Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN SISTEMAS DE INFORMACIÓN

Trabajo Fin de Grado

Investigación de Extracción de Datos en Redes Sociales

ESCUELA POLITÉCNICA SUPERIOR

Autor: Laura Lorenzo López

Tutor: Manuel Sánchez Rubio

2018

GRADO EN SISTEMAS DE INFORMACIÓN

Trabajo Fin de Grado

Investigación de Extracción de Datos en Redes Sociales

Autor: Laura Lorenzo López

Tutor: Manuel Sánchez Rubio

Tribunal

Presidente:

Vocal 1º:

Vocal 2º:

Calificación: _____

Alcalá de Henares a, de del 2018

Tabla de contenido

1 SUMARIO 4

1.1 SUMARIO 4
1.2 SUMMARY 4
1.3 PALABRAS CLAVE: 4

2 MEMORIA 6

2.1 INTRODUCCIÓN 6
2.2 BASE TEÓRICA. 7
2.2.1 TWITTER 7
2.2.2 API 8
2.2.3 REST APIS 9
2.2.4 STREAMING APIS 9
2.2.5 FIREHOSE API 10
2.3 CONFIGURACIÓN DEL ENTORNO DE DESARROLLO 10
2.3.1 CUENTA *TWITTER DEVELOPERS* 10
2.3.2 CREDENCIALES DE APLICACIÓN 13
2.3.3 LIBRERÍA TWITTER4J 18
2.3.4 SOFTWARE NETBEANS 19
2.3.5 SOFTWARE DE JAVA 20
2.4 CÓDIGO FUENTE 21
2.5 DESARROLLO 45
2.5.1 CREACIÓN DEL PROYECTO 45
2.5.2 FUNCIONALIDADES 47
2.6 CONCLUSIONES 55
2.7 TRABAJO A FUTURO 56

3 MANUAL DE USUARIO 58

3.1 VISTA 58

4 BIBLIOGRAFÍA 65

4.1 BIBLIOGRAFÍA 65
- TWITTER4J 65
- TWITTER DEVELOPERS 65
- NETBEANS 65
- TWITTER APIS 65
- TWITTER WIKIPEDIA 65
- PUBLICAR TWEETS 65
- TWITTER4J EJEMPLOS 65

5 ÍNDICE DE FIGURAS 66

5.1 ILUSTRACIONES 66

Capítulo 1

1 Sumario

1.1 Sumario

El proyecto consiste en una aplicación de *Twitter* para realizar distintas funciones. Se ha desarrollado en *Java* usando la librería de código libre *Twitter4j*. Al iniciarse la aplicación vemos la pantalla principal que nos muestra el usuario y el icono de *Twitter* de la cuenta con la que se ha creado la aplicación, y podremos publicar *Tweets*, ver el *Timeline* y realizar búsquedas tanto por palabras, etiquetas o usuarios. Los datos de las búsquedas se mostrarán en la tabla inferior con los distintos datos del *Tweet* y Usuario al que pertenecen.

1.2 Summary

This project consists on a Twitter application to do different tasks with your account. It has been developed on Java using the free code library Twitter4j. When the application is executed, we see the main view that shows us the user and icon of the Twitter account we created the application with, here we can post Tweets on our account, see our Timeline and do searches by words, hash tags or users. The searches data will be displayed on the below table showing the different data from the Tweets and Users whom they belong.

1.3 Palabras Clave:

Twitter – Tweet – Timeline – Búsqueda - Hashtag

Capítulo 2

2 Memoria

2.1 Introducción

En esta memoria primero se pondrá en contexto al usuario sobre el tema que se tratará y explicar los conceptos y términos necesarios a la hora de ver como se ha desarrollado la aplicación. También se describirá qué es *Twitter*, su funcionamiento y un poco de su historia. Además de ver cómo se realiza la conexión entre nuestra aplicación y la cuenta de *Twitter* usada.

Se expondrán los pasos a realizar para obtener una cuenta de Twitter de Desarrollador con una cuenta existente normal, para poder crear una aplicación en su web, y así obtener las cuatro claves necesarias, llamadas de *consumer* y *token*, para realizar la autenticación de la cuenta desde nuestra aplicación Java.

Para usar la librería de código libre *Twitter4j* con la que vamos a programar nuestra aplicación, será necesario descargarla e incluirla en nuestro proyecto Java. Además, se necesita el *IDE NetBeans*, que es con el que en este caso hemos desarrollado la aplicación. También, si no se dispone ya de ello, es necesario la descarga e instalación del software de *Java* para poder ejecutar la aplicación.

Tras ello, expondremos el código fuente de la aplicación y se irán explicando paso a paso el funcionamiento y las funciones que realiza cada parte. También se explicará la interfaz y la utilidad de cada zona de la vista.

A continuación, se llevarán a cabo unas conclusiones sobre el proyecto y se reflexionara sobre los posibles ámbitos de mejoras que se podrían llegar a desarrollar en el futuro.

Para finalizar, se desarrollará un manual de usuario de la aplicación, explicando paso a paso como ejecutar la aplicación, su uso y sus distintas funcionalidades, así cómo se podrán ver ejemplos de cómo luce la vista de la aplicación en funcionamiento.

2.2 Base Teórica.

En esta sección vamos a ver en que consiste *Twitter* y como se realizará la conexión entre este servicio y nuestra aplicación *Java*.

Explicaremos la diferencia entre las dos formas de conexión con *Twitter* que son *REST APIs* y *Streaming APIs*, y veremos las distintas funciones que se pueden realizar con ellas.

2.2.1 Twitter

Twitter es una red social basada en microblogging, consiste en publicar mensajes cortos de un máximo de 280 caracteres (140 anteriormente) llamados *Tweets*, al que se le pueden añadir hasta cuatro imágenes, o un vídeo, o links, o, recientemente, encuestas de hasta cuatro opciones de respuesta.

Cualquier persona se puede crear una cuenta para publicar estos *Tweets*, y seguir a otras cuentas y ser seguido. Las cuentas pueden ser publicas para que cualquiera pueda ver nuestro contenido, o privados, donde solo tendrán acceso nuestros seguidores al contenido, y para que alguien nos siga debemos aceptarlo. Las interacciones entre cuentas, a parte de poder responder a *Tweets* de otros personar o mencionarlos en los nuestros, se produce principalmente por medio de los *Likes* (Me gusta) a un *Tweet,* o *Retweet*, que consiste en publicar en tu perfil ese mismo Tweet.

Twitter fue creado en Estados Unidos en el año 2006, y actualmente cuenta con más de 335 millones de usuarios activos en todo el mundo. Debido a la falta de crecimiento e incluso pérdidas que ha ido sufriendo el servicio a lo largo de sus años en activo, están diversificando y ampliando sus funciones para volver a atraer a más usuarios, como con la compra del servicio *Periscope* y su posterior integración dentro de *Twitter*, que permite a cualquier usuario retransmitir videos en directo fácilmente.

El servicio se desarrolló inicialmente con el *framework* de *Ruby*, *Ruby on Rails*, y almacenaba la información en bases de datos *MySQL*, pero esta infraestructura fue siendo cada vez más lenta y pesada con el gran crecimiento que iba teniendo *Twitter*. Por lo que se fue reemplazando gradualmente a servidores *Java* y software desarrollado en *Scala*, lo que aumento exponencialmente la capacidad del servicio.

La API de *Twitter* es abierta por lo que cualquier desarrollador podría usarla para integrar el servicio en sus propias aplicaciones, como vamos a hacer en este proyecto. Para ello, es necesario que las aplicaciones de terceros se identifiquen por medio del protocolo *OAuth*, se trata de un protocolo abierto que permite la autenticación de una forma segura y sencilla, y con un método estándar desde web, móvil o aplicaciones.

Twitter proporciona dos tipos de autenticación con *OAuth*:

1. Autenticación de usuario (contexto de usuario): permite a la aplicación actuar en nombre del usuario, como el usuario, por ejemplo, para publicar *Tweets*. Esta forma requiere las cuatro claves de la aplicación *Twitter* que crearemos en *Twitter Developer*, las dos de *consumer* y las de *token*. Ésta será la forma que usaremos en este proyecto.
2. Autenticación de solo aplicación: esta forma realiza peticiones de API por sí misma, sin el contexto de usuario. Este método esta pensado para quien solo necesita acceder a información pública del servicio, como ver tweets o listas públicos. Para usar este método se necesita un *bearer token* (token de portador), se consigue con las dos claves de *consumer*.

2.2.2 API

API, *Application Program Interface*, quiere decir interfaz de programación de aplicación, y consiste en una serie de rutinas, protocolos y herramientas para construir aplicaciones de software. Básicamente, una API especifica como deben interactuar los componentes del software.

En este proyecto vamos a usar la librería *Twitter4J* que es una librería no oficial en *Java* para la API de *Twitter*. Esta librería te permite integrar fácilmente *Twitter* en tu

aplicación *Java*, además de contar con múltiples métodos y clases específicas para realizar las distintas funciones que ofrece *Twitter*, como publicar *Tweets*, ver el *Timeline* o realizar búsquedas.

2.2.3 REST APIs

La REST API, por sus siglas en inglés *Representational State Transfer* (Transferencia de Estado Representacional), o también conocida como *Search* API (API de Búsqueda) permite a los desarrolladores acceder a la información y los recursos usando una simple invocación http, es decir, a leer y escribir información dentro de *Twitter*. Algunos ejemplos de su uso serían:

- Publicar *Tweets*
- Mostrar nuestro *Timeline*
- Ver el perfil de un usuario

Para poder usar la REST API deberemos autenticarnos en nuestra aplicación usando *OAuth* de forma segura, como indicamos antes. Las respuestas de estas peticiones son devueltas en formato JSON.

2.2.4 Streaming APIs

La *Streaming* API de *Twitter* permite acceder al flujo global de datos del servicio. El usuario podrá ir recibiendo los Tweets que se van publicando en tiempo real, al contrario que con la REST API que te devuelve Tweets que ya han ocurrido. Existen tres tipos distintos de *streaming*:

1. Streams Públicos: son flujos de datos públicos de *Twitter*, se usa para seguir el contenido de usuarios, algún tema o hashtag específico, y también para la minería de datos.
2. Streams de usuarios: se trata de flujos de datos específicos de un solo usuario de *Twitter*.

3. Streams de localización: este tipo de flujo de datos se caracteriza por contener el flujo en especifico de un lugar dado. Suele ser usado a través de coordenadas.

2.2.5 Firehose API

La *Firehose* API de *Twitter* es la API más completa y que da acceso al 100% de los flujos de datos que coinciden con tu criterio de búsqueda, ya que las dos anteriores no llegan a proporcionarlos en su totalidad. Es muy similar a la *Streaming* API, proporcionando los datos en tiempo real. Está API es manejada por dos proveedores de datos GNIP y DataSift, y es además de pago.

2.3 Configuración del entorno de desarrollo

2.3.1 Cuenta *Twitter Developers*

Para poder crear nuestra aplicación con *Twitter* debemos crearnos una cuenta en *Twitter Developers* en la dirección https://developer.twitter.com/en/dashboard por medio de una cuenta existente de *Twitter*. Como vemos en la ilustración, saldrá nuestra cuenta de *Twitter* y si queremos aplicar a una cuenta de desarrollador, pulsaremos en continuar.

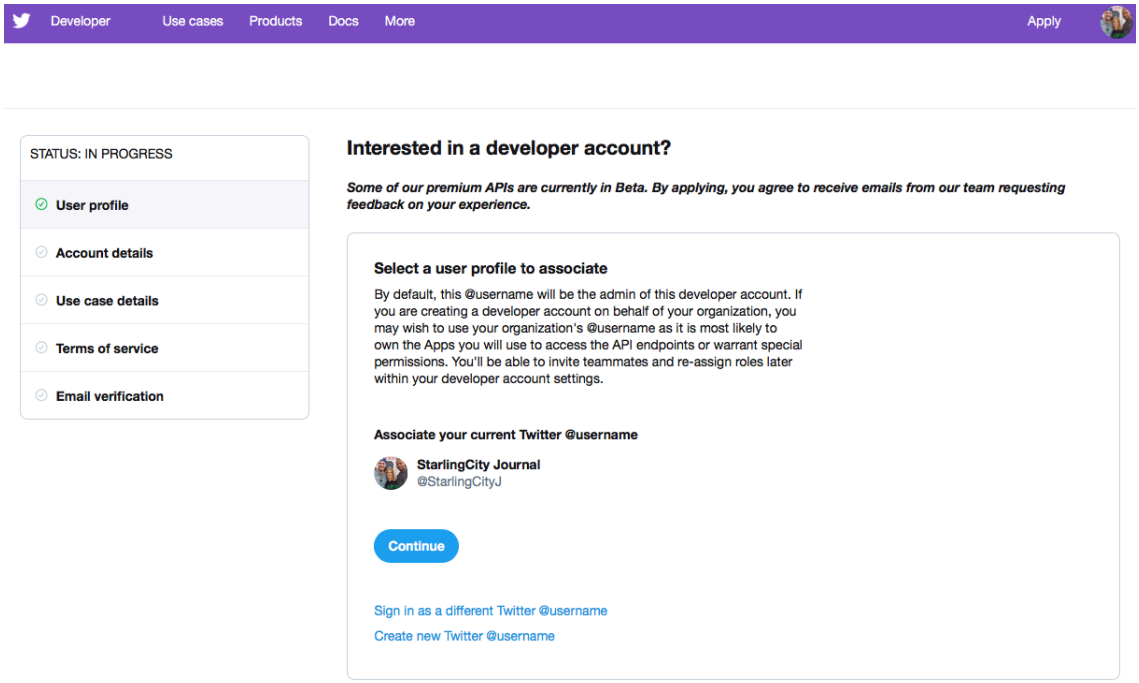

Ilustración 1: Creación de Cuenta de Desarrollador

En la siguiente pantalla deberemos elegir si se trata de una cuenta para organización o de uso personal, en este caso elegimos de uso personal y escribimos el nombre de cuenta, elegimos nuestro país y pulsamos en continuar.

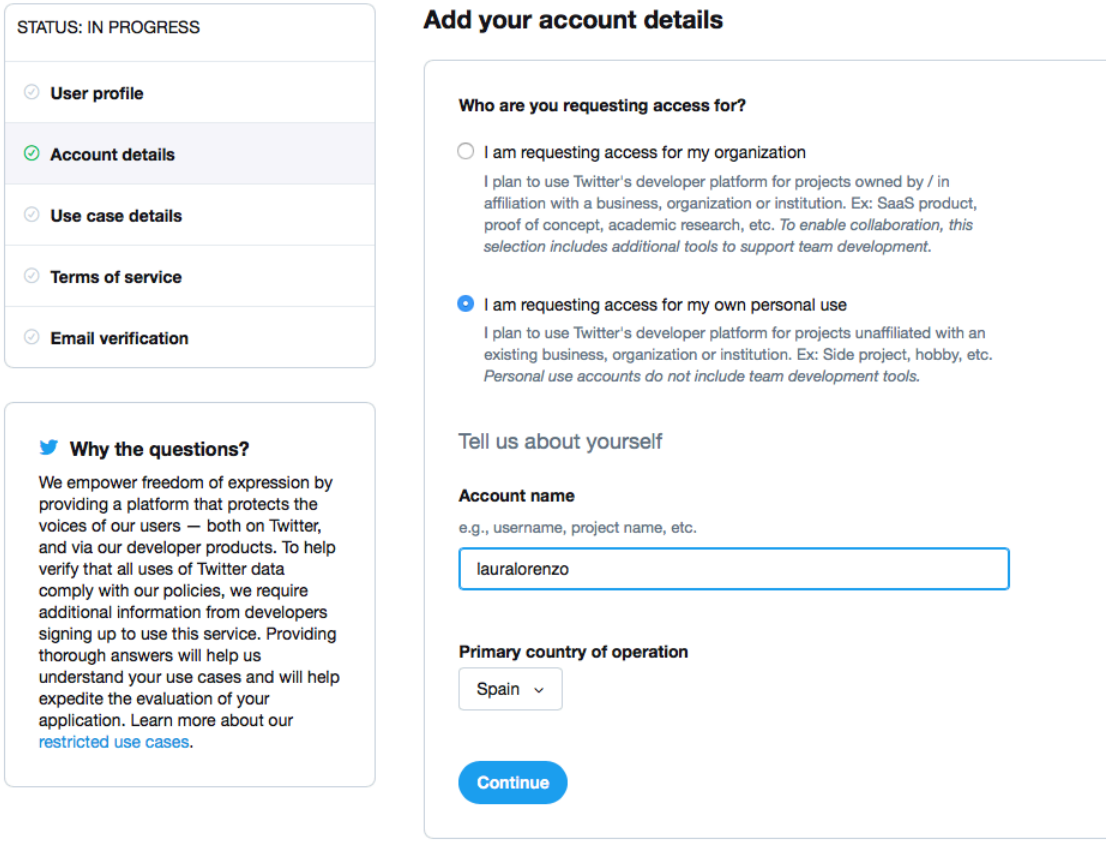

Ilustración 2: Creación de Cuenta de Desarrollador 2

A continuación, debemos elegir el uso que le daremos a la cuenta, académico en este caso, describir en un mínimo de 300 caracteres lo que vamos a construir, y contestar si nuestra aplicación proporcionará datos al gobierno, siempre será que no, ya que por motivos de seguridad si se contesta que sí, en principio no concederán la cuenta.

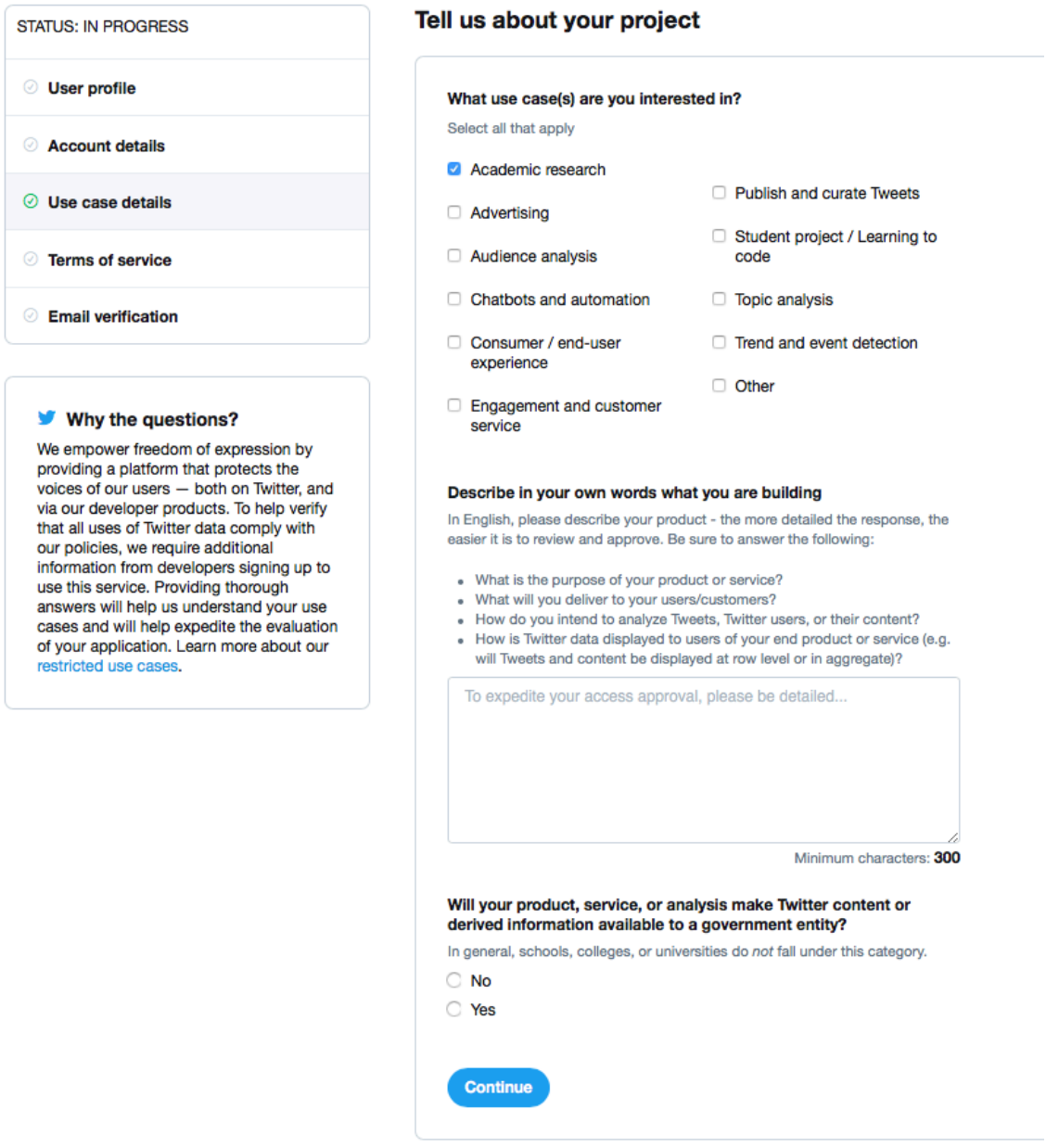

Ilustración 3: Creación de Cuenta de Desarrollador 3

Finalmente hay que leer y aceptar los términos del servicio y enviar la aplicación.

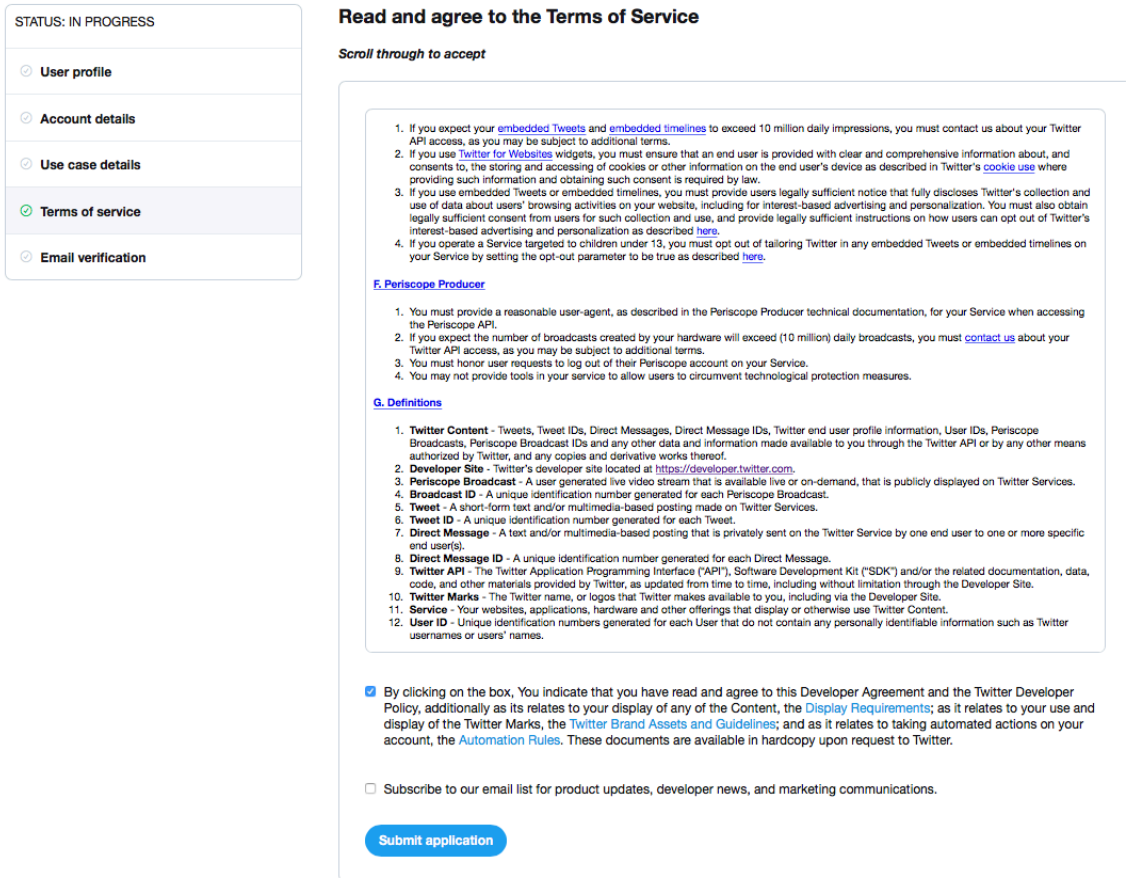

Ilustración 4: Creación de Cuenta de Desarrollador 4

Tras todo esto deberemos verificar nuestro email y, en principio, si no ha habido ningún problema, ya deberíamos tener acceso a nuestra cuenta de desarrollador en la web.

2.3.2 Credenciales de Aplicación

Dentro de la web debemos desplegar el menú de la cuenta arriba a la derecha, como vemos en la ilustración, y pulsar sobre la pestaña Apps.

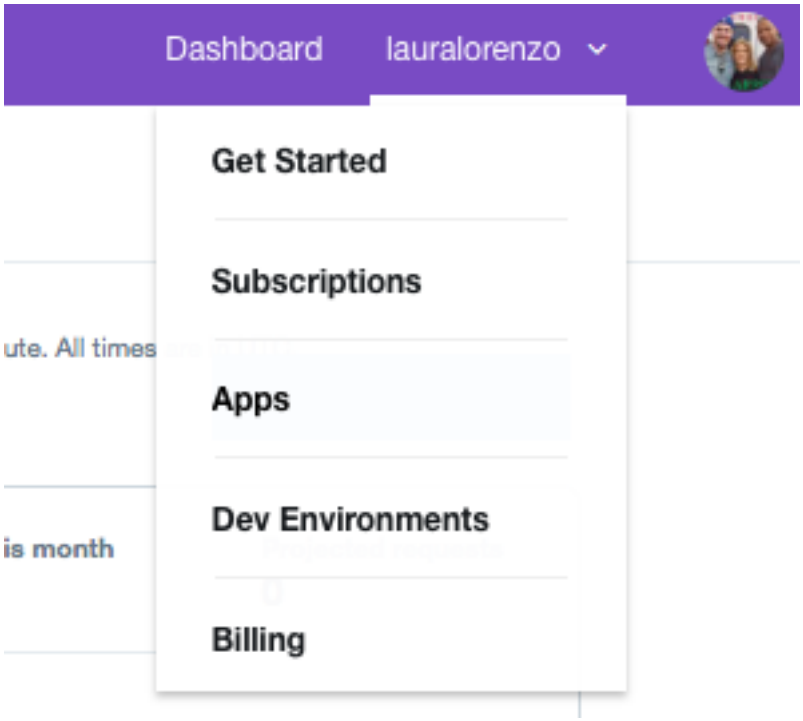

Ilustración 5: Creación de App

En la siguiente pantalla pulsaremos sobre Crear una app.

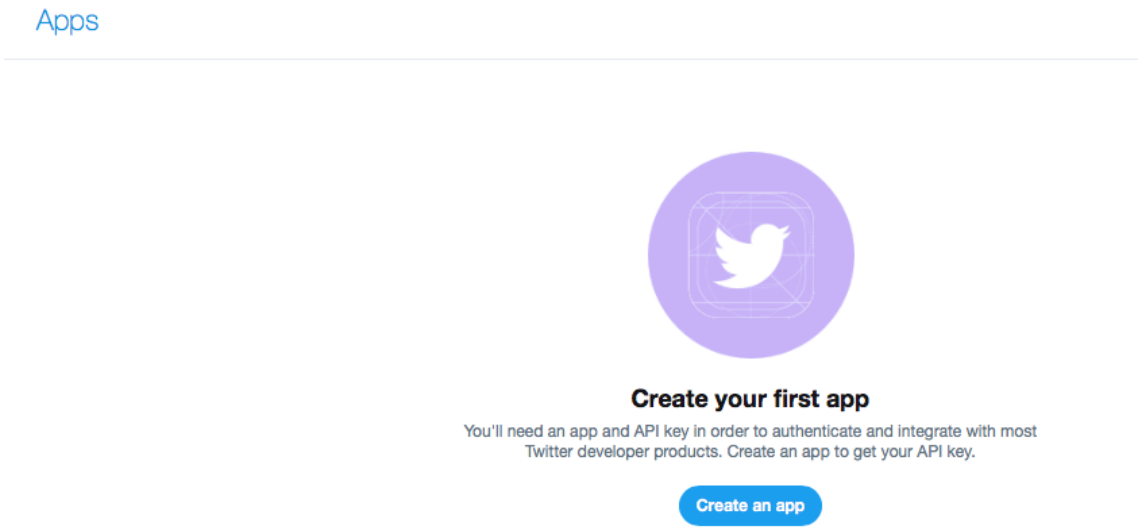

Ilustración 6: Creación de App 2

Luego debemos rellenar todos los datos que nos piden para la aplicación, como nombre, url, descripción, solo serán necesarios los que dice *required*.

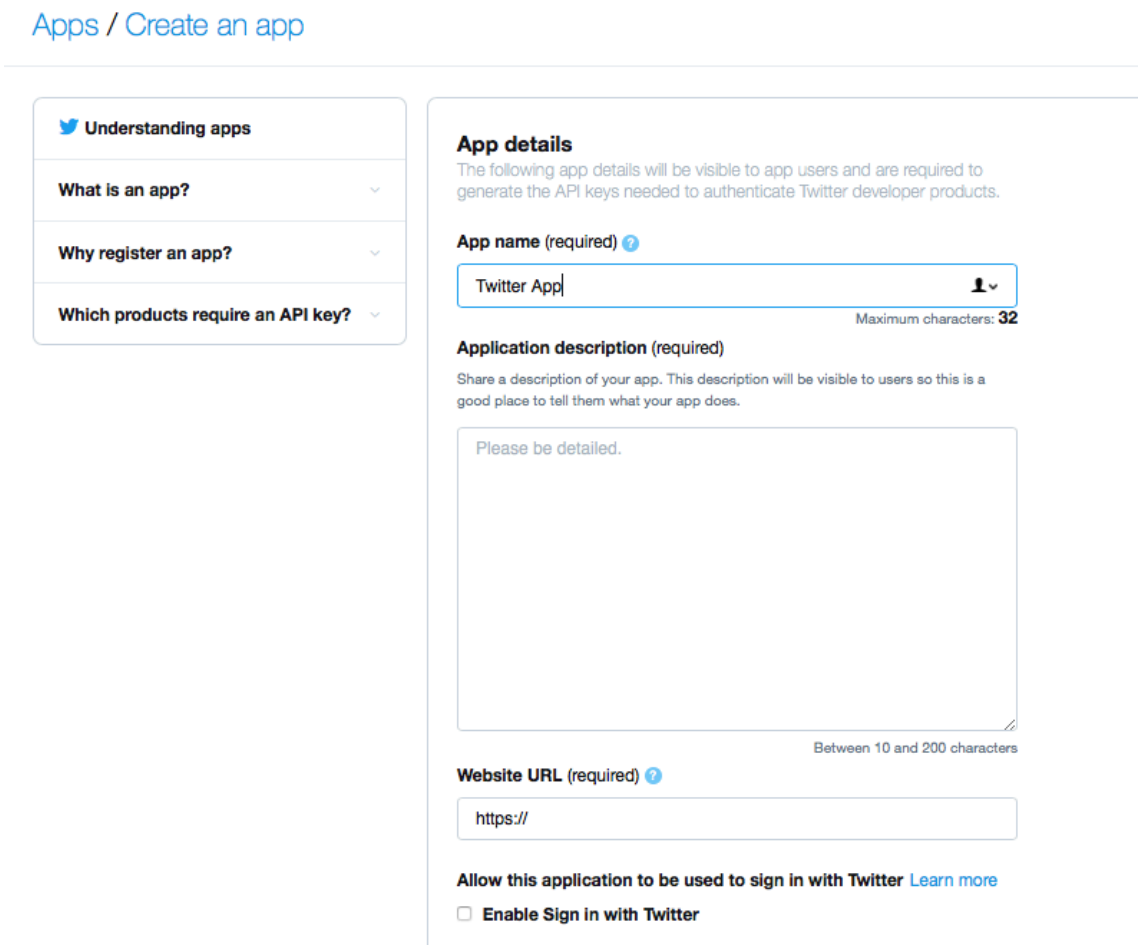

Ilustración 7: Creación de App 3

Al finalizar, ya tendremos nuestra aplicación creada, y veremos la siguiente pantalla con todos los datos sobre ella, aquí podremos editarlos si queremos.

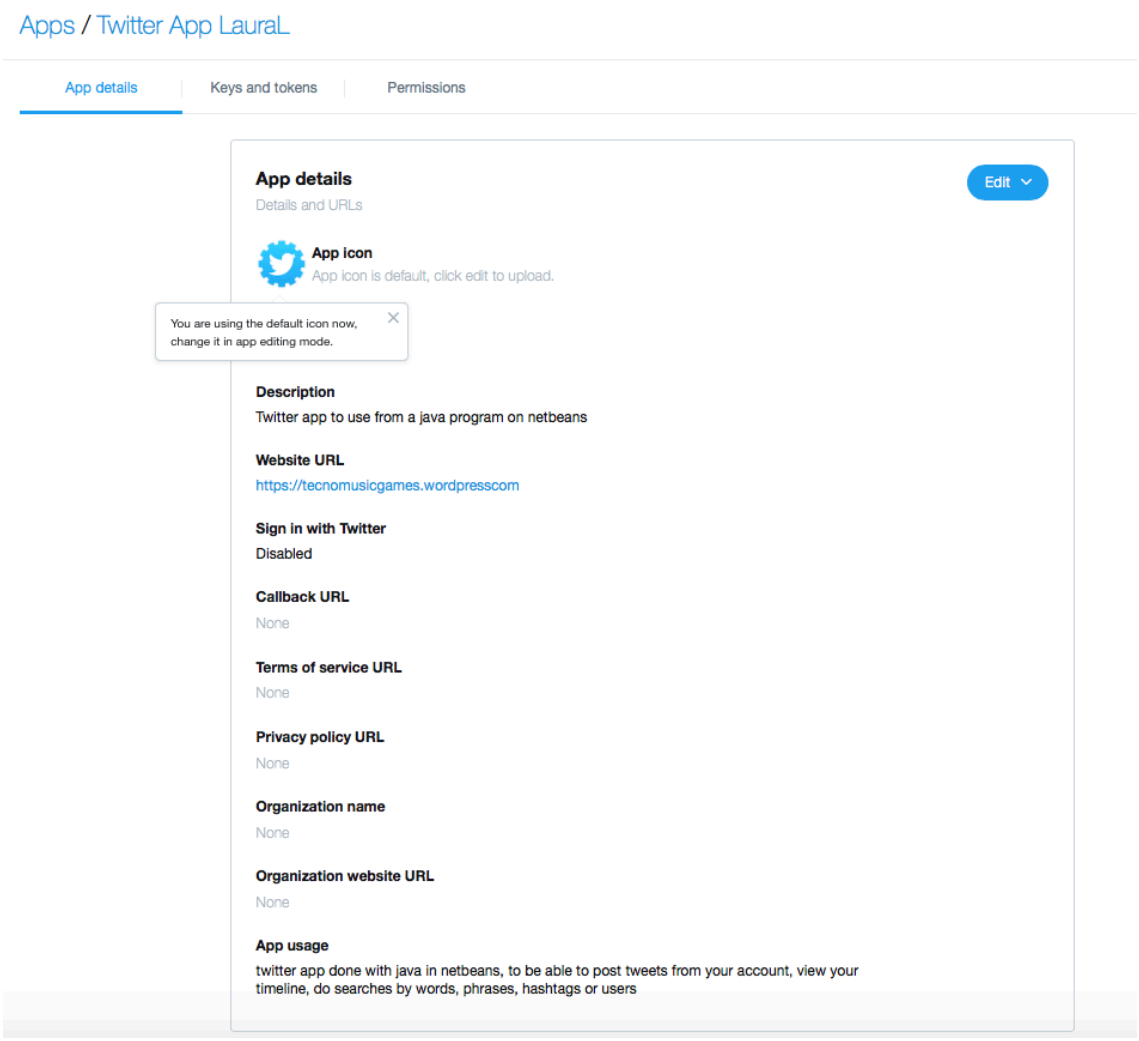

Ilustración 8: Creación de App 4

En esa misma imagen, debemos ir a la pestaña de Keys and tokens para coger nuestras claves para usar en nuestra aplicación Java.

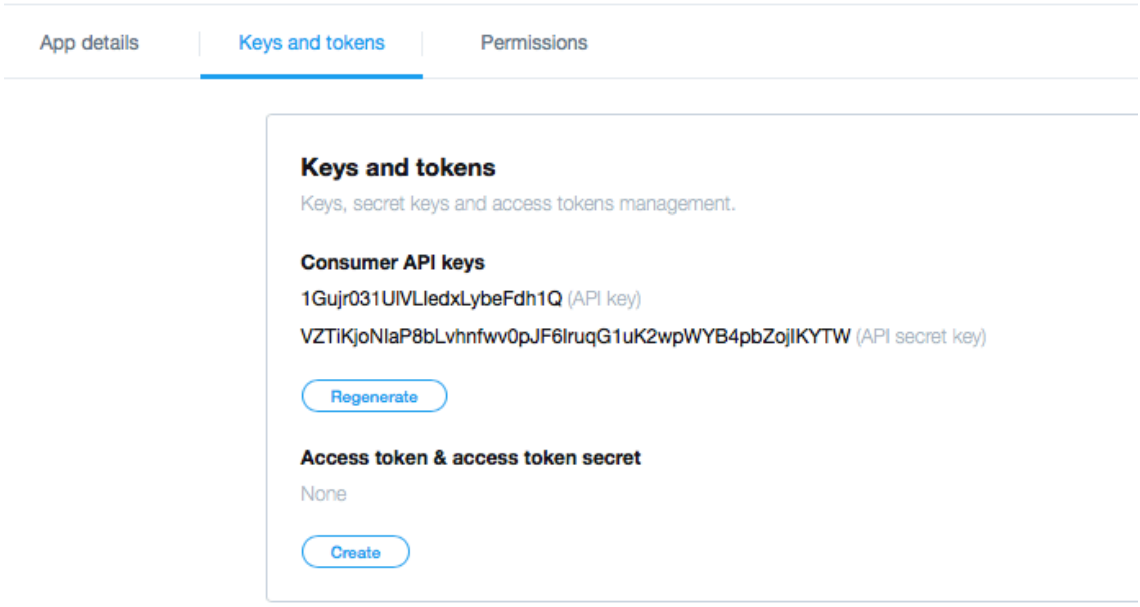

Ilustración 9: Obtención de claves 1

Por defecto solo nos crea las claves de *Consumer*, la normal y la secreta, nosotros aparte de estas dos necesitamos las de *token*, para crearlas, pulsamos en el botón de *Create* en la sección de *Access token & Access token secret*.

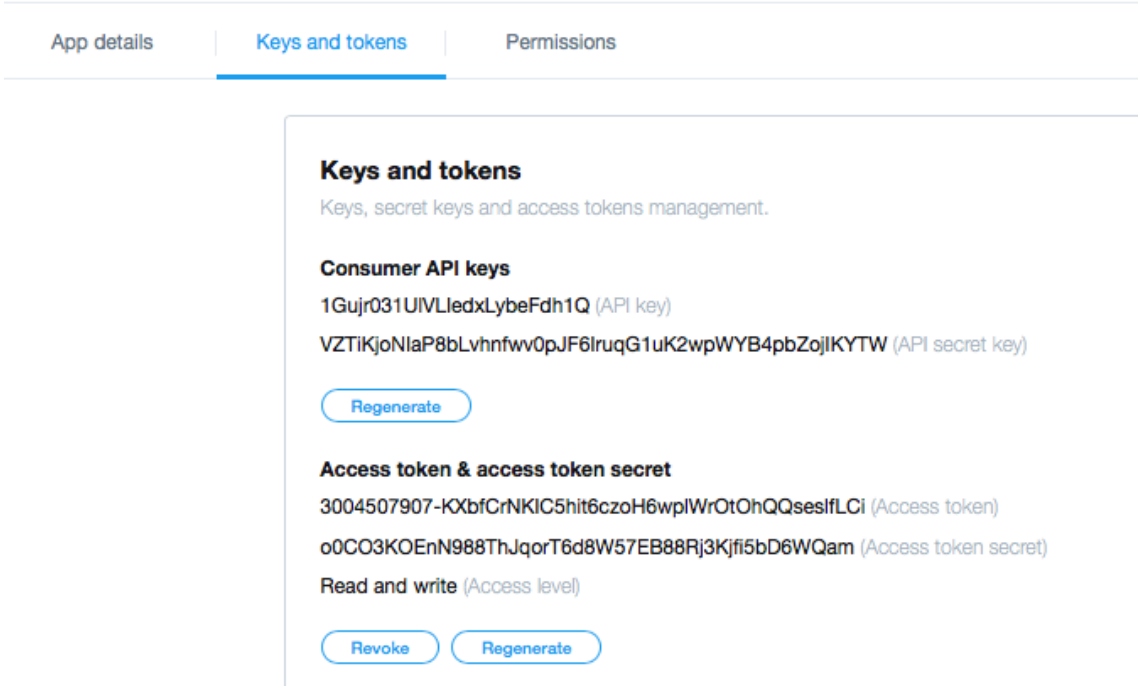

Ilustración 10: Obtención de claves 2

Ya tenemos nuestras cuatro claves, nos las guardamos bien para luego introducirlas en nuestro programa Java.

2.3.3 Librería Twitter4j

Existen muchas librerías para desarrollar aplicaciones de *Twitter* y en diversos lenguajes de programación. Nosotros como vamos a programar en *Java*, hemos elegido la librería *Twitter4J*, pues es una de las más usadas y amplias en funcionalidades.

Debemos descargar la librería e incluirla en nuestro proyecto *Java*. Para ello vamos a su página oficial http://twitter4j.org/en/index.html y descargamos el comprimido en la sección *Downloads* como vemos en la imagen.

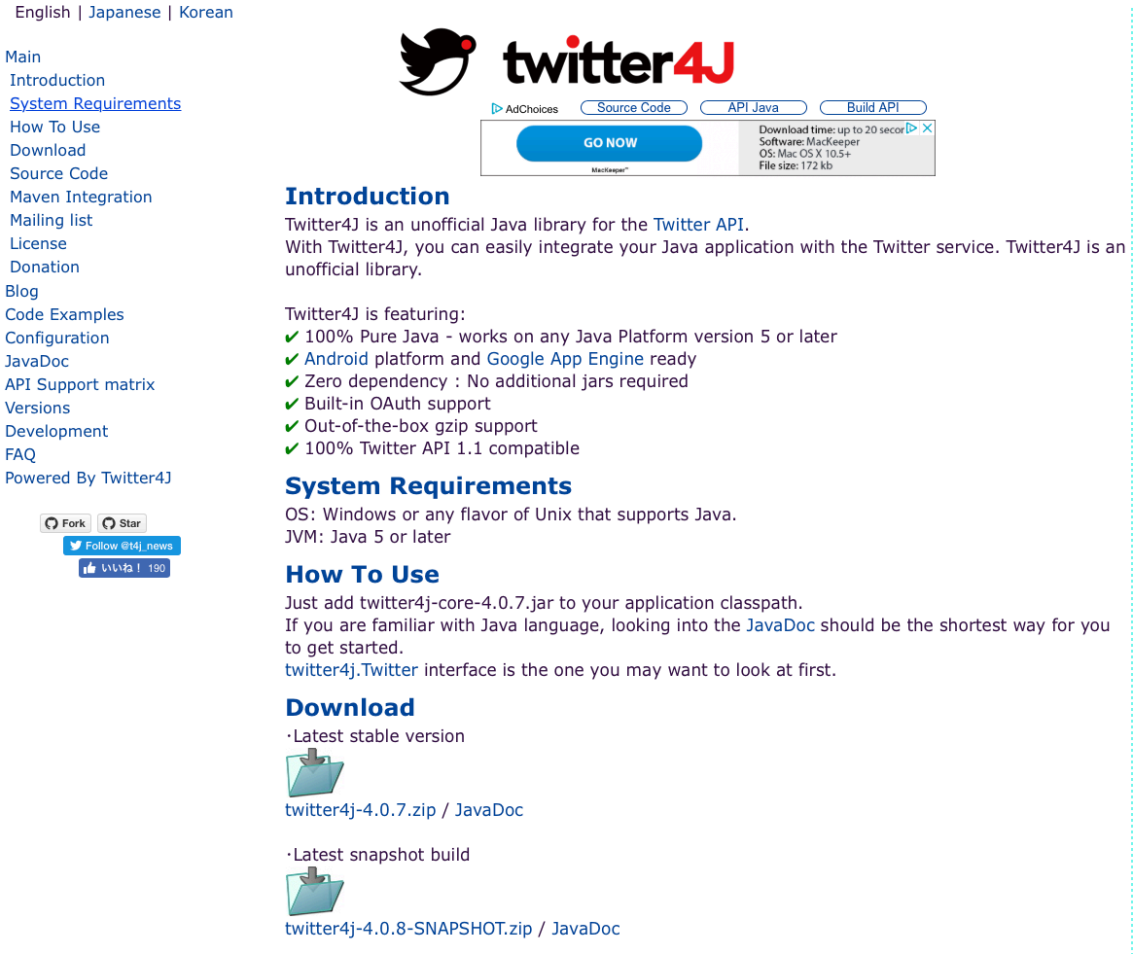

Ilustración 11: Descarga Twitter4J

Una vez descargado debemos descomprimir el archivo y guardarlo en una carpeta conocida para después poder añadirla a nuestro proyecto.

2.3.4 Software NetBeans

Hemos decidido usar el entorno de desarrollo *NetBeans* para el desarrollo de nuestra aplicación, para poder usarlo debemos descargarlo de su web oficial. Para ello en la página de inicio vamos donde pone *NetBeans* IDE 8.2 y pulsamos el botón de *Download*.

Ilustración 12: Descarga NetBeans 1

En la siguiente pantalla nos mostrará las distintas versiones disponibles para descargar. Para nuestro proyecto con la versión básica de Java SE será suficiente, pulsamos en su botón *Download*.

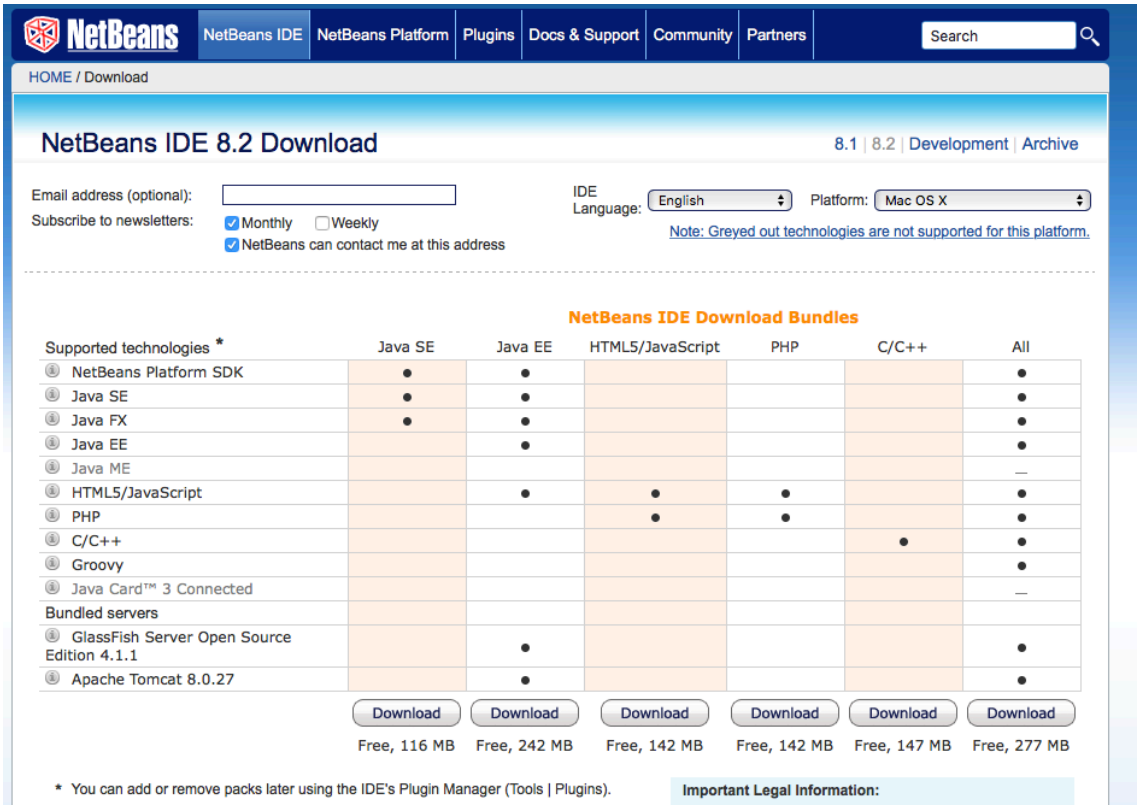

Ilustración 13: Descarga NetBeans 2

2.3.5 Software de Java

Para poder usar y ejecutar nuestra aplicación necesitaremos tener el software de *Java* instalado. Si no lo tenemos ya, iremos a la web oficial https://www.java.com/es/download/ y pulsaremos en el botón de Descarga, cuando acabe la descarga, lo instalamos y listo.

Ilustración 14: Descarga Java

2.4 Código fuente

```
package twitterApp;
//Imports needed
import java.awt.Component;
import java.awt.Window;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.net.MalformedURLException;
import java.net.URL;
import java.util.Date;
import java.util.List;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.ImageIcon;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JTable;
import javax.swing.JTextArea;
```

```
import javax.swing.table.DefaultTableCellRenderer;
import javax.swing.table.DefaultTableModel;
import twitter4j.Paging;
import twitter4j.Query;
import twitter4j.QueryResult;
import twitter4j.ResponseList;
import twitter4j.Status;
import twitter4j.TwitterException;
import twitter4j.TwitterFactory;
import twitter4j.conf.ConfigurationBuilder;
import twitter4j.Twitter;
import twitter4j.User;

/**
 *
 * @author Laura Lorenzo
 */
public class TwitterView extends javax.swing.JFrame {
    //Variable general de Twitter para aplicar sus funciones
    Twitter twitter;
    //String para almacenar el mensaje a Tweetear
    String message = null;
    //Modelo de la tabla que mostrará los datos
    DefaultTableModel model;

    /**
     *
     * @throws TwitterException
     */
    //Método para publicar un Tweet
    public void Tweet() throws TwitterException {
        Paging page = new Paging();
        Status tweet = twitter.updateStatus(message);
    }
```

```
    /**
     * Creates new form TwitterView
     */
    public TwitterView() {
        initComponents();
        //Constructor con las claves de nuestra app para conectar con Twitter
        ConfigurationBuilder cb = new ConfigurationBuilder();
        cb.setDebugEnabled(true)
                .setOAuthConsumerKey("Cw3UAKyEDRxdbMSnTfXnoTr3X")

.setOAuthConsumerSecret("rsY1Wfrs4vgdYn5nBSmTXEoZ1bB6hRdHZK7xnK8caUy
pvmzJWk")
                .setOAuthAccessToken("328013467-
Yqnk6LaLXVx51BMMd4p1SEciJPVhu8ey9TXaPO8X")

.setOAuthAccessTokenSecret("eE8tll1F7OD3ddl6uZrC9bgrYN4pSIgAhlc3hASNSTiU
Z");
        twitter = new TwitterFactory(cb.build()).getInstance();

        try {
            initUserInfo();
        } catch (MalformedURLException | TwitterException ex) {
            Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE,     null,
ex);
        }
        jTextArea1.setLineWrap(true);
        //Fijamos las medidas y el modelo de la tabla
        jTable1.setDefaultRenderer(Object.class, new ImgTable());
        jTable1.getColumnModel().getColumn(0).setPreferredWidth(10);
        jTable1.getColumnModel().getColumn(1).setPreferredWidth(40);
        jTable1.getColumnModel().getColumn(2).setPreferredWidth(50);
        jTable1.getColumnModel().getColumn(3).setPreferredWidth(40);
        jTable1.getColumnModel().getColumn(4).setPreferredWidth(400);
```

```
            jTable1.getColumnModel().getColumn(4).setResizable(true);

            model = (DefaultTableModel) jTable1.getModel();
            //Indicamos el número de filas al inicio
            model.setNumRows(0);
        }

        //Método para que dentro de la tabla se muestren los iconos en formato
        //imagen, y el Tweet en un área de texto correctamente
        public class ImgTable extends DefaultTableCellRenderer {
            public Component getTableCellRendererComponent(JTable table, Object value,
boolean isSelected, boolean hasFocus, int row, int column) {
                if (value instanceof JLabel) {
                    JLabel label = (JLabel) value;
                    return label;
                } else if (value instanceof JTextArea) {
                    JTextArea area = (JTextArea) value;
                    return area;
                } else {
                    return super.getTableCellRendererComponent(table, value, isSelected,
hasFocus, row, column);
                }
            }
        }
        /**
         * This method is called from within the constructor to initialize the form.
         * WARNING: Do NOT modify this code. The content of this method is always
         * regenerated by the Form Editor.
         */
        @SuppressWarnings("unchecked")
        // <editor-fold defaultstate="collapsed" desc="Generated Code">
        private void initComponents() {
    //Declaración de los componentes Visuales
            jScrollPane1 = new javax.swing.JScrollPane();
```

```
jTextArea1 = new javax.swing.JTextArea();
jButton1 = new javax.swing.JButton();
jButton2 = new javax.swing.JButton();
jLabel1 = new javax.swing.JLabel();
jScrollPane2 = new javax.swing.JScrollPane();
jTable1 = new javax.swing.JTable();
textField1 = new java.awt.TextField();
jLabel2 = new javax.swing.JLabel();
jLabel3 = new javax.swing.JLabel();
jLabel4 = new javax.swing.JLabel();
jButton3 = new javax.swing.JButton();
jButton4 = new javax.swing.JButton();
jButton5 = new javax.swing.JButton();
jButton6 = new javax.swing.JButton();
jButton7 = new javax.swing.JButton();
jButton8 = new javax.swing.JButton();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setBackground(new java.awt.Color(204, 255, 204));

jTextArea1.setColumns(20);
jTextArea1.setRows(5);
jScrollPane1.setViewportView(jTextArea1);

jButton1.setText("Clean");
jButton1.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton1ActionPerformed(evt);
   }
});

jButton2.setText("Tweet");
jButton2.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
```

```
        jButton2ActionPerformed(evt);
    }
});

jLabel1.setFont(new java.awt.Font("Microsoft Himalaya", 3, 48)); // NOI18N
jLabel1.setForeground(new java.awt.Color(0, 204, 51));
jLabel1.setText("Twitter App");

jTable1.setModel(new javax.swing.table.DefaultTableModel(
    new Object [][] {
        {null, null, null, null, null},
        {null, null, null, null, null},
        {null, null, null, null, null},
        {null, null, null, null, null}
    },
    new String [] {
        "Icon", "Username", "Date", "Location", "Tweet"
    }
) {
    boolean[] canEdit = new boolean [] {
        false, false, false, false, false
    };

    public boolean isCellEditable(int rowIndex, int columnIndex) {
        return canEdit [columnIndex];
    }
});
jTable1.setRowHeight(50);
jScrollPane2.setViewportView(jTable1);

textField1.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        textField1ActionPerformed(evt);
    }
```

```
});

jLabel2.setText("Search:");

jLabel3.setText("Icon");
jLabel3.setMaximumSize(new java.awt.Dimension(48, 48));
jLabel3.setMinimumSize(new java.awt.Dimension(48, 48));
jLabel3.setPreferredSize(new java.awt.Dimension(48, 48));

jLabel4.setText("jLabel4");
jButton3.setText("By Word");
jButton3.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton3ActionPerformed(evt);
   }
});
jButton4.setText("Users");
jButton4.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton4ActionPerformed(evt);
   }
});
jButton5.setText("Get Timeline");
jButton5.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton5ActionPerformed(evt);
   }
});
jButton6.setText("Clean Table");
jButton6.addActionListener(new java.awt.event.ActionListener() {
   public void actionPerformed(java.awt.event.ActionEvent evt) {
      jButton6ActionPerformed(evt);
   }
});
```

```
        jButton7.setText("By Hashtag");
        jButton7.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton7ActionPerformed(evt);
            }
        });
        jButton8.setText("Save in File");
        jButton8.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton8ActionPerformed(evt);
            }
        });
        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(22, 22, 22)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(jLabel3,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(39, 39, 39)
                        .addComponent(jLabel4)
                        .addGap(398, 398, 398)
                        .addComponent(jLabel1)
                        .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
                    .addGroup(layout.createSequentialGroup()
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILIN
G)
                    .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILIN
G)
                        .addGroup(layout.createSequentialGroup()
                            .addComponent(jButton1)
                            .addGap(84, 84, 84)
                            .addComponent(jButton2))
                        .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
                        .addGroup(layout.createSequentialGroup()
                            .addGap(63, 63, 63)
                            .addComponent(jButton5)
                            .addGap(433, 433, 433)
                            .addComponent(jLabel2)
                            .addGap(28, 28, 28)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
, false)
                                .addGroup(layout.createSequentialGroup()
                                    .addComponent(jButton3)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                    .addComponent(jButton7)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
```

```
                                .addComponent(jButton4))
                            .addComponent(textField1,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)))
                        .addGroup(layout.createSequentialGroup()
                            .addGap(190, 190, 190)
                            .addComponent(jButton6)
                            .addGap(114, 114, 114)
                            .addComponent(jButton8))))
                    .addComponent(jScrollPane2))
                .addGap(35, 35, 35))))
        );
        layout.setVerticalGroup(
            layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addGap(17, 17, 17)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
                    .addComponent(jLabel3,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel4)
                    .addComponent(jLabel1,
javax.swing.GroupLayout.PREFERRED_SIZE,                                    48,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addGap(20, 20, 20)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
                    .addGroup(layout.createSequentialGroup()
```

```
                    .addComponent(textField1,
javax.swing.GroupLayout.PREFERRED_SIZE,                                    33,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addGap(22, 22, 22)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
                        .addComponent(jButton3)
                        .addComponent(jButton4)
                        .addComponent(jButton7))
                    .addGap(9, 9, 9)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELIN
E)
                        .addComponent(jButton6)
                        .addComponent(jButton8)))
                .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
                        .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jLabel2)
                        .addComponent(jButton5))
                    .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING
)
                        .addComponent(jButton1)
                        .addComponent(jButton2))))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
```

```
                .addComponent(jScrollPane2,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(9, Short.MAX_VALUE))
        );

        pack();
    }// </editor-fold>

    //Método para limpiar el área de texto con el botón Clean
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        jTextArea1.setText("");
    }
    //Método del botón Tweet que llama al método Tweet() si todo está bien,
    //o muestra los mensajes correspondientes de error
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            if (jTextArea1.getText().length() > 280) {
                throw new Exception("The maximum characters allowed are 280");
            }
            if (jTextArea1.getText().isEmpty()) {
                throw new Exception("You must write something to tweet");
            }
            message = jTextArea1.getText();
            try {
                Tweet();
                JOptionPane.showMessageDialog(null, "Tweet was sent successfully");
                jTextArea1.setText("");
            } catch (TwitterException te) {
                Logger.getLogger(Window.class.getName()).log(Level.SEVERE, null, te);
            }

        } catch (Exception e) {
```

```
            JOptionPane.showMessageDialog(null, e);
        }
    }
    private void textField1ActionPerformed(java.awt.event.ActionEvent evt) {
    }
    //Método del botón de Búsqueda por palabras
    private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            //Si no hay nada escrito muestra el mensaje
            if (textField1.getText() == null || textField1.getText().isEmpty()) {
                try {
                    throw new Exception("You must write a term to search");
                } catch (Exception ex) {
                    Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE,
null, ex);
                }
            }
            String queryString = textField1.getText();
            Integer numberOfTweets = 100;
            //Pasa el texto de String a query
            Query query = new Query(queryString);
            query.setCount(numberOfTweets);
            QueryResult result;
            //Realiza la búsqueda de la query en Twitter
            result = twitter.search(query);

            //Guarda los datos en una lista de status
            List<Status> tweets = result.getTweets();
            //Bucle para recorrer la lista status
            for (int i = 0; i < tweets.size(); i++) {
                //Cogemos el usuario del Tweet para sacar algunos de sus datos
                User user = tweets.get(i).getUser();
                System.out.println("Date: " + tweets.get(i).getCreatedAt());
                System.out.println("User: " + tweets.get(i).getUser().getScreenName());
```

```
System.out.println("Location: " + user.getLocation());
System.out.println("ID: " + tweets.get(i).getText() + "\n");

URL urlIcon;
try {
   //Cogemos la URL del icono del usuario
   urlIcon = new URL(user.getProfileImageURL());
   //Convertimos la URL en una imagen tipo icono
   ImageIcon userIcon = new ImageIcon(urlIcon);
   //Metemos el icono en una label como su icono
   //para que se pueda visualizar dentro de la tabla
   JLabel icon = new JLabel();
   icon.setIcon(userIcon);
   icon.setText("");

   //Cogemos el nombre de usuario
   String name = tweets.get(i).getUser().getScreenName();
   //Su localización si la tiene
   String location = user.getLocation();
   //La fecha y hora de creación del tweet
   Date date = tweets.get(i).getCreatedAt();

   //Cogemos el texto del tweet y lo metemos en un área de texto
   JTextArea tweet = new JTextArea();
   tweet.setText(tweets.get(i).getText());
   //Características para que salte de línea y verlo entero
   tweet.setLineWrap(true);
   tweet.setWrapStyleWord(true);
   tweet.setOpaque(true);

   //Añadimos como fila de la tabla el icono, usuario, fecha,
   //localización y tweet
   model.addRow(new Object[]{
      icon, name, date, location, tweet
```

```
                });

            } catch (MalformedURLException ex) {
                Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE,
null, ex);
            }
          }
        } catch (TwitterException ex) {
          Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE,     null,
ex);
        }
      }

      //Botón para ver el Timeline de la cuenta
      private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
        try {
          //Paginamos los datos para ver los máximos posibles
          int totalTweets = 100;
          Paging paging = new Paging(1, totalTweets);
          //COgemos el timeline y guardamos los tweets en una lista de Status
          List<Status> timeline = twitter.getHomeTimeline(paging);

          //Recorremos la lista de status
          for (int i = 0; i < timeline.size(); i++) {
            //Cogemos el usuario del Tweet para sacar algunos de sus datos
            User user = timeline.get(i).getUser();
            System.out.println("Date: " + timeline.get(i).getCreatedAt());
            System.out.println("User: " + timeline.get(i).getUser().getScreenName());
            System.out.println("Location: " + user.getLocation());
            System.out.println("ID: " + timeline.get(i).getText() + "\n");

            //Igual que el anterior botón, sacamos el icono, usuario, fecha,
            //localización y tweet
            URL urlIcon;
```

```
                try {
                    urlIcon = new URL(user.getProfileImageURL());
                    ImageIcon userIcon = new ImageIcon(urlIcon);
                        JLabel icon = new JLabel();
                    icon.setIcon(userIcon);
                    icon.setText("");
                    String name = timeline.get(i).getUser().getScreenName();
                    String location = user.getLocation();
                    Date date = timeline.get(i).getCreatedAt();
                    JTextArea tweet = new JTextArea();
                    tweet.setText(timeline.get(i).getText());
                    tweet.setLineWrap(true);
                    tweet.setWrapStyleWord(true);
                    tweet.setOpaque(true);

                    //Lo añadimos todo como fila a la tabla
                    model.addRow(new Object[]{
                       icon, name, date, location, tweet
                    });
                } catch (MalformedURLException ex) {
                    Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE,
null, ex);
                }
              }
          } catch (TwitterException ex) {
              Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE,     null,
ex);
          }
      }

      //Botón Clean para limpiar la tabla
      private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {
         model.setNumRows(0);
      }
```

```
        //Botón de búsqueda por Usuario
        private void jButton4ActionPerformed(java.awt.event.ActionEvent evt) {
            try {
                //Si no hay nada escrito nos muestra el mensaje
                if (textField1.getText() == null || textField1.getText().isEmpty()) {
                    try {
                        throw new Exception("You must write a term to search");
                    } catch (Exception ex) {
                        Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE,
null, ex);
                    }
                }

                //Cogemos el texto escrito y lo pasamos a query
                String queryString = textField1.getText();
                //Creamos una lista de usuarios para almacenar las respuestas
                ResponseList<User> result;
                //Bucle para sacar el máximos número de usuarios
                for (int j = 1; j < 10; j++) {
                    //Realizamos la búsqueda específica de usuarios con la query
                    //y guardamos los resultados  en la lista
                    result = twitter.searchUsers(queryString, j);

                    //Bucle para recorrer todos los resultados
                    for (int i = 0; i < result.size(); i++) {
                        System.out.println("Date: " + result.get(i).getCreatedAt());
                        System.out.println("User: " + result.get(i).getScreenName());
                        System.out.println("Location: " + result.get(i).getLocation());

                        URL urlIcon;
                        //Como en los anteriores casos, sacamos el icono, usuario,
                        //localización, y, en este caso, la fecha de creación del perfil
                        //y su último tweet publicado
```

```
                try {
                    urlIcon = new URL(result.get(i).getProfileImageURL());
                    ImageIcon userIcon = new ImageIcon(urlIcon);

                    JLabel icon = new JLabel();
                    icon.setIcon(userIcon);
                    icon.setText("");
                    String name = result.get(i).getScreenName();
                    String location = result.get(i).getLocation();
                    Date date = result.get(i).getCreatedAt();

                    JTextArea tweet = new JTextArea();
                    if (result.get(i).getStatus().getText() != null) {
                        tweet.setText(result.get(i).getStatus().getText());
                        tweet.setLineWrap(true);
                        tweet.setWrapStyleWord(true);
                        tweet.setOpaque(true);
                    }

                    //Añadimos los datos como fila a la tabla
                    model.addRow(new Object[]{
                        icon, name, date, location, tweet
                    });
                } catch (MalformedURLException ex) {
                    Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE,
null, ex);
                }
            }
        }

        } catch (TwitterException ex) {
            Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE,     null,
ex);
        }
```

```
    }

    //Botón de Búsqueda por hashtag (etiqueta - #)
    private void jButton7ActionPerformed(java.awt.event.ActionEvent evt) {
        try {
            //Si no hay nada escrito nos muestra el mensaje
            if (textField1.getText() == null || textField1.getText().isEmpty()) {
                try {
                    throw new Exception("You must write a term to search");
                } catch (Exception ex) {
                    Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE,
null, ex);
                }
            }

            //Cogemos el texto escrito y añadimos el símbolo de hashtag (#)
            String queryString = "#" + textField1.getText();
            Integer numberOfTweets = 100;
            //Lo pasamos a query
            Query query = new Query(queryString);
            query.setCount(numberOfTweets);

            QueryResult result;
            //Realizamos la búsqueda, igual que la búsqueda por palabras
            result = twitter.search(query);
            //Guardamos los resultados en una lista de status
            List<Status> tweets = result.getTweets();

            //Bucle para recorrer la lista
            for (int i = 0; i < tweets.size(); i++) {
                User user = tweets.get(i).getUser();
                System.out.println("Date: " + tweets.get(i).getCreatedAt());
                System.out.println("User: " + tweets.get(i).getUser().getScreenName());
                System.out.println("Location: " + user.getLocation());
```

```
                System.out.println("ID: " + tweets.get(i).getText() + "\n");

                URL urlIcon;
                try {
                    //Igual que en las demás búsquedas, cogemos el icono, usuario
                    //fecha, localización y tweet
                    urlIcon = new URL(user.getProfileImageURL());
                    ImageIcon userIcon = new ImageIcon(urlIcon);

                    JLabel icon = new JLabel();
                    icon.setIcon(userIcon);
                    icon.setText("");
                    String name = tweets.get(i).getUser().getScreenName();
                    String location = user.getLocation();
                    Date date = tweets.get(i).getCreatedAt();

                    JTextArea tweet = new JTextArea();
                    tweet.setText(tweets.get(i).getText());
                    tweet.setLineWrap(true);
                    tweet.setWrapStyleWord(true);
                    tweet.setOpaque(true);

                    //Añadimos los datos como fila a la tabla
                    model.addRow(new Object[]{
                        icon, name, date, location, tweet
                    });
                } catch (MalformedURLException ex) {
                    Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE,
null, ex);
                }
            }
        } catch (TwitterException ex) {
            Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE,     null,
ex);
```

```
    }
}

//Botón para guardar los datos de la tabla en un archivo .txt
private void jButton8ActionPerformed(java.awt.event.ActionEvent evt) {
    try {
        //Declaramos la ruta donde se guardará el archivo
        String data = "/Users/Laura/Desktop/DatosTabla.txt";
        BufferedWriter bfw = new BufferedWriter(new FileWriter(data));

        //Bucle para recorrer las filas de la tabla
        for (int i = 0; i < jTable1.getRowCount(); i++) {
            //Bucle para recorrer las columnas de la tabla después del icono
            for (int j = 1; j < jTable1.getColumnCount(); j++) {
                //si es el area de texto (tweet), cogemos todo su texto
                if (jTable1.getValueAt(i, j) instanceof JTextArea) {
                    JTextArea area = (JTextArea) jTable1.getValueAt(i, j);
                    String tweet = area.getText();
                    bfw.write(tweet);
                    if (j < jTable1.getColumnCount() - 1) {
                        bfw.write(",");
                    }
                } else {
                    //Para el resto de contenidos cogemos su valor
                    bfw.write(jTable1.getValueAt(i, j).toString());
                    if (j < jTable1.getColumnCount() - 1) {
                        bfw.write(", ");
                    }
                }
            }
            bfw.newLine();
            bfw.newLine();
        }
        bfw.close();
```

```
            System.out.println("The file was saved succesfully!");
            //Mensaje si el archivo se ha creado bien
            JOptionPane.showMessageDialog(null, "The file was saved succesfully!");
        } catch (IOException e) {
            //Mensjae si ha habido problemas
            JOptionPane.showMessageDialog(null, "ERROR: There was a problem saving the file!");
            System.out.println("ERROR: There was a problem saving the file!" + e.getMessage());
        }
    }

    //Método para coger los datos de la cuenta logeada
    private void initUserInfo() throws MalformedURLException, TwitterException {
        try {
            //Cogemos el nombre de usuario
            Status status = twitter.getUserTimeline().get(0);
            User user = status.getUser();
            //Y su icono
            URL urlIcon = new URL(user.getProfileImageURL());
            ImageIcon userIcon = new ImageIcon(urlIcon, user.getScreenName());
            //Se muestran arriba a la izquierda de la pantalla
            jLabel3.setIcon(userIcon);
            jLabel4.setText("@" + user.getScreenName());
        } catch (MalformedURLException ex) {
            Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
```

```
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {
            java.util.logging.Logger.getLogger(TwitterView.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (InstantiationException ex) {
            java.util.logging.Logger.getLogger(TwitterView.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (IllegalAccessException ex) {
            java.util.logging.Logger.getLogger(TwitterView.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {
            java.util.logging.Logger.getLogger(TwitterView.class.getName()).log(java.util.logging.Level.SEVERE, null, ex);
        }
        //</editor-fold>
```

```
        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new TwitterView().setVisible(true);
            }
        });
    }

    // Variables declaration
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JButton jButton3;
    private javax.swing.JButton jButton4;
    private javax.swing.JButton jButton5;
    private javax.swing.JButton jButton6;
    private javax.swing.JButton jButton7;
    private javax.swing.JButton jButton8;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JScrollPane jScrollPane2;
    private javax.swing.JTable jTable1;
    private javax.swing.JTextArea jTextArea1;
    private java.awt.TextField textField1;
    // End of variables declaration
}
```

2.5 Desarrollo

2.5.1 Creación del Proyecto

Abrimos el IDE *NetBeans*, para crear un nuevo proyecto pulsamos sobre la pestaña superior *File*, y elegimos la opción de *New Project*. Nos mostrará una ventana emergente como la de la ilustración, donde debemos elegir la categoría *Java*, y en proyectos *Java Application.*

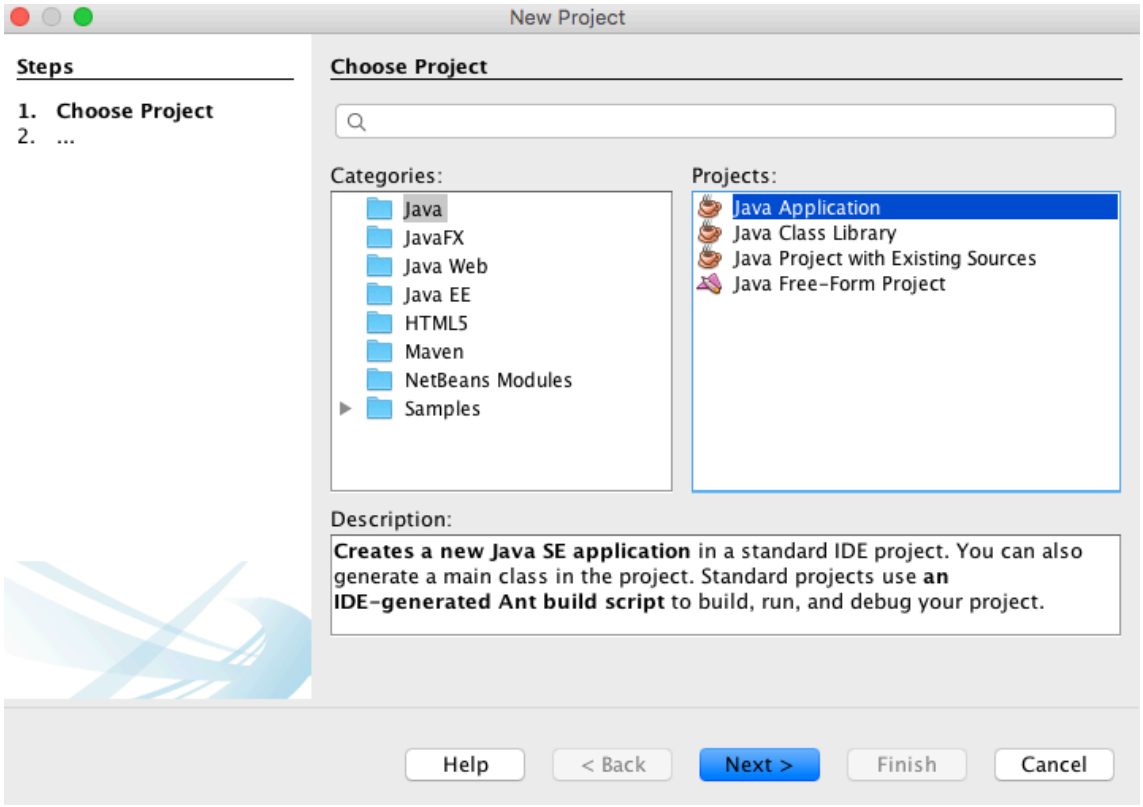

Ilustración 15: Creación del Proyecto en NetBeans

Seguiremos los pasos del *Wizard*, rellenando los campos necesarios que nos pidan hasta finalizar, y ya tendremos nuestro proyecto Java creado.

Ahora debemos agregar la librería Twitter4J, que descargamos anteriormente, a nuestro proyecto Java. Para ello, sobre el proyecto que hemos creado en *NetBeans* pulsaremos con el botón izquierdo y elegimos la opción de *Properties.*

Una nueva ventana emergente se abrirá, en el menú de la izquierda seleccionamos la opción *Libraries*. Para añadirlas librerías pulsamos el botón de *Add Jar/Folder* y se nos abrirá un explorador para elegirla librería de la ruta donde la guardamos previamente. Del archivo de la librería debemos abrir la carpeta llamada *Lib* donde veremos cinco archivos .jar como vemos en la ilustración.

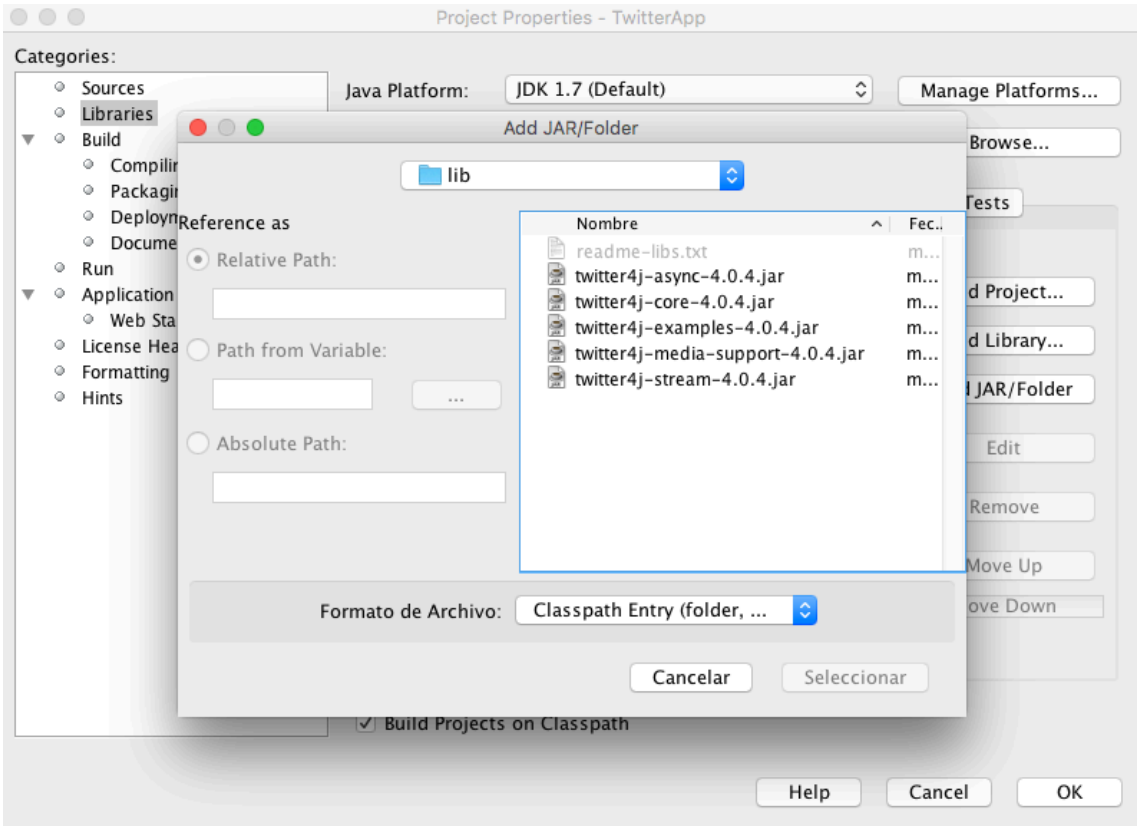

Ilustración 16: Añadir librería Twitter4j

Seleccionamos todos los archivos y los añadimos a nuestro proyecto, la ventana de librerías debería quedar como la siguiente ilustración.

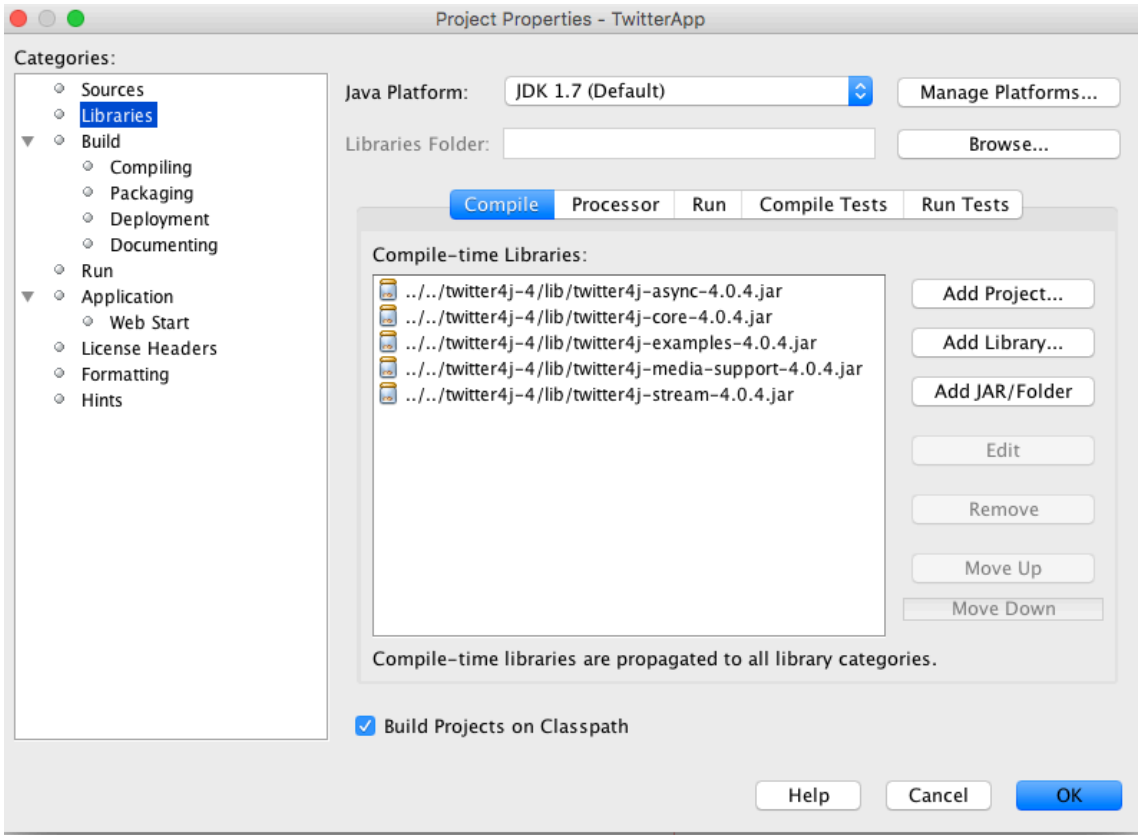

Ilustración 17: Librerías Twitter4J

2.5.2 Funcionalidades

Tenemos dos formas de visualizar el proyecto en *NetBeans*, con la vista de código fuente, y la vista de diseño, que vemos en la siguiente ilustración.

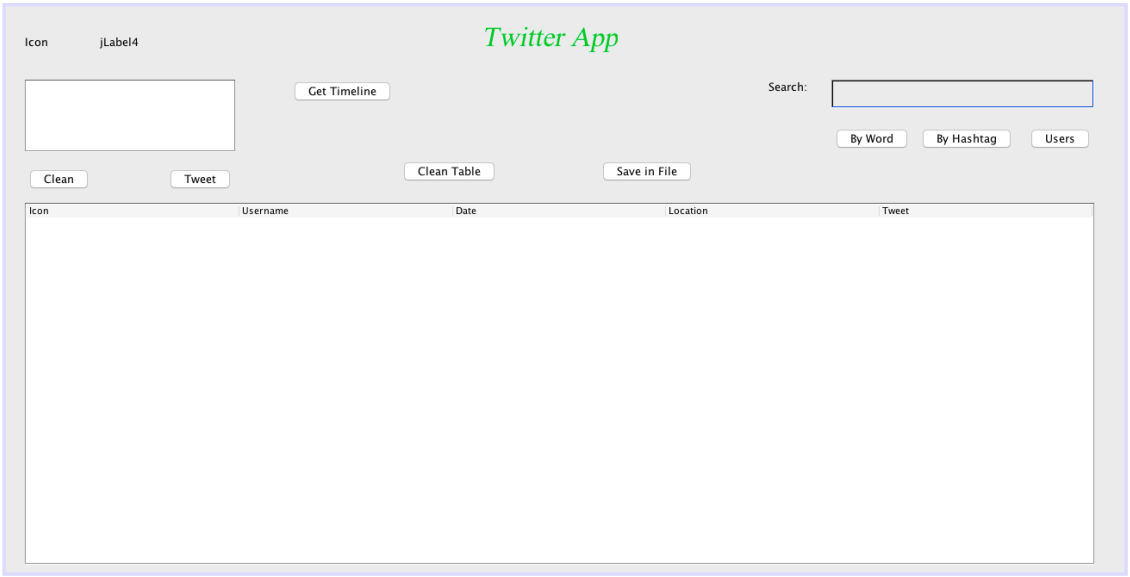

Ilustración 18: Librerías Twitter4J

En la vista de diseño añadimos y colocamos los componentes visuales como queremos que se muestren en la vista de nuestra aplicación final. Sus funcionalidades y características las desarrollaremos en la vista del código fuente.

A continuación, mostraremos extractos del código para explicar las funcionalidades de la aplicación.

Primero, tenemos declarados unas variables generales necesarias en el proyecto, que son una de tipo *Twitter*, proveniente de la librería *Twitter4J*, que permite realizar las diversas funcionalidades que incluye el servicio de *Twitter*; un *String* para guardar el mensaje a publicar en el *Tweet*; y una de tipo *DefaultTableModel*, que servirá para generar el modelo de la tabla donde se mostrarán los datos.

```
43      //Variable general de Twitter para aplicar sus funciones
44      Twitter twitter;
45      //String para almacenar el mensaje a Tweetear
46      String message = null;
47      //Modelo de la tabla que mostrará los datos
48      DefaultTableModel model;
```

Ilustración 19: Variables generales

El método llamado *Tweet(),* es el que nos permite publicar un nuevo *Tweet* en la cuenta a la que pertenecen los *tokens* y *keys* que nos conectan con *Twitter*. Como vemos en el código usamos la variable declarada anteriormente *twitter*, para publicar nuestro *Tweet*, con el método existente de la librería *updateStatus(),* al que le pasamos el mensaje escrito.

```
54      //Método para publicar un Tweet
55      public void Tweet() throws TwitterException {
56
57          Paging page = new Paging();
58          Status tweet = twitter.updateStatus(message);
59
60      }
```

Ilustración 20: Método Tweet()

Este método será llamado posteriormente dentro de la funcionalidad del botón llamado *Tweet* (*tweetear* / publicar).

Dentro del método principal de la clase, crearemos el constructor para introducir las claves de *consumer* y *tokens* de nuestra app, que conseguimos antes en la web de *Twitter Developers*. Se añaden las cuatro claves al constructor en el orden que se ve en código. Después se debe añadir una factoría de *Twitter* a nuestra variable *Twitter*, construyendo el constructor, como vemos en la última línea, para que realice la conexión.

```
68          //Constructor con las claves de nuestra app para conectar con Twitter
69          ConfigurationBuilder cb = new ConfigurationBuilder();
70          cb.setDebugEnabled(true)
                    .setOAuthConsumerKey("Cw3UAKyEDRxdbMSnTfXnoTr3X")
72                  .setOAuthConsumerSecret("rsY1Wfrs4vgdYn5nBSmTXEoZ1bB6hRdHZK7xnK8caUypvmzJWk")
73                  .setOAuthAccessToken("328013467-Yqnk6LaLXVx51BMMd4p1SEciJPVhu8ey9TXaPO8X")
74                  .setOAuthAccessTokenSecret("eE8tll1F7OD3ddl6uZrC9bgrYN4pSIgAhlc3hASNSTiUZ");
75          twitter = new TwitterFactory(cb.build()).getInstance();
```

Ilustración 21: Conexión con las claves

Se fijan las medidas y características del modelo de la tabla que mostrará los datos, y en la última línea se fija que se inicialice con cero filas.

```
84          //Fijamos las medidas y el modelo de la tabla
85          jTable1.setDefaultRenderer(Object.class, new ImgTable());
86          jTable1.getColumnModel().getColumn(0).setPreferredWidth(10);
87          jTable1.getColumnModel().getColumn(1).setPreferredWidth(40);
88          jTable1.getColumnModel().getColumn(2).setPreferredWidth(50);
89          jTable1.getColumnModel().getColumn(3).setPreferredWidth(40);
90          jTable1.getColumnModel().getColumn(4).setPreferredWidth(400);
91          jTable1.getColumnModel().getColumn(4).setResizable(true);
92
93          model = (DefaultTableModel) jTable1.getModel();
94
95          //Indicamos el número de filas al inicio
96          model.setNumRows(0);
```

Ilustración 22: Modelo tabla 1

El siguiente método es muy importante para el modelo de la tabla, pues extenderá del modelo por defecto, y añade las opciones de mostrar una *label* y un área texto, para poder mostrar correctamente los iconos y los *tweets* en la tabla.

```java
//Método para que dentro de la tabla se muestren los iconos en formato
//imagen, y el Tweet en un area de texto correctamente
public class ImgTable extends DefaultTableCellRenderer {

    public Component getTableCellRendererComponent(JTable table, Object value, boolean isSelected, boolean hasFocus, int row, int column) {
        if (value instanceof JLabel) {
            JLabel label = (JLabel) value;
            return label;
        } else if (value instanceof JTextArea) {
            JTextArea area = (JTextArea) value;
            return area;
        } else {
            return super.getTableCellRendererComponent(table, value, isSelected, hasFocus, row, column);
        }
    }
}
```

Ilustración 23: Modelo tabla 2

Este es el método del botón *Clean*, que limpia el área de texto de los tweets, simplemente fijamos el texto del área de texto a vacío.

```java
    //Método para limpiar el area de texto con el botón 1
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {

        jTextArea1.setText("");
    }
```

Ilustración 24: Botón Clean

El siguiente método es el del botón *Tweet*, que sirve para publicar los *tweets*. Si se escriben más de 280 caracteres en el área de texto nos mostrará un mensaje de error de que el máximo de caracteres permitidos es de 280, el límite actual de *Twitter*. Si la caja de texto está vacía, nos mostrará un mensaje diciendo que debemos escribir algo para publicar. Y si todo va bien, llamará al método *Tweet()* que vimos antes, y publicará nuestro *Tweet* mostrándonos un mensaje satisfactorio.

```java
    //Método del botón 2 que llama al método Tweet() si todo está bien,
    //o muestra los mensajes correspondientes de error
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {

        try {
            if (jTextArea1.getText().length() > 280) {
                throw new Exception("The maximum characters allowed are 280");
            }
            if (jTextArea1.getText().isEmpty()) {
                throw new Exception("You must write something to tweet");
            }
            message = jTextArea1.getText();
            try {
                Tweet();
                JOptionPane.showMessageDialog(null, "Tweet was sent successfully");
                jTextArea1.setText("");
            } catch (TwitterException te) {
                Logger.getLogger(Window.class.getName()).log(Level.SEVERE, null, te);
            }

        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, e);
        }
    }
```

Ilustración 25: Botón Tweet

A continuación, tenemos el método del botón de Búsqueda por palabra. Si no hay nada escrito nos mostrará el mensaje de error relevante a ello. Si hay término escrito cogerá el texto escrito en el *String queryString*, como vemos en el código, y lo pasará a tipo *query*. Para coger el resultado declaramos la variable *result* de tipo *QueryResult*, y ejecutamos la búsqueda a través de nuestra variable general twitter, con la funcionalidad de la librería *search*, pasándole nuestra *query*. Después guardamos el resultado en una lista de tipo *Status,* que es el tipo como guarda la librería los tweets y sus datos.

```
368     private void jButton3ActionPerformed(java.awt.event.ActionEvent evt) {
369
370         try {
371             //Si no hay nada escrito muestra el mensaje
372             if (textField1.getText() == null || textField1.getText().isEmpty()) {
373                 try {
374                     throw new Exception("You must write a term to search");
375                 } catch (Exception ex) {
376                     Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE, null, ex);
377                 }
378             }
379
380             String queryString = textField1.getText();
381             Integer numberOfTweets = 100;
382
383             //Pasa el texto de String a query
384             Query query = new Query(queryString);
385             query.setCount(numberOfTweets);
386
387             QueryResult result;
388
389             //Realiza la búsqueda de la query en Twitter
390             result = twitter.search(query);
391
392             //Guarda los datos en una lista de status
393             List<Status> tweets = result.getTweets();
```

Ilustración 26: Botón Búsqueda por Palabras 1

Tras esto, recorremos la lista de los resultados con un bucle *for*, en el que iremos cogiendo los datos que queremos mostrar en la tabla posteriormente. Para poder mostrar el icono, debemos coger la URL de la imagen, como vemos en la línea 407 de código, y crear una nueva variable de tipo *ImageIcon* con esta URL, para luego asignársela como icono a una *label* sin texto, y así se nos mostrará correctamente en la tabla. También cogeremos el nombre de usuario, la fecha, la localización y el texto del *tweet*, todo con métodos que proporciona la librería *Twitter4J* para su clase *Status*. Finalmente, añadimos los datos que hemos sacado a una nueva fila de la tabla.

```
395             //Bucle para recorrer la lista status
                for (int i = 0; i < tweets.size(); i++) {
397                 //Cogemos el usuario del Tweet para sacar algunos de sus datos
398                 User user = tweets.get(i).getUser();
399                 System.out.println("Date: " + tweets.get(i).getCreatedAt());
400                 System.out.println("User: " + tweets.get(i).getUser().getScreenName());
401                 System.out.println("Location: " + user.getLocation());
402                 System.out.println("ID: " + tweets.get(i).getText() + "\n");
403 
404                 URL urlIcon;
405                 try {
406                     //Cogemos la URL del icono del usuario
407                     urlIcon = new URL(user.getProfileImageURL());
408                     //Convertimos la URL en una imagen tipo icono
409                     ImageIcon userIcon = new ImageIcon(urlIcon);
410 
411                     //Metemos el icono en una label como su icono
412                     //para que se pueda visualizar dentro de la tabla
413                     JLabel icon = new JLabel();
414                     icon.setIcon(userIcon);
415                     icon.setText("");
416 
417                     //Cogemos el nombre de usuario
418                     String name = tweets.get(i).getUser().getScreenName();
419                     //Su localización si la tiene
420                     String location = user.getLocation();
421                     //La fecha y hora de creación del tweet
422                     Date date = tweets.get(i).getCreatedAt();
423 
424                     //COgemos el texto del tweet y lo metemos en un area de texto
425                     JTextArea tweet = new JTextArea();
426                     tweet.setText(tweets.get(i).getText());
427                     //Características para que salte de línea y verlo entero
428                     tweet.setLineWrap(true);
429                     tweet.setWrapStyleWord(true);
430                     tweet.setOpaque(true);
431 
432                     //Añadimos como fila de la tabla el icono, usuario, fecha,
433                     //localización y tweet
434                     model.addRow(new Object[]{
435                         icon, name, date, location, tweet
436                     });
```

Ilustración 27: Botón Búsqueda por Palabras 2

Posteriormente tenemos el método del botón *Get Timeline*, que nos mostrará en la tabla nuestro *Timeline*. Como vemos en código, aquí debemos crear una lista de *Status* y guardar los *tweets* con el método *getHomeTimeline(),* el resto del método es igual al método anterior de la búsqueda para mostrar los resultados.

```
450     //Botón para ver el Timeline de la cuenta
451     private void jButton5ActionPerformed(java.awt.event.ActionEvent evt) {
452 
453         try {
454             //Paginamos los datos para ver los máximos posibles
455             int totalTweets = 100;
456             Paging paging = new Paging(1, totalTweets);
457             //COgemos el timeline y guardamos los tweets en una lista de Status
458             List<Status> timeline = twitter.getHomeTimeline(paging);
```

Ilustración 28: Botón Get Timeline

Ahora tenemos el método del botón *Clean Table,* que sirve para limpiar la tabla de datos. Tan solo cogemos la variable de *model*, el modelo de datos de la tabla, y fijamos el número de filas a cero con el método *setNumRows():*

```
504     //Botón Clean para limpiar la tabla
505     private void jButton6ActionPerformed(java.awt.event.ActionEvent evt) {
506
507         model.setNumRows(0);
508     }
```

Ilustración 29: Botón Clean Table

Después nos encontramos con el método del botón de Búsqueda por Usuario, para realizar búsquedas en *Twitter* de usuarios. Este método funciona prácticamente igual que el de búsqueda por palabras, solo que aquí debemos realizar un bucle anterior para sacar un mayor número de usuarios, ya que el método solo te devuelve unos 20 usuarios, entonces son el bucle se van paginando, aquí hasta sacar 100 usuarios si los hay. También, la búsqueda aquí se realiza con el método *searchUsers()* en vez de solo *search()* como en la anterior. Tras esto el método es igual al otro recorriendo los resultados y añadiéndolos a la tabla.

```
529             //Bucle para sacar el máximos número de usuarios
530             for (int j = 1; j < 10; j++) {
531                 //Realizamos la búsqueda específica de usuarios con la query
532                 //y guardamos los resultados  en la lista
533                 result = twitter.searchUsers(queryString, j);
534
535                 //Bucle para recorrer todos los resultados
                    for (int i = 0; i < result.size(); i++) {
```

Ilustración 30: Botón Búsqueda por Usuario

Ahora vamos a por el método del botón de Búsqueda por *Hashtag* (etiqueta - #). Este método es exactamente igual al de búsqueda por palabra, solo tenemos que añadirle el símbolo de *hashtag* (#) al término de búsqueda para que la realice correctamente como vemos en la ilustración del código.

```
594             //Cogemos el texto escrito y añadimos el símbolo de hashtag (#)
595             String queryString = "#" + textField1.getText();
```

Ilustración 31: Botón Búsqueda por Hashtag

Después tenemos el método del botón *Save in File,* para guardar todos los datos que contiene la tabla en un archivo .txt. Aquí declaramos la ruta donde se creará el archivo y su nombre, y luego hacemos dos bucles para recorrer todas sus filas y columnas,

excepto la del icono pues no es texto. Cuando sea la columna del texto del *tweet*, al ser un área de texto, lo declaramos de forma distinta para que guarde todo el texto del área, mientras que en el resto de los casos guarda el valor de la celda.

```
654     //Botón para guardar los datos de la tabla en un archivo .txt
655     private void jButton8ActionPerformed(java.awt.event.ActionEvent evt) {
656
657         try {
658             //Declaramos la ruta donde se guardará el archivo
659             String data = "/Users/Laura/Desktop/DatosTabla.txt";
660             BufferedWriter bfw = new BufferedWriter(new FileWriter(data));
661
662             //Bucle para recorrer las filas de la tabla
663             for (int i = 0; i < jTable1.getRowCount(); i++) {
664                 //Bucle para recorrer las columnas de la tabla después del icono
665                 for (int j = 1; j < jTable1.getColumnCount(); j++) {
666                     //si es el area de texto (tweet), cogemos todo su texto
667                     if (jTable1.getValueAt(i, j) instanceof JTextArea) {
668                         JTextArea area = (JTextArea) jTable1.getValueAt(i, j);
669                         String tweet = area.getText();
670                         bfw.write(tweet);
671                         if (j < jTable1.getColumnCount() - 1) {
672                             bfw.write(",");
673                         }
674                     } else {
675                         //Para el resto de contenidos cogemos su valor
676                         bfw.write(jTable1.getValueAt(i, j).toString());
677                         if (j < jTable1.getColumnCount() - 1) {
678                             bfw.write(", ");
679                         }
680                     }
681
682                 }
683                 bfw.newLine();
684                 bfw.newLine();
685             }
686
687             bfw.close();
688             System.out.println("The file was saved succesfully!");
689             //Mensaje si el archivo se ha creado bien
690             JOptionPane.showMessageDialog(null, "The file was saved succesfully!");
```

Ilustración 32: Botón Save in FIle

Finalmente, tenemos el método que cogerá el usuario e icono de nuestra cuenta, con la que creamos la aplicación *Twitter Developers* y cogimos las claves. Para ello sacamos uno de nuestros *status (tweets)* el cero mismamente, y cogemos el nombre de usuario, y el icono igual que hicimos para mostrarlo en la tabla cogiendo su URL y creando una nueva *ImageIcon*. Después mostraremos ambos en nuestra vista con los *labels*.

```
699     //Método para coger los datos de la cuenta logeada
700     private void initUserInfo() throws MalformedURLException, TwitterException {
701
702         try {
703             //Cogemos el nombre de usuario
704             Status status = twitter.getUserTimeline().get(0);
705             User user = status.getUser();
706             //Y su icono
707             URL urlIcon = new URL(user.getProfileImageURL());
708             ImageIcon userIcon = new ImageIcon(urlIcon, user.getScreenName());
709             //Se muestran arriba a la izquierda de la pantalla
710             jLabel3.setIcon(userIcon);
711             jLabel4.setText("@" + user.getScreenName());
712         } catch (MalformedURLException ex) {
713             Logger.getLogger(TwitterView.class.getName()).log(Level.SEVERE, null, ex);
714         }
715
716     }
```

Ilustración 33: Mostrar Usuario e Icono logeados

2.6 Conclusiones

Con este proyecto se pretende mostrar cómo realizar fácilmente una aplicación con diversas funcionalidades de *Twitter* desarrollando nosotros mismos el código fuente y una interfaz accesible.

Se ha realizado usando el lenguaje *Java* y la librería *Twitter4J*, pero existen muchas otras posibilidades con otros lenguajes y librerías para desarrollar nuestras propias aplicaciones de *Twitter*.

Esta librería nos proporciona multitud de funciones y métodos para hacer nuestro desarrollo sencillo y rápido, pero siempre contamos con la posibilidad de usar directamente la API de *Twitter* para desarrollar nosotros mismos las funcionalidades que deseemos. En esta ocasión se han usado funciones tanto de publicación, muestra de *Timeline* y de distintos tipos de búsqueda, pero la librería cuenta con muchas más funcionalidades que cubren la mayoría de las funciones del servicio, como seguir usuarios, responder a *Tweets* o enviar mensajes directos. Además, es muy sencilla de configurar y añadir a nuestro proyecto, como hemos visto tan solo necesitamos descargarla y añadir los ficheros .jar a nuestro proyecto, lo que la convierte en una librería muy accesible y su uso está muy extendido también al ser de código libre.

2.7 Trabajo a futuro

El proyecto es una muestra de lo que puede llegar a hacerse con una aplicación propia de *Twitter* usando está librería, con usos de publicación de *Tweets*, ver el *Timeline* y realizar búsquedas por palabras, *hashtags* y usuarios. Estás funcionalidades se podrían ampliar en el fututo con la inclusión de contestar a *tweets* del *Timeline*, *Retweetear* o darle a *Like*, el uso de los mensajes privados, y las listas, además de poder seguir y dejar de seguir a usuarios, para finalmente llegar a hacer una aplicación completa del servicio de *Twitter*.

La función principal que sería más interesante de añadir sería la de poder hacer *Login* con cualquier cuenta de *Twitter*, como si fuera el mismo servicio, usando el nombre de usuario y su contraseña, poder entrar en tu cuenta con la aplicación. Y no ser siempre la cuenta dependiente con la que se creo la aplicación en *Twitter Developers*.

Otros puntos que mejorar a futuro serían la interfaz de la vista, haciéndola más profesional y con un diseño mejorado.

Capitulo 3

3 Manual de Usuario

A continuación, vamos a presentar el Manual de Usuario de la aplicación, donde veremos las distintas funcionalidades, cómo funcionan y capturas de cómo se ve cada paso.

3.1 Vista

AL ejecutar la aplicación veremos la siguiente pantalla que es la vista de inicio. Arriba a la izquierda podemos ver el icono y el usuario que estamos usando, en este caso *@sanchezrum*.

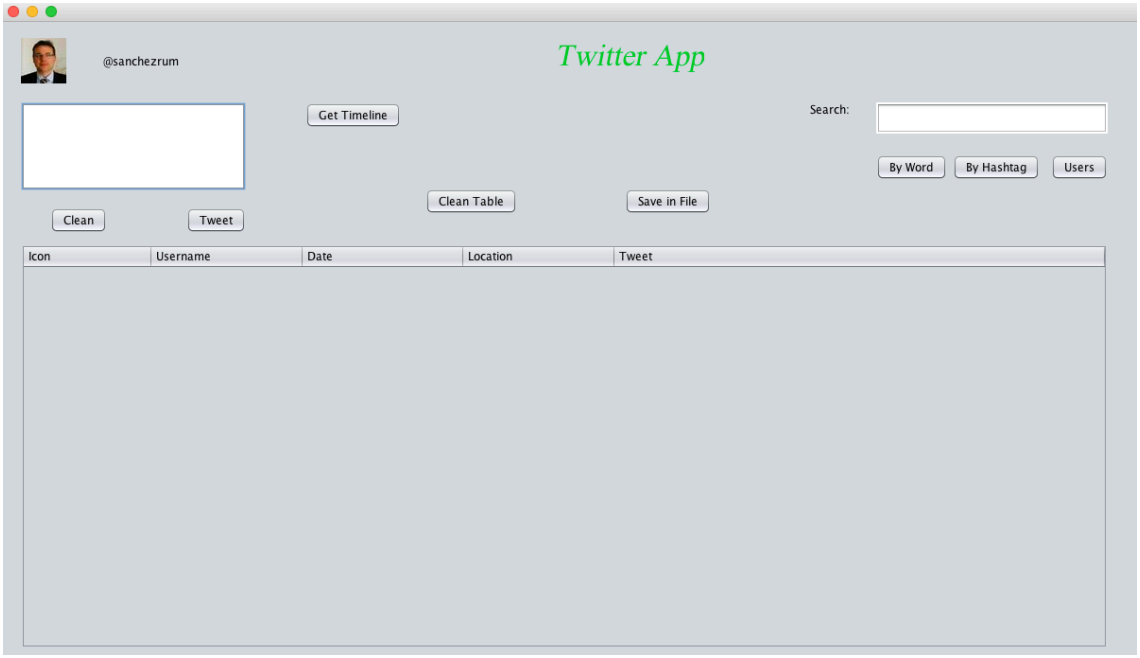

Ilustración 34: Vista general

En el cuadro de texto de la izquierda podemos escribir para publicar un *Tweet*, y al pulsar el botón *Tweet* si todo ha ido bien, nos mostrará un mensaje satisfactorio, como el de la imagen

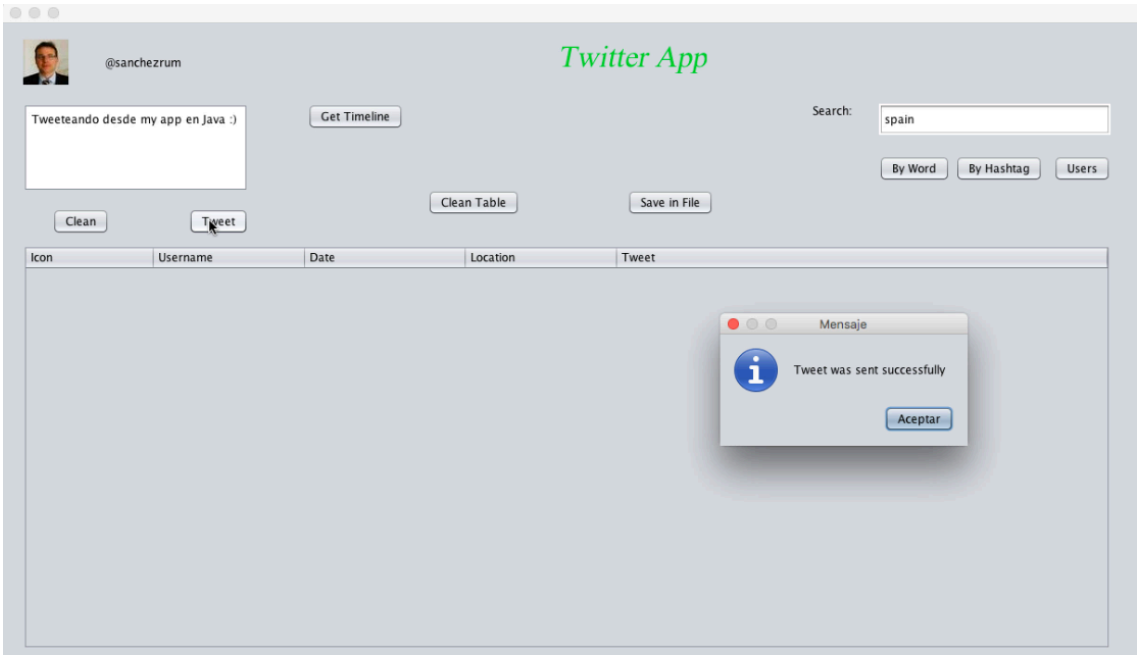

Ilustración 35: Publicar Tweet 1

Por el contrario, puede haber dos casos conocidos de error, si no escribimos nada y pulsamos el botón, nos mostrará un mensaje como el siguiente.

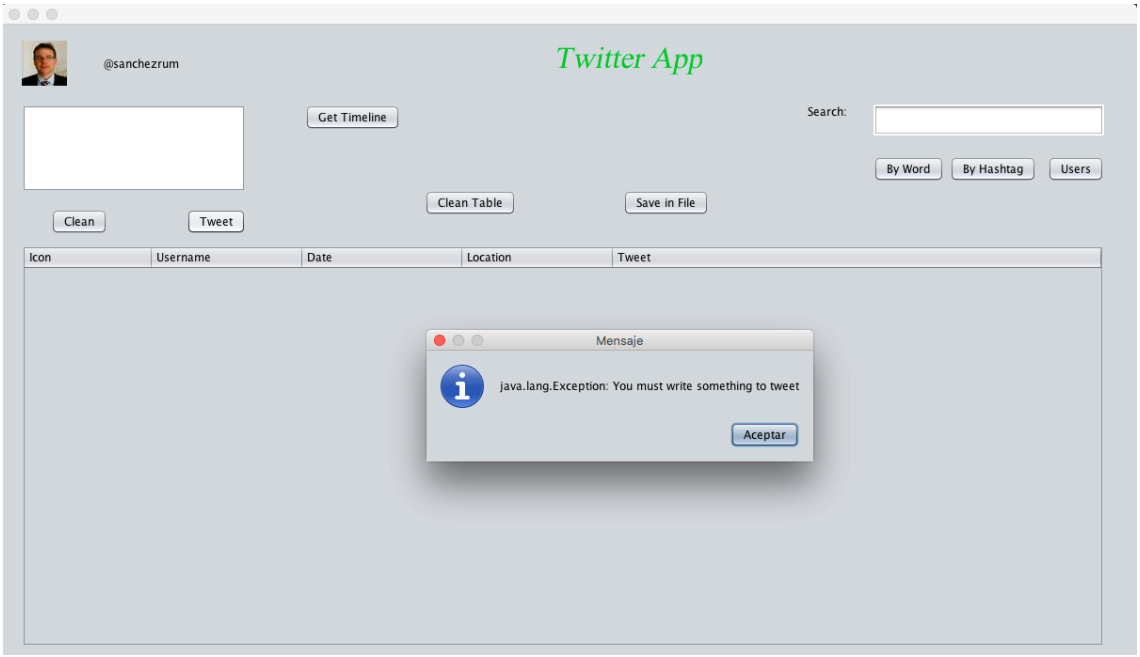

Ilustración 36: Publicar Tweet 2

Y el otro error conocido sería si se escriben más de 280 caracteres, que es el límite de *Twitter*, nos dirá que no podemos escribir más de 280, como en la imagen.

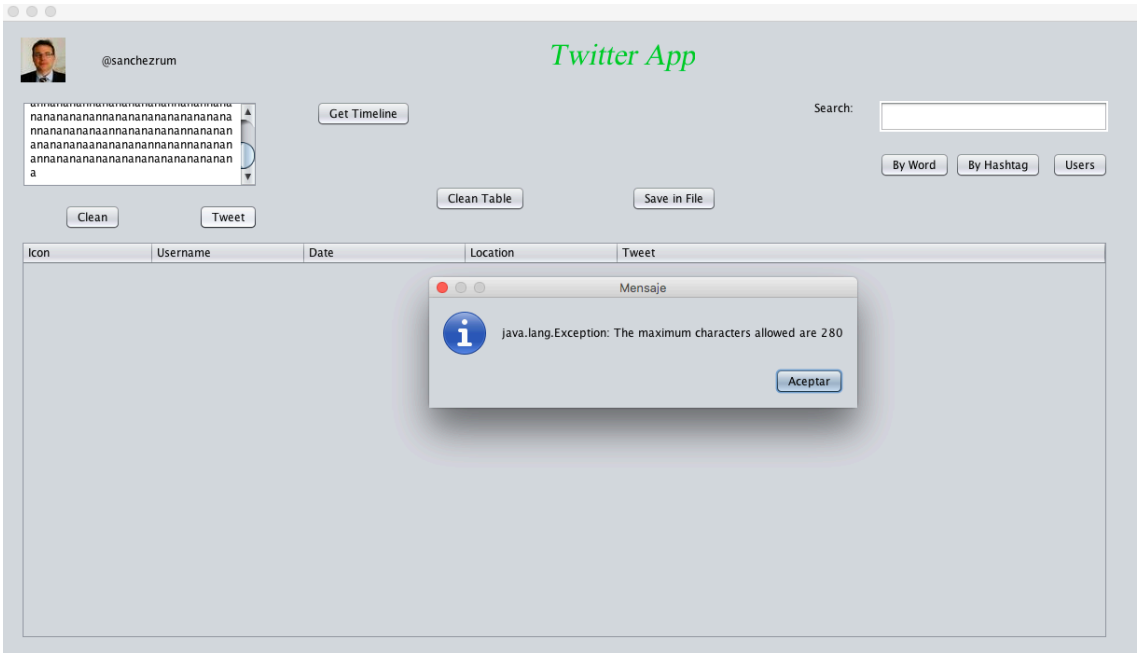

Ilustración 37: Publicar Tweet 3

Con el botón *Clea*n de al lado podemos borrar todo lo escrito en esa caja de texto.

Al pulsar el botón *Get Timeline*, nos mostrará en la tabla los últimos *Tweets* de la gente que seguimos, mostrándonos icono, usuario, fecha, localización y el texto del *Tweet*. En la imagen podemos ver un ejemplo de cómo se vería.

Pulsando el botón *Clean Table* podemos limpiar todo el contenido de la tabla, o si queremos podemos seguir usando la aplicación y los nuevos datos aparecerán debajo de los ya existentes.

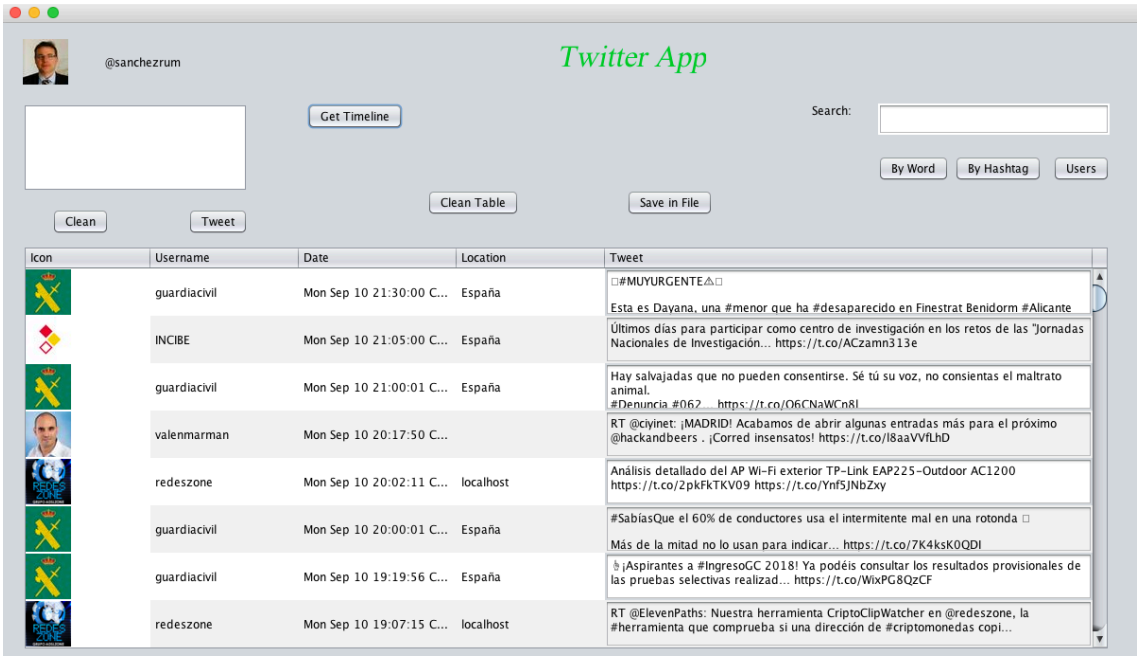

Ilustración 38: Get Timeline

En la caja de texto de la derecha podemos realizar las búsquedas, pueden ser de tres tipos:

1. Por palabra
2. Por Hashtag
3. Por Usuario

Por palabra buscara que contenga esa palabra o frase dentro del Tweet. (Ej.: Madrid)

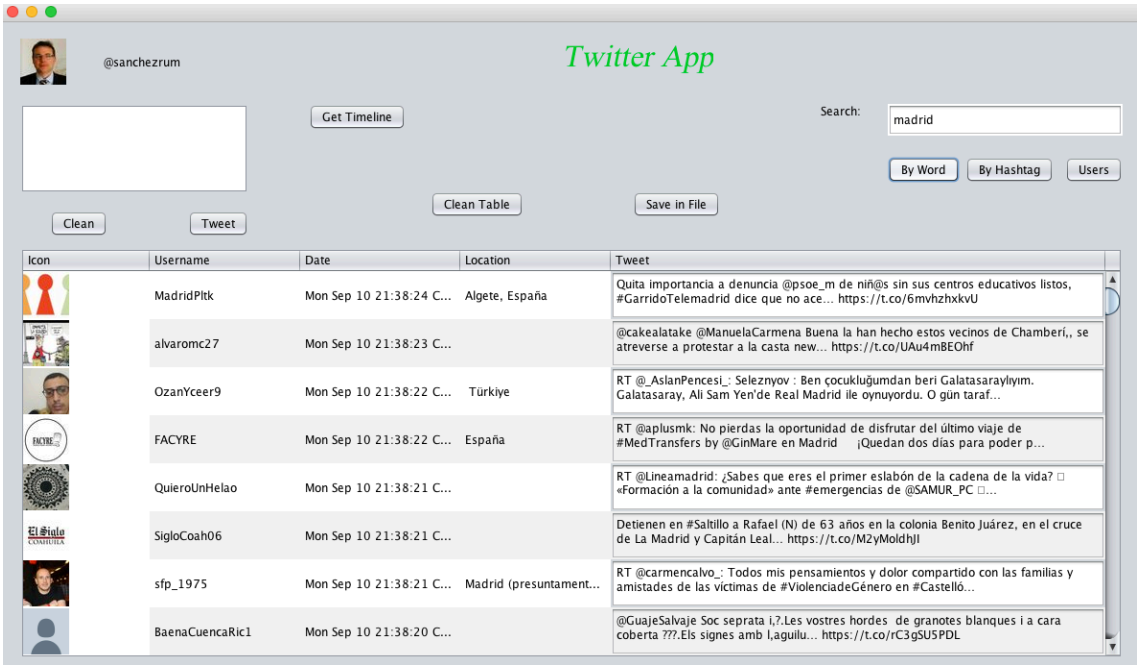

Ilustración 39: Búsqueda por palabra

Por *hashtag* realizará la búsqueda que contenga esa palabra en el *Tweet* con el icono de etiqueta (Ej.: #madrid)

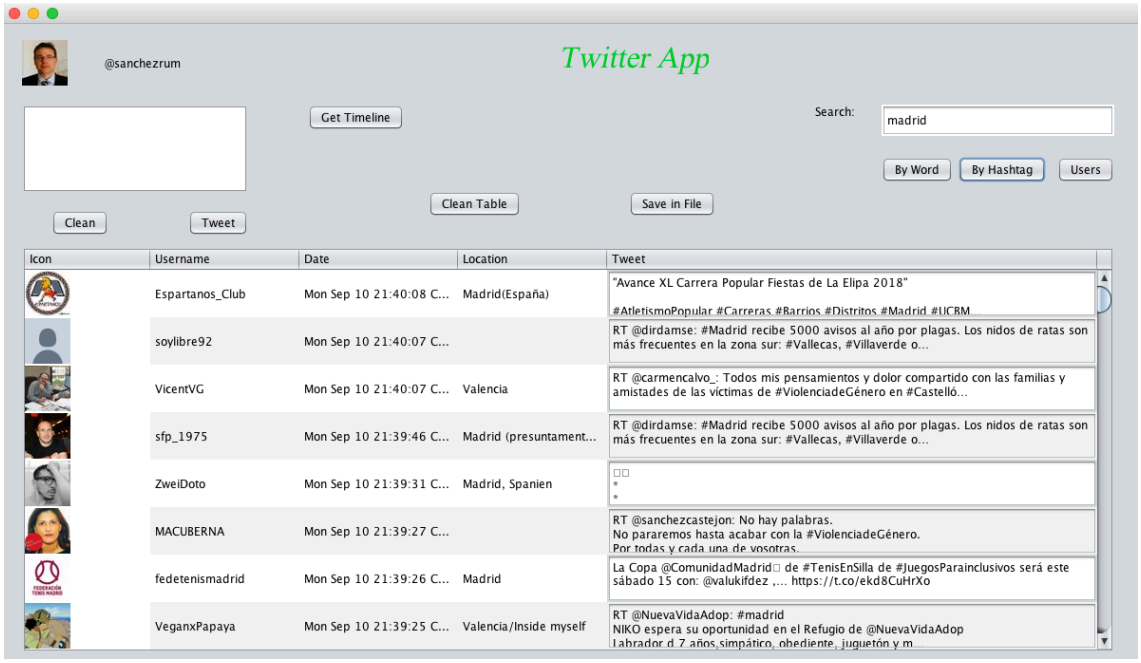

Ilustración 40: Búsqueda por hashtag

Y por usuario realiza la búsqueda por nombre de usuario, aunque al usar el algoritmo de Twitter, los resultados que devuelve son como los que devuelve Twitter en sus propias búsquedas, es decir, que no solo devuelve los usuarios que incluyen la palabra en su nombre, si no también perfiles que están relacionados con esa palabra. Como vemos en el ejemplo no solo nos devuelve el perfil del *@realmadrid* que contiene la palabra Madrid, si no también el del *@Atleti*, que está muy relacionado con el termino Madrid.

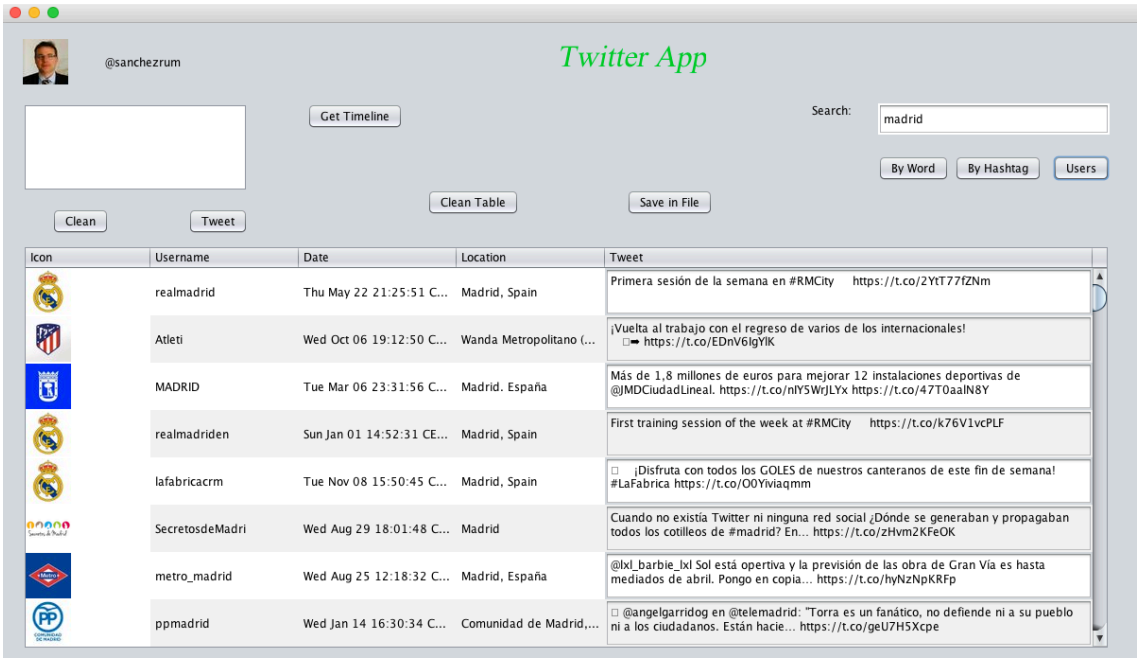

Ilustración 41: Búsqueda por usuario

La última funcionalidad es poder guardar toda la información que se encuentra en la tabla en un archivo .txt. Se puede hacer pulsando el botón *Save in File*, si todo va bien nos mostrará un mensaje satisfactorio como en la imagen.

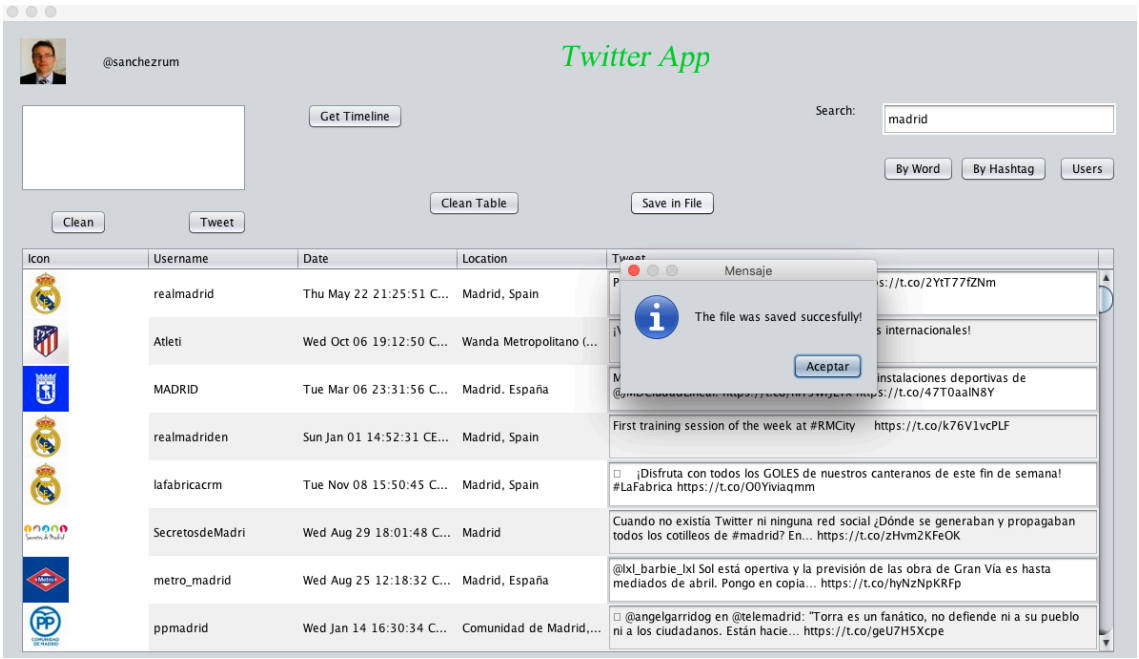

Ilustración 42: Guardar fichero 1

El archivo se creará en la ruta que pusimos en el código y se verá por ejemplo como en la siguiente imagen.

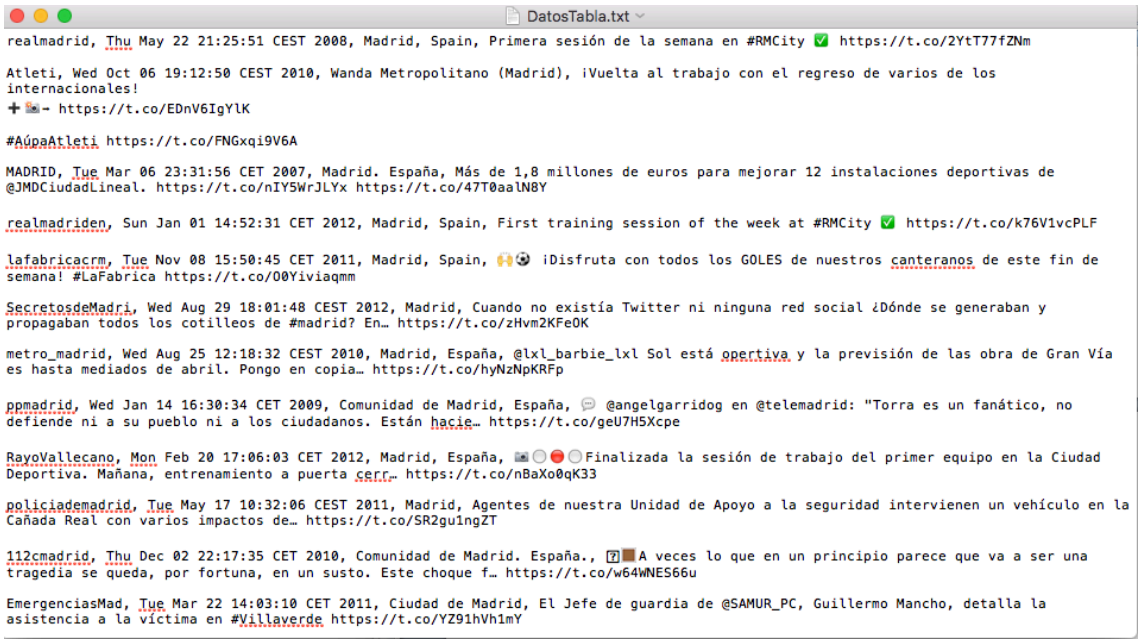

Ilustración 43: Guardar fichero 2

Capítulo 4

4 Bibliografía

4.1 Bibliografía

- Twitter4J: http://twitter4j.org/en/index.html
- Twitter Developers: https://developer.twitter.com
- NetBeans: https://netbeans.org
- Twitter APIs: https://brightplanet.com/2013/06/twitter-firehose-vs-twitter-api-whats-the-difference-and-why-should-you-care/
- Twitter Wikipedia: https://en.wikipedia.org/wiki/Twitter
- Publicar Tweets: http://panamahitek.com/utilizando-twitter-desde-java-con-twitter4j/
- Twitter4J ejemplos: http://www.tothenew.com/blog/?s=twitter

Capítulo 5

5 Índice de Figuras

5.1 Ilustraciones

Ilustración 1: Creación de Cuenta de Desarrollador 11
Ilustración 2: Creación de Cuenta de Desarrollador 2 11
Ilustración 3: Creación de Cuenta de Desarrollador 3 12
Ilustración 4: Creación de Cuenta de Desarrollador 4 13
Ilustración 5: Creación de App 14
Ilustración 6: Creación de App 2 14
Ilustración 7: Creación de App 3 15
Ilustración 8: Creación de App 4 16
Ilustración 9: Obtención de claves 1 17
Ilustración 10: Obtención de claves 2 17
Ilustración 11: Descarga Twitter4J 18
Ilustración 12: Descarga NetBeans 1 19
Ilustración 13: Descarga NetBeans 2 20
Ilustración 14: Descarga Java 21
Ilustración 15: Creación del Proyecto en NetBeans 45
Ilustración 16: Añadir librería Twitter4J 46
Ilustración 17: Librerías Twitter4J 47
Ilustración 18: Librerías Twitter4J 47
Ilustración 19: Variables generales 48
Ilustración 20: Método Tweet() 48
Ilustración 21: Conexión con las claves 49
Ilustración 22: Modelo tabla 1 49
Ilustración 23: Modelo tabla 2 50
Ilustración 24: Botón Clean 50
Ilustración 25: Botón Tweet 50
Ilustración 26: Botón Búsqueda por Palabras 1 51
Ilustración 27: Botón Búsqueda por Palabras 2 52
Ilustración 28: Botón Get Timeline 52
Ilustración 29: Botón Clean Table 53
Ilustración 30: Botón Búsqueda por Usuario 53
Ilustración 31: Botón Búsqueda por Hashtag 53
Ilustración 32: Botón Save in File 54
Ilustración 33: Mostrar Usuario e Icono logeados 55
Ilustración 34: Vista general 58
Ilustración 35: Publicar Tweet 1 59
Ilustración 36: Publicar Tweet 2 59
Ilustración 37: Publicar Tweet 3 60
Ilustración 38: Get Timeline 61
Ilustración 39: Búsqueda por palabra 61
Ilustración 40: Búsqueda por hashtag 62

ILUSTRACIÓN 41: BÚSQUEDA POR USUARIO 63
ILUSTRACIÓN 42: GUARDAR FICHERO 1 63
ILUSTRACIÓN 43: GUARDAR FICHERO 2 64