Universidad Autónoma Metropolitana
Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería
Licenciatura en Ingeniería en Computación

Reporte final del Proyecto de Integración

Sistema web para la geolocalización de incidentes delictivos a partir de publicaciones en Twitter

Modalidad: Proyecto Tecnológico

Trimestre 2017 Otoño

Alejandro López Arteaga
2122001103
lopartale@gmail.com

Asesor:
José Alejandro Reyes Ortiz
Doctor en Ciencias de la Computación
Profesor Asociado
Departamento de Sistemas
jaro@correo.azc.uam.mx

Coasesor:
Leonardo Daniel Sánchez Martínez
Doctor en Ciencias y Tecnologías de la Información
Profesor Asociado
Departamento de Sistemas
ldsm1985@gmail.com

11 de diciembre de 2017

Declaratoria

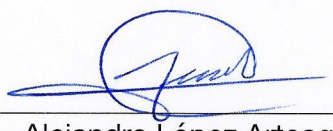
Yo, Dr. José Alejandro Reyes Ortiz, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Dr. José Alejandro Reyes Ortiz

Yo, Dr. Leonardo Daniel Sánchez Martínez , declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Dr. Leonardo Daniel Sánchez Martínez

Yo, Alejandro López Arteaga, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unida d Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.



Alejandro López Arteaga

Resumen

La inseguridad y la delincuencia son de los temas que más preocupan y aquejan hoy en día a los ciudadanos, asimismo otro tema de la actualidad son las redes sociales y los datos que inherentemente contienen, así que con el fin de contribuir a la sociedad, se desarrolló un sistema web que permite a los ciudadanos la visualización, geolocalización y navegación a través de un mapa de los incidentes delictivos suscitados en la Ciudad de México y el Estado de México, tomando como fuente de información las publicaciones en Twitter de cuentas pertenecientes a periódicos y organizaciones confiables de estas entidades.

Este sistema fue desarrollado bajo el patrón de arquitectura de software MVC (Modelo-Vista-Controlador), además de hacer uso de distintas tecnologías como son el lenguaje de programación Java y el gestor de bases de datos MyQSL ambos usados en la implementación del backend, el conjunto de lenguajes HTML, CSS y JavaScript para la construcción del frontend, de igual forma se utilizaron frameworks que permiten un desarrollo ágil como son Struts 2, Bootstrap, Jquery y AJAX.

A lo largo de este reporte se abordan los aspectos teóricos y técnicos fundamentales para comprender la estructura y el desarrollo de este Proyecto de Integración, también se presenta un análisis sobre los resultados obtenidos tras la conclusión y puesta en producción del sistema web.

Índice

Índice de figuras

1. Introducción

A lo largo de los últimos años en la Ciudad de México se ha incrementado la tasa de delitos y por tal motivo no se tiene garantizada la integridad física ni mucho menos la seguridad de los bienes. De la misma forma ha crecido la desconfianza hacia las autoridades y por ello la cantidad de delitos denunciados ha ido a la baja. Se estima que en la Ciudad de México el 84.6% de la población de 18 años y más considera la inseguridad y delincuencia como el problema más importante que los aqueja hoy en día. La percepción general de los ciudadanos es que la ciudad es muy insegura [1] y dado que las estadísticas de seguridad pública proporcionadas por las autoridades se basan en los delitos denunciados, no se tiene una información certera sobre las zonas de alta incidencia delictiva.

Por ello se desarrolló un sistema web que extraiga los delitos ocurridos en la Ciudad de México y el Estado de México a partir de tweets de cuentas oficiales de periódicos y organizaciones confiables, para dar a conocer las zonas de alta incidencia delictiva por medio de la geolocalización en un mapa de dichos delitos, esto con el fin de contribuir con la sociedad para una mejor seguridad, proporcionado a los ciudadanos una herramienta que los ayude a tomar medidas preventivas para evitar ser víctimas de algún delito cuando se transite por la ciudad.

Este sistema web permite a los ciudadanos la visualización, geolocalización y navegación en un mapa de los siguientes tipos de delitos: asalto, explotación, homicidio, secuestro, suicidio y violación, basados en la clasificación de delitos del INEGI en 2008 “delitos contra las personas” [2].

2. Antecedentes

2.1. Proyectos Terminales

1. Aprendizaje automático de patrones textuales para la identificación de eventos sobre la seguridad ciudadana [3].

El objetivo principal de este proyecto es identificar patrones textuales sobre eventos delictivos a partir de notas periodísticas, lo que tienen en común es el uso de patrones textuales y la fuente de extracción, pero en esta propuesta se enfoca solo a publicaciones de periódicos y organizaciones confiables en Twitter, además de dar a conocer su ubicación por medio de la geolocalización.

2. Sistema de información para el análisis de la seguridad social o pública a través de periódicos [4].

Este proyecto se enfoca en analizar y clasificar notas periodísticas sobre seguridad extraídas de un sitio web específico examinando el texto contenido en cada página web, la similitud con este proyecto es que se usan fuentes periodísticas para la extracción de información y se analiza el contenido de las mismas, sin embargo el presente proyecto se centró en la localización de delitos específicos y así delimita la clase de información a extraer.

3. Diseño y desarrollo de un prototipo de aplicación web y móvil nativa android para el monitoreo de seguridad en el Distrito Metropolitano de Quito a través de la red social twitter [5].

En este proyecto se desarrolló un prototipo de sistema web y de aplicación móvil, para mostrar las zonas de mayor peligro en la ciudad de Quito, los delitos cometidos son publicados por los usuarios en Twitter con el hashtag (#) QuitoSeguridad, seguido de la dirección, se recopilan los datos tomados de las publicaciones y se coloca la información en un mapa de calor. Este proyecto comparte la idea de obtención de datos de Twitter y su exposición en un mapa, pero difieren en la forma de obtener la localización del delito dado que se usan patrones textuales, además la extracción se basa en cuentas de periódicos y organizaciones confiables.

2.2. Propuestas

4. Sistema para gestionar incidentes delictivos [6].

En este trabajo se propuso llevar a cabo un sistema web que permitiera la publicación, consulta y la búsqueda basada en ubicación, tiempo e información de delitos, en similitud con el presente proyecto ambos pretenden dar a conocer la ubicación de los delitos pero su fuente de información está basada en las publicaciones de usuarios y no en la extracción de delitos de tweets de periódicos y organizaciones confiables.

2.3. Software

5. Alertux [7].

Es una aplicación móvil y web que recopila, organiza y distribuye en tiempo real alertas de seguridad generadas por sus usuarios. Con esta aplicación se comparte el fin de ayudar a la sociedad proporcionando información para evitar lugares peligrosos, sin embargo la recopilación de

información proviene de fuentes distintas en particular para este proyecto se extrae de publicaciones en Twitter.

6. Mi policía [8].

Es una aplicación móvil que tiene como objetivo acercar los servicios de la policía al ciudadano, y reducir los tiempos de respuesta en las llamadas de emergencia, de igual manera tiene la función de geolocalizar los delitos, pero con el fin de agilizar el auxilio de las autoridades y para este proyecto la geolocalización se usa para brindar una mejor percepción acerca de las zonas inseguras.

3. Justificación

En la Ciudad de México casi todas las víctimas de un delito prefieren no denunciar debido a la tardanza y baja calidad del proceso a seguir. Las autoridades a su vez, repiten que si no hay denuncias ellas no pueden hacer su trabajo para reducir el crimen.

En la última Encuesta Nacional de Victimización y Percepción sobre Seguridad Pública del INEGI (ENVIPE) correspondiente al 2016, se reporta que en el 94.7 % de los delitos no hubo denuncia o no se inició averiguación previa, además se dio a conocer que el principal motivo para no denunciar es debido a ser una “pérdida de tiempo” [1].

Las redes sociales como Twitter representan datos frescos, en tiempo real y al instante; la popularidad de los periódicos crece de manera sostenida ya que maximizan su difusión a través de la viralidad y alcance que le pueden llegar a aportar las redes sociales [9].

Debido a los factores anteriores es importante contar con un sistema de información web que refleje la realidad de la seguridad pública en la Ciudad de México y el Estado de México, es por eso que en este proyecto se da a conocer la localización de los delitos denunciados o no denunciados y que han sido reportados en en redes sociales por periódicos y organizaciones confiables, de esta manera se brinda la sociedad conciencia de las zonas de alta incidencia delictiva y así se ayuda a reducir los riesgos al transitar por ciertas zonas de la ciudad.

4. Objetivos

4.1. Objetivo General

Desarrollar un sistema de información web para la visualización de delitos y su geolocalización extraídos a partir de publicaciones en Twitter de las cuentas oficiales de periódicos de la Ciudad de México.

4.2. Objetivos Específicos

- Diseñar un modelo de clases para estructurar la aplicación e implementar una base de datos relacional para dar persistencia a la información de los delitos cometidos en la Ciudad de México y su geolocalización.

- Extraer delitos a partir de publicaciones en Twitter, obtener su geolocalización y almacenar la información conseguida en la base de datos.

- Desarrollar una aplicación web para la visualización de los delitos extraídos y su geolocalización mediante un mapa.

5. Marco teórico

5.1. Streaming API de Twitter

El Streaming API de Twitter utiliza el protocolo Streaming HTTP para entregar datos a través de una conexión API de transmisión abierta. En lugar de entregar datos en lotes por medio de solicitudes repetidas de una aplicación cliente, como podría esperarse de una API REST, se abre una única conexión entre su aplicación y la API, y cada vez que se producen nuevas coincidencias se envían nuevos resultados a través de esa conexión. Esto da como resultado un mecanismo de entrega de baja latencia que puede soportar un rendimiento muy alto [10].

El Streaming API de Twitter permite conectar nuestra aplicación con Twitter. Nos da acceso a los tweets globales en tiempo real. Para poder conectarnos nos pide las Keys y Tokens que son datos personales y secretos para la conexión. Una vez que se conecta se pueden hacer consultas, sobre los tweets y la información de los usuarios.

La interacción general entre una aplicación y la API de Twitter para estas transmisiones se describe de la siguiente manera:

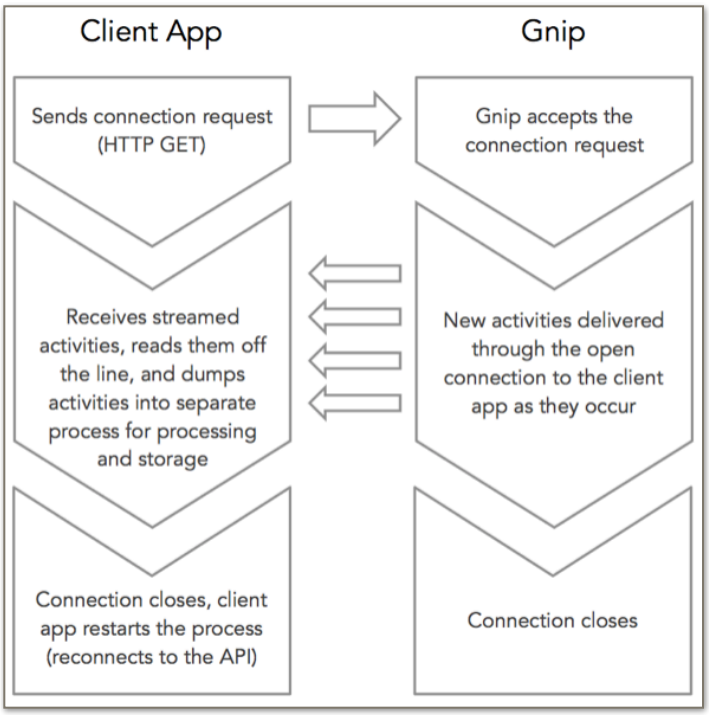


Figura 1. Interacción entre una aplicación y la API Streaming.

5.2. Formato de un Tweet

Todas las API de Twitter que devuelven tweets proporcionan esa información codificada utilizando JavaScript Object Notation (JSON) que se basa en pares clave-valor, con atributos nombrados y valores asociados. Estos atributos y su estado se usan para describir objetos, incluidos los tweets y los usuarios [11].

Todos estos objetos encapsulan atributos principales que describen el objeto. Cada tweet tiene un autor, un mensaje, una ID única, una marca de tiempo de cuándo se publicó y, a veces, datos geográficos compartidos por el usuario. Entonces, además del contenido de texto en sí, un tweet puede tener más de 140 atributos asociados. A continuación se muestra un ejemplo de un tweet publicado:

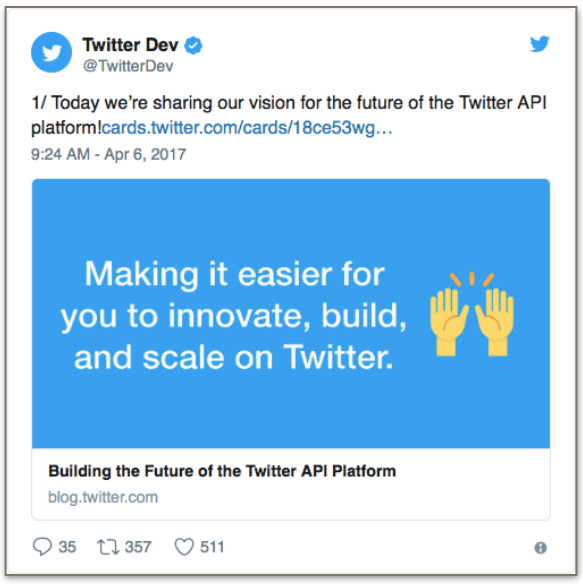


Figura 2. Ejemplo de un tweet.

El siguiente JSON ilustra la estructura de un Tweet y algunos de sus atributos:


```
{
  "tweet": {
    "created_at": "Thu Apr 06 15:24:15 +0000 2017",
    "id_str": "850006245121695744",
    "text": "1\/ Today we\u2019re sharing our vision for the future of the Twitter API platform!\nhttps:\/\/t.
    "user": {
      "id": 2244994945,
      "name": "Twitter Dev",
      "screen_name": "TwitterDev",
      "location": "Internet",
      "url": "https:\/\/dev.twitter.com\/",
      "description": "Your official source for Twitter Platform news, updates & events. Need technical help? V
      \u2328\ufe0f #TapIntoTwitter"
    },
    "place": {

    },
    "entities": {
      "hashtags": [

      ],
      "urls": [
        {
          "url": "https:\/\/t.co\/XweGngmxlP",
          "unwound": {
            "url": "https:\/\/cards.twitter.com\/cards\/18ce53wgo4h\/3xo1c",
            "title": "Building the Future of the Twitter API Platform"
          }
        }
      ],
      "user_mentions": [

      ]
    }
  }
}
```


Figura 3. Ejemplo del JSON de un tweet.

5.3. Twitter4J

Twitter4J es una librería Java para manejar el Twitter API y con ella se puede integrar fácilmente una aplicación Java con el servicio de Twitter. Es una librería no oficial aunque por su sencillez, potencia y capacidades es probablemente la más usada. Entre sus características más interesantes están:

- 100% Java. Funciona con Java 5 en adelante
- No tiene dependencias, para usarla simplemente basta añadir twitter4j-core-4.0.2.jar al classpath
- Preparado para Android y Google App Engine
- Soporte OAuth
- Soporte API Streaming

La clase TwitterStream tiene varios métodos preparados para usar la Streaming API de Twitter, todo lo que necesita es tener una clase que implemente la interfaz StatusListener. Twitter4J usará la interfaz StatusListener creando un hilo y consumiendo el streaming de Twitter [12].

5.4. Geolocalización

La geolocalización es el proceso que se utiliza para relacionar la posición de un objeto en un plano con su posición sobre la superficie terrestre medida en coordenadas geográficas de latitud y longitud [13].

La geolocalización implica el posicionamiento que define la localización de un objeto, de un dispositivo, en un sistema de coordenadas determinado de nuestro planeta tierra. Este proceso es generalmente empleado por los sistemas de información geográfica, un conjunto organizado de hardware y software, más datos geográficos, que se encuentra diseñado especialmente para capturar, almacenar, manipular y analizar en todas sus posibles formas la información geográfica referenciada.

5.5. Google Maps Geocoding API

La Google Maps Geocoding API es un servicio que proporciona geocodificación convencional e inversa de direcciones. La geocodificación es el proceso que convierte direcciones (como la dirección de una calle) en coordenadas geográficas (latitud y longitud) que puedes usar para disponer marcadores en un mapa o posicionar el mapa. La geocodificación inversa es el proceso de conversión de coordenadas geográficas en direcciones en lenguaje natural [14].

Se puede acceder a Google Maps Geocoding API a través de una interfaz HTTP. A continuación, se proporciona un ejemplo de solicitud de geocodificación convencional y se especifica que la salida debe tener formato JSON:

```
https://maps.googleapis.com/maps/api/geocode/json?
address=UAM+Azcapotzalco&key=YOUR_API_KEY
```

En la respuesta se incluyen la latitud y longitud de la dirección. A continuación, se muestra un ejemplo de respuesta de geocodificación, en JSON:


```
{
   "results": [{
           "formatted_address": "Av San Pablo Xalpa 180, Reynosa Tamaulipas, 02200 Ciudad de México, CDMX, Mexico",
           "geometry": {
               "viewport": {
                   "southwest": {
                       "lng": -99.1883217802915,
                       "lat": 19.50206971970849
                   },
                   "northeast": {
                       "lng": -99.1856238197085,
                       "lat": 19.5047676802915
                   }
               },
               "location": {
                   "lng": -99.18697279999999,
                   "lat": 19.5034187
               },
               "location_type": "ROOFTOP"
           },
           "address_components": [{
                   "types": ["street_number"],
                   "short_name": "180",
                   "long_name": "180"
               }, {
                   "types": ["political", "sublocality", "sublocality_level_1"],
                   "short_name": "Reynosa Tamaulipas",
                   "long_name": "Reynosa Tamaulipas"
               }, {
                   "types": ["locality", "political"],
                   "short_name": "México D.F.",
                   "long_name": "Ciudad de México"
               }, {
                   "types": ["administrative_area_level_1", "political"],
                   "short_name": "CDMX",
                   "long_name": "Ciudad de México"
               }, {
                   "types": ["country", "political"],
                   "short_name": "MX",
                   "long_name": "Mexico"
               }],
           "place_id": "ChIJJ0pSC3T40YUR9ouL455SAvs"
       }],
   "status": "OK"
}
```


Figura 4. Ejemplo del JSON de una geocodificación.

5.6. Google Maps JavaScript API

Esta API está diseñada para comenzar rápidamente a explorar y desarrollar aplicaciones con la Google Maps, permite personalizar mapas con imágenes y contenido propios. Se basa en programación mediante JavaScript y conceptos de programación orientados a objetos [15]. Los siguientes puntos son los pasos básicos para usar esta API:

1. Se declara la aplicación como HTML5 a través de la declaración:

```
<!DOCTYPE html>
```

2. Se crea un elemento div con el nombre “map” para que contenga el mapa.

```
<div id=“map”></div>
```

3. Se define una función JavaScript que crea un mapa en el elemento div.

```
map = new google.maps.Map(document.getElementById('map'),
{
    center: {lat: -34.397, lng: 150.644},
    zoom: 8
});
```

4. Se carga la Maps JavaScript API usando una etiqueta script.

```
<script async defer
      src="https://maps.googleapis.com/maps/api/js?
    key=YOUR_API_KEY&callback=initMap">
</script>
```

5.7. Expresiones regulares

Las expresiones regulares son patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto, las expresiones regulares se pueden usar para buscar, editar y manipular texto.

Una expresión regular define un patrón de búsqueda para cadenas. El patrón de búsqueda puede ser cualquier cosa, desde un carácter simple, una cadena fija o una expresión compleja que contiene caracteres especiales que describen el patrón. El patrón definido por la expresión regular puede coincidir una o varias veces o no para una cadena dada.

El proceso de analizar o modificar un texto con una expresión regular se llama: “La expresión regular se aplica al texto / cadena”. El patrón definido por la expresión regular se aplica al texto de izquierda a derecha. Una vez que se ha utilizado un carácter fuente en una coincidencia, no se puede reutilizar. Por ejemplo, la expresión regular aba coincidirá con abababa solo dos veces (aba_aba__) [16]. En la siguiente tabla se muestran algunos símbolos que son comunes en expresiones regulares:

Expresión	**Descripción**
.	Un punto indica cualquier carácter
^expresión	El símbolo ^ indica el principio del String. En este caso el String debe contener la expresión al principio.
expresión$	El símbolo $ indica el final del String. En este caso el String debe contener la expresión al final.
[abc]	Los corchetes representan una definición de conjunto. En este ejemplo el String debe contener las letras a ó b ó c.
[abc][12]	El String debe contener las letras a ó b ó c seguidas de 1 ó 2.
[^abc]	El símbolo ^ dentro de los corchetes indica negación. En este caso el String debe contener cualquier carácter excepto a ó b ó c.
[a-z1-9]	Rango. Indica las letras minúsculas desde la a hasta la z (ambas incluidas) y los dígitos desde el 1 hasta el 9 (ambos incluidos).
A|B	El carácter | es un OR. A ó B.
AB	Concatenación. A seguida de B

Tabla 1. Tabla de expresiones regulares comunes.

Las expresiones regulares de java nos ayudan a hacer estos análisis, el siguiente es un ejemplo en código java de una expresión regular para validar una dirección de email:

```
String emailRegexp = "[^@]+@[^@]+\\.[a-zA-Z]{2,}";
// Lo siguiente devuelve true
System.out.println(Pattern.matches(emailRegexp, "a@b.com"));
// Lo siguiente devuelve false
System.out.println(Pattern.matches(emailRegexp, "@b.com"));
```

6. Desarrollo del proyecto

Con el fin de poder comprender mejor la arquitectura y las fases de desarrollo de este proyecto se parte de un modelo que engloba la estructura general del sistema web. En este modelo se incluyen los módulos que conforman el proyecto los cuales son descritos a detalle en los próximos temas de esta sección partiendo de lo general hacia aspectos más específicos. Los módulos que integran el sistema web son la **Base de Datos**, el **Extractor** y la **Aplicación**, el modelo mencionado anteriormente y sus módulos pueden apreciarse en la Figura 5.

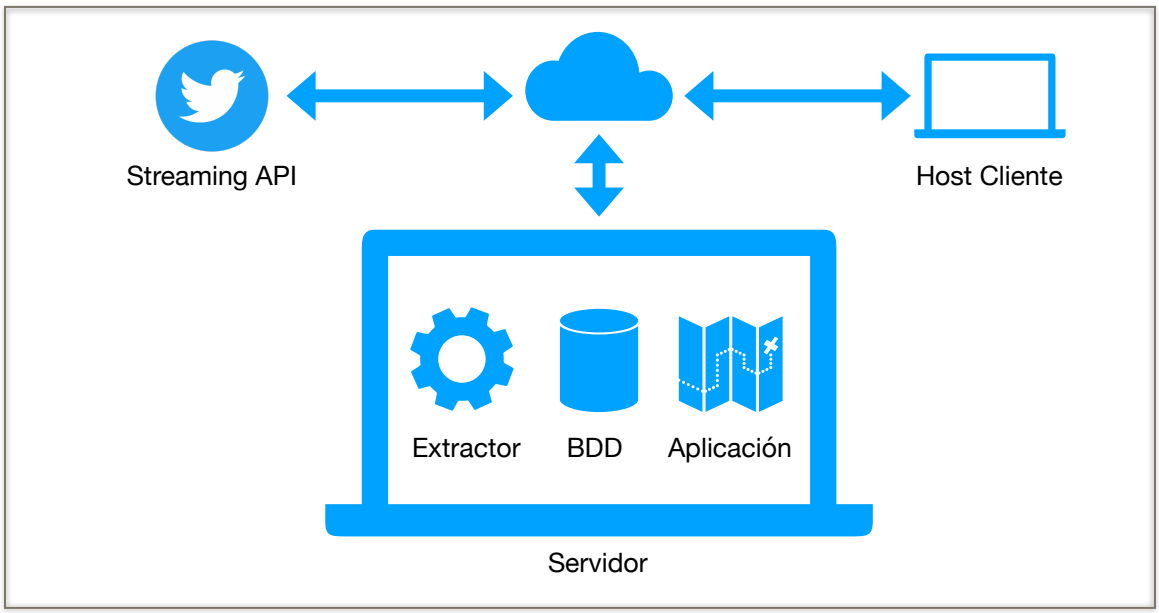


Figura 5. Modelo de la arquitectura del proyecto.

6.1. Base de Datos

Este módulo consiste en una base de datos construida bajo el modelo relacional, tiene la función de dar persistencia a la información con la que trabaja el sistema web, es decir, almacena los incidentes delictivos ocurridos en la Ciudad de México y el Estado de México. La base de datos está compuesta por dos tablas, la primera tabla llamada **tweet** contiene información propia de los tweets extraídos y la otra tabla llamada **delito** contiene solo los tipos de delito que se obtienen por medio del módulo Extractor.

La información que se almacena en la base de datos consiste en los datos obtenidos de los tweets extraídos en formato JSON que devuelve la Streaming API de Twitter al Extractor, los datos almacenados son específicamente el identificador del tweet, la fecha de extracción, la cuenta que pública el tweet, el texto contenido en el tweet, las coordenadas de latitud y longitud obtenidas a partir de geocodificar la ubicación contenida en el tweet y el tipo de delito identificado en el texto del tweet por medio de expresiones regulares y patrones textuales. El modelo relacional de la base de datos y sus tablas se presentan en la Figura 6.

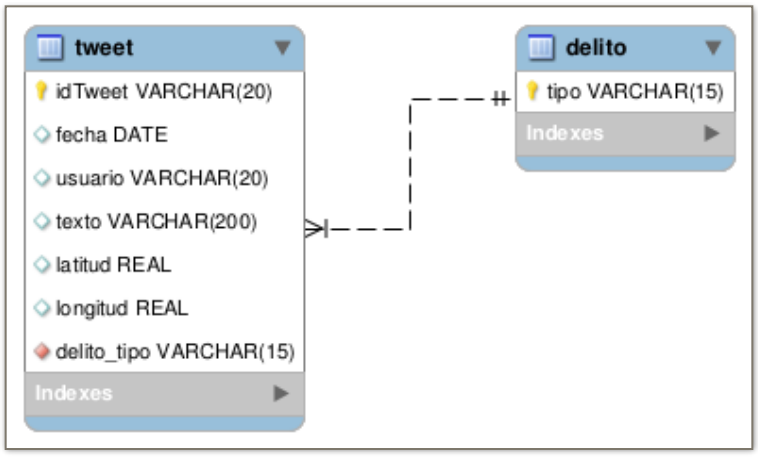


Figura 6. Modelo relacional de la base de datos.

Como puede verse en el modelo relacional de la base de datos existe una relación muchos a uno entre las tablas tweet y delito, esto quiere decir que cada registro de la tabla tweet está relacionado con un registro de la tabla delito y que cada registro de la tabla delito está relacionado a muchos registros de la tabla tweet. Gracias a este diseño se ahorra espacio de almacenamiento en la base de datos ya que se evitan registros que contengan valores repetidos para el atributo de tipo de delito y se optimizan los tiempos de consultas por tipo de delito.

El script para la creación de la base de datos y para la inserción de registros en la tabla “delito” puede observarse en la sección de apéndices 11.1.

6.2. Extractor

Este módulo se encarga de ejecutar las funciones necesarias para lograr la identificación y obtención efectiva de los incidentes delictivos, estas funciones se presentan a continuación:

- Establecer la conexión a la Streaming API de Twitter.
- Obtener los tweets publicados por las cuentas oficiales de periódicos y organizaciones confiables de la Ciudad de México y del Estado de México.
- Identificar los incidentes delictivos contenidos en los tweets extraídos por medio de patrones textuales.
- Extraer las ubicaciones en donde se suscitaron los incidentes delictivos, a partir de los tweets extraídos y por medio de patrones textuales.
- Geocodificar las ubicaciones obtenidas en el punto anterior.
- Almacenar los incidentes delictivos que fueron identificados y geocodificados.

El módulo Extractor está integrado por los siguientes componentes: la **Conexión**, el **Listener**, el **FilterQuery** y el **Processor**, los cuales están contenidos en el diagrama de clases de la Figura 7 y en conjunto llevan a cabo un proceso que se describe a continuación.

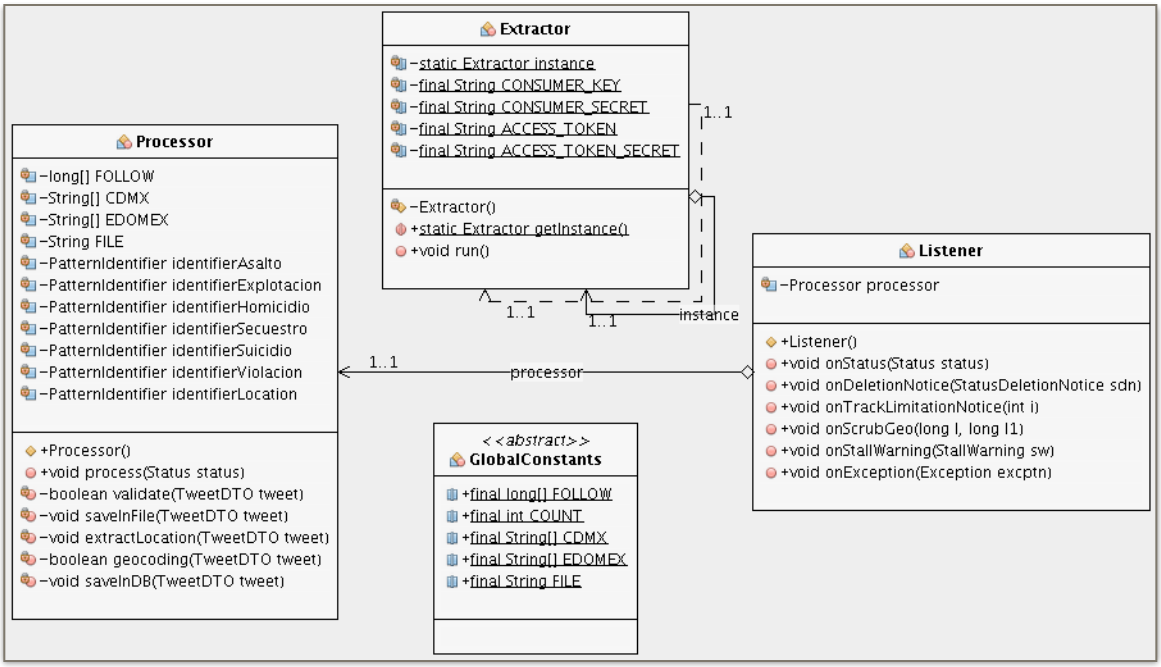


Figura 7. Diagrama de clases del paquete extractor.logic.extraction.

El proceso de la obtención de tweets comienza estableciendo una conexión al Streaming API de Twitter, esta “Conexión” se mantiene activa permanentemente y escuchando a través de su “Listener” los nuevos tweets publicados. La conexión también cuenta con un “FilterQuery" que es un filtro aplicado a la misma conexión, es decir, hace que Twitter solo envíe los tweets requeridos de acuerdo al filtro aplicado.

Luego de que un tweet ha sido obtenido, este pasa al “Processor” quien es el encargado de comprobar el origen de la publicación, identificar patrones textuales en el texto del tweet, extraer la ubicación contenida en el texto del tweet, geocodificar la ubicación extraída y almacenar el tweet en la base de datos. El proceso anterior se ejecuta de forma cíclica cada vez que haya un nuevo tweet es obtenido, de esta forma el Extractor alimenta a la base de datos y esta a su vez es la fuente de información para ser presentada en la aplicación web.

En los siguientes puntos se explica paso a paso como se llevó a cabo la construcción del módulo Extractor, también se abordan detalles de su implementación a nivel de código y los aspectos importantes de su funcionamiento.

6.2.1. Crear una aplicación en Twitter

Para poder conectarse a la Streaming API de Twitter y consumir su flujo de tweets es necesario tener una cuenta en Twitter y registrarse como desarrollador en developer.twitter.com, el siguiente paso es crear una aplicación en apps.twitter.com como se muestra a continuación en la Figura 8.

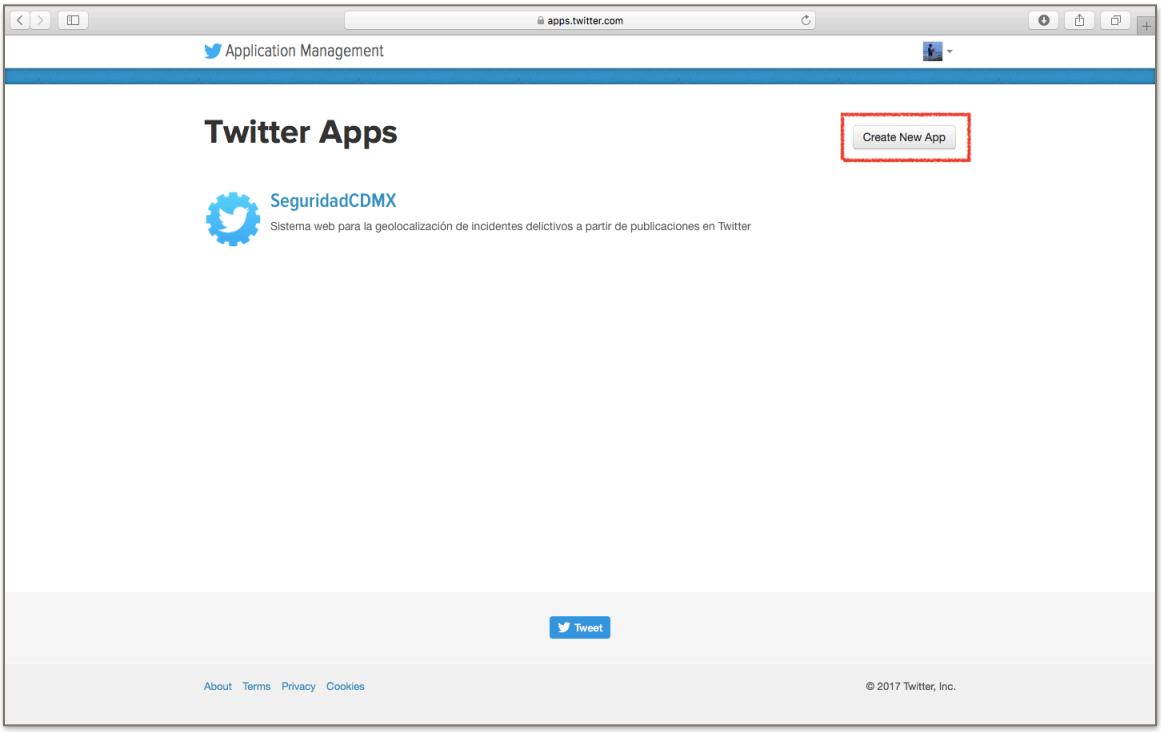


Figura 8. Creación de una aplicación en Twitter.

Una vez que se ha creado la aplicación entonces es posible acceder a un menú de opciones donde se puede, agregar detalles y configuraciones, pero lo más importante, se puede generar las **Keys** y los **Tokens** de acceso a Twitter, como se muestra en la Figura 9, los parámetros anteriores permiten establecer la conexión a Twitter desde el sistema web y consumir el flujo de tweets por medio del módulo Extractor.

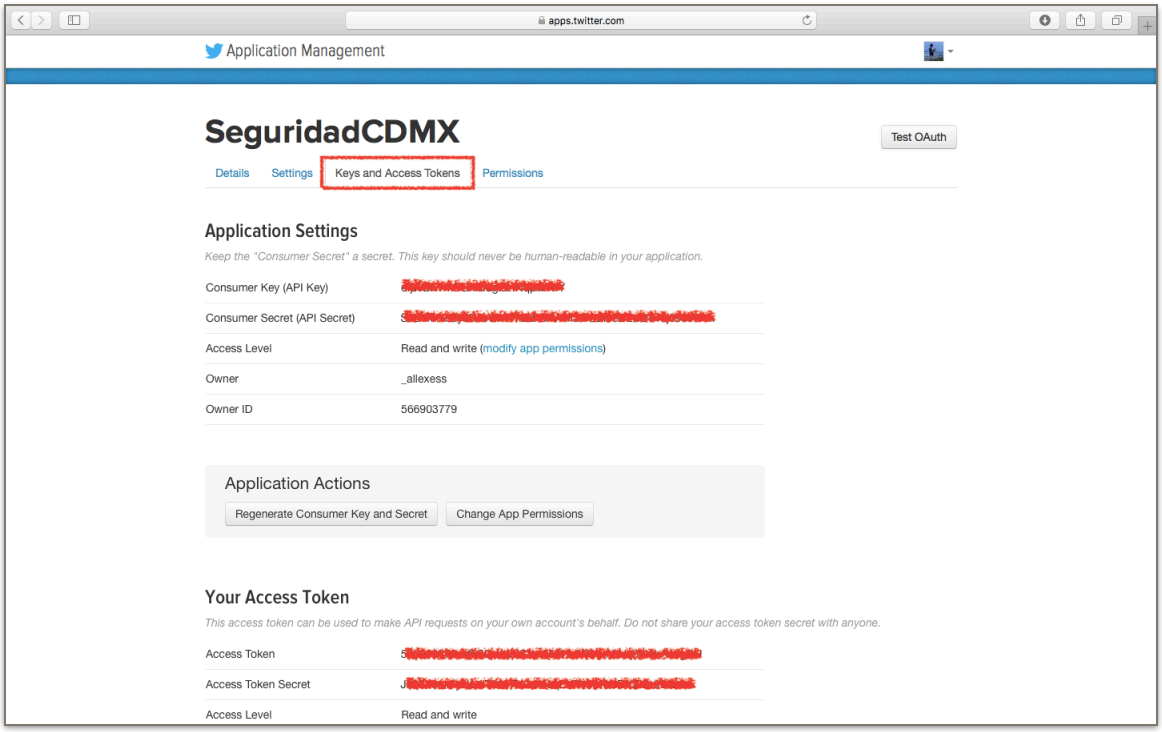


Figura 9. Creación Keys y Tokens de acceso a Twitter.

6.2.2. Conexión a la Streaming API de Twitter

Para establecer la conexión a la Streaming API de Twitter además de la obtención de los parámetros de acceso, se hizo uso de las clases **ConfigurationBuilder**, **TwitterStreamFactory** y **TwitterStream** pertenecientes a la librería Twitter4J.

Con la clase ConfigurationBuilder se construyó un objeto al que se le establecieron los Keys y Tokens de acceso, es decir que este objeto contiene la configuración necesaria para realizar la conexión. También se usó la clase TwitterStreamFactory que permite construir objetos a partir de otro objeto que contenga una configuración válida para conectarse a Twitter, así que esta clase se usó para instanciar un objeto TwitterStream el cual contiene la conexión a la Streaming API de Twitter.

Las clases anteriores y su implementación pueden apreciarse de forma más clara en el código de la clase **Extractor** presentado en la sección de apéndices 11.2.

6.2.3. Escuchador de tweets

La conexión requiere un objeto que este escuchando de manera permanente el flujo de tweets, así que se usó la clase **StatusListener** de la librería Twitter4J para crear el objeto anterior y añadirlo al objeto TwitterStream, para ello se creó una clase llamada **Listener** que implementa la interfaz StatusListener y sobre escribe los métodos de esta interfaz, en este caso solo fue necesario sobre escribir el método **onStatus**.

El método onStatus es llamado por TwitterStream cuando se recibe un nuevo tweet como un objeto de tipo **Status** que es una clase de la librería Twitter4J, esta clase proporciona los métodos necesarios para obtener los datos del JSON de un tweet. Se decidió que solo se tomarían en cuenta los nuevos tweets y se descartarían los retweets para limitar la duplicidad de los incidentes delictivos, una vez que el tweet entrante pasa esta condición Listener inicia su procesamiento para extraer la información requerida.

La implementación de las clases y métodos anteriores puede verse en los códigos de las clases **Extractor** y **Listener** encontrados en la sección de apéndices 11.2 y 11.3 respectivamente.

6.2.4. Filtrado de los tweets entrantes

Hasta este punto ya es posible hacer la conexión a la Streaming API y recibir tweets a través del Listener, pero de esta forma se recibiría todo el flujo de tweets, es decir, todos los tweets sin restricción alguna. Para limitar el flujo recibido se usó la clase **FilterQuery** de la librería Twitter4J que nos permite filtrar el flujo de tweets entrantes, en este caso el filtro aplicado fue por un conjunto de cuentas de Twitter definidas en el atributo **FOLLOW** de la clase **GlobalConstans** (sección de apéndices 11.4).

Para crear el filtro se instanció un objeto FilterQuery, así que se le pasaron dos parámetros a su constructor, el primero fue el atributo **COUNT** de la clase GlobalConstants que contiene el número de tweets a recuperar del historial (COUNT tiene un valor de 0 ya que el historial solo aplica para cuentas premium), el segundo parámetro fue FOLLOW, este último es un arreglo que contiene los id's de las cuentas de Twitter que nos interesan.

Luego se llamó al método **filterLevel** de la clase FilterQuery y se le pasó como parámetro la cadena "**low**", este método junto con el parámetro anterior establecen el nivel de filtrado para el flujo de tweets, en otras palabras limitan el número de tweets recibidos a aquellos que cumplen con los parámetros establecidos por el filtro, o sea los que pertenecen a los id's de las cuentas definidas en FOLLOW.

Por último se le añadió el objeto FilterQuery al objeto TwitterStream por medio su método **filter** y es este último quien inicia la conexión al Streaming API y el consumo del flujo de tweets.

La implementación de las clases y métodos anteriores puede verse en el código de la clase **Extractor** encontrada en la sección de apéndices 11.2.

6.2.5. Procesamiento y extracción de información de un tweet

Como ya se había mencionado anteriormente, tras la obtención de un tweet Listener inicia su procesamiento y se llevan a cabo una serie de tareas con el fin de extraer el posible incidente delictivo. Lo anterior se hace llamando al método **process** de la clase **Processor** para comenzar la extracción del posible incidente delictivo, este método recibe como parámetro el objeto Status quien contiene el tweet entrante. Estas tareas se listan y describen a continuación de manera más detallada y pueden ser vistas en el diagrama de clases de Figura 10.

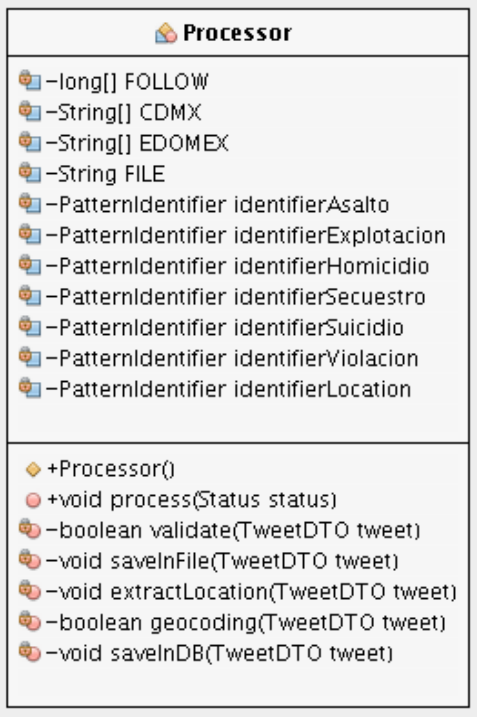


Figura 10. Diagrama de clases de Processor.

I. Comprobación del origen de un tweet

La primera tarea de Processor consiste en comprobar el origen del tweet entrante, es decir, que provenga de alguna de las cuentas de Twitter que han sido determinadas como confiables. Esta comprobación se lleva a cabo en el método process (sección de apéndices 11.5) y consiste en comparar si el id de la cuenta que publicó el tweet coincide con alguno de los id's contenidos en el arreglo de la constante FOLLOW, si la condición anterior se cumple entonces puede dar inicio la siguiente tarea, de lo contrario el tweet es descartado y las tareas de Processor se dan por terminadas.

II. Identificación de patrones textuales en un tweet

Esta tarea de Processor consiste en validar el tweet entrante por medio de expresiones regulares y de la identificación de patrones textuales de incidentes delictivos presentes en el texto contenido en el tweet.

Para realizar esta tarea se hace uso del método **validate** (sección de apéndices 11.6) de la clase Processor y de la clase **TextualPattern** (sección de apéndices 11.7) que contiene la definición de los patrones textuales. El método validate determina si el texto contenido en el tweet analizado cumple o no con los patrones textuales, identifica el tipo de delito y devuelve un valor booleano **true** o **false** para indicar si se tuvo éxito en este análisis, lo anterior se realiza con ayuda de la clase **PatternIdentifier** (sección de apéndices 11.8) la cual aplica las expresiones regulares definidas por los patrones textuales. En la Tabla 2 se muestran tres ejemplos de patrones textuales aplicados a tweets para identificar incidentes delictivos de tipo Homicidio.

Patron textual	Texto que identifica
(asesina|asesinar|asesinaron|asesinó|asesinan)	"Pasajero **asesina** a custodio del reclusorio Oriente en microbús #Iztapalapa"
(matar|mata|mató|mataron|matan|murieron)	"Irrumpen hombres armados en Cruz Verde y **matan** a paciente"
(homicidio|homicidios)	"La @PGR_mx participará en la invetigación del **homicidio** del vicepresidente de @Televisa"

Tabla 2. Patrones textuales para identificar homicidios.

Si el tweet resulta valido entonces se procede con la siguiente tarea, de lo contrario el tweet es descartado y las tareas de Processor se dan por terminadas. En la Figura 11 se presenta el diagrama de clases del paquete que contiene a TextualPattern y PatternIndentifier.

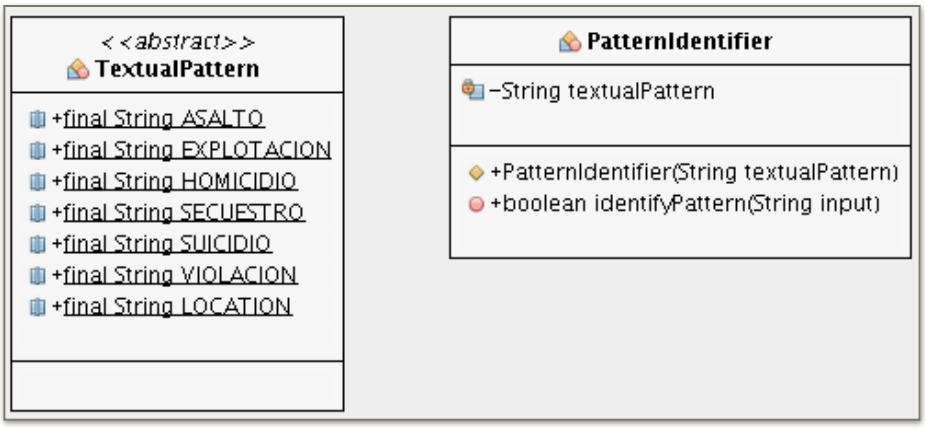


Figura 11. Diagrama de clases del paquete extractor.logic.regex.

III. Extracción y geocodificación de la ubicación de un tweet

Esta tarea se divide en dos partes, una es extracción de la ubicación donde se suscitó el incidente delictivo del texto contenido en el tweet y la otra es la geocodificacion de la ubicación anterior.

La extracción de la ubicación se realiza de la siguiente forma:

i. Se divide el texto del tweet en tokens donde cada palabra es un token.

ii. Se busca token por token la palabra “en”, la cual se considera un patrón textual que indica una ubicación, los tokens anteriores al que contiene esta palabra son descartados.

iii. Se busca la primer aparición de un token que contenga una palabra que comience con letra mayúscula, a partir de aquí se toman al menos 1 token y a lo más 4, esto si es que existen más tokens. Si no se encuentra la palabra buscada el tweet es descartado y las tareas del Processor se dan por terminadas.

iv. Con los tokens tomados se forma una cadena que contiene la información de la ubicación donde se suscitó el incidente delictivo.

La serie de pasos anteriormente descrita es llevada a cabo por el método **extractLocation** (sección de apéndices 11.9) de la clase Processor, la cadena obtenida es usada en la siguiente tarea.

La geocodificación de la ubicación es ejecutada por el método **geocoding** (sección de apéndices 11.10) de la clase Processor y consiste en consumir el servicio web de geocodificación de la Geocoding API de Google Maps, para esto se envía como parámetro la ubicación contenida en la cadena extraída anteriormente y se recibe como respuesta un JSON que contiene las coordenadas de latitud y longitud así como el país, el estado, la ciudad y otros datos asociados a la ubicación geocodificada.

Si la respuesta obtenida satisface las condiciones de ubicación definidas en los atributos **CDMX** y **EDOMEX** de la clase GlobalConstants, es decir, que la respuesta JSON tenga como resultados que la ubicación pertenezca a México y que se encuentre dentro de la Ciudad de México o del Estado de México, entonces la ubicación y su geocodificación son validas por tanto se procede con la siguiente tarea, en caso contrario el tweet es descartado y las tareas de Processor se dan por terminadas.

Lo anterior se realiza con ayuda de las clases **Gecodic** (sección de apéndices 11.11) y **Geocodificator** (sección de apéndices 11.12), en la Figura 12 se muestra el diagrama de clases del paquete que las contiene.

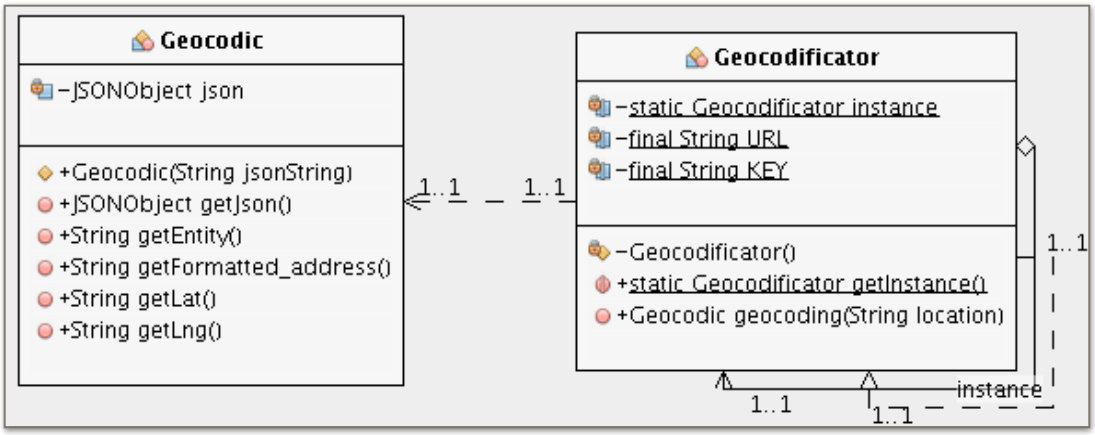


Figura 12. Diagrama de clases del paquete extractor.logic.geocodification.

IV. Almacenamiento de un tweet en la base de datos

Al llegar a este punto el tweet ha pasado los filtros necesarios para considerar que contiene la información buscada y requerida para el fin del sistema web, es decir que se consiguió extraer un incidente delictivo identificado y geocodificado. Por tanto la última tarea de Processor da comienzo y consiste en crear un registro con los datos obtenidos a partir del JSON del tweet procesado y las coordenadas de geolocalización obtenidas de la geocodificación de la ubicación del incidente delictivo.

Esta tarea es efectuada por el método **saveInDB** (sección de apéndices 11.13) de la clase Processor y con ayuda de las clases **TweetDAO** (sección de apéndices 11.14), **TweetDTO** (sección de apéndices 11.15) y la conexión a la base de datos proporcionada por **ConnectionMySQL** (sección de apéndices 11.16), en la Figura 13 se muestra su diagrama de clases.

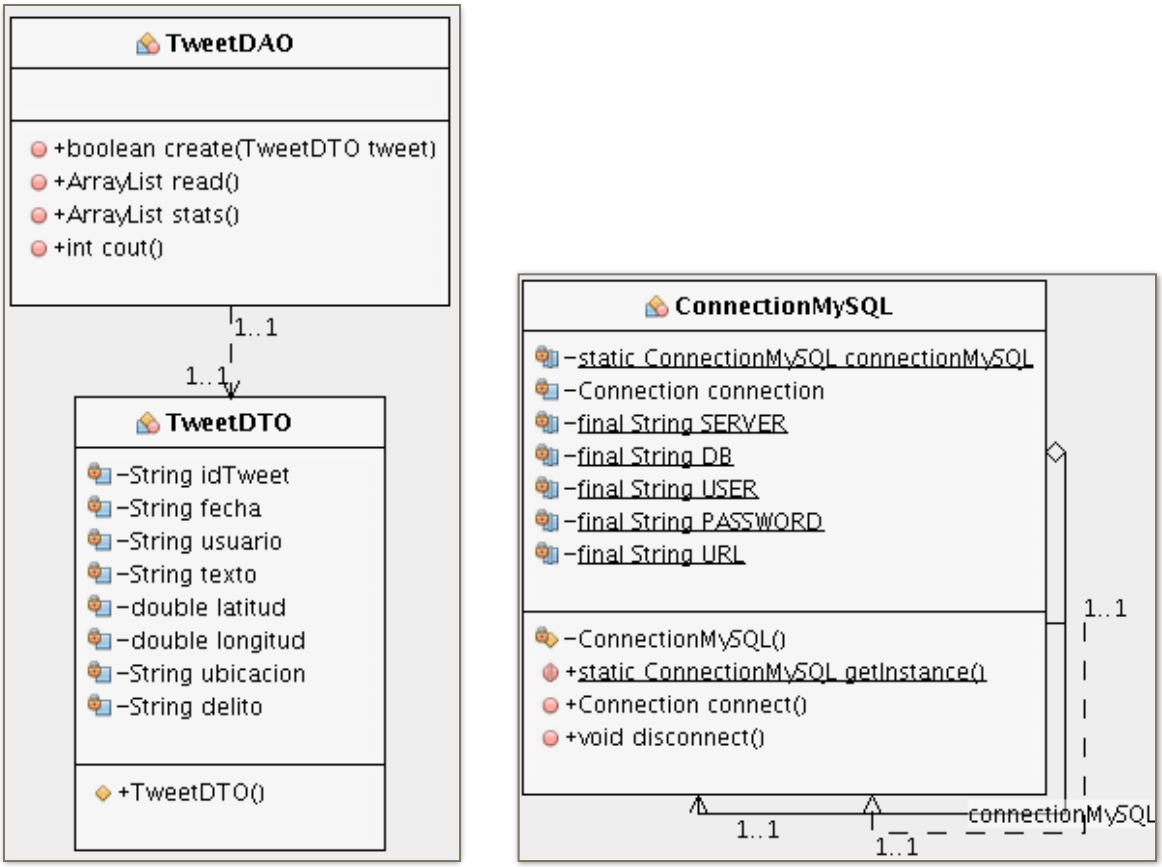


Figura 13. Diagrama de clases de TweetDAO, TweetDTO y ConnectionMySQL.

6.3. Aplicación

Este último módulo corresponde a la aplicación web que presenta a través del navegador los incidentes delictivos extraídos a partir de los tweets obtenidos por el módulo Extractor. Esta aplicación web fue construida con un diseño responsivo, es decir, que es adaptable para los distintos tamaños de pantalla de móviles, tablets, desktops, laptops, etc., se encuentra accesible en aisii.azc.uam.mx:8080/SeguridadCDMX y en ella se muestra de forma gráfica los incidentes delictivos y su geolocalización en un mapa de Google Maps. En la Figura 14 se muestra algunas capturas de pantalla de la versión para móviles.

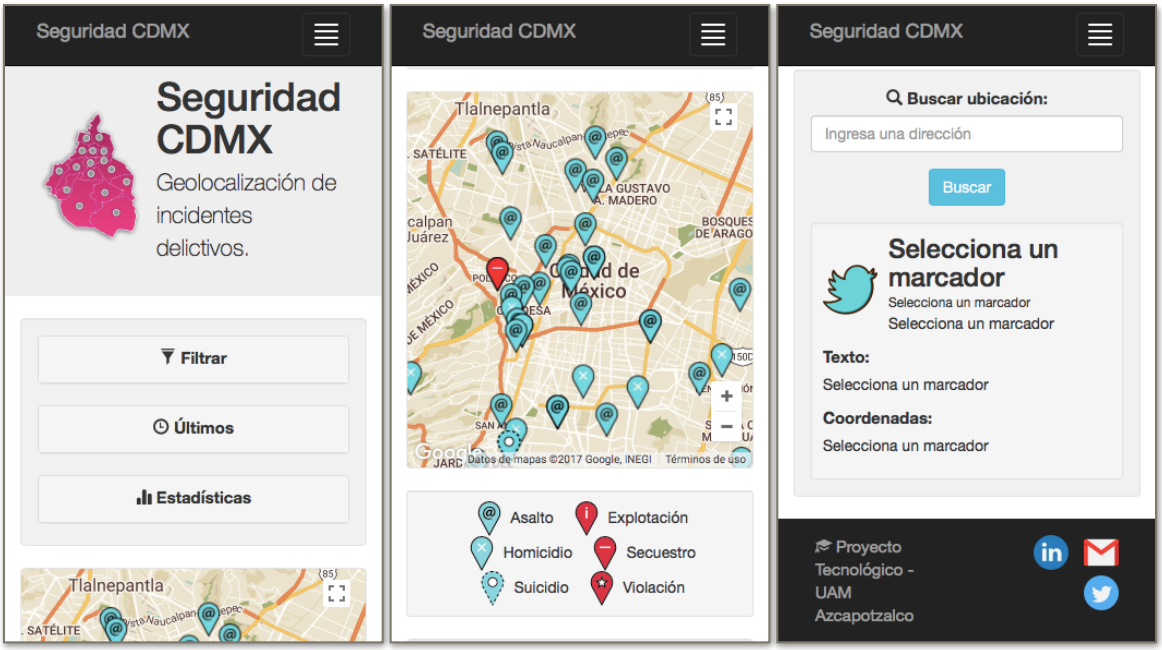


Figura 14. Versión para móviles de la aplicación web.

La página principal está dividida en tres secciones primordiales que son: **Mapa con geolocalización de incidentes delictivos, Panel de información** y **Panel de herramientas**, estas secciones pueden apreciarse en la Figura 15.

Figura 15. Página principal de la aplicación web (versión de escritorio).

6.3.1. Mapa con geolocalización de incidentes delictivos

Esta sección contiene el mapa construido por medio de Google Maps JavaScript API en él se presenta la geolocalización de los incidentes delictivos por medio de marcadores específicos para cada tipo de delito y por cada tweet alojado en la base de datos, estos marcadores indican la posición geográfica donde se suscitó el incidente delictivo.

La sección también incluye las leyendas que indican el tipo de delito para cada marcador, además al hacer clic sobre alguno de los marcadores posicionados sobre el mapa se muestra una ventana con el tipo de delito y cuenta que publicó el tweet, lo anterior se presenta en la Figura 16.

Figura 16. Ventana de marcador.

6.3.2. Panel de información

En esta sección se muestra la información más relevante sobre el incidente delictivo, específicamente se presenta el tipo de delito, la fecha de extracción, la cuenta que publicó el tweet, el texto contenido en el tweet y sus coordenadas de geolocalización. Esta información se muestra al hacer clic en algún marcador del mapa o al tener activo algún marcador por medio del panel de herramientas.

La sección también incluye un buscador de ubicaciones, este requiere que el usuario ingrese una dirección o lugar y que presione “Enter” o haga clic en el botón “Buscar”, luego la aplicación posiciona y centra el mapa en la ubicación buscada colocando un marcador distintivo. Las funcionalidades de esta sección pueden verse en la Figura 17.

Figura 17. Funcionalidades del Panel de información.

6.3.3. Panel de herramientas

Esta sección contiene tres subpaneles con funcionalidades que ayudan a los usuarios en la navegación a través del mapa, los cuales se describen enseguida.

- **Filtrar**: este subpanel permite aplicar un filtro sobre los tipos de delito presentados en el mapa, es decir, da la posibilidad de ocultar y mostrar los marcadores del mapa. (Figura 18)

- **Últimos**: este subpanel presenta los últimos cinco incidentes delictivos extraídos junto con su tipo de delito, además al hacer clic sobre alguno de ellos la aplicación centra el mapa sobre el marcador correspondiente y muestra la información del incidente delictivo tanto en la venta del marcador como en el Panel de Información. (Figura 19)

- **Estadísticas**: este subpanel da a conocer el número de incidentes delictivos extraídos por tipo de delito, este siempre se encuentra desplegado en la página principal para la versión de escritorio.

Tanto los subpaneles como el mapa se actualizan en caso de que se identifique un nuevo incidente y la aplicación notifica al usuario lanzando una alerta a la pantalla.

Figura 18. Subpanel Filtrar.

Figura 19. Subpanel Últimos.

7. Resultados

En un lapso de 10 días que el Módulo Extractor se mantuvo en funcionamiento se lograron identificar en los tweets extraídos 40 incidentes delictivos, de estos 28 pertenecieron a la categoría de asaltos, 11 a la de homicidio y 1 a la de secuestro, en términos de porcentajes se obtuvieron los resultados mostrados en la Figura 20.

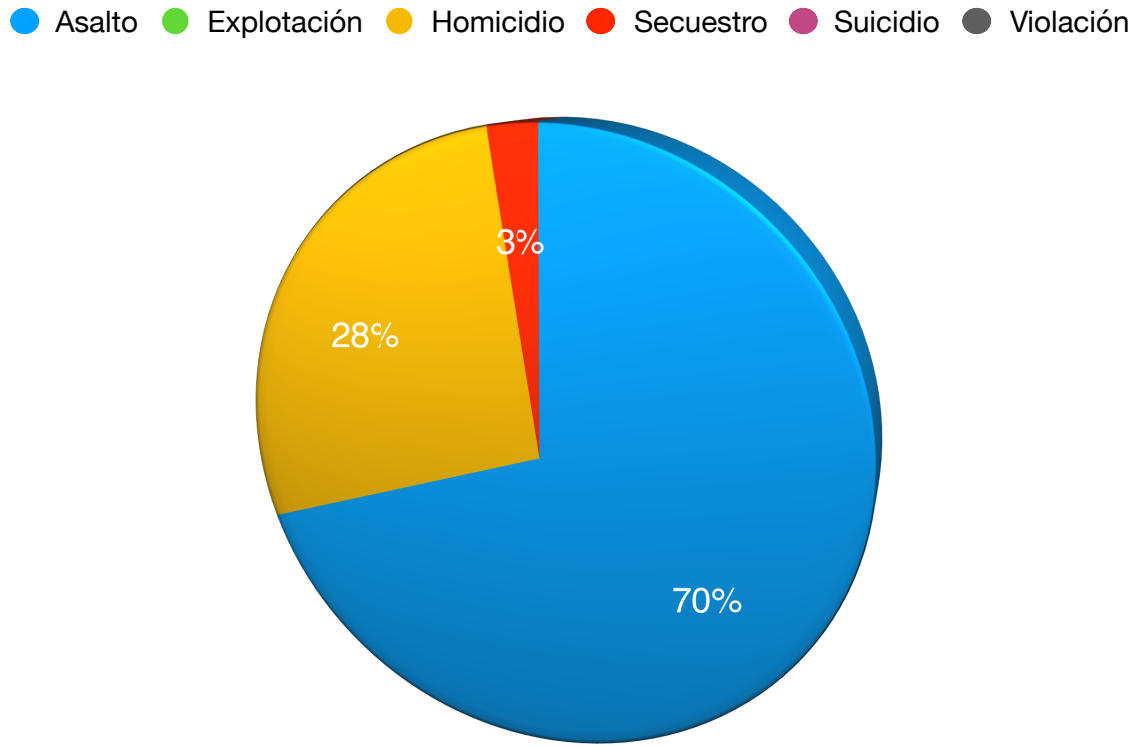


Figura 20. Porcentajes de incidentes identificados por tipo de delito.

Estos 40 incidentes delictivos identificados fueron etiquetados de forma manual para determinar la precisión de los resultados obtenidos y las tareas realizadas por el módulo Extractor, para medir esta precisión se definieron cinco factores a evaluar **Origen**, **Patrones textuales**, **Ubicación**, **Geocodificación** y **Contexto**, además de tres rangos de precisión, **Buena**, **Aceptable** y **Mala**, los resultados obtenidos después de etiquetar se presentan en la Tabla 3.

Factor / Precisión	Buena	Aceptable	Mala
Origen	40	0	0
Patrones textuales	36	1	3
Ubicación	30	2	8
Geogodificación	25	8	7
Contexto	33	4	3

Tabla 3. Precisión del módulo Extractor.

Para fines de comparación los resultados de la Tabla 2 se presentan de forma más clara en la Figura 21.

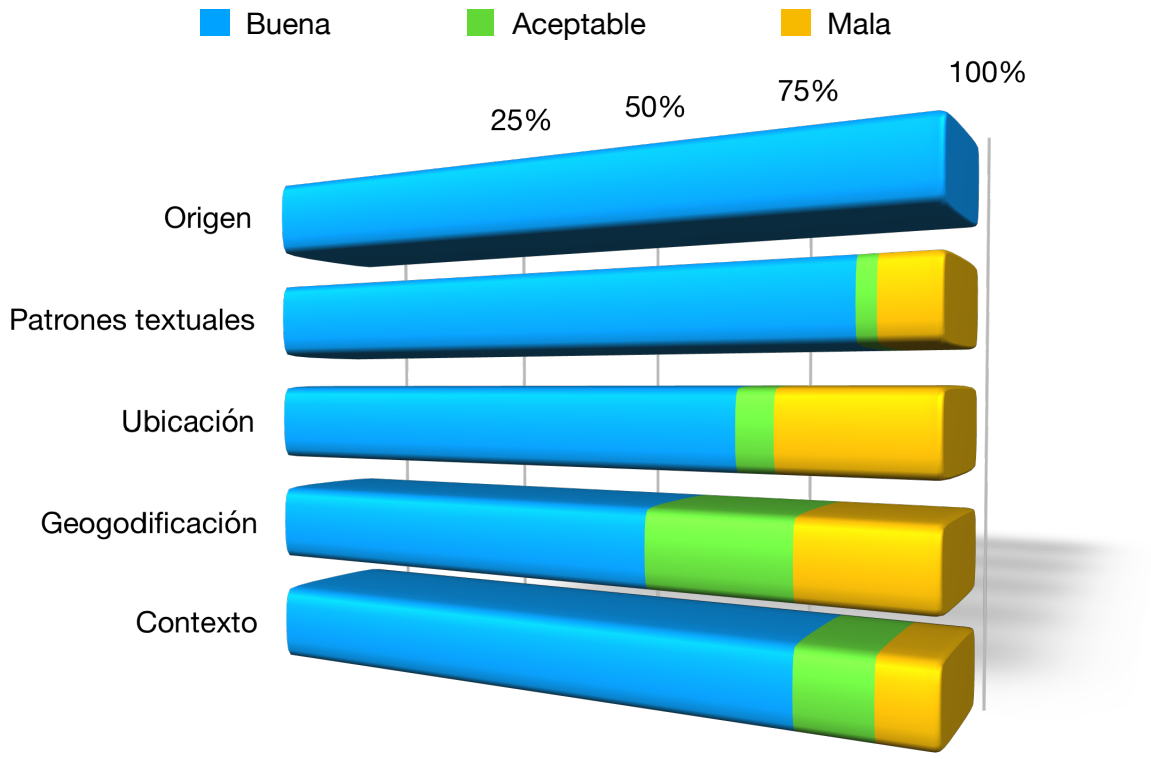


Figura 21. Comparativo de la precisión del módulo Extractor.

8. Análisis y discusión de resultados

De acuerdo a los resultados presentados en la sección anterior, se realizó un análisis cuantitativo de la precisión de las tareas involucradas en la extracción de los incidentes delictivos respecto a los factores de evaluación designados previamente, para poder clasificar las fortalezas del sistema web y las áreas de oportunidad en las que este puede mejorar. En la Tabla 4 se presenta la clasificación de los factores de evaluación por el porcentaje de precisión obtenido luego de etiquetar manualmente los tweets extraídos.

Factor / % de precisión	Buena	Aceptable	Mala
Origen	100.00%	0.00%	0.00%
Patrones textuales	90.00%	2.50%	7.50%
Ubicación	75.00%	5.00%	20.00%
Geocodificación	62.50%	20.00%	17.50%
Contexto	82.50%	10.00%	7.50%

Tabla 4. Clasificación de los factores de evaluación por su % de precisión.

De a cuerdo a la clasificación presentada en la tabla anterior, el sistema web tiene como principales fortalezas el “Origen”, es decir, la obtención de tweets de las fuentes confiables preestablecidas y los “Patrones textuales” o sea la identificación de incidentes delictivos en los textos de los tweets obtenidos.

Asimismo se determinaron como labores aceptables pero mejorables la “Ubicación” que se refiere extracción de las ubicaciones donde se suscitaron los incidentes delictivos de los tweets obtenidos y el “Contexto” el cual describe la relación real que existe entre la información contenida en los tweets y los incidentes delictivos extraídos, en otras palabras, si los tweets realmente hablan de delitos ocurridos en las entidades preestablecidas.

Sin embargo la “Geocodificación” se determinó como un aspecto a replantear y optimizar en cuanto a su proceso de obtención, debido a que fue el factor con el menor porcentaje de precisión “Buena” y a su vez el segundo mayor en cuanto a precisión “Mala”. Estos porcentajes se debieron a que la

ubicación contenida en algunos tweets no se refería a un lugar localizado en la Ciudad de México o el Estado de México, pero en estas entidades si existe una ubicación con ese nombre.

Debido a lo anterior al realizar la geocodificación con la Google Maps Geocoding API esta nos devuelve un resultado correcto de acuerdo a los parámetros enviados, pero incorrecto en relación al contexto geográfico, un ejemplo de ello se muestra en el tweet de la Figura 22 el cual se refiere a la ciudad de Tampa en Florida sin embargo el tweet se geolocaliza en una calle llamada Tampa de la Ciudad de México.

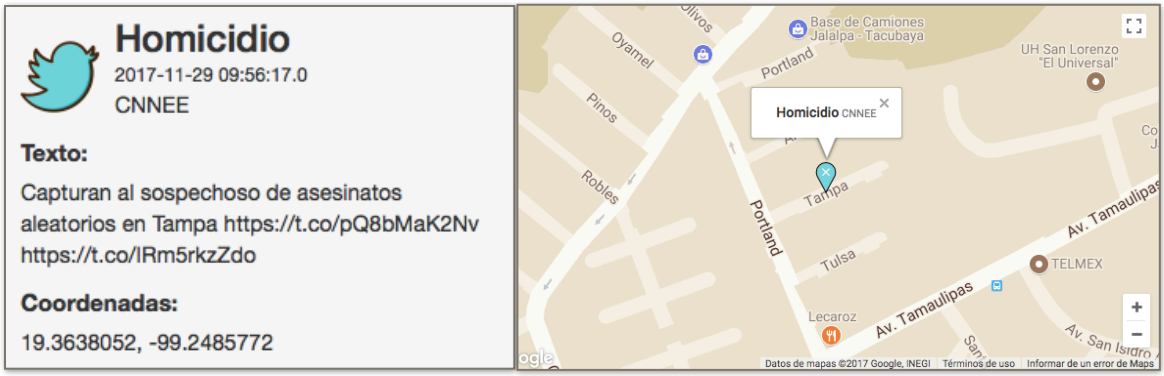


Figura 22. Tweet con error de contexto geográfico.

Otras áreas de oportunidad en las que el sistema web puede ser optimizado son:

- Detección de dos o mas tipos de delitos presentes en un solo tweet. Ejemplo: “Viola y asesina a menor de edad en la delegación Iztapalapa”.

- Mejorar los patrones textuales para la extracción de la ubicación, por ejemplo: “Reportando en vivo autoridades detienen al Chapo Guzman en la Ciudad de México”, ya que de acuerdo a la lógica del módulo Extractor la ubicación extraída sería la cadena “Chapo Guzman en la”.

- Duplicidad de incidentes delictivos debido a acontecimientos muy destacados, es decir que distintas cuentas publican tweet refiriéndose al mismo incidente delictivo, pero el módulo Extractor considera que son distintos.

9. Conclusiones

Se consiguió desarrollar un sistema web que provee un vista actual de los incidentes delictivos que se suscitan día a día en la Ciudad de México y el Estado de México, tras el término de este proyecto y su puesta en funcionamiento, el sistema web hizo claro la gran utilidad y potencial de este tipo de herramientas, debido a la cantidad y calidad de los datos que son recabados y clasificados, pues la forma en que la aplicación web presenta al público esta información permite tener una perspectiva diferente a lo convencional, pero real de la situación en cuestión de seguridad y delincuencia que se vive en el presente de la Ciudad de México y el Estado de México.

De igual forma este desarrollo demostró que sin duda las redes sociales como Twitter y los datos que nos proporcionan son muy valiosos, ya que si son analizados de forma que se les pueda abstraer una interpretación sencilla y entendible, se pueden obtener grandes beneficios, ya sea para contribuir con la sociedad como en este caso o para llevar la aplicación de estas tecnologías a otros sectores como empresas, marcas, industrias o gobiernos.

Lo anterior se debe a que si se cuenta con una gran cantidad de información sobre un tema específico y se tiene estructurada para darle sentido con el propósito de conseguir la mejor interpretación de acuerdo a una necesidad, entonces se tendrá un panorama único sobre la actualidad y la tendencia del tema, así como una mejor aproximación a una posible solución, de esta forma la interpretación y el panorama obtenidos pueden ser aprovechados para fines sociales, políticos o incluso económicos.

El análisis de los resultados obtenidos y la precisión en la extracción de los incidentes delictivos que ronda el 80% de correctitud, se traduce en que la aplicación web proporciona información certera sobre los tipos de delito y su geolocalización, los factores anteriores permiten afirmar que el sistema web cumple la finalidad para la que fue desarrollado, es decir, que se logró producir una herramienta tecnológica que da a conocer las zonas con alta incidencia delictiva y permite a la sociedad tomar medidas preventivas al respecto.

Por último cabe mencionar que tanto la información obtenida como la lógica del sistema y la aplicación web pueden ser reutilizadas e implementadas para extender la funcionalidad de esta herramienta hacia una aplicación para dispositivos móviles y mediante técnicas de minería de datos, prever y alertar en tiempo real a los ciudadanos del posible peligro al transitar en la vía pública.

10. Referencias bibliográficas

[1] (INEGI), "Encuesta Nacional de Victimización y Percepción sobre Seguridad Pública (ENVIPE) 2016", Beta.inegi.org.mx, 2017. [Online]. Available: http://www.beta.inegi.org.mx/proyectos/enchogares/regulares/envipe/2016/. [Accessed: 15- Jun- 2017].

[2] "Clasificación Estadística de Delitos (CED)", Www3.inegi.org.mx, 2017. [Online]. Available: http://www3.inegi.org.mx/sistemas/clasificaciones/delitos.aspx. [Accessed: 15- Jun- 2017].

[3] M. Reyes García, "Aprendizaje automático de patrones textuales para la identificación de eventos sobre la seguridad ciudadana", proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2016.

[4] E. F. Rosas Coronado, "Sistema de información para el análisis de la seguridad social o pública a través de periódicos", proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2014.

[5] A. Amores Bassante and C. Aldheir Suárez Loaiza, "Diseño y desarrollo de un prototipo de aplicación web y móvil nativa android para el monitoreo de seguridad en el Distrito Metropolitano de Quito a través de la red social twitter", proyecto terminal, Licenciatura, Universidad de las Américas, 2017.

[6] R. A. Ruvalcaba Flores, "Sistema Web para gestionar incidentes delictivos", propuesta de proyecto terminal, División de Ciencias Básicas e Ingeniería, Universidad Autónoma Metropolitana Azcapotzalco, México, 2017.

[7] "Alertux - Alertas Ciudadanas que salvan vidas", Alertux.com, 2017. [Online]. Available: http://alertux.com. [Accessed: 15- Jun- 2017].

[8] Mi policía, "En defensa de la sociedad", [Online]. Available: en: https://play.google.com/store/apps, [Accessed: 15- Jun- 2017]

[9] "La popularidad en redes sociales de los periódicos crece de manera sostenida", PuroMarketing, 2017. [Online]. Available: http://www.puromarketing.com/42/28042/popularidad-redes-sociales-periodicos-crece-manera-sostenida.html. [Accessed: 15- Jun- 2017].

[10] Developer.twitter.com. (2017). Docs. [online] Available at: https://developer.twitter.com/en/docs [Accessed 1 Dec. 2017].

[11] Developer.twitter.com. (2017). Introduction to Tweet JSON. [online] Available at: https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/intro-to-tweet-json.html [Accessed 1 Dec. 2017].

[12] Twitter4j.org. (2017). Twitter4J - A Java library for the Twitter API. [online] Available at: http://twitter4j.org [Accessed 1 Dec. 2017].

[13] Beltran Lopez, G. (2012). Geolocalizacion y redes sociales. Bubok Publishing S L.

[14] Google Developers. (2017). Primeros pasos | Google Maps Geocoding API | Google Developers. [online] Available at: https://developers.google.com/maps/documentation/geocoding/start?hl=es [Accessed 1 Dec. 2017].

[15] Google Developers. (2017). Primeros pasos | Google Maps JavaScript API | Google Developers. [online] Available at: https://developers.google.com/maps/documentation/javascript/tutorial?hl=es [Accessed 1 Dec. 2017].

[16] Mozilla Developer Network. (2017). Expresiones Regulares. [online] Available at: https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Regular_Expressions [Accessed 1 Dec. 2017].

11. Apéndices

11.1. Script para la Base de Datos e inserción de registros

```
SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

CREATE SCHEMA IF NOT EXISTS `seguridadCDMX` DEFAULT CHARACTER SET utf8 ;
USE `seguridadCDMX` ;

CREATE TABLE IF NOT EXISTS `seguridadCDMX`.`delito` (
  `tipo` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`tipo`))
ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS `seguridadCDMX`.`tweet` (
  `idTweet` VARCHAR(20) NOT NULL,
  `fecha` DATETIME NULL DEFAULT CURRENT_TIMESTAMP,
  `usuario` VARCHAR(20) NULL,
  `texto` VARCHAR(200) NULL,
  `latitud` REAL NULL,
  `longitud` REAL NULL,
  `delito_tipo` VARCHAR(15) NOT NULL,
  PRIMARY KEY (`idTweet`),
  INDEX `fk_Tweet_Delito_idx` (`delito_tipo` ASC),
  CONSTRAINT `fk_Tweet_Delito`
    FOREIGN KEY (`delito_tipo`)
    REFERENCES `seguridadCDMX`.`delito` (`tipo`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

INSERT INTO `seguridadCDMX`.`delito` (`tipo`) VALUES ('Asalto');
INSERT INTO `seguridadCDMX`.`delito` (`tipo`) VALUES ('Explotación');
INSERT INTO `seguridadCDMX`.`delito` (`tipo`) VALUES ('Homicidio');
INSERT INTO `seguridadCDMX`.`delito` (`tipo`) VALUES ('Secuestro');
INSERT INTO `seguridadCDMX`.`delito` (`tipo`) VALUES ('Suicidio');
INSERT INTO `seguridadCDMX`.`delito` (`tipo`) VALUES ('Violación');
```

11.2. Extractor.java

```
public class Extractor {

    private static Extractor instance = null;
    private static final String CONSUMER_KEY = "fOfbyxZrR8IpsPURmusss";
    private static final String CONSUMER_SECRET = "sGzf8t8n0x1HAqnLxR";
    private static final String ACCESS_TOKEN = "125642462-6BsAg8XVGOf";
    private static final String ACCESS_TOKEN_SECRET = "bUajsbFhcRRLA0";

    private Extractor() {}

    public static Extractor getInstance() {
        if (instance == null) {
            instance = new Extractor();
        }
        return instance;
    }

    public void run() throws TwitterException {
        ConfigurationBuilder cb = new ConfigurationBuilder();
        cb.setDebugEnabled(true)
                .setOAuthConsumerKey(CONSUMER_KEY)
                .setOAuthConsumerSecret(CONSUMER_SECRET)
                .setOAuthAccessToken(ACCESS_TOKEN)
                .setOAuthAccessTokenSecret(ACCESS_TOKEN_SECRET);
        TwitterStreamFactory tsf = new TwitterStreamFactory(cb.build());
        TwitterStream ts = tsf.getInstance();
        ts.addListener(new Listener());
        FilterQuery fq = new
               FilterQuery(GlobalConstants.COUNT, GlobalConstants.FOLLOW);
        fq.filterLevel("low");
        ts.filter(fq);
    }

}
```

11.3. Listener.java

```
public class Listener implements StatusListener {

    private final Processor processor;

    public Listener() {
        processor = new Processor();
    }

    @Override
    public void onStatus(Status status) {
        if (!status.isRetweet()) {
            processor.process(status);
        }
    }

    @Override
    public void onDeletionNotice(StatusDeletionNotice sdn) {}

    @Override
    public void onTrackLimitationNotice(int i) {}

    @Override
    public void onScrubGeo(long l, long l1) {}

    @Override
    public void onStallWarning(StallWarning sw) {}

    @Override
    public void onException(Exception excptn) {}

}
```

11.4. GlobalConstants.java

```
public abstract class GlobalConstants {

    public static final long[] FOLLOW = new long[]{
       16676396, 27708897, 24969337, 104683173, 14917589, 62844209, 84613584,
       403486458, 158511120,111110077, 520653311, 33884545, 14135853, 459516221,
       35977487, 551446885, 104822018, 35541975,19889168, 121165363, 246485892,
       246450356, 60486251, 105594215, 109631064, 25401359, 566903779
    };

    public static final int COUNT = 0;

    public static final String[] CDMX = new String[]{
       "CDMX", "Ciudad de México”
    };

    public static final String[] EDOMEX = new String[]{
       "Méx.", "Estado de México”
    };

    public static final String FILE = "tweets.txt";
}
```

11.5. Código del método process

```
public void process(Status status) {
        TweetDTO tweet = new TweetDTO();
        long id = status.getUser().getId();
        for (long idFollow : FOLLOW) {
            if (id == idFollow) {
                tweet.setIdTweet(String.valueOf(status.getId()));
                tweet.setFecha(status.getCreatedAt().toString());
                tweet.setUsuario(status.getUser().getScreenName());
                tweet.setTexto(status.getText());
                if (validate(tweet)) {
                    saveInFile(tweet);
                    System.out.println(tweet);
                    extractLocation(tweet);
                    if (geocoding(tweet)) {
                        saveInDB(tweet);
                        break;
                    }
                }
            }
        }
}
```

11.6. Código del metodo validate

```
private boolean validate(TweetDTO tweet) {
        if (identifierLocation.identifyPattern(tweet.getTexto())) {
            if (identifierAsalto.identifyPattern(tweet.getTexto())) {
                tweet.setDelito(Delito.ASALTO);
                return true;
            } else if (identifierExplotacion.identifyPattern(tweet.getTexto())) {
                tweet.setDelito(Delito.EXPLOTACION);
                return true;
            } else if (identifierHomicidio.identifyPattern(tweet.getTexto())) {
                tweet.setDelito(Delito.HOMICIDIO);
                return true;
            } else if (identifierSecuestro.identifyPattern(tweet.getTexto())) {
                tweet.setDelito(Delito.SECUESTRO);
                return true;
            } else if (identifierSuicidio.identifyPattern(tweet.getTexto())) {
                tweet.setDelito(Delito.SUICIDIO);
                return true;
            } else if (identifierViolacion.identifyPattern(tweet.getTexto())) {
                tweet.setDelito(Delito.VIOLACION);
                return true;
            }
        }
        return false;
}
```

11.7. TextualPattern.java

```
public abstract class TextualPattern {

    public static final String ASALTO = "((.)* )*("
            + "robo"
            + "|asaltar"
            + "|asalto"
            + "|robar"
            + "|asaltos"
            + "|asaltó"
            + "|despojar (a|de|con)"
            + "|robó"
            + "|amagar (a|con)"
            + "|atraco"
            + "|fue (asaltado|asaltada)"
            + "|robo (agravado|agraviado|con violencia|organizado)"
            + "|meter a robar"
            + "|ser asaltado"
            + "|atracar"
            + "|(han|habia|haber) robado"
            + "|se (roba|roban|robaron)"
            + ")( (.)*)*";

    public static final String EXPLOTACION = "((.)* )*("
            + "explotación sexual"
            + "|trata de (personas|menores|infantes|mujeres|blancas)"
            + "|obligadas a (ejercer |dedicarse )?(a )?(el )?"
            + "(sexoservicio|prostituirse|tener relaciones sexuales)"
            + "|(el|un) lenocinio"
            + "|tráfico sexual"
            + "|(explotaba|explotó) sexualmente a"
            + "|turismo sexual"
            + "|explotación"
            + "|prostitución"
            + "|trata sexual"
            + ")( (.)*)*";

    public static final String HOMICIDIO = "((.)* )*("
            + "(asesina|asesinar|asesinaron|asesinó|asesinan)"
            + "|(fue|fueran|fueron) (asesinada(s)|asesinado(s))"
            + "|(homicidio|homicidios)"
            + "|(asesinato|asesinatos)"
            + "|(matar|mata|mató|mataron|matan|murieron)"
            + "|feminicidio"
            + "|(fue|fueran|fueron) (baleados|baleado|baleadas|baleada)"
            + "|(ejecutan|ejecutaron|ejecutó|ejecuta|ejecutar) a"
            + "|(disparan|dispararon|disparó|dispara) a"
            + "|(fue|fueran|fueron) (ejecutados|ejecutadas|ejecutado|ejecutada)"
            + "|(balean|balearon|baleó|balea) a"
            + "|(fue|fueran|fueron|sido) (privada|privados|privado|”
            + "privadas) de la vida"
            + "|(fue|fueran|fueron|sido) (hallada|halladas|hallado|hallados) sin vida"
            + "|(fue|fueran|fueron) (ultimado|ultimados|ultimadas|ultimada)"
            + "|(ejecución|ejecuciones) de"
            + "|dio muerte a"
            + "|multihomicidio"
            + ")( (.)*)*";
```

```
    public static final String SECUESTRO = "((.)* )*("
            + "fue secuestrad(a|o)"
            + "|secuestr(o|ó)"
            + "|secuestrar"
            + "|privar de (su|la) libertad"
            + "|privación (ilegal )?de (la|su) libertad"
            + "|raptar"
            + "|(perpetrar|reportar) un secuestro"
            + "|acusar de secuestro"
            + ")( (.)*)*";

    public static final String SUICIDIO = "((.)* )*("
            + "se (suicida|suicidó|suicidaría|suicidan|suicidará|suicidaron)"
            + "|se (quita|quitó|quitarán|quitaron|quitara|quitan) la vida"
            + "|se (ahorca|ahorcó|ahorcaba)"
            + "|(el|un|su) suicidio"
            + "|quitarse la vida"
            + "|suicidarse"
            + "|(termina|terminan|terminó|terminar) con su vida"
            + "|se (lanza|lanzó) (a|de|desde)"
            + "|se (arroja|arrojó|arrojan) (a|de|desde)"
            + "|se (colgó|cuelga|cuelgan)"
            + "|se (había|habría) (suicidado|ahorcado|quitado la vida)"
            + "|fue (encontrad(a|o)|hallad(a|o)) (ahorcad(a|o)|colgad(a|o)|sin vida)"
            + "|lanzarse (a|de|desde)"
            + "|se (dispara|disparó)"
            + "|se dio un (disparo|tiro)"
            + "|se cuelga (a|de|desde)"
            + "|acabar con su (vida|existencia)"
            + "|se (avienta|aventó|avientan) (a|de|desde)"
            + ")( (.)*)*";

    public static final String VIOLACION = "((.)* )*("
            + "abuso(s) (sexual|sexuales|incestuoso|de orden sexual)"
            + "|violar a"
            + "|abusar"
            + "|violación"
            + "|(acoso|hostigamiento) sexual"
            + "|fue (presuntamente )?(violada|violado)"
            + "|violencia sexual"
            + "|agresión sexual|agresiones sexuales"
            + "|haber (sido )?(violada|violado) a"
            + "|pederastia"
            + "|ataque sexual|ataque fisico-sexual"
            + "|violación (equiparada|sexual)"
            + "|(hacer|realizar) tocamientos (indebidos |lascivos )?a"
            + "|fue (abusado|agredido|agredida) sexualmente"
            + "|vejación (a|de)"
            + ")( (.)*)*";

    public static final String LOCATION = "((.)* )*("
            + "en"
            + ")( (.)*)*”;

}
```

11.8. PatternIdentifier.java

```
public class PatternIdentifier {
    private final String textualPattern;

    public PatternIdentifier(String textualPattern) {
        this.textualPattern = textualPattern;
    }

    public boolean identifyPattern(String input) {
        String regex;
        Pattern pattern;
        Matcher matcher;

        regex = textualPattern;
        pattern = Pattern.compile(regex);
        matcher = pattern.matcher(input);
        return matcher.matches();
    }
}
```

11.9. Código del método extractLocation

```
private void extractLocation(TweetDTO tweet) {
        String location = "";
        String aux;
        String text = tweet.getTexto();
        StringTokenizer tokenizer = new StringTokenizer(text, " ");
        while (tokenizer.hasMoreTokens()) {
            if (tokenizer.nextToken().equals("en")) {
                while (tokenizer.hasMoreTokens()) {
                    aux = tokenizer.nextToken();
                    if (Character.isUpperCase(aux.charAt(0)) ||
                        Character.isUpperCase(aux.charAt(1))) {
                        location += aux + " ";
                        for (int i = 0; i < 3; i++) {
                            if (tokenizer.hasMoreTokens()) {
                               location += tokenizer.nextToken() + " ";
                            }
                        }
                        break;
                    }
                }
                break;
            }
        }
        tweet.setUbicacion(location);
}
```

11.10. Código del metodo geocoding

```
private boolean geocoding(TweetDTO tweet) {
        String direccion = tweet.getUbicacion();
        String entity;
        while (true) {
            Geocodic geocodic =
               Geocodificator.getInstance().geocoding(direccion);
            entity = geocodic.getEntity();
            if (entity != null) {
                for (String CDMX1 : CDMX) {
                    if (entity.equals(CDMX1)) {
                             tweet.setLatitud(Double.valueOf(geocodic.getLat()));
                             tweet.setLongitud(Double.valueOf(geocodic.getLng()));
                             tweet.setUbicacion(geocodic.getFormatted_address());
                             return true;
                    }
                }
                for (String EDOMEX1 : EDOMEX) {
                    if (entity.equals(EDOMEX1)) {
                             tweet.setLatitud(Double.valueOf(geocodic.getLat()));
                             tweet.setLongitud(Double.valueOf(geocodic.getLng()));
                             tweet.setUbicacion(geocodic.getFormatted_address());
                             return true;
                    }
                }
                return false;
            } else {
                int index = direccion.lastIndexOf(" ");
                if (index != -1) {
                    direccion =
                      direccion.substring(0, direccion.lastIndexOf(" "));
                } else {
                    return false;
                }
            }
        }
}
```

11.11. Geocodic.java

```
public class Geocodic {

    private JSONObject json;

    public JSONObject getJson() {
        return json;
    }

    public Geocodic(String jsonString) {
        try {
            json = new JSONObject(jsonString);
        } catch (JSONException ex) {
            System.err.println("Error al parsear String... " +
           ex.getMessage());
        }
    }

    public String getEntity() {
        try {
            JSONArray jsonArray = json.getJSONArray(“results”).getJSONObject(0)
                                      .getJSONArray("address_components");
            for (int i = 0; i < jsonArray.length(); i++) {
                String types = jsonArray.getJSONObject(i).getString("types");
                if(types.equals("[\"administrative_area_level_1\",\"political\"]")) {
                    return jsonArray.getJSONObject(i).getString("short_name");
                }
            }
        } catch (JSONException ex) {
            System.err.println("Error al recuperar entidad federativa... " +
              ex.getMessage());
            return null;
        }
        return null;
    }

    public String getFormatted_address() {
        try {
            return json.getJSONArray(“results").getJSONObject(0)
                       .getString("formatted_address");
        } catch (JSONException ex) {
            System.err.println("Error al recuperar dirección con formato... " +
                       ex.getMessage());
            return null;
        }
    }

    public String getLat() {
        try {
            return json.getJSONArray("results")
                    .getJSONObject(0).getJSONObject("geometry")
                    .getJSONObject("location").getString("lat");
        } catch (JSONException ex) {
            System.err.println("Error al recuperar latitud... " + ex.getMessage());
            return null;
        }
    }
```

```
    public String getLng() {
        try {
            return json.getJSONArray("results")
                    .getJSONObject(0)
                    .getJSONObject("geometry")
                    .getJSONObject("location")
                    .getString("lng");
        } catch (JSONException ex) {
            System.err.println("Error al recuperar longitud... " + ex.getMessage());
            return null;
        }
    }

}
```

11.12. Geocodificator.java

```
public class Geocodificator {

    private static Geocodificator instance = null;
    private static final String URL =
       "https://maps.googleapis.com/maps/api/geocode/json?";
    private static final String KEY = "AIzaSyC-H_Y1FdK5kaDU1-NBwcP-nBYL9PVGISI";

    private Geocodificator() {}

    public static Geocodificator getInstance() {
        if (instance == null) {
            instance = new Geocodificator();
        }
        return instance;
    }

    public Geocodic geocoding(String location) {
        String request = URL + "address=" + location.replace(" ", "+") +
              "&components=country:MX" + "&key=" + KEY;
        String line;
        String jsonString = "";
        try {
            URL url = new URL(request);
            URLConnection connection = url.openConnection();
            BufferedReader in = new BufferedReader(
              new InputStreamReader(connection.getInputStream()));
            while ((line = in.readLine()) != null) {
                jsonString += line;
            }
            return new Geocodic(jsonString);
        } catch (MalformedURLException ex) {
            System.err.println("Error en URL de Google Maps Geocoding API... " +
ex.getMessage());
            return null;
        } catch (IOException ex) {
            System.err.println("Error de IO en Google Maps Geocoding API... " +
ex.getMessage());
            return null;
        }
    }
}
```

11.13. Código del metodo saveInDB

```
private void saveInDB(TweetDTO tweet) {
        TweetDAO tweetDAO = new TweetDAO();
        tweetDAO.create(tweet);
}
```

11.14. TweetDAO.java

```
public class TweetDAO {

    public boolean create(TweetDTO tweet) {
        TweetDTO tweetDTO = tweet;
        String sql = "INSERT INTO tweet (idTweet, usuario, texto, latitud, longitud,
               delito_tipo) VALUES (?, ?, ?, ?, ?, ?)";

        ConnectionMySQL cmsql = ConnectionMySQL.getInstance();
        Connection connection = cmsql.connect();
        if (connection != null) {
            try {
                PreparedStatement preparedStatement = connection.prepareStatement(sql);
                preparedStatement.setString(1, tweetDTO.getIdTweet());
                preparedStatement.setString(2, tweetDTO.getUsuario());
                preparedStatement.setString(3, tweetDTO.getTexto());
                preparedStatement.setDouble(4, tweetDTO.getLatitud());
                preparedStatement.setDouble(5, tweetDTO.getLongitud());
                preparedStatement.setString(6, tweetDTO.getDelito());
                preparedStatement.executeUpdate();
                cmsql.disconnect();
                return true;
            } catch (SQLException ex) {
                System.err.println("Error al insertar TweetDTO en la base de datos... "
                      + ex.getMessage());
                cmsql.disconnect();
                return false;
            }
        }
        return false;
    }

    public ArrayList read() {
        String sql = "SELECT * FROM tweet ORDER BY fecha DESC";
        ArrayList array = new ArrayList();

        ConnectionMySQL cmsql = ConnectionMySQL.getInstance();
        Connection connection = cmsql.connect();
        if (connection != null) {
            try {
                PreparedStatement preparedStatement = connection.prepareStatement(sql);
                ResultSet resultSet = preparedStatement.executeQuery();
                while (resultSet.next()) {
                    TweetDTO tweetDTO = new TweetDTO(
                            resultSet.getString("idTweet"),
                            resultSet.getString("fecha"),
                            resultSet.getString("usuario"),
                            resultSet.getString("texto"),
                            resultSet.getDouble("latitud"),
```

```
                        resultSet.getDouble("longitud"),
                        resultSet.getString("delito_tipo")
                );
                array.add(tweetDTO);
            }
            cmsql.disconnect();
            return array;
        } catch (SQLException ex) {
            System.err.println("Error al recuperar los TweetDTO's de la base de
                  datos... " + ex.getMessage());
            cmsql.disconnect();
            return null;
        }
    }
    return null;
}

public ArrayList stats() {
    String sql = "SELECT delito_tipo, COUNT(delito_tipo) AS stats FROM tweet GROUP
           BY delito_tipo";
    ArrayList array = new ArrayList();

    ConnectionMySQL cmsql = ConnectionMySQL.getInstance();
    Connection connection = cmsql.connect();
    if (connection != null) {
        try {
            PreparedStatement preparedStatement = connection.prepareStatement(sql);
            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                array.add(resultSet.getString("delito_tipo"));
                array.add(resultSet.getInt("stats"));
            }
            cmsql.disconnect();
            return array;
        } catch (SQLException ex) {
            System.err.println("Error al recuperar estadísticas de la base de
                  datos... " + ex.getMessage());
            cmsql.disconnect();
            return null;
        }
    }
    return null;
}

public int cout() {
    String sql = "SELECT COUNT(idTweet) AS count FROM tweet";
    int count = -1;

    ConnectionMySQL cmsql = ConnectionMySQL.getInstance();
    Connection connection = cmsql.connect();
    if (connection != null) {
        try {
            PreparedStatement preparedStatement = connection.prepareStatement(sql);
            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                count = resultSet.getInt("count");
            }
            cmsql.disconnect();
            return count;
```

```
            } catch (SQLException ex) {
                System.err.println("Error al recuperar numero de tweets de la base de
                      datos... " + ex.getMessage());
                cmsql.disconnect();
                return -1;
            }
        }
        return -1;
    }

}
```

11.15. TweetDTO.java

```
public class TweetDTO {

    private String idTweet;
    private String fecha;
    private String usuario;
    private String texto;
    private double latitud;
    private double longitud;
    private String ubicacion;
    private String delito;

    public TweetDTO() {}

    public TweetDTO(String idTweet, String fecha, String usuario, String texto, double
       latitud, double longitud, String delito) {
        this.idTweet = idTweet;
        this.fecha = fecha;
        this.usuario = usuario;
        this.texto = texto;
        this.latitud = latitud;
        this.longitud = longitud;
        this.delito = delito;
    }

    public String getIdTweet() {
        return idTweet;
    }

    public void setIdTweet(String idTweet) {
        this.idTweet = idTweet;
    }

    public String getFecha() {
        return fecha;
    }

    public void setFecha(String fecha) {
        this.fecha = fecha;
    }

    public String getUsuario() {
        return usuario;
    }
```

```
    public void setUsuario(String usuario) {
        this.usuario = usuario;
    }

    public String getTexto() {
        return texto;
    }

    public void setTexto(String texto) {
        this.texto = texto;
    }

    public double getLatitud() {
        return latitud;
    }

    public void setLatitud(double latitud) {
        this.latitud = latitud;
    }

    public double getLongitud() {
        return longitud;
    }

    public void setLongitud(double longitud) {
        this.longitud = longitud;
    }

    public String getUbicacion() {
        return ubicacion;
    }

    public void setUbicacion(String ubicacion) {
        this.ubicacion = ubicacion;
    }

    public String getDelito() {
        return delito;
    }

    public void setDelito(String delito) {
        this.delito = delito;
    }

    @Override
    public String toString() {
        return idTweet + "  " + fecha + "  " + usuario + "  " + texto + "  " + delito;
    }

}
```

11.16. ConnectionMySQL.java

```
public class ConnectionMySQL {

    private static ConnectionMySQL connectionMySQL;
    private Connection connection;

    private static final String SERVER = "jdbc:mysql://localhost:3406/";
    private static final String DB = "seguridadcdmx";
    private static final String USER = "root";
    private static final String PASSWORD = "123";
    private static final String URL = SERVER + DB;

    private ConnectionMySQL() {}

    public static ConnectionMySQL getInstance() {
        if (connectionMySQL == null) {
            connectionMySQL = new ConnectionMySQL();
        }
        return connectionMySQL;
    }

    public Connection connect() {
        try {
            Class.forName("com.mysql.jdbc.Connection");
            Class.forName("com.mysql.jdbc.Driver");
            connection = (Connection) DriverManager.getConnection(URL, USER, PASSWORD);
            return connection;
        } catch (ClassNotFoundException ex) {
            System.err.println("No se encontro la clase... " + ex.getMessage());
            return null;
        } catch (SQLException ex) {
            System.err.println("No se logro la conexión a la base de datos... " +
ex.getMessage());
            return null;
        }
    }

    public void disconnect() {
        try {
            connection.close();
        } catch (SQLException ex) {
            System.err.println("Error al desconectar... " + ex.getMessage());
        }
    }

}
```

11.17. Diagrama de clases del módulo Extractor