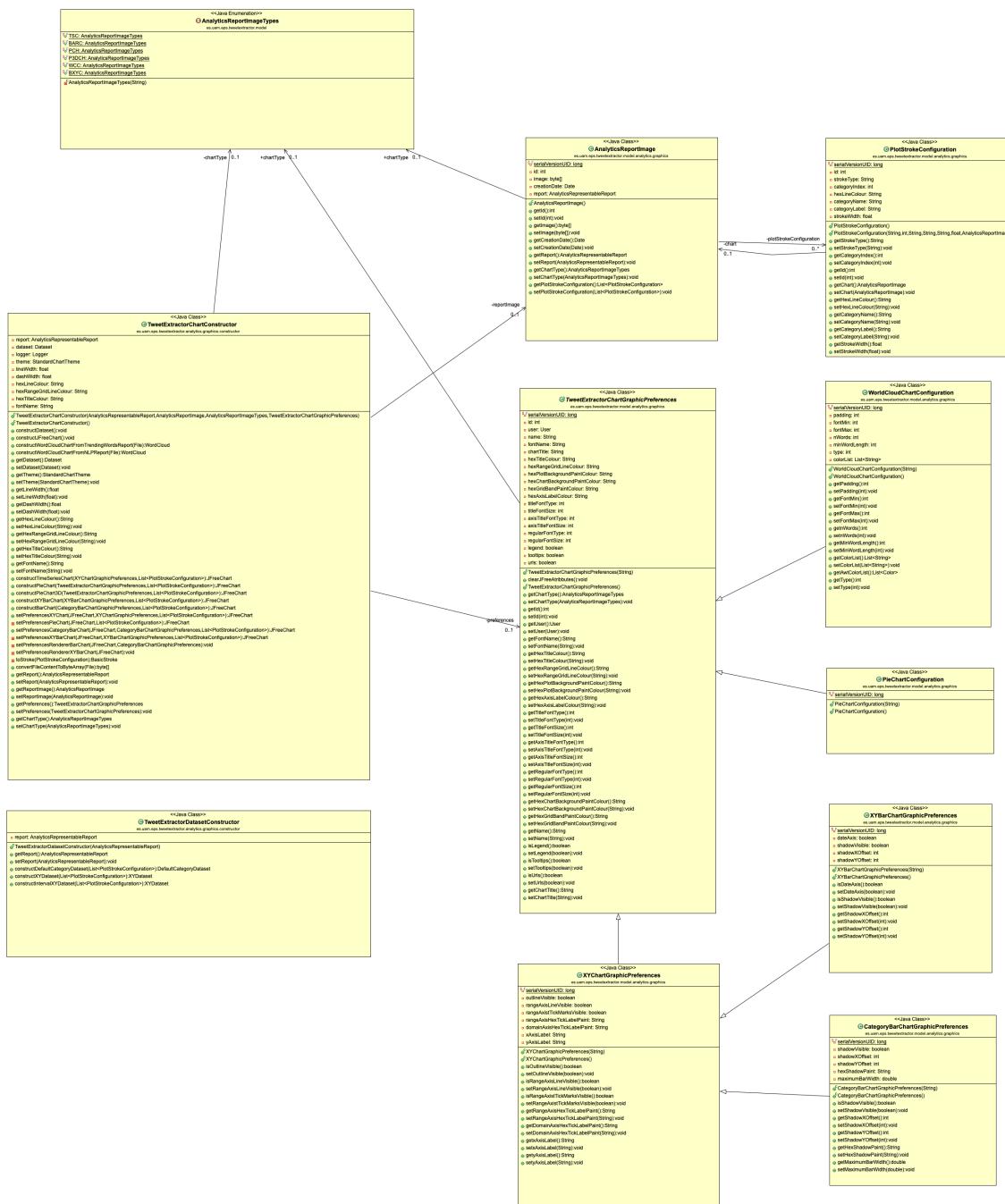
Para generar un gráfico, esta librería nos ofrece un constructor que requiere un conjunto de datos o “Dataset” (clases incluidas en la librería que encapsularán los valores de los registros) y una configuración para el gráfico (leyendas, fuentes, tipos de linea, colores, etc.) Este funcionamiento se muestra en este diagrama:



**Figura 4.4:** Diagrama que muestra el proceso de generación de un gráfico con la librería JFreeChart

Para modelar los gráficos que se vayan generando se creará la clase GráficoReporteAnalítico (guardaremos las gráficas en base de datos). También crearemos clases para encapsular las configuraciones posibles de cada tipo de gráfico (estas clases heredarán de la clase TweetExtractorPreferencias-Gráfico).

A continuación aparece el diagrama UML completo que muestra esta parte del modelo:



**Figura 4.5:** Diagrama UML que muestra el modelo de datos que representan los gráficos de los reportes y las configuraciones para generarlos.

## Análisis semántico de los Tweets

Por último, y de acuerdo con el último apartado de los requisitos funcionales, se añadirán al modelo los objetos que se usarán para el análisis semántico de los tweets.

Si hablamos de semántica, lo primero que debemos tener en cuenta es el idioma del texto que vamos a analizar.

La API de Twitter ya nos indica de forma nativa en qué idioma está escrito cada tweet. En el anexo puede encontrarse una referencia de los idiomas que la API es capaz de identificar (ver B.1). Para modelar los idiomas se creará un referencial inmutable de idiomas encapsulados en la clase IdiomaTwitterDisponible.

Dado un idioma, se podrán crear distintas listas de palabras ignorables que no se tendrán en cuenta para el cálculo de frecuencias. Dado un idioma y un conjunto de extracciones, un usuario debe poder partir los tweets extraídos en términos y almacenarlos para su posterior clasificación, y para ello construiremos una clase ConjuntoTokensPersonal.

Dado un idioma un usuario también podrá crear diferentes configuraciones para la clasificación de los términos. Estas configuraciones se modelarán como un conjunto de categorías y subcategorías. Los términos obtenidos se podrán clasificar en estas categorías, y esta clasificación se almacenará también en base de datos para tenerla en cuenta en los posteriores análisis.

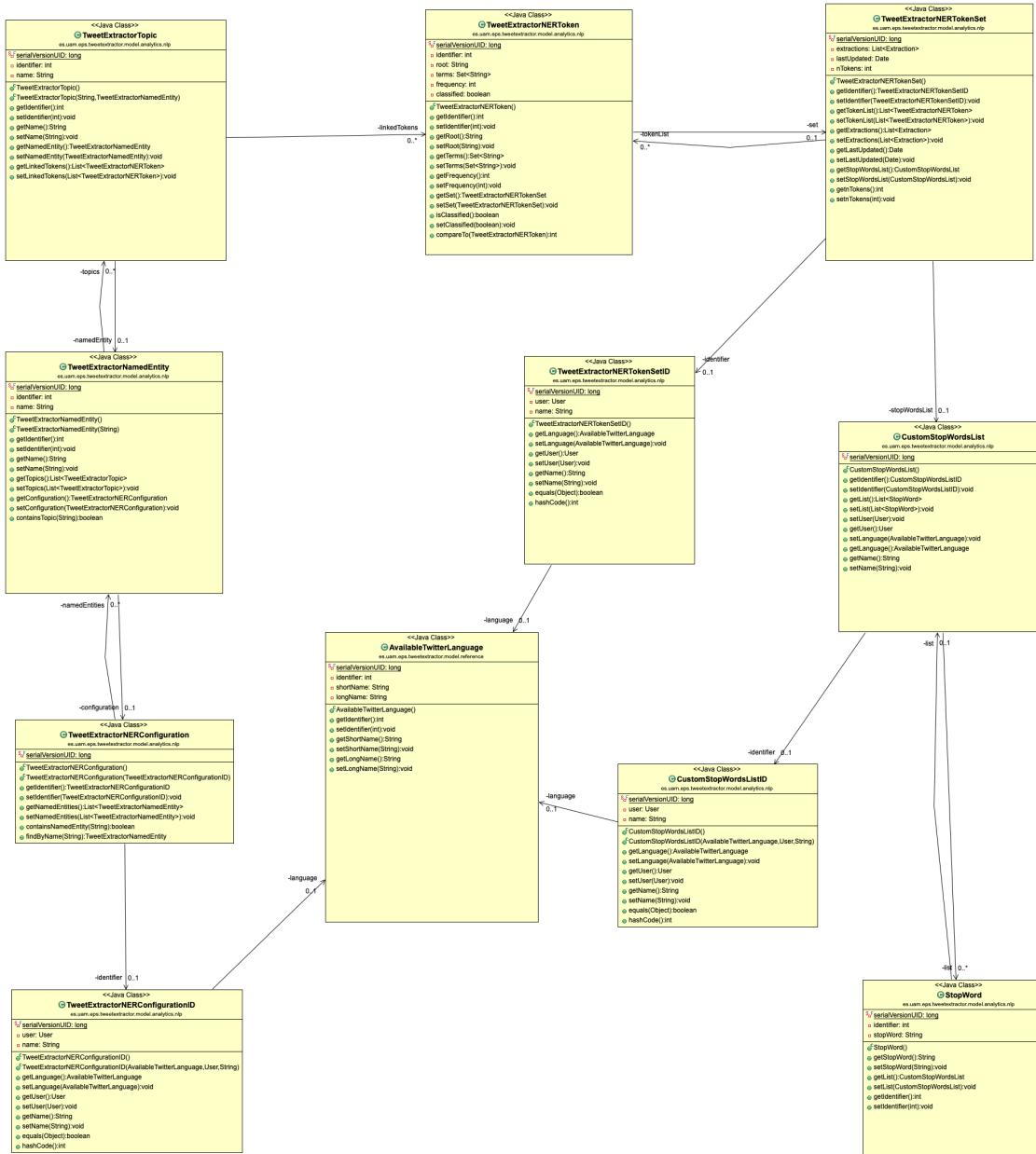
En la figura (ver 4.6) se puede apreciar el diagrama UML completo que contiene las clases que se usarán para modelar estos objetos, así como sus relaciones entre ellas.

## Base de datos

Gracias a nuestra decisión de utilizar Hibernate para gestionar el mapeo objeto-relacional de nuestro sistema, no tendremos que preocuparnos por la creación y configuración de las tablas de la base de datos.

Una vez diseñado como está nuestro modelo, solo tendremos que añadir anotaciones en las clases durante el desarrollo para que Hibernate reconozca todas nuestras clases, sus atributos, sus interrelaciones ,etc.

Ayudándose de estas anotaciones, Hibernate se encargará de generar el esquema, las tablas, etc. Sólo tenemos que facilitarle al framework las credenciales para acceder a la base de datos.



**Figura 4.6:** Diagrama UML que muestra el modelo de datos utilizado para modelar los objetos relacionados con el análisis semánticos de los datos.

# DESARROLLO Y MANTENIMIENTO

Vamos a desarrollar nuestra aplicación en el IDE Eclipse. Como herramienta de gestión y construcción de proyectos usaremos Maven, que nos permitirá de forma fácil añadir, actualizar y eliminar dependencias en nuestras aplicaciones, así como compilar todos los módulos y construir los ejecutables.

Todo el código estará disponible en este repositorio de GitHub, desde el que se pueden ver las distintas versiones de la aplicación y la evolución del código a lo largo del tiempo.

A continuación se describen los distintos módulos que contiene nuestro sistema:

- **tweetextractor-commons:** Es el módulo que contiene todas las librerías que son compartidas por el resto de módulos, es decir, las clases que son comunes. Al compilarlo obtendremos un contenedor .jar que será incluido como librería en los módulos que lo requieran. Contiene:
  - La configuración de conexión a la base de datos, así como todas las interfaces que necesitamos para interactuar con ella (DAO's y servicios).
  - La configuración del contexto aplicativo a través de SpringFramework.
  - Las interfaces de los servicios web SOAP.
  - La mayor parte del modelo de datos, así como la configuración para el mapeo objeto-relacional.
- **tweetextractor-fx:** Es el módulo que contendrá y generará la interfaz gráfica del usuario TweetExtractorFX. Al compilarlo generará un ejecutable .jar desde el cual se podrá acceder a la GUI. Contiene:
  - La vista de nuestra GUI. Se compone de ficheros .fxml que contienen las diferentes pantallas y diálogos que componen la interfaz gráfica.
  - El controlador de nuestra GUI. Cada elemento de la vista está conectado a su elemento controlador. Son clases Java que se encargarán de mediar entre la vista y el modelo.
  - JavaFX nos proporciona unas tareas (bastante similares a los Threads de Java) para realizar acciones más largas (como por ejemplo, recuperar 15000 palabras de la base de datos) en segundo plano sin bloquear la aplicación durante su ejecución. Hemos implementado muchas de las acciones duraderas a través de estas tareas.
  - La interfaz que conecta la GUI con la API de Twitter. Se usará para alimentar extracciones desde la GUI.
  - La interfaz que gestiona las preferencias de nuestra aplicación que se guardan en el registro del sistema operativo (como la conexión con el módulo servidor).

- **tweetextractor-server:** Es el módulo que contendrá y generará el servidor web de nuestro sistema: TweetExtractorServer. Al compilarlo, generará un contenedor web Java .war que podrá desplegarse sobre un servidor de aplicaciones. Contiene:

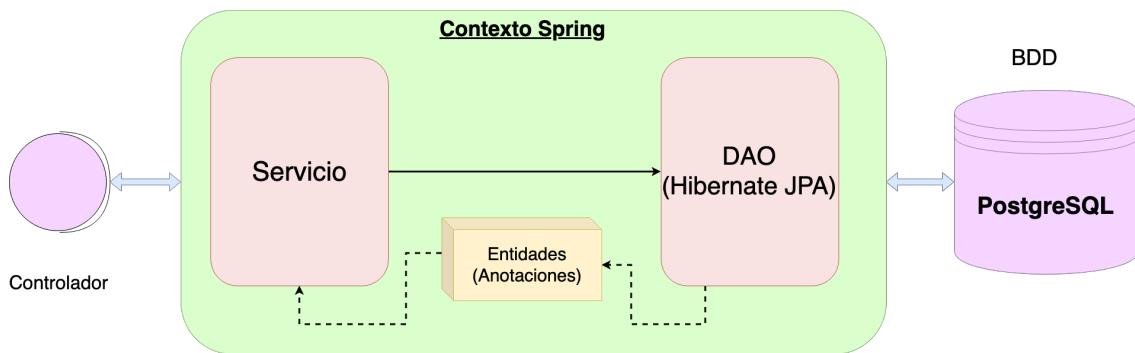
- El servidor. Este se encarga de la gestión de las tareas asíncronas del usuario (las carga, ejecuta, detiene, crea, elimina,...). Tiene en cuenta y lleva a cabo las peticiones de los distintos clientes que acceden a él.
- La implementación de los servicios web SOAP. Son las clases donde se especifican las acciones que se ejecutan en el servidor cuando algún cliente envía una petición desde la interfaz.
- La interfaz que conecta el servidor con la API de Twitter. Se usará para alimentar extracciones desde el servidor.

## Contextos: Hibernate + SpringFramework

Nuestro sistema utilizará contextos Spring en los que guardaremos objetos (o “beans”, como se les conoce en el universo Spring) que son accesibles desde cualquier parte del sistema y que son construidos, modificados y liberados según las necesidades de forma automática.

Los beans que nos interesa declarar en nuestro caso son nuestros servicios que nos otorgan los distintos métodos para encapsular el acceso a base de datos.

Gracias a Spring, cuando vamos a usar un servicio este es automáticamente construido y entregado desde el contexto (están todos marcados con la anotación @Service). Entonces Spring inyecta (gracias a la anotación @Autowired) en nuestro servicio el objeto de acceso a datos (DAO) correspondiente al tipo de servicio y este ya está listo para ser utilizado. Este funcionamiento puede observarse en el diagrama que se muestra a continuación:



**Figura 5.1:** Diagrama que muestra las distintas capas del funcionamiento de nuestros accesos a la base de datos. Se distinguen los servicios (nivel entidad), los objetos de acceso a datos (nivel DAO), y los objetos relacionales en sí (las tablas de la base de datos).

Todos los métodos de nuestros servicios serán transaccionales ayudándonos de la anotación @Transactional. El código completo del DAO y Servicio genéricos se encuentran en el anexo (ver apéndice C). A continuación se muestra un fragmento de código perteneciente al servicio genérico que ilustra estos últimos comentarios:

---

**Código 5.1:** Ejemplo de declaración, atributos y anotaciones de nuestro servicio genérico (al que extienden el resto de servicios específicos)

```
20  @Service
21  public abstract class GenericService<V extends Serializable, K extends Serializable>
22      implements GenericServiceInterface<V, K> {
23      @Autowired
24      private AbstractGenericDAO<V, K> genericDao;
25
26      public GenericService(AbstractGenericDAO<V,K> genericDao) {
27          this.genericDao=genericDao;
28      }
29
30      public GenericService() {
31      }
32
33      @Override
34      @Transactional(propagation = Propagation.REQUIRED)
35      public void saveOrUpdate(V entity) {
36          genericDao.saveOrUpdate(entity);
37      }
```

Una vez está lista la interfaz Objeto <->BDD, ahora sólo falta añadir anotaciones a las clases del modelo de datos para que Hibernate automáticamente cree y modifique las tablas de forma automática (ver apéndice C.5).

Hibernate junto con JPA nos proporciona también una excelente compatibilidad entre el polimorfismo Java y el mapeo objeto-relacional asociado a esta. Se nos ofrecen múltiples opciones: crear una tabla para cada tipo de clase que extiende a la clase abstracta (de este modo hibernate sabe directamente qué tipo de objeto debe construir para mapear una fila de dicha tabla), crear una misma tabla para todos los tipos (de este modo Hibernate usará un campo al que llamaremos discriminante que le indicará saber con qué clase se corresponde cada fila de la tabla conjunta), etc.

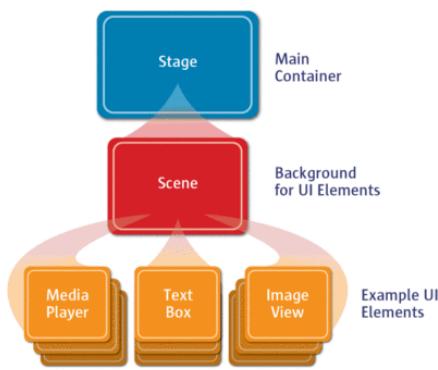
Un ejemplo de implementación de la herencia en tabla única es el de los reportes analíticos (todos se guardan en la tabla perm\_analytics\_report). Para distinguir de qué tipo es cada uno se usa un discriminante que contiene una cadena de caracteres que identifica a cada tipo en su clase correspondiente. Un fragmento del código está disponible en el anexo (ver apéndice C.6,7).

Por otro lado, un ejemplo de herencia implementado en una tabla para cada clase son los registros de estos reportes analíticos (ver apéndice C.8).

## GUI. JavaFX: Vista y controlador

Como habíamos indicado, la GUI ha sido construída con la librería JavaFX. Esta librería funciona con el diseño Modelo-Vista-Controlador. Nos permite diseñar cada pantalla o diálogo de la vista en un fichero XML (.fxml) de una forma muy similar a la que diseñaríamos una página web HTML (distintos tipos de objetos predefinidos con distintos atributos para controlarlos). Una vez lista la vista, la conectamos a un controlador java a través del cual gestionaremos la vista en contacto con el modelo (con ayuda de los servicios).

En el siguiente diagrama que nos proporciona Oracle en la documentación se puede ver de forma muy generalizada la estructura de una aplicación JavaFX:



**Figura 5.2:** Diagrama que muestra de forma muy generalizada la estructura de una aplicación JavaFX

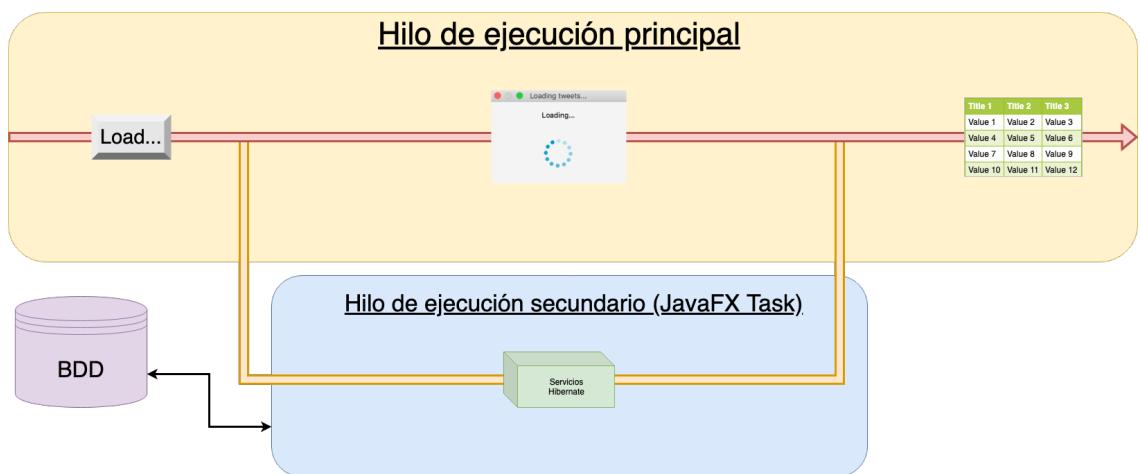
En TweetExtractorFX, la clase MainApplication creará el escenario (Stage) primario e incluirá en él una escena (Scene) que está controlada por nuestra clase RootLayoutControl. El RootLayout contiene la barra de menús de la aplicación en la parte superior, así como una escena central que irá cambiando dependiendo del elemento de la vista que mostremos en cada momento. Este funcionamiento se ve claramente en el código de los apéndices (ver apéndice C.9).

Unos elementos especiales de la vista son los diálogos, todos sus controladores extienden a nuestra clase TweetExtractorFXDialogController, y se muestran en modo ventana modal, es decir, mientras el diálogo se muestra sólo se puede interactuar con él, bloqueándose el resto de la aplicación. Además, por necesidad de algunos tipos de diálogos en los que se requiere al usuario una entrada de datos, sus controladores pueden devolver una respuesta a la aplicación principal al cerrarse el diálogo. En la aplicación principal hay un método que se encarga de mostrar estos diálogos y recoger sus respuestas (ver apéndice C.10).

Uno de los principales problemas del desarrollo de la interfaz gráfica de usuario es que si se tiene un flujo de aplicación en un solo hilo de ejecución, cuando se realice una acción no instantánea (accesos largos a base de datos, conexiones a servidores en remoto...) el hilo quedará ocupado (puesto que está realizando dicha acción) y la interfaz gráfica queda completamente bloqueada (desatendida por el hilo).

Para lidiar con este problema JavaFX nos ofrece una reinterpretación de los Thread de Java hecha en exclusiva para esta librería. Cuando necesitamos ejecutar una tarea no instantánea lo que haremos en el hilo principal será crear el tipo de tarea concreto que queremos realizar (nuestras tareas extienden todas a nuestra clase abstracta TweetExtractorFXTTask).

Una vez creada, le pasaremos los parámetros necesarios (si es que los necesita), crearemos un hilo paralelo en el que arrancamos la tarea y mostramos un diálogo de carga mientras esperamos que el hilo acabe y nos de una respuesta (a diferencia de los Treads de Java, las tareas de JavaFX devuelven un objeto de la clase que queramos). Este funcionamiento se puede ver en este diagrama:



**Figura 5.3:** Diagrama que muestra el flujo de ejecución cuando se utiliza un hilo paralelo para la ejecución de tareas largas (como el acceso a la base de datos) sin bloquear la GUI completamente.

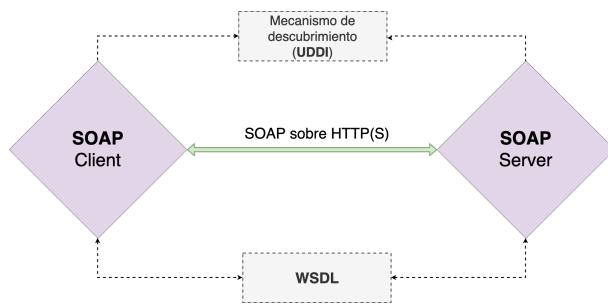
En los apéndices se puede encontrar el código de nuestra clase abstracta que encapsula las tareas concurrentes (ver apéndice C.11), un ejemplo de tarea concurrente desarrollada que se encarga de conectarse a la base de datos para recuperar los tweets de una extracción (ver apéndice C.12), y el código que lanza y espera a esta tarea desde la GUI (ver apéndice C.13).

## Servidor: Tareas asíncronas y servicios web SOAP

La aplicación TweetExtractorServer se compone de una clase principal (homónima) que crea y carga el contexto del servidor al arrancarlo, gestionarlo mientras esté funcionando y destruirlo al apagar el servidor. Nace dada la necesidad de que algunas tareas deban poder ser ejecutadas en segundo plano aún estando cerrada la GUI (RF-15).

Todas las tareas asíncronas son cargadas por el servidor al arranque, y éste las mantiene almacenadas en una lista Java. El servidor tiene métodos por los cuales se pueden realizar distintas acciones con las tareas (añadir, eliminar, lanzar, detener...).

Para acceder a estos métodos exclusivos del servidor, un usuario debe conectarse al servidor desde la GUI por medio de los servicios web SOAP. Para la implementación de estos servicios hemos optado por JAX-WS, montando el protocolo SOAP directamente sobre HTTPS, como se puede ver en este diagrama:



**Figura 5.4:** Diagrama que pone en contraste la conexión SOAP tradicional (con los descriptores WSDL y el descubrimiento de métodos) con la conexión directa sobre el protocolo HTTP.

Una tabla de referencia con todos los servicios web implementados, incluyendo una breve descripción de sus funcionalidades y un enlace a sus descriptores WSDL (a los que se accede por HTTPS obviamente) está disponible en los apéndices de esta memoria (ver B.2).

## Encapsulado XML

Tanto para la transmisión de objetos a través de los servicios web como para la exportación de datos a ficheros de formato XML, es necesario disponer de un mapeo Objeto <->XML. Para este fin se utiliza XML Bind, que nos proporciona unas anotaciones sobre las clases para automatizar el proceso (ver apéndice C.5).

---

## Filtros y consultas a la API de Twitter

Para la codificación de los filtros presentados en la sección anterior (ver 4.1), se crea la clase abstracta Filter de la que heredan todos los tipos de filtro que existen. Esta clase contiene el prototipo del método toQuery() que deberá implementar cada tipo de filtro. Los tipos de filtro implementados se enumeran a continuación:

Clase	Comentario	Operador API
FilterContains	Contiene una lista de palabras que queremos que contengan los tweets	palabra1 ... palabraN
FilterContainsExact	Contiene una expresión literal que queremos que contengan los tweets	"expresión"
FilterFrom	Contiene el nombre de una cuenta de Twitter	from:cuenta
FilterHashtag	Contiene una lista de hashtags que queremos que contengan los tweets	#hashtag1 ... #hashtagN
FilterList	Contiene una cuenta y el nombre de una lista en la que queremos que estén los tweets	list:cuenta/lista
FilterMention	Contiene el nombre de una cuenta que queremos que se mencione en los tweets	@cuenta
FilterOr	Contiene dos filtros y actúa de disyunción lógica entre ellos	filtro1 OR filtro2
FilterNot	Contiene un filtro y actúa de negación lógica sobre él	-filtro
FilterSince	Contiene una fecha desde la que queremos que se envíen los tweets	since:"YYYY-MM-DD"
FilterUntil	Contiene una fecha hasta la que queremos que se envíen los tweets	until:"YYYY-MM-DD"
FilterTo	Contiene el nombre de una cuenta de Twitter hacia la que queremos que vaya dirigido el tweet	to:cuenta

**Tabla 5.1:** En esta tabla se muestran los distintos tipos de filtro implementados, así como su traducción a los operadores de la API de Twitter

Una vez añadidos los filtros a una extracción desde la GUI (con el controlador QueryConstructorControl) se pueden traducir estos filtros a una consulta de la API de Twitter a través de un método de la clase estática FilterManager (ver apéndice C.14).

Con la consulta ya lista, utilizamos nuestra interfaz que se apoya en la librería Twitter4J para lanzar la consulta a Twitter y tratar la respuesta. Se gestionan también las distintas excepciones que pueden ocurrir (error de red, tasa temporal permitida por la API superada, error desconocido...) Parte del código que se utiliza para realizar estas consultas puede leerse en los apéndices (ver apéndice C.15).

## Reportes analíticos.

Tras modelar los distintos tipos de reportes y los diferentes tipos de registros que cada de ellos contiene, hay que diseñar la manera de analizar dichos resultados.

En la mayoría de las ocasiones se ha optado por realizar la consulta directamente a la base de datos a través del servicio TweetService, que contiene métodos para el análisis de los tweets como extractGlobalTimelineVolumeReport(User u), el cual nos devolvería un reporte del volumen de tweets por día que ha extraído un usuario. Para ello se utiliza la siguiente consulta SQL directamente en la base de datos:

**Código 5.2:** Consulta que se realiza para obtener como resultado el volumen de tweets que ha extraído un usuario cada día del año. Recibe el parámetro :user, que es el identificador del usuario que realiza la consulta.

```

1  SELECT date_part('year', t.creation_date) AS "Year",
2      date_part('month', t.creation_date) AS "Month",
3      date_part('day', t.creation_date) AS "Day",
4      COUNT(t.*) AS "Volume"
5  FROM perm_tweet t
6  JOIN perm_extraction e ON e.identifier=t.extraction_identifier
7  JOIN perm_user u ON e.user_identifier=u.identifier
8  WHERE u.identifier=:user
9  GROUP BY date_part('day', t.creation_date),
10      date_part('month', t.creation_date),
11      date_part('year', t.creation_date)
12 ORDER BY "Year","Month","Day";

```

## Gráficos : Word Cloud

Para cumplir el requisito RF-37, se ha importado la librería externa Kumo. Tiene un funcionamiento similar al que describimos de la librería JFreeChart (ver 4.4).

En este caso los registros de los reportes de frecuencia de palabras se encapsulan en una clase llamada FrequencyAnalyzer (proporcionada por la librería), en la que vamos introduciendo cada término y su frecuencia. Después, configuramos el gráfico con los atributos de nuestra clase WorldCloudChartConfiguration que encapsulará la parametrización de WordClouds.

Una vez con ambos elementos, podemos proceder a generar el gráfico y guardarlo en la base de datos. Como ejemplo se ha incluído en los apéndices el método que genera un WordCloud a partir de un reporte de frecuencia de términos (ver apéndice C.16).

Se da soporte a tres tipos de Word Cloud: circular, rectangular y bordear por píxeles (se le pasa una imagen con fondo transparente y generará un WordCloud con la forma de dicha imagen).

### Clase Constants

En la clase Constants se definen todos los datos inmutables que se utilizarán como referencia en nuestro sistema. Como es obvio, esta clase pertenece al módulo tweetextractor-commons ya que la consultan todos los módulos.

Contiene una serie de objetos Java inmutables (mayoritariamente con los atributos “final” y “static”) tales como enumeraciones (por ejemplo, los discriminantes para el polimorfismo Hibernate, que se explicarán en el siguiente apartado), mapeos entre números y cadenas de caracteres (por ejemplo, los estados de una tarea son números en la base de datos, mientras que nosotros vemos los estados como palabras en la GUI), la matriz de compatibilidad entre tipos de reportes y tipos de gráficos, etc.

### Encriptado de contraseñas en la base de datos

Para cumplir el requisito RNF-7 (ver 3.2), vamos a utilizar la herramienta BCrypt que viene incluída con el conjunto de herramientas de seguridad de SpringFramework.

Ayudándonos de las funciones hash (unidireccionales o de un solo sentido) que nos proporciona, podemos encriptar la contraseña de un usuario en el momento que se registra. De este modo, la contraseña se envía ya cifrada a la base de datos desde la GUI (y se cifra dos veces, debido a la conexión SSL/TLS para acceder a la base de datos).

Cuando el usuario quiere iniciar sesión, la contraseña que proporciona al sistema es rehasheada para comprobarla con el hash que ya teníamos guardado en base de datos, de este modo la contraseña nunca sale de ningún módulo del sistema sin cifrar.

El algoritmo criptográfico que usa BCrypt es Blowfish.

## 5.1. Mantenimiento

### SonarQube



## RESULTADOS Y EJEMPLOS

---



## CONCLUSIONES Y TRABAJO FUTURO

---



# BIBLIOGRAFÍA

---

[1] YUSUKE YAMAMOTO, *Twitter4J - A Java library for the Twitter API* Visitar.



# DEFINICIONES

---

**acrónimo** Sigla cuya configuración permite su pronunciación como una palabra; por ejemplo, ovni: objeto volador no identificado; TIC, tecnologías de la información y la comunicación.

**definición** Proposición que expone con claridad y exactitud los caracteres genéricos y diferenciales de algo material o inmaterial.

**opción de estilo** Son los valores que modifican el funcionamiento del estilo. Se ponen entre corchetes y separadas por comas en el comando \documentclass y antes de el nombre del estilo que irá entre llaves.



# ACRÓNIMOS

---

**IEEE** Institute of Electrical and Electronics Engineers.

**WYSIWYG** What You See Is What You Get.

**WYTIWYG** What You Think Is What You Get.



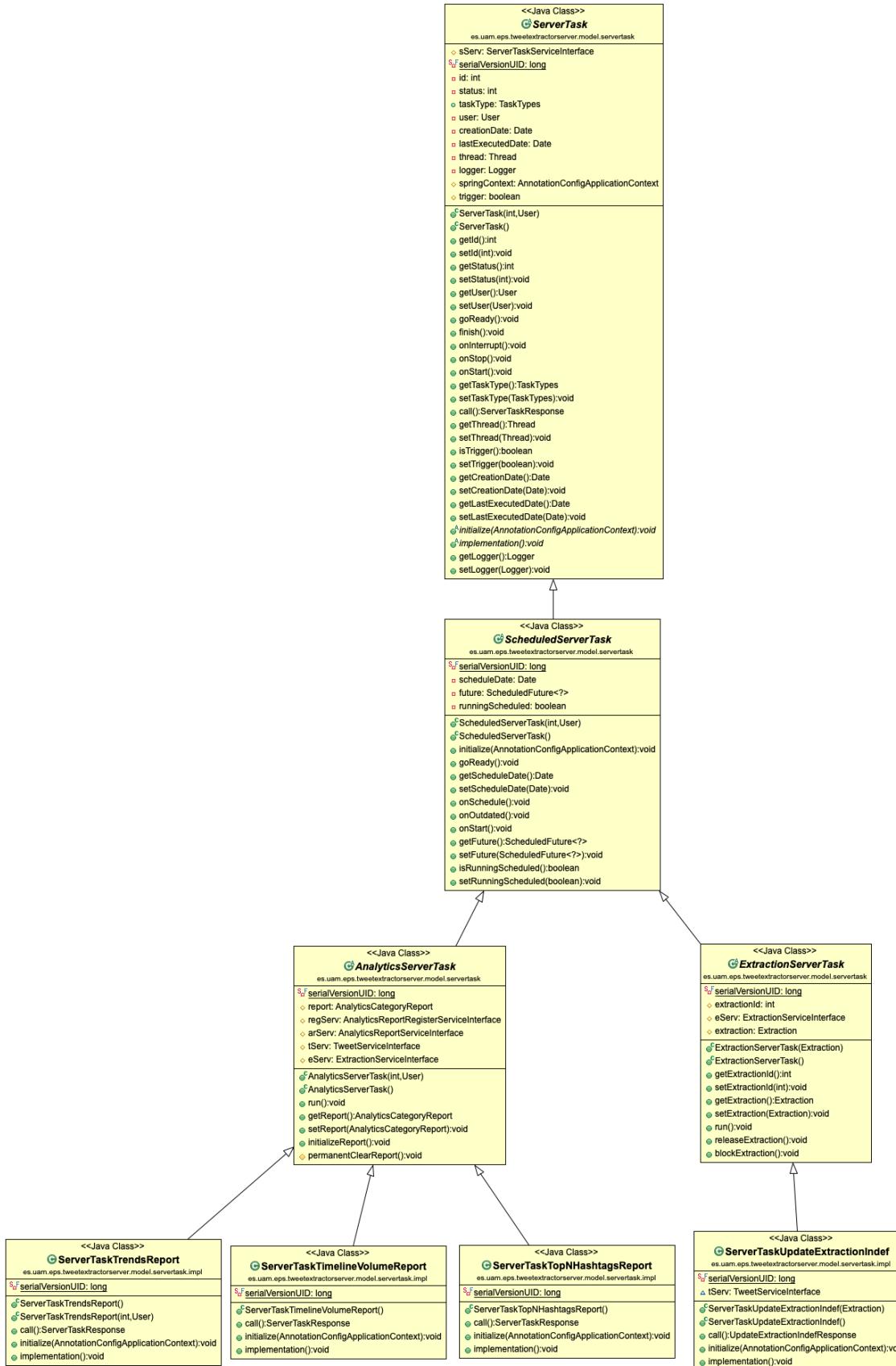
# **APÉNDICES**



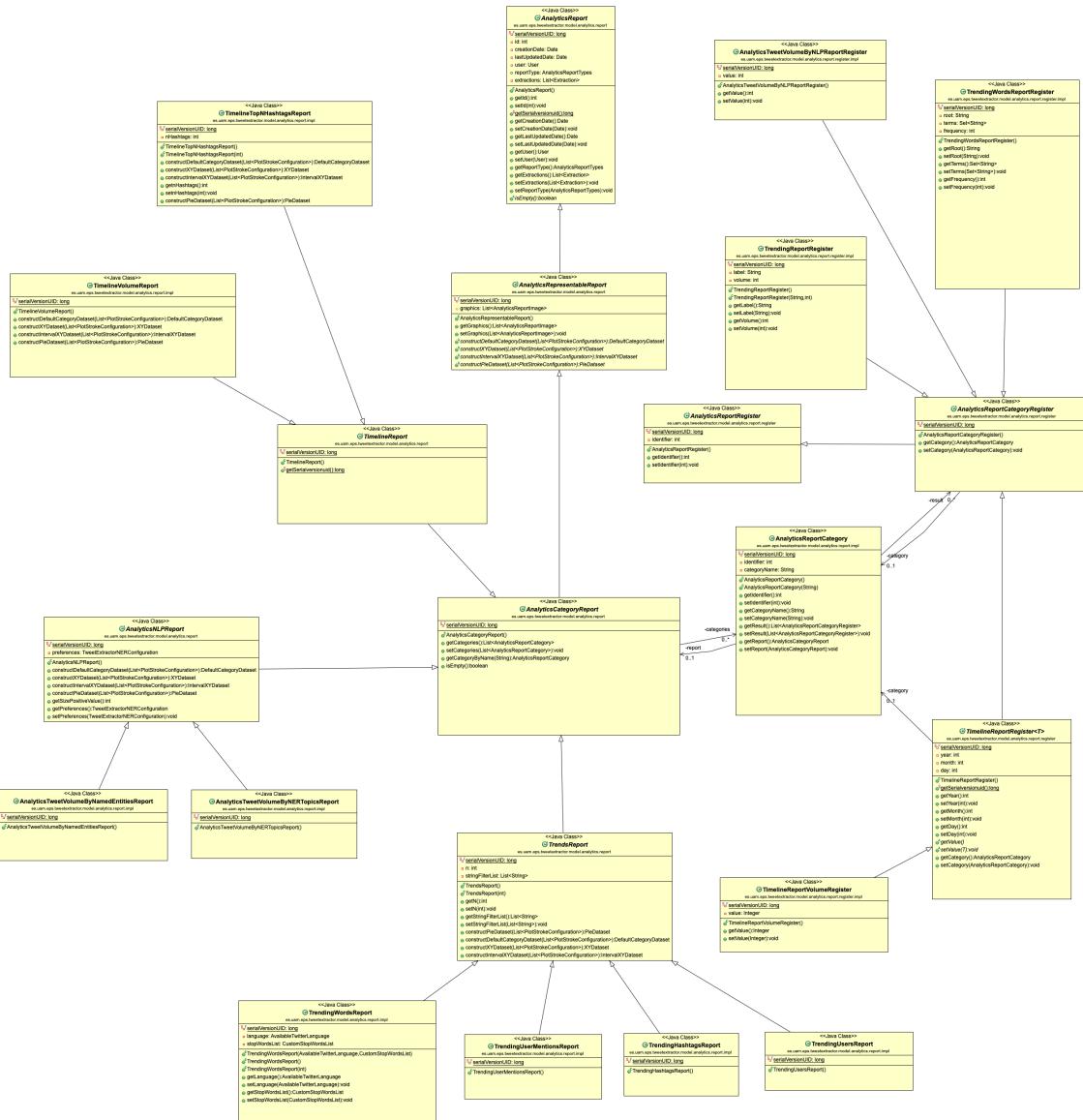
| A

## DIAGRAMAS

---



**Figura A.1:** Diagrama UML que muestra las clases que representan las tareas asíncronas del servidor, con todos sus diferentes tipos.



**Figura A.2:** Diagrama UML que muestra de forma general las clases que representan los reportes de los análisis



# TABLAS

---

Idioma	Código	Idioma	Código
French	fr	Hungarian	hu
English	en	Persian	fa
Arabic	ar	Hebrew	he
Japanese	ja	Urdu	ur
Spanish	es	Thai	th
German	de	Ukrainian	uk
Italian	it	Catalan	ca
Indonesian	id	Irish	ga
Portuguese	pt	Greek	el
Korean	ko	Basque	eu
Turkish	tr	Czech	cs
Russian	ru	Gallegan	gl
Dutch	nl	Romanian	ro
Filipino	fil	Croatian	hr
Msa	msa	En-gb	en-gb
Zh-tw	zh-tw	Vietnamese	vi
Zh-cn	zh-cn	Bengali	bn
Hindi	hi	Bulgarian	bg
Norwegian	no	Serbian	sr
Swedish	sv	Slovak	sk
Finnish	fi	Gujarati	gu
Danish	da	Marathi	mr
Polish	pl	Tamil	ta
		Kannada	kn

(a) Idiomas (I)

(b) Idiomas (II)

**Tabla B.1:** En esta tabla se muestran los idiomas que la API de Twitter es capaz de identificar. Serán los idiomas para los que funcione nuestro análisis semántico

URL	Comentario
<a href="https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/createServerTaskTimelineTopNHashtagsReportImpl">https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/createServerTaskTimelineTopNHashtagsReportImpl</a>	Crea una tarea ServerTaskTimelineTopNHashtagsReport
<a href="https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/createServerTaskTimelineVolumeReport">https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/createServerTaskTimelineVolumeReport</a>	Crea una tarea ServerTaskTimelineVolumeReport
<a href="https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/createServerTaskTrendsReport">https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/createServerTaskTrendsReport</a>	Crea una tarea ServerTaskTrendsReport
<a href="https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/createServerTaskTweetVolumeByNERTopicsReport">https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/createServerTaskTweetVolumeByNERTopicsReport</a>	Crea una tarea ServerTaskTweetVolumeByNERTopicsReport
<a href="https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/createServerTaskTweetVolumeByNamedEntitiesReport">https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/createServerTaskTweetVolumeByNamedEntitiesReport</a>	Crea una tarea ServerTaskTweetVolumeByNamedEntitiesReport
<a href="https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/createServerTaskUpdateExtractionIndef">https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/createServerTaskUpdateExtractionIndef</a>	Crea una tarea ServerTaskUpdateExtractionIndef
<a href="https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/deleteServerTask">https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/deleteServerTask</a>	Borra una tarea y todos sus datos
<a href="https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/getServerStatus">https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/getServerStatus</a>	Se usa a modo de ping.
<a href="https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/getServerTaskStatus">https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/getServerTaskStatus</a>	Devuelve el estado de una tarea
<a href="https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/getUserServerTasks">https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/getUserServerTasks</a>	Devuelve una lista con datos de todas las tareas
<a href="https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/interruptServerTask">https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/interruptServerTask</a>	Interrumpe una tarea ejecutándose
<a href="https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/launchServerTask">https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/launchServerTask</a>	Lanza una tarea “Preparada”
<a href="https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/scheduleServerTask">https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/scheduleServerTask</a>	Programa una tarea
<a href="https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/setServerTaskReady">https://app.preciapps.com:8080/tweetextractor-server-1.0.1.0/setServerTaskReady</a>	Pasa una tarea al estado “Preparada”

**Tabla B.2:** En esta tabla se muestran los diferentes servicios web SOAP disponibles en el módulo servidor, así como una breve descripción sobre su funcionalidad y un hipervínculo a sus descriptores WSDL accesibles desde HTTPS

C

# CÓDIGOS

---

**Código C.1:** Ejemplo de declaración, atributos, métodos y anotaciones de nuestro servicio genérico  
(al que extienden el resto de servicios específicos)

```

1  @Service
2  public abstract class GenericService<V extends Serializable, K extends Serializable>
3      implements GenericServiceInterface<V, K> {
4      @Autowired
5      private AbstractGenericDAO<V, K> genericDao;
6
7      public GenericService(AbstractGenericDAO<V,K> genericDao) {
8          this.genericDao=genericDao;
9      }
10
11     public GenericService() {
12     }
13
14     @Override
15     @Transactional(propagation = Propagation.REQUIRED)
16     public void saveOrUpdate(V entity) {
17         genericDao.saveOrUpdate(entity);
18     }
19
20     @Override
21     @Transactional(propagation = Propagation.REQUIRED, readOnly = true)
22     public List<V> findAll() {
23         return genericDao.getAll();
24     }
25
26     @Override
27     @Transactional(propagation = Propagation.REQUIRED, readOnly = true)
28     public V findById(K id) {
29         return genericDao.find(id);
30     }
31     @Override
32     @Transactional(propagation = Propagation.REQUIRED, readOnly = true)
33     public void detach(V entity) {
34         genericDao.detach(entity);
35     }
36     @Override
37     @Transactional(propagation = Propagation.REQUIRED, readOnly = true)
38     public void detachList(List<V> entityList) {
39         genericDao.detachList(entityList);
40     }
41     @Override
42     @Transactional(propagation = Propagation.REQUIRED, readOnly = true)
43     public void refresh(V entity) {
44         genericDao.refresh(entity);
45     }
46     @Override
47     @Transactional(propagation = Propagation.REQUIRED, readOnly = true)
48     public void initialize(Object lazyObject) {
49         genericDao.initialize(lazyObject);
50     }

```

---

**Código C.2:** Ejemplo de declaración, atributos, métodos y anotaciones de nuestro servicio genérico  
(al que extienden el resto de servicios específicos)

```
51  @Override
52  @Transactional(propagation = Propagation.REQUIRED)
53  public void persist(V entity) {
54      genericDao.persist(entity);
55  }
56  @Override
57  @Transactional(propagation = Propagation.REQUIRED)
58  public void persistList(List<V> entityList) {
59      genericDao.persistList(entityList);
60  }
61  @Override
62  @Transactional(propagation = Propagation.REQUIRED)
63  public void deleteList(List<V> entityList) {
64      genericDao.deleteList(entityList);
65  }
66  @Override
67  @Transactional(propagation = Propagation.REQUIRED)
68  public void merge(V entity) {
69      genericDao.merge(entity);
70  }
71  @Override
72  @Transactional(propagation = Propagation.REQUIRED)
73  public void update(V entity) {
74      genericDao.update(entity);
75  }
76  @Override
77  @Transactional(propagation = Propagation.REQUIRED,readonly = true)
78  public boolean existsAny(K id) {
79      return genericDao.existsAny(id);
80  }
81  @Override
82  @Transactional(propagation = Propagation.REQUIRED)
83  public void delete(V entity) {
84      genericDao.delete(entity);
85  }
86  @Override
87  @Transactional(propagation = Propagation.REQUIRED)
88  public void deleteById(K id) {
89      V entity = findById(id);
90      if(entity!=null)genericDao.delete(entity);
91  }
92  @Override
93  @Transactional(propagation = Propagation.REQUIRED)
94  public void deleteAll() {
95      genericDao.deleteAll();
96  }
97  @Override
98  @Transactional(propagation = Propagation.REQUIRED)
99  public void flush() {
100     genericDao.flush();
101 }
102 }
```

**Código C.3:** Implementación del objeto de acceso a datos (DAO) genérico (del cual heredan todos los DAO específico)

```

1  @SuppressWarnings("unchecked")
2  @Repository
3  public abstract class AbstractGenericDAO<V extends Serializable, K extends Serializable>
4      implements GenericDAOInterface<V, K> {
5      @Autowired
6      private SessionFactory sessionFactory;
7
8      protected Class<V> daoType;
9
10     public AbstractGenericDAO() {
11         Type t = getClass().getGenericSuperclass();
12         ParameterizedType pt = (ParameterizedType) t;
13         daoType = (Class<V>) pt.getActualTypeArguments()[0];
14     }
15
16     protected Session currentSession() {
17         return sessionFactory.getCurrentSession();
18     }
19     @Override
20     public void saveOrUpdate(V entity) {
21         currentSession().saveOrUpdate(entity);
22     }
23     @Override
24     public void persist(V entity) {
25         currentSession().save(entity);
26     }
27     @Override
28     public boolean existsAny(K id) {
29         V entity = find(id);
30         return (entity==null) ? false : true;
31     }
32     @Override
33     public void update(V entity) {
34         currentSession().saveOrUpdate(entity);
35     }
36
37     @Override
38     public void delete(V entity) {
39         currentSession().delete(entity);
40     }
41     @Override
42     public void detach(V entity) {
43         currentSession().detach(entity);
44     }
45     @Override
46     public void detachList(List<V> entityList) {
47         if(entityList==null) return;
48         for(V entity : entityList) {
49             currentSession().detach(entity);
50         }
51     }

```

---

**Código C.4:** Implementación del objeto de acceso a datos (DAO) genérico (del cual heredan todos los DAO específico)

```
52     @Override
53     public void merge(V entity) {
54         currentSession().merge(entity);
55     }
56
57     @Override
58     public V find(K key) {
59         return currentSession().get(daoType, key);
60     }
61     @Override
62     public List<V> getAll() {
63         return currentSession().createQuery("from " + daoType.getName()).list();
64     }
65     @Override
66     public void refresh(V entity) {
67         currentSession().refresh(entity);
68     }
69     @Override
70     public void deleteById(K idDB) {
71         V entity = find(idDB);
72         if(entity!=null)delete(entity);
73     }
74     @Override
75     public void deleteAll() {
76         List<V> entityList = getAll();
77         for (V entity : entityList) {
78             delete(entity);
79         }
80     }
81     @Override
82     public void persistList(List<V> entityList) {
83         if(entityList==null) return;
84         for(V entity : entityList) {
85             persist(entity);
86         }
87     }
88     @Override
89     public void deleteList(List<V> entityList) {
90         if(entityList==null) return;
91         for(V entity : entityList) {
92             delete(entity);
93         }
94     }
95     @Override
96     public void initialize(Object lazyObject) {
97         Hibernate.initialize(lazyObject);
98     }
99     @Override
100    public void flush() {
101        currentSession().flush();
102    }
103
104 }
```

**Código C.5:** Se muestran todas las anotaciones de Hibernate que le ayudarán a traducir entre objetos y filas de la tabla perm\_extraction. También se muestran las anotaciones XML Bind para la traducción XML <->Objeto

```

50  @XmlRootElement(name = "extraction")
51  @XmlSeeAlso({ es.uam.eps.tweetextractor.model.filter.impl.FilterHashtag.class,
52      es.uam.eps.tweetextractor.model.filter.impl.FilterContains.class,
53      es.uam.eps.tweetextractor.model.filter.impl.FilterContainsExact.class,
54      es.uam.eps.tweetextractor.model.filter.impl.FilterList.class,
55      es.uam.eps.tweetextractor.model.filter.impl.FilterMention.class,
56      es.uam.eps.tweetextractor.model.filter.impl.FilterSince.class,
57      es.uam.eps.tweetextractor.model.filter.impl.FilterUntil.class,
58      es.uam.eps.tweetextractor.model.filter.impl.FilterOr.class,
59      es.uam.eps.tweetextractor.model.filter.impl.FilterNot.class,
60      es.uam.eps.tweetextractor.model.filter.impl.FilterFrom.class })
61  @XmlType(propOrder={"id","idDB","creationDate","lastModificationDate","filterXmlList"})
62  @Entity
63  @Controller
64  @Table(name = "perm_extraction")
65  public class Extraction implements Serializable {
66      @XmlTransient
67      @Transient
68      transient UserServiceInterface service;
69      @Transient
70      @XmlTransient
71      private static final long serialVersionUID = 5761562204007375522L;
72      @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
73      @Column(name = "identifier")
74      private int idDB;
75      @Column(name = "id", length=36, unique=true, nullable=false)
76      private String id;
77      @Column(name = "creation_date")
78      @Temporal(TemporalType.TIMESTAMP)
79      private Date creationDate;
80      @Column(name = "last_modification_date")
81      @Temporal(TemporalType.TIMESTAMP)
82      private Date lastModificationDate;
83      @XmlTransient
84      @OneToMany(fetch=FetchType.LAZY,cascade =
85          {CascadeType.PERSIST,CascadeType.REMOVE},mappedBy="extraction")
86      private List<Tweet> tweetList;
87      @OneToMany(fetch=FetchType.EAGER,cascade =
88          {CascadeType.PERSIST,CascadeType.REMOVE},mappedBy="extraction")
89      @XmlTransient
90      private List<Filter> filterList;
91      @Transient
92      private transient List<Filter> filterXmlList = new ArrayList<>();
93      @ManyToOne
94      private User user;
95      @Column(name = "extracting")
96      private boolean extracting;
97      public Extraction() {
98          create();
      }

```

---

**Código C.6:** Se muestran todas las anotaciones de Hibernate que ayudarán tanto a la integración con el polimorfismo Java como aquellas que ayudarán a la traducción entre objetos y filas.

```
47 @Entity
48 @Table(name="perm_analytics_report")
49 @Inheritance(strategy = InheritanceType.SINGLE_TABLE)
50 @Polymorphism(type = PolymorphismType.IMPLICIT)
51 @DiscriminatorColumn(name = "report_type",length=6, discriminatorType =
    DiscriminatorType.STRING)
52 public abstract class AnalyticsReport implements Serializable{
53     @Transient
54     @XmlTransient
55     private static final long serialVersionUID = 3263875661727113958L;
56     @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
57     @Column(name = "identifier")
58     private int id;
59     @Column(name = "generation_date")
60     @Temporal(TemporalType.TIMESTAMP)
61     private Date creationDate;
62     @Column(name = "last_updated_date")
63     @Temporal(TemporalType.TIMESTAMP)
64     private Date lastUpdatedDate;
65     @ManyToOne
66     @XmlTransient
67     private User user;
68     @XmlTransient
69     @Column(name = "report_type", length=6 ,nullable = false, insertable = false, updatable = false)
70     @Enumerated(EnumType.STRING)
71     public AnalyticsReportTypes reportType;
72     @ManyToMany(fetch = FetchType.LAZY,cascade =
        {CascadeType.PERSIST,CascadeType.MERGE,CascadeType.REFRESH})
73     @JoinTable(name = "analytics_report_extraction",
74         joinColumns = { @JoinColumn(name = "report_identifier") },
75         inverseJoinColumns = { @JoinColumn(name = "extraction_identifier") })
76     private List<Extraction> extractions = new ArrayList<>();
77     public AnalyticsReport() {
78         super();
79         this.creationDate= new Date();
80     }
```

**Código C.7:** Se muestran todas las anotaciones de Hibernate que ayudarán tanto a la integración con el polimorfismo Java como aquellas que ayudarán a la traducción entre objetos y filas.

```
22  @Entity
23  @DiscriminatorValue(value = AnalyticsReportTypes.Values.TYPE_TRENDING_WORDS_REPORT)
24  @XmlRootElement(name = "trendingWordsReport")
25  public class TrendingWordsReport extends TrendsReport {
26      @Transient
27      @XmlTransient
28      private static final long serialVersionUID = -6935441753670688388L;
29      @ManyToOne
30      private AvailableTwitterLanguage language;
31      @ManyToOne
32      private CustomStopWordsList stopWordsList;
33      /**
34       * @param language
35       * @param stopWordsList
36       */
37      public TrendingWordsReport(AvailableTwitterLanguage language, CustomStopWordsList
38          stopWordsList) {
39          super();
40          this.language = language;
41          this.stopWordsList = stopWordsList;
42      }
```

---

**Código C.8:** Se muestran todas las anotaciones de Hibernate que ayudarán tanto a la integración con el polimorfismo Java como aquellas que ayudarán a la traducción entre objetos y filas.

```
23  @Entity
24  @Inheritance(strategy = InheritanceType.TABLE_PER_CLASS)
25  public abstract class AnalyticsReportRegister implements Serializable{
26      @Transient
27      private static final long serialVersionUID = -1129809341413845048L;
28      @Id
29      @GeneratedValue(strategy = GenerationType.TABLE)
30      @Column(name = "identifier")
31      private int identifier;
32      public AnalyticsReportRegister() {
33          super();
34      }
35      /**
36      * @return the identifier
37      */
38      public int getIdentifier() {
39          return identifier;
40      }
41      /**
42      * @param identifier the identifier to set
43      */
44      public void setIdentifier(int identifier) {
45          this.identifier = identifier;
46      }
47  }
```

**Código C.9:** Su muestran los métodos de la clase MainApplication que se encargan de inicializar el Stage principal y añadirle el RootLayout como Scene. Se muestra además el método que cargará cualquier pantalla de la vista en el centro del RootLayout (será fácilmente el método más usado en toda la aplicación).

```

75     @Override
76     public void start(Stage primaryStage) {
77         logger.info("Starting_TweetExtractorFX...");
78         this.primaryStage = primaryStage;
79         initializeIcons();
80         this.primaryStage.setTitle("TweetExtractorFX");
81         initRootLayout();
82         TweetExtractorFXPreferences.initializePreferences();
83         springContext = new AnnotationConfigApplicationContext(TweetExtractorSpringConfig.class);
84     }
85     public void initRootLayout() {
86         try {
87             // Load root layout from fxml file.
88             FXMLLoader loader = new FXMLLoader();
89             loader.setLocation(MainApplication.class.getResource("view/RootLayout.fxml"));
90             rootLayout = loader.load();
91             // Show the scene containing the root layout.
92             Scene scene = new Scene(rootLayout);
93             primaryStage.setScene(scene);
94             // Give the controller access to the main app.
95             rootLayoutController = loader.getController();
96             rootLayoutController.setMainApplication(this);
97             this.showScreenInCenterOfRootLayout("view/WelcomeScreen.fxml");
98             primaryStage.show();
99         } catch (IOException e) {
100             logger.error(e.getMessage());
101         }
102     }
103     public void showScreenInCenterOfRootLayout(String fxmlPath) {
104         try {
105             FXMLLoader loader = new FXMLLoader();
106             loader.setLocation(MainApplication.class.getResource(fxmlPath));
107             Node rootNode = loader.load();
108             // Set query constructor into the center of root layout.
109             rootLayout.setCenter(rootNode);
110             // Give the controller access to the main app.
111             TweetExtractorFXController controller = loader.getController();
112             controller.setMainApplication(this);
113         } catch (IOException e) {
114             logger.error(e.getMessage());
115             e.printStackTrace();
116         }
117     }

```

---

**Código C.10:** Se muestra el método de la clase MainApplication que recibe la ruta de un diálogo fxml y la clase que se corresponde con su controlador para mostrarlo en pantalla. Devuelve una respuesta que será distinta (incluso vacía en algunos casos) según el tipo de diálogo que se quiera mostrar.

```
118 public TweetExtractorFXDialogResponse showDialogLoadFXML(String fxmlPath, Class<?> clazz) {  
119     try {  
120         FXMLLoader loader = new FXMLLoader();  
121         loader.setLocation(MainApplication.class.getResource(fxmlPath));  
122         AnchorPane page = loader.load();  
123         // Create the dialog Stage.  
124         Stage dialogStage = new Stage();  
125         dialogStage.initModality(Modality.WINDOW_MODAL);  
126         dialogStage.initOwner(this.getPrimaryStage());  
127         Scene scene = new Scene(page);  
128         dialogStage.setScene(scene);  
129         // Set the dialogStage to the controller.  
130         TweetExtractorFXDialogController controller = loader.getController();  
131         Method meth = clazz.getMethod("setDialogStage", Stage.class);  
132         meth.invoke(controller, dialogStage);  
133         meth = clazz.getMethod("setMainApplication", MainApplication.class);  
134         meth.invoke(controller, this);  
135         // Show the dialog and wait until the user closes it, then add filter  
136         dialogStage.showAndWait();  
137         return controller.getResponse();  
138     } catch (Exception e) {  
139         e.printStackTrace();  
140         logger.error(e.getMessage());  
141         return null;  
142     }  
143 }
```

**Código C.11:** Esta clase abstracta encapsula las tareas que se ejecutan en segundo plano para no bloquear la GUI. Todos los tipos de tareas extenderán a esta clase.

```
1 public abstract class TweetExtractorFXTask<V> extends Task<V> {  
2  
3     protected AnnotationConfigApplicationContext springContext;  
4  
5     public TweetExtractorFXTask(AnnotationConfigApplicationContext springContext) {  
6         super();  
7         this.springContext = springContext;  
8     }  
9  
10    public AnnotationConfigApplicationContext getSpringContext() {  
11        return springContext;  
12    }  
13  
14    public void setSpringContext(AnnotationConfigApplicationContext springContext) {  
15        this.springContext = springContext;  
16    }  
17  
18 }
```

---

**Código C.12:** Se muestra la clase que representa la tarea que utiliza el servicio de Tweets para conectarse a la base de datos y cargar todos los tweets de una extracción.

```
1 public class LoadTweetsTask extends TweetExtractorFXTask<Integer>{
2     private Extraction extraction;
3     private TweetServiceInterface tweetService;
4
5     public LoadTweetsTask(AnnotationConfigApplicationContext springContext, Extraction extraction) {
6         super(springContext);
7         this.extraction = extraction;
8         tweetService= springContext.getBean(TweetServiceInterface.class);
9     }
10
11    @Override
12    protected Integer call() throws Exception {
13        if(extraction==null)return 0;
14        List<Tweet>ret =tweetService.findByExtraction(this.getExtraction());
15        if (ret==null)return 0;
16        this.getExtraction().setTweetList(ret);
17        Logger logger = LoggerFactory.getLogger(this.getClass());
18        logger.info("Loaded_"+ret.size()+"tweets_from_database");
19        return ret.size();
20    }
21
22    public Extraction getExtraction() {
23        return extraction;
24    }
25
26    public void setExtraction(Extraction extraction) {
27        this.extraction = extraction;
28    }
29
30 }
```

**Código C.13:** Se muestra el proceso por el cual se llama a la tarea LoadTweetsTask desde la GUI, se lanza el diálogo de carga y se espera el éxito o el fracaso de la tarea para actuar en consecuencia. Obsérvese el uso de funciones lambda de Java.

```

279 public void refreshTweetObservableList() {
280     if (extraction != null && extraction.getFilterList() != null) {
281         LoadTweetsTask loadTask = new LoadTweetsTask(mainApplication.getSpringContext(),
282             extraction);
283         loadTask.setOnSucceeded(e -> {
284             this.tweetObservableList.clear();
285             this.tweetObservableList.setAll(extraction.getTweetList());
286             if (loadingDialog != null)
287                 loadingDialog.close();
288         });
289         loadTask.setOnFailed(e -> {
290             if (loadingDialog != null)
291                 loadingDialog.close();
292         });
293         Thread thread = new Thread(loadTask);
294         thread.setName(loadTask.getClass().getCanonicalName());
295         thread.start();
296         loadingDialog = mainApplication.showLoadingDialog("Loading_tweets...");
297         loadingDialog.showAndWait();
298     }
}

```

**Código C.14:** Método de la clase estática FilterManager que construye un consulta de la API de Twitter a partir de una lista de filtros.

```

37 public static String getQueryFromFilters(List<Filter>filterList) {
38     if(filterList==null) {
39         return null;
40     }else {
41         String ret= "";
42         for(Filter filter:filterList) {
43             ret=ret.concat(filter.toQuery());
44         }
45         return ret;
46     }
47 }

```

---

**Código C.15:** Método que realiza la consulta preparada a la API de Twitter y gestiona el resultado, así como las posibles excepciones que se puedan dar (error de red, exceso de la tasa temporal permitida por la API, etc.)

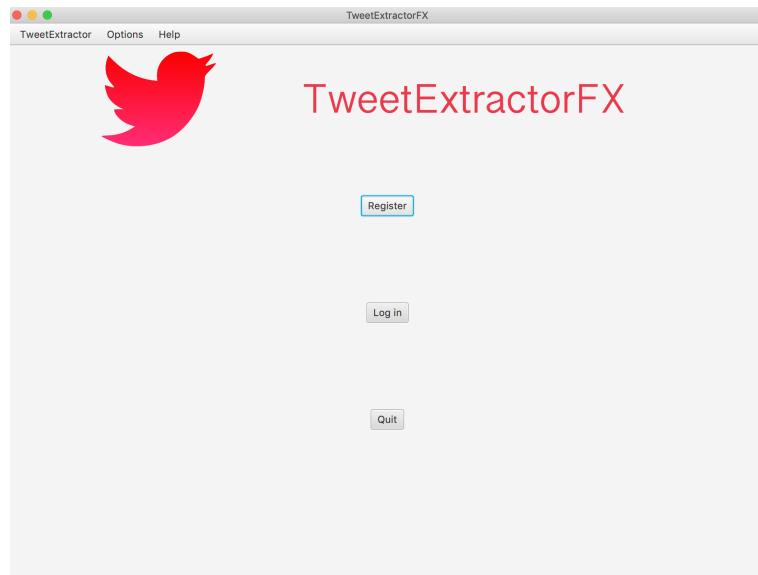
```
184 public UpdateStatus getStatusListExecution() {  
185     UpdateStatus ret= new UpdateStatus(0);  
186     List<Status>resultList=new ArrayList<>();  
187     try {  
188         QueryResult result;  
189         do {  
190             ret.setStatus(Constants.SUCCESS_UPDATE);  
191             ret.setError(false);  
192             result = twitter.search(query);  
193             List<Status> tweets = result.getTweets();  
194             for (Status tweet : tweets) {  
195                 resultList.add(tweet);  
196             }  
197         } while ((query = result.nextQuery()) != null);  
198         ret.setStatusList(resultList);  
199         return ret;  
200     } catch (TwitterException te) {  
201         //Network Issue  
202         if(te.getStatusCode()==-1&&te.getErrorCode()==-1) {  
203             ret.setStatus(Constants.CONNECTION_UPDATE_ERROR);  
204             ret.setErrorMessage(te.getErrorMessage());  
205             ret.setError(true);  
206         } else  
207             //Exceeded Rate Limit  
208             if(te.getStatusCode()==429&&te.getErrorCode()==88) {  
209                 ret.setStatus(Constants.RATE_LIMIT_UPDATE_ERROR);  
210                 ret.setError(true);  
211                 if(!resultList.isEmpty()) {  
212                     ret.setStatus(Constants.SUCCESS_UPDATE);  
213                     ret.setStatusList(resultList);  
214                     ret.setErrorMessage(te.getErrorMessage());  
215                     return ret;  
216                 }  
217             } else {  
218                 ret.setStatus(Constants.UNKNOWN_UPDATE_ERROR);  
219                 ret.setError(true);  
220                 ret.setErrorMessage(te.getErrorMessage());  
221                 return ret;  
222             }  
223         }  
224         return ret;  
225     }  
}
```

**Código C.16:** Creando WordCloud con una configuración y un reporte de frecuencia de palabras.

```

225 public WordCloud constructWordCloudFromTrendingWordsReport(File pixelBoundaryFile)
226     throws IOException {
227     WorldCloudChartConfiguration config = (WorldCloudChartConfiguration) preferences;
228     TrendingWordsReport trendingWordsReport = (TrendingWordsReport) getReport();
229     final FrequencyAnalyzer frequencyAnalyzer = new FrequencyAnalyzer();
230     frequencyAnalyzer.setWordFrequenciesToReturn(config.getnWords());
231     frequencyAnalyzer.setMinWordLength(config.getMinWordLength());
232     List<WordFrequency> wordFrequencies = new ArrayList<>();
233     for (AnalyticsReportCategoryRegister register :
234         trendingWordsReport.getCategories().get(0).getResult()) {
235         TrendingWordsReportRegister castedRegister = (TrendingWordsReportRegister) register;
236         WordFrequency newWord = new WordFrequency((String)
237             castedRegister.getTerms().toArray()[0],
238             castedRegister.getFrequency());
239         wordFrequencies.add(newWord);
240     }
241     wordFrequencies = frequencyAnalyzer.loadWordFrequencies(wordFrequencies);
242     Dimension dimension = null; WordCloud wordCloud = null;
243     switch (config.getType()) {
244     case Constants.WCC_CIRCULAR:
245         dimension = new Dimension(1080, 1080);
246         wordCloud = new WordCloud(dimension, CollisionMode.PIXEL_PERFECT);
247         wordCloud.setBackground(new CircleBackground(540));
248         break;
249     case Constants.WCC_PIXEL_BOUNDARY:
250         try {
251             dimension = new Dimension(1920, 1080);
252             wordCloud = new WordCloud(dimension, CollisionMode.PIXEL_PERFECT);
253             wordCloud.setBackground(new PixelBoundaryBackground(pixelBoundaryFile));
254         } catch (IOException e) {
255             logger.warn("An_exception_has_been_thrown_opening_pixel_boundary_file: " +
256                         e.getMessage());
257         }
258         break;
259     case Constants.WCC_RECTANGULAR:
260         dimension = new Dimension(1920, 1080);
261         wordCloud = new WordCloud(dimension, CollisionMode.RECTANGLE);
262         wordCloud.setBackground(new RectangleBackground(dimension));
263         break;
264     default:
265         break;
266     }
267     if (wordCloud != null) {
268         wordCloud.setPadding(2);
269         wordCloud.setColorPalette(new ColorPalette(config.getAwtColorList()));
270         wordCloud.setFontScalar(new LinearFontScalar(config.getFontMin(), config.getFontMax()));
271         wordCloud.build(wordFrequencies);
272         final ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
273         wordCloud.writeToStreamAsPNG(byteArrayOutputStream);
274         final byte[] bytes = byteArrayOutputStream.toByteArray();
275         this.reportImage.setImage(bytes);
276     }
277     return wordCloud;
278 }
```

# IMÁGENES DE LA INTERFAZ GRÁFICA



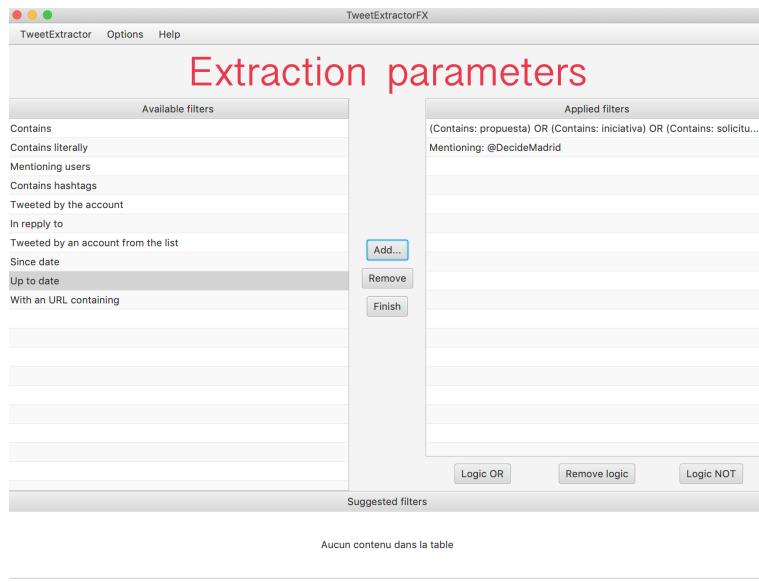
**Figura D.1:** Pantalla principal de bienvenida al abrir la aplicación

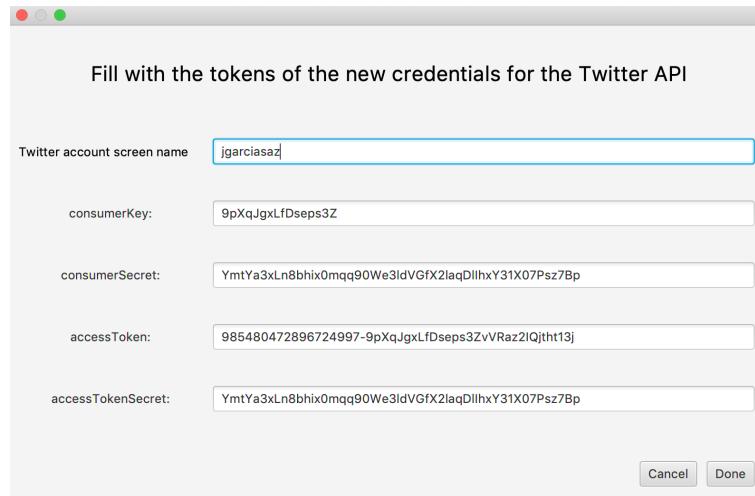
The image contains two separate dialog boxes. The left dialog box is titled "Log in" and has fields for "Username" (containing "jose") and "Password" (containing a masked password). It includes buttons for "Cancel", "Log in", and "New account". The right dialog box is titled "New account" and has fields for "Username" (containing "newUser"), "Password" (containing a masked password), and "Repeat password" (containing a masked password). It includes buttons for "Cancel" and "Create account".

(a) Inicio de Sesión

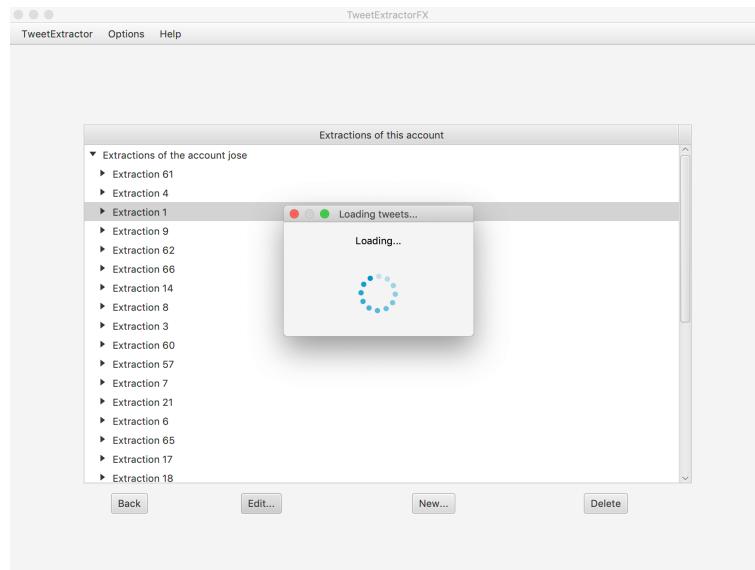
(b) Registro

**Figura D.2:** Diálogos de autentificación en la aplicación.

**Figura D.3:** Pantalla de inicio**Figura D.4:** Pantalla para configurar el perímetro de cada extracción. A la izquierda los filtros disponibles, a la derecha los añadidos.



**Figura D.5:** Configurando los credenciales de la API de Twitter (los credenciales mostrados en la foto no son reales)



**Figura D.6:** Las tareas costosas no bloquean la aplicación. Un nuevo hilo nos muestra un diálogo informativo sobre lo que se está haciendo.



# ÍNDICE TERMINOLÓGICO

---

`budgettitle`, 25

colores, 2

  predefinidos, 2

`eigenvalue`, 40

opciones, 47





