Tutorial cliente Twitter para Android

Paco Zamora Martínez (francisco.zamora@uch.ceu.es)

Grado en Ingeniería Informática en Sistemas de Información, Universidad CEU Cardenal Herrera, Enero 2015

Este tutorial va a ser una introducción a conceptos básicos de Android a través del desarrollo de un cliente muy básico para Twitter. No pretende ser un documento cerrado, y mucho menos una guía de diseño para aplicaciones basadas en Twitter, ya que el diseño de la aplicación requeriría mayor rigurosidad de la aquí expuesta. Esperamos que aun así os pueda servir para conocer algunos conceptos e ideas de Android y que os anime a utilizarlo en el futuro.

1. Instalación de Android Studio

Necesitaremos descargar Android Studio[1] e instalarlo en nuestro sistema. Esta aplicación dispone de todas las herramientas necesarias para empezar a programar aplicaciones para Android, excepto la instalación del JDK de Java, que es preciso que sea realizada **antes** de instalar Android Studio. Verificad que tenéis la versión 6.0 o más actual del JDK, o si no, decargadlo de aquí[2].

Seguid las instrucciones de la página de Android Studio para proceder a su instalación. El proceso puede ser ligeramente diferente dependiendo del sistema operativo y/o plataforma de vuestro ordenador, pero a grandes rasgos consiste en:

1. Desempaquetar el archivo que os habéis descargado.
2. Instalar y/o mover el binario.
3. Abrir Android Studio y seguir las instrucciones de la guía inicial.

1.1. Instalando paquetes para el SDK

Una vez abrimos Android Studio, la primera vez comenzará a descargar paquetes y utilidades que precisa instalar para su correcto funcionamiento. Una vez finalice este proceso nos aparecerá la pantalla que vemos en la figura 1.

Para este tutorial vamos a instalar soporte del SDK para la versión 4.0 de Android (API 14). Pincharemos en el botón donde pone **Configure** y después en **SDK Manager**. Se abrirá una nueva ventana donde podremos configurar el SDK. Esperar unos segundos a que la aplicación termine de iniciarse, y después señalar con el ratón las casillas donde pone:

- Android 4.0 (API 14) y señalamos todas las opciones para esa versión.
- Google Play Services.

Tendremos que esperar a que todos los paquetes se descarguen e instalen.

[1] http://developer.android.com/sdk/index.html
[2] http://www.oracle.com/technetwork/java/javase/downloads/index.html

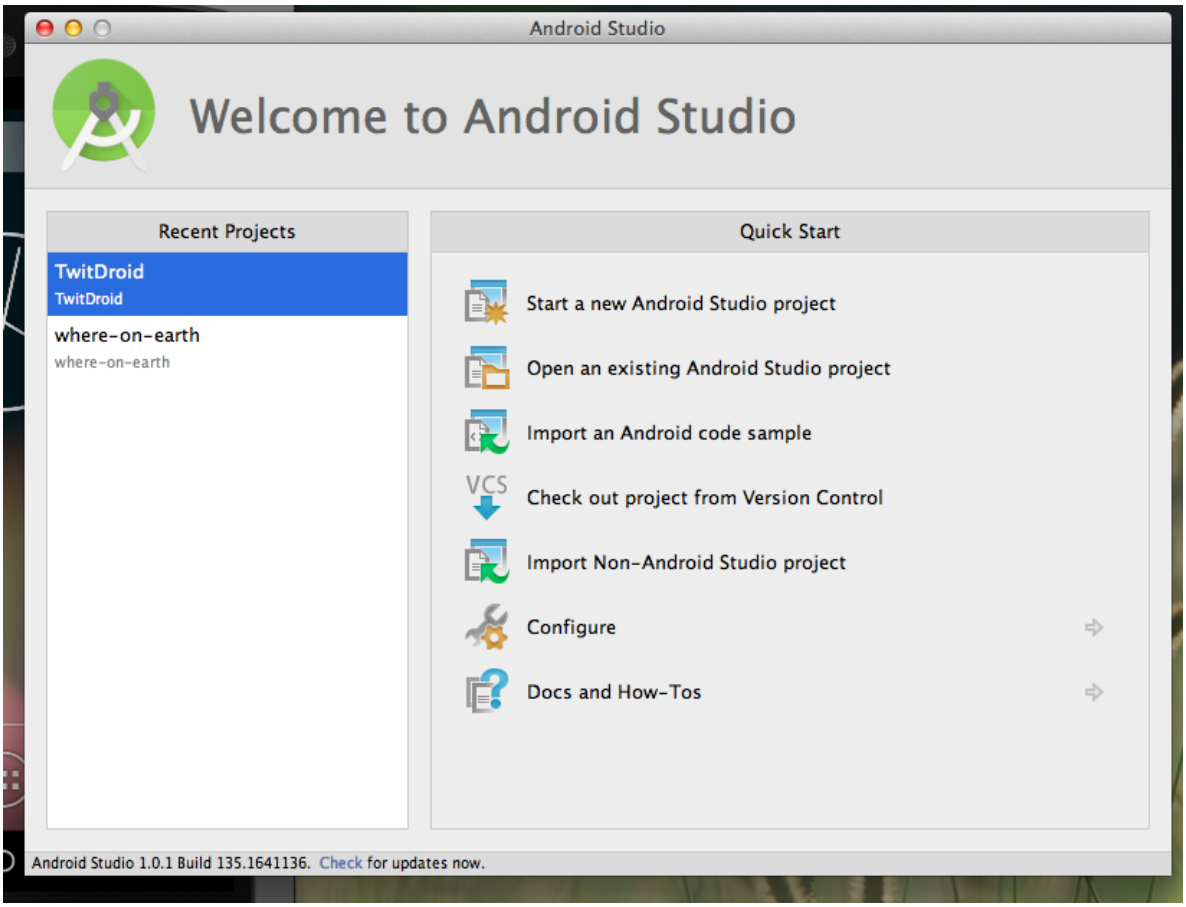

Figura 1: Estado inicial de Android Studio

2. Aplicación TwitDroid

2.1. Unas nociones básicas de Android

En Android, hay varios conceptos importantes que debemos conocer. El más importante es el concepto de **actividad** (*Activity* en inglés) y de **fragmento** (*Fragment* en inglés). Cada pantalla que tiene nuestra aplicación es una actividad. Por lo tanto, nuestra aplicación estará formada por actividades que definen las pantallas que vamos a tener disponibles, y estas pantallas estarán formados por *widgets*. Al mismo tiempo, un fragmento es un *trozo de una actividad*, lo que nos permite reutilizar esos trozos para componer actividades de mayor tamaño. Los fragmentos son introducidos en Android a partir de la versión 3.0, con la llegada de Android a las tabletas, debido a la pantalla de mayor tamaño disponible en las tabletas.

Otro aspecto importante de Android es el uso de XMLs para definir diferentes componentes estáticos de la aplicación, como pueden ser las vistas relacionadas con las actividades, menús de opciones, cadenas de texto, estilos, etc.

También es importante comprender la importancia del proceso de compilación, enlazado, ejecución e instalación de la aplicación. Dicho proceso involucra a varios comandos disponibles en el SDK, y está automatizado utilizando lo que se ha denominado **Gradle**. A veces dicho proceso puede ser lento, sobretodo la primera vez que lanzamos la aplicación.

La aplicación se ejecuta por defecto en una máquina virtual, normalmente configurada por defecto en el proceso de inicialización de Android Studio (la primera vez que lo arrancáis). La máquina virtual puede tener un inicio lento, con lo que intentaremos lanzarla lo antes posible para que así esté lista cuando la necesitemos.

2.2. Crear un proyecto nuevo

Volvemos a la ventana donde se abrió Android Studio y pinchamos en el botón **Start a new project**. Se abrirá un asistente con el aspecto de la figura 2. En la primera pantalla le damos estos valores:

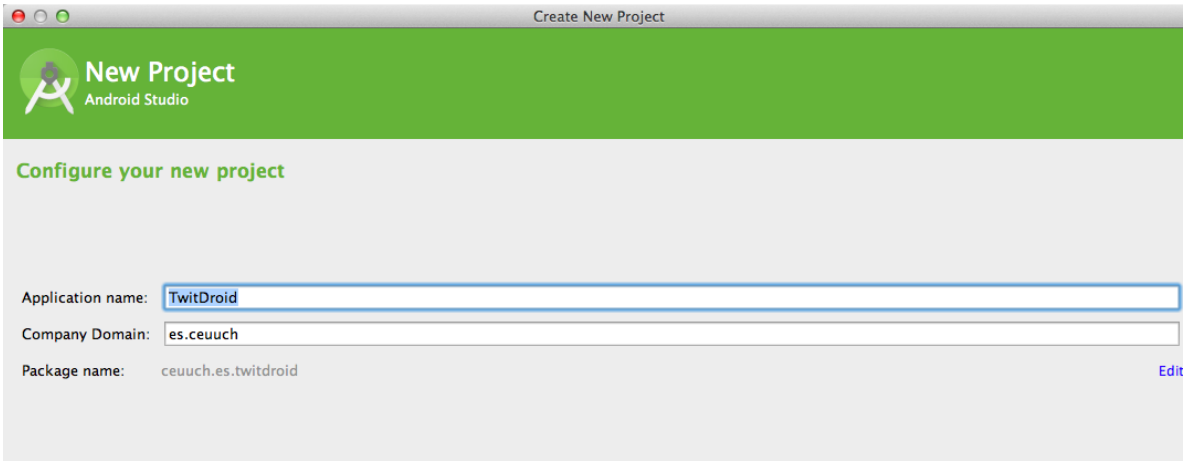

Figura 2: Nuevo proyecto 1

- Nombre aplicación: TwitDroid
- Compañía: es.ceuuch

Pinchamos en siguiente y en la segunda pantalla (véase figure 3) nos aseguramos que en la lista **Minimum SDK** ponga *API 14: Android 4.0 (IceCreamSandwich)*.

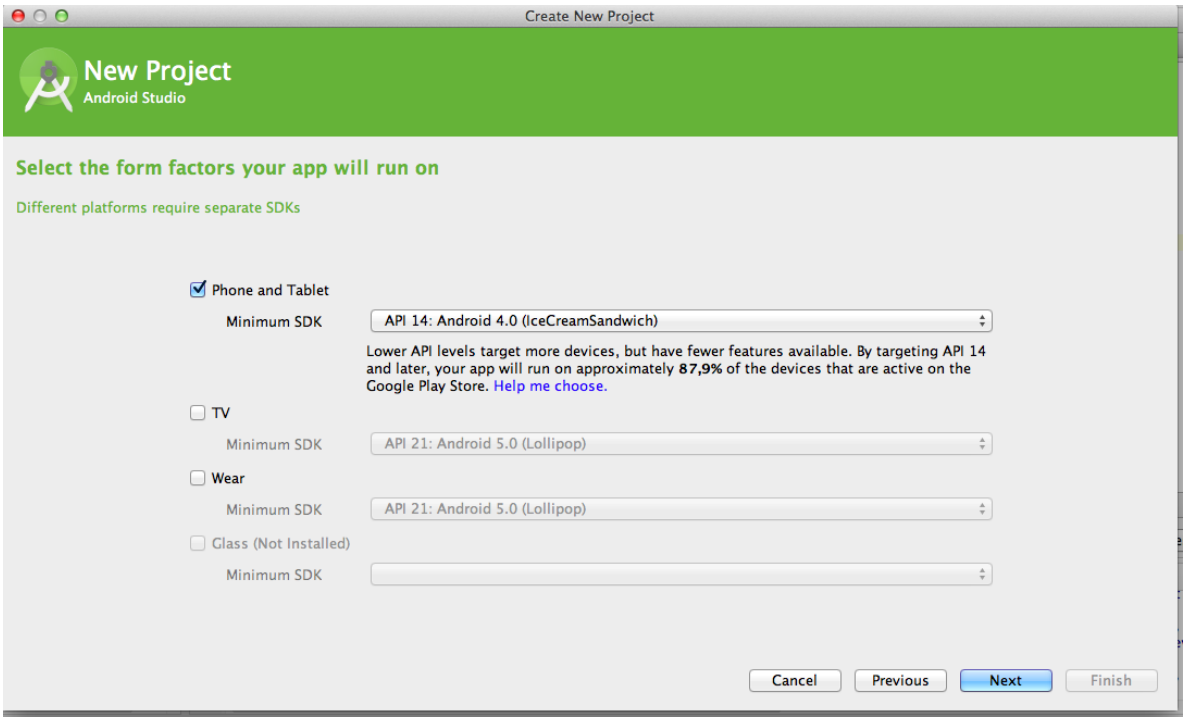

Figura 3: Nuevo proyecto 2

Pinchamos en el botón siguiente y llegamos a la figura 4. Seleccionamos la opción **Blank Activity** y pinchamos en siguiente.

Llegamos a la figure 5. Cambiamos el nombre de la actividad principal y le ponemos donde dice **Activity Name** el nombre *TimeLineActivity*. Dejamos el resto como esté y pinchamos en el botón **Finish**.

2.3. El interfaz de desarrollo de Android Studio

Una vez hemos creado o abierto un proyecto, llegamos al interfaz de desarrollo. El aspecto es similar al que podéis ver en la figura 6. A la izquierda tiene varias pestañas, nosotros vamos a trabajar con la que pone **1. Project**. Esa pestaña contiene la jerarquía de directorios de las aplicaciones de Android Studio. Resumidamente, esa jerarquía contiene:

- **app**: es la carpeta que contiene nuestra aplicación.
  - **manifests**: archivos XML donde se describe nuestra aplicación y los permisos que necesita para ser ejecutada en un dispositivo real.
  - **java**: contiene el código fuente. Normalmente está dividido en dos carpetas, la que contiene el paquete Java con nuestro código y la que contiene tests unitarios.
    - **ceuuch.es.twitdroid**: esta es la carpeta que contiene el código Java de nuestra aplicación.
    - **ceuuch.es.twitdroid (androidTest)**: contiene los tests unitarios. No vamos a utilizar esta carpeta.

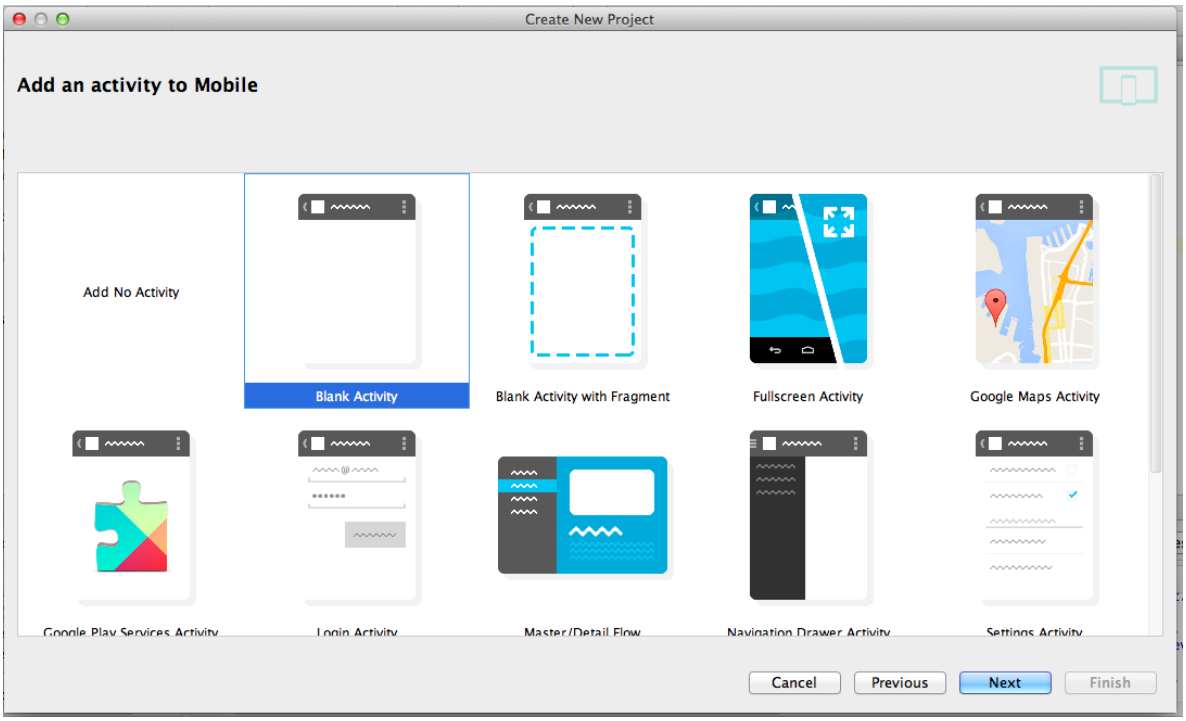

Figura 4: Nuevo proyecto 3

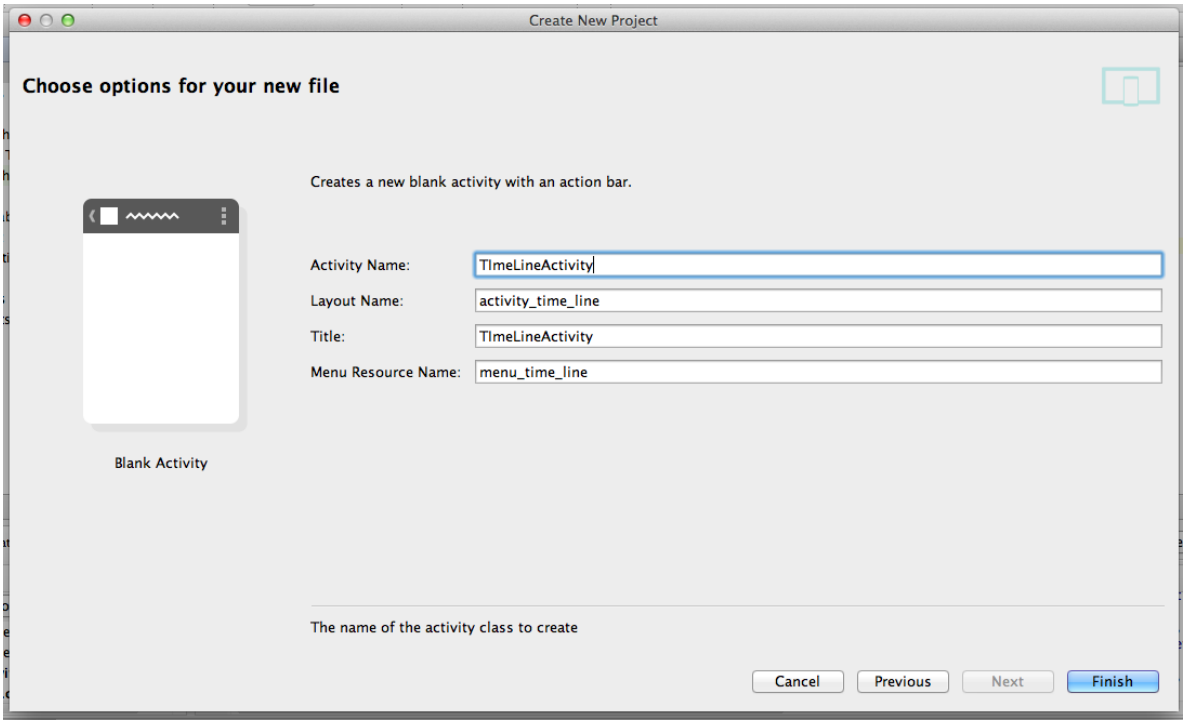

Figura 5: Nuevo proyecto 4

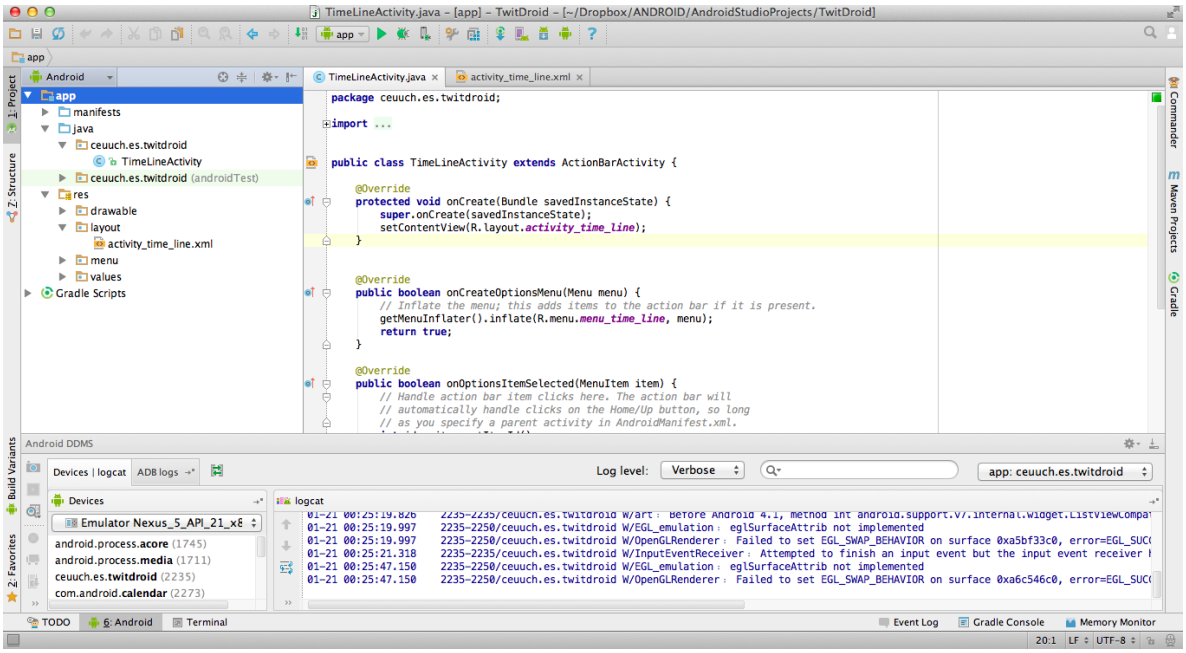

Figura 6: Interfaz de desarrollo

- **res**: contiene recursos de diferente tipo, como son el aspecto de nuestras actividades, cadenas de texto constantes, figuras y demás objetos multimedia, ...
  - **drawable**: contiene objetos y figuras que se pueden dibujar en pantalla.
  - **layout**: contiene ficheros XML con la descripción del aspecto de nuestras actividades.
  - **menus**: contiene ficheros XML con los menús desplegables de opciones que hay en cada actividad.
  - **values**: contiene constantes utilizadas en la interfaz (*layout*), son distintos archivos XML que contienen la definición de dichas constantes.

- **Gradle Scripts**: scripts necesarios para la compilación, enlazado, ejecución e instalación de nuestra aplicación.

En la mitad derecha de la pantalla tendremos código XML, Java, o el interfaz editable, dependiendo del tipo de archivo y el tipo de vista que escojamos utilizar. En Android todos los interfaces se pueden definir en XML o bien usar el editor de interfaces que a su vez modifica el XML. También es posible crear el interfaz programáticamente, este eso, escribiendo código que crea los widgets y los organiza jerárquicamente.

En la mitad inferior aparecerán detalles sobre la ejecución de la aplicación, la salida por consola (para depuración) y los errores de compilación y ejecución que se encuentren y/o produzcan.

En la parte superior, está la barra de botones, donde tenemos una zona muy importante y que voy a detallar aquí (véase figura 7) de izquierda a derecha:

1. **Make Project** permite construir la aplicación.
2. **Select run/debug** permite cambiar el objetivo de nuestra compilación.
3. **Run** permite ejecutar la aplicación, bien sea en la máquina virtual, bien en el dispositivo Android.
4. **Debug** permite ejecutar la aplicación para debug, permitiendo poner puntos de corte, ejecución paso a paso, etc.

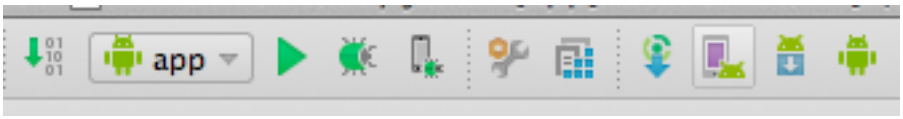

Figura 7: Fragmento de la barra de botones

5. **Attach debugger** permite enlazar el entorno de depuración con un dispositivo Android, pudiendo depurar la salida por consola de las aplicaciones que está ejecutando el dispositivo.
6. **Preferences** permite cambiar la configuración de Android Studio.
7. **Project Structure** nos permite cambiar la localización del proyecto en disco.
8. **Sync project** fuerza la sincronización entre el proyecto y los scripts de Gradle.
9. **AVD Manager** nos permite lanzar el gestor de máquinas virtuales.
10. **SDK Manager** lanza el gestor de paquetes del SDK.
11. **Monitor** permite monitorizar el dispositivo Android, para controlar el uso que hace la aplicación del dispositivo y poder optimizar así optimizarlo.

ATENCIÓN antes de continuar, vamos a pinchar en el botón **AVD Manager** y a lanzar la máquina virtual, para que esté lista en los siguientes pasos. Aprovecharemos también para cambiar la configuración de la máquina virtual, poniéndole 512MB de memoria RAM, haciendo que vaya más rápida la ejecución en PCs con poca memoria. A continuación solicitarle a la aplicación que lance la máquina virtual en el emulador.

2.4. Integración con Twitter

Para poder integrar nuestra aplicación con la API de Twitter, usaremos la biblioteca *twitter4j*. Para ello vamos a editar los scripts de Gradle, añadiendo una nueva dependencia. Abrimos `Gradle scripts/build.gradle (Module:app)` y añadimos las dependencias de twitter4j:

```
dependencies {
    compile fileTree(dir: "libs", include: ["*.jar"])
    compile "com.android.support:appcompat-v7:21.0.3"
    compile "org.twitter4j:twitter4j-core:3.0.5"
    compile "org.twitter4j:twitter4j-async:3.0.5"
}
```

Y los siguientes comandos dentro de `android { ... }` para evitar futuros problemas con Gradle:

```
android {
  ...
  packagingOptions {
      exclude "META-INF/LICENSE.txt"
  }
}
```

La dependencia con twitter4j será resuelta de manera automática por Gradle, utilizando el gestor de paquetes Java conocido como *Maven*. De esa manera, dichas dependencias se descargarán de internet sin necesidad de hacer nada más que especificarlas como hemos hecho.

Seguidamente modificamos el archivo `app/manifests/AndroidManifest.xml` y añadimos las siguientes permisos:

```
<!-- Permission - Internet Connect -->
<uses-permission android:name="android.permission.INTERNET" />
<!-- Network State Permissions -->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"
    />
```

Para poder usar la API de Twitter vamos a necesitar una *API key* y permiso explícito del usuario para acceder a la información de su cuenta. Para no complicarlo, vamos a utilizar nuestro propio usuario y poner explícitamente nuestro *token* de autenticación.

Vamos a ir a la sección *developers* de Twitter[3], buscamos el enlace donde dice *Manage your apps*, pinchamos en él, y creamos una nueva aplicación.

Una vez dentro de la aplicación de Twitter, modificamos los permisos para permitirle *read*, *write* y *direct messages*. Creamos un *Token de Acceso* y copiamos los cuatro campos: *Consumer Key*, *Consumer Secret*, *Access Token*, *Access Token Secret*. Estos 4 campos serán necesarios para poder configurar twitter4j y permitir el acceso a nuestra cuenta. Hay muchas formas diferentes de llevar a cabo este proceso, aquí vamos a crear una nueva clase que nos permitirá obtener el flujo de mensajes del usuario.

NOTA es importante cambiar los permisos antes de crear el token de acceso, en caso contrario el token no tendrá los permisos necesarios y deberá ser regenerado tras modificar los permisos a la aplicación.

2.4.1. Clase TwitterConnection

Vamos de nuevo a Android Studio y le decimos que queremos una nueva clase. Le damos como nombre **TwitterConnection**, y ponemos este código para la clase:

```
import twitter4j.AsyncTwitterFactory;
import twitter4j.AsyncTwitter;
import twitter4j.conf.ConfigurationBuilder;
import twitter4j.TwitterListener;

public class TwitterConnection {
  private static TwitterConnection instance;

  private TwitterConnection() {

  }

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

  private ConfigurationBuilder getConfBuilder() {
    ConfigurationBuilder cb = new ConfigurationBuilder();
    cb.setDebugEnabled(true)
      .setOAuthConsumerKey("AQUI LA CONSUMER KEY")
      .setOAuthConsumerSecret("AQUI EL CONSUMER SECRET")
```

[3] https://dev.twitter.com/

```
      .setOAuthAccessToken("AQUI EL ACCESS TOKEN")
      .setOAuthAccessTokenSecret("AQUI EL TOKEN SECRET");
    return cb;
  }

  public void getTimelineFeedInBackground(TwitterListener listener) {
    AsyncTwitterFactory factory = new
        AsyncTwitterFactory((getConfBuilder().build()));
    AsyncTwitter async = factory.getInstance();
    async.addListener(listener);
    async.getHomeTimeline();
  }
}
```

Esta clase sigue el patrón de diseño conocido como **Singleton**, esto es, una clase de la que *sólo* existe una única instancia en toda la aplicación. El método `getTimelineFeedInBackground()` nos permitirá acceder al *time line* de forma asíncrona (en segundo plano). Para poder hacer esto, habrá que pasarle un objeto Java que recibirá los datos en el momento en que estén listos (patrón de diseño *Listener*). Esta clase hace las veces de *callback*, y es otro patrón de uso común. Esta forma de trabajar permite que la aplicación siga ejecutándose de manera que la recepción de datos desde internet es realizada por otro hilo de ejecución.

2.4.2. Mostrando el *time line*

Para empezar, vamos a capturar el *time line* desde nuestra aplicación y a mostrarlo en la consola de Android, sin tocar todavía el interfaz de usuario de nuestra actividad. Para ello vamos a modificar el método `onCreate()` de la clase `TimeLineActivity` y a añadir un nuevo método privado `showTimeLine()`. El código que quedará de esta manera:

```
    private void showTimeLine(ResponseList<Status> statuses) {
        int id=0;
        for (Status st : statuses) {
            Log.d("TIME LINE " + String.valueOf(id), st.getText());
            ++id;
        }
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_time_line);
        TwitterConnection.getInstance().
          getTimeLineFeedInBackground(new TwitterAdapter(){
            @Override
            public void gotHomeTimeline(ResponseList<Status> statuses) {
                showTimeLine(statuses);
            }

            @Override
            public void onException(TwitterException e, TwitterMethod
                method) {
```

```
                Log.e("TwitterException",
                        Integer.toString(e.getErrorCode()) + " :: " +
                        e.getErrorMessage());
            }

        });
    }
```

Este código se encarga de registrar un *Listener* (en este caso, *TwitterAdapter*), reimplementando (`@Override`) los métodos `gotHomeTimeline()` y `onException()`. El método `gotHomeTimeline()` recibe una lista de twits (clase `Status`) y llama al método `showTimeLine()` pasándole dicha lista. El método `onException()` es necesario para poder capturar problemas de conexión durante el proceso de descarga del time line. Dicho método simplemente muestra por la consola el mensaje de error de dicha excepción.

2.4.3. Actualizando la fecha

IMPORTANTE, una vez llegados a este punto, y antes de ejecutar el ejemplo, debemos asegurarnos de que la fecha de la máquina virtual esté actualizada. Simplemente vamos a los **ajustes del dispositivo** virtual, como si de un teléfono real se tratara, y cambiamos la fecha para que sea la misma que la de nuestros ordenadores. Este proceso se puede llevar a cabo también a través de la **consola o terminal**, aunque para esto necesitáis conocer las herramientas de Android. No obstante, aquí os dejo como hacerlo desde la consola.

- Linux/MacOS

```
adb shell date -s $(date +"%Y%m%d.%H%M%S")
```

- Windows

```
adb shell date -s $(get-date -format yyyyMMdd.HHmmss)
```

Otra posibilidad es no utilizar el emulador y ejecutar el programa **en un móvil real**.

2.4.4. Ejecutando el ejemplo

Vamos a ver nuestro time line en funcionamiento, simplemente le damos al botón de ejecutar en la barra superior y esperamos a que la aplicación arranque en el dispositivo donde queramos ejecutarla. En la pantalla de Android Studio saldrá algo similar a la figura 8.

2.5. Mostrando el time line en pantalla

Vamos a modificar el interfaz de TimeLineActivity para que muestre un widget de progreso y un ListView. Abrimos el fichero XML relacionado con TimeLineActivity, para ello usamos el gestor del proyecto (lado izquierdo de la pantalla, pestaña **1. Project**) y vamos al fichero `app/res/layout/activity_time_line.xml`. Normalmente este archivo viene abierto cuando se crea el proyecto, pero es preciso esperar a que esté todo cargado y disponible en pantalla. Este proceso abrirá la pantalla de edición de interfaces, como se ve en la figura 9. Lo primero que haremos es añadir un contenedor de tipo **ListView**, que ocupe toda la pantalla. Encima de este contenedor añadimos un

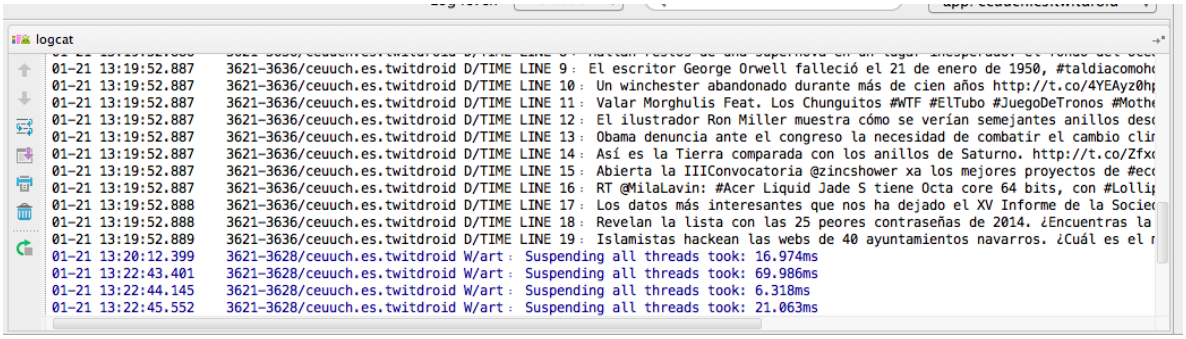

Figura 8: Time line en Android Studio

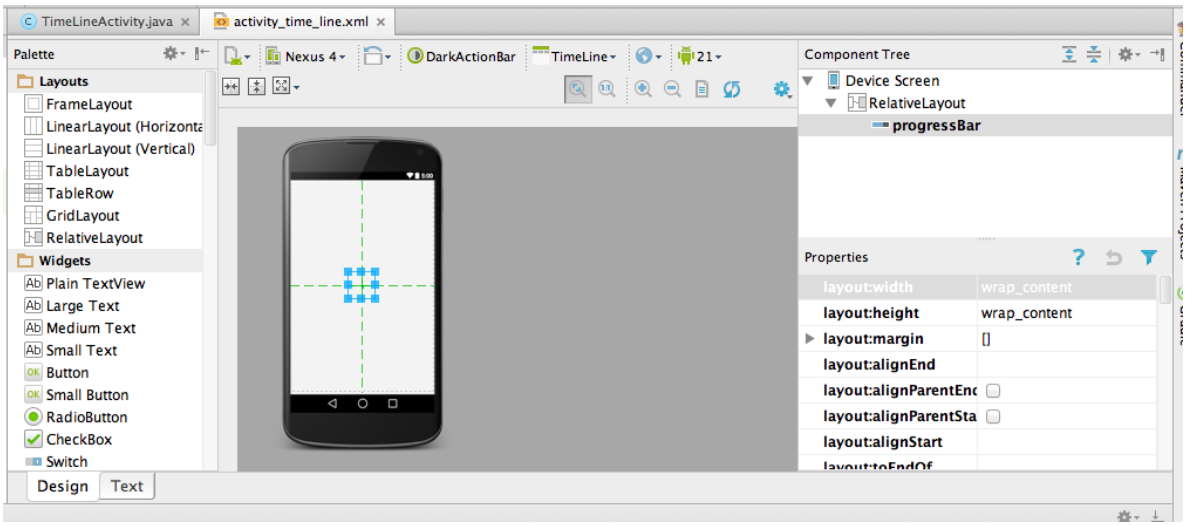

Figura 9: Editor de interfaces

widget de tipo **ProgresBar (Large)**, y lo ponemos centrado en pantalla. Le damos al botón ejecutar y veremos el resultado y además el time line en la consola de Android Studio.

Ahora vamos a modificar el método `showTimeLine()` para que, además de mostrar los twits en la consola, los concatene en el ListView, permitiéndole al usuario verlos en pantalla. Cada twit vamos a convertirlo a un string de texto, tal y como ya hicimos en la consola. Además, controlaremos la barra de progreso para hacerla desaparecer cuando los datos hayan sido descargados.

```
    private void showTimeLine(ResponseList<Status> statuses) {
        final ListView list_view = (ListView)
           findViewById(R.id.listView);
        final ArrayList<String> array = new ArrayList<String>();

        // Recorremos todos los twits
        int id=0;
        for (Status st : statuses) {
            Log.d("TIME LINE " + String.valueOf(id), st.getText());
            ++id;
            array.add(st.getText());
        }
        final Context context = this;
        // Usamos runOnUiThread porque necesitamos hacer cosas en la UI y
        // este metodo esta ejecutandose en otro hilo diferente.
        this.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                // Damos el adapter a la vista
                final ArrayAdapter adapter = new ArrayAdapter(context,
                  // Usamos un layout por defecto de android.
                  android.R.layout.simple_list_item_1, array);
                list_view.setAdapter(adapter);

                // Ocultamos la barra de progreso
                View progressBar = findViewById(R.id.progressBar);
                progressBar.setVisibility(View.GONE);
            }
        });
    }
```

2.6. Tweet, retweet y responder

En esta sección vamos a añadir la funcionalidad de tweet, retweet y responder. Con estas opciones aprenderemos a transitar de una actividad a otra, y a enviar información a Twitter.

2.6.1. Tweet

Primero modificaremos TimeLineActivity para añadirle un botón *Tweet* en la parte superior de la pantalla. Algo como lo que se ve en la figura 10.

Seguidamente crearemos una nueva actividad para nuestra aplicación. La llamaremos **TweetActivity** y seleccionaremos el tipo *Blank Activity*. Le añadiremos un campo texto simple, un campo texto editable

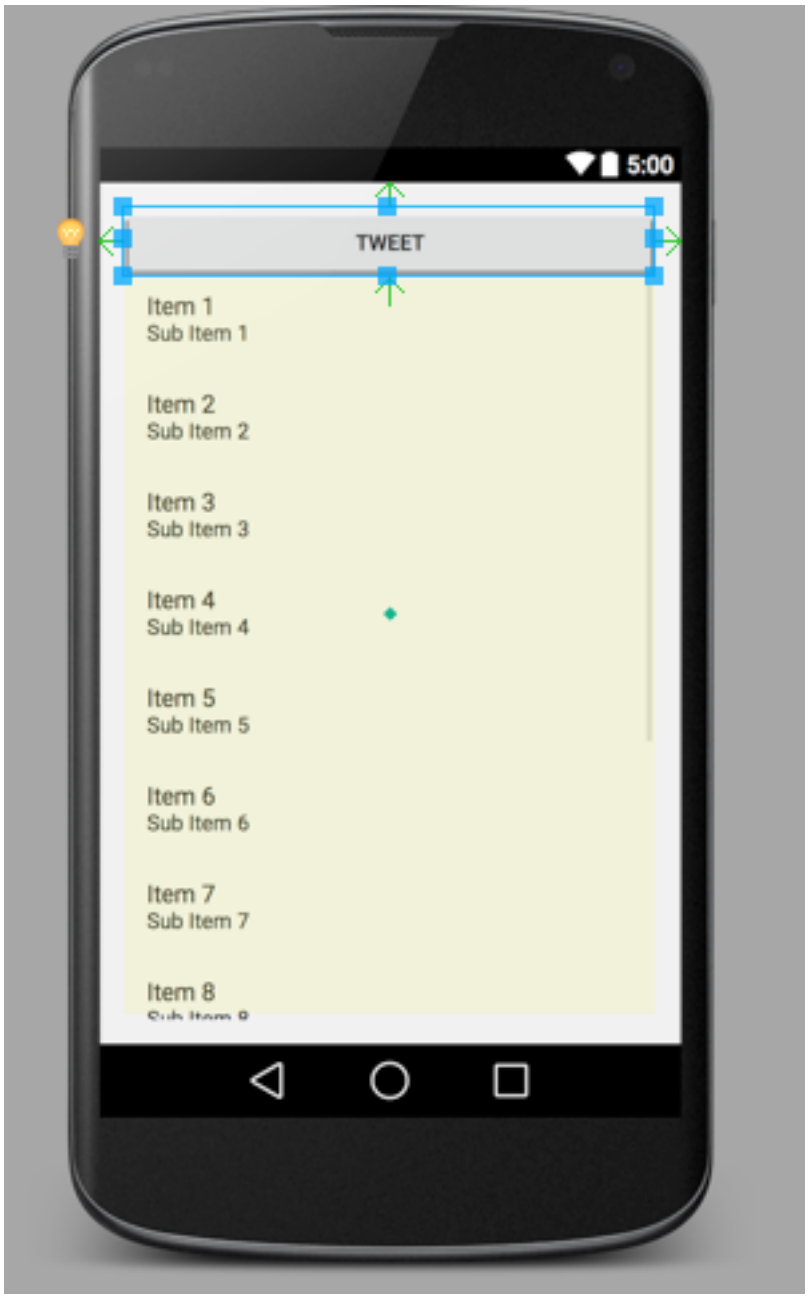

Figura 10: Botón tweet

de múltiples líneas y un botón enviar. Al botón le cambiaremos el campo *id*, para que sea `sendButton`. Debe quedar algo similar a la figura 11.

Para enlazar la TimeLineActivity con TweetActivity, iremos al *layout* de la primera, y asignaremos al campo `onClick` del botón un evento de la clase TimeLineActivity. Escribiremos en dicho campo `onTweetClick` (véase figura 12).

Añadiremos a la clase **TimeLineActivity** el método `onTweetClick()`, que tendrá este código:

```
    public void onTweetClick(View view) {
        Intent intent = new Intent(this, TweetActivity.class);
        startActivity(intent);
    }
```

Ahora ampliaremos la funcionalidad de nuestra clase **TwitterConnection**, añadiéndole un método `sendTweetInBackground()`, que recibe un String y ejecute una llamada asíncrona para actualizar nuestro estado (Tweet). La operación es similar a la que ya hicimos en el método `getTimeLineFeedInBackGround()`

```
    public void sendTweetInBackground(String text, TwitterListener
       listener) {
        // Crea una instancia de twitter
        AsyncTwitterFactory factory = new
          AsyncTwitterFactory((getConfBuilder().build()));
        AsyncTwitter async = factory.getInstance();
        async.addListener(listener);
        async.updateStatus(text);
    }
```

Ahora necesitamos actualizar la clase **TweetActivity** para que el botón esté asociado al método `onSendClick()` y dicho método nos permita enviar el tweet y capturar errores en caso de producirse.

```
    public void onSendClick(View view) {
        EditText edit_text = (EditText) findViewById(R.id.editText);
        String tweet_text = edit_text.getText().toString();
        TwitterConnection.getInstance().sendTweetInBackground(tweet_text,
                new TwitterAdapter() {
                    @Override
                    public void updatedStatus(Status status) {
                        onUpdatedStatus();
                    }

                    @Override
                    public void onException(TwitterException e,
                       TwitterMethod method) {
                        onStatusException(e);
                    }
                });
    }
```

El método `onSendClick()` utiliza dos auxiliares, que son:

```
    private void onUpdatedStatus() {
        final Activity context = this;
```

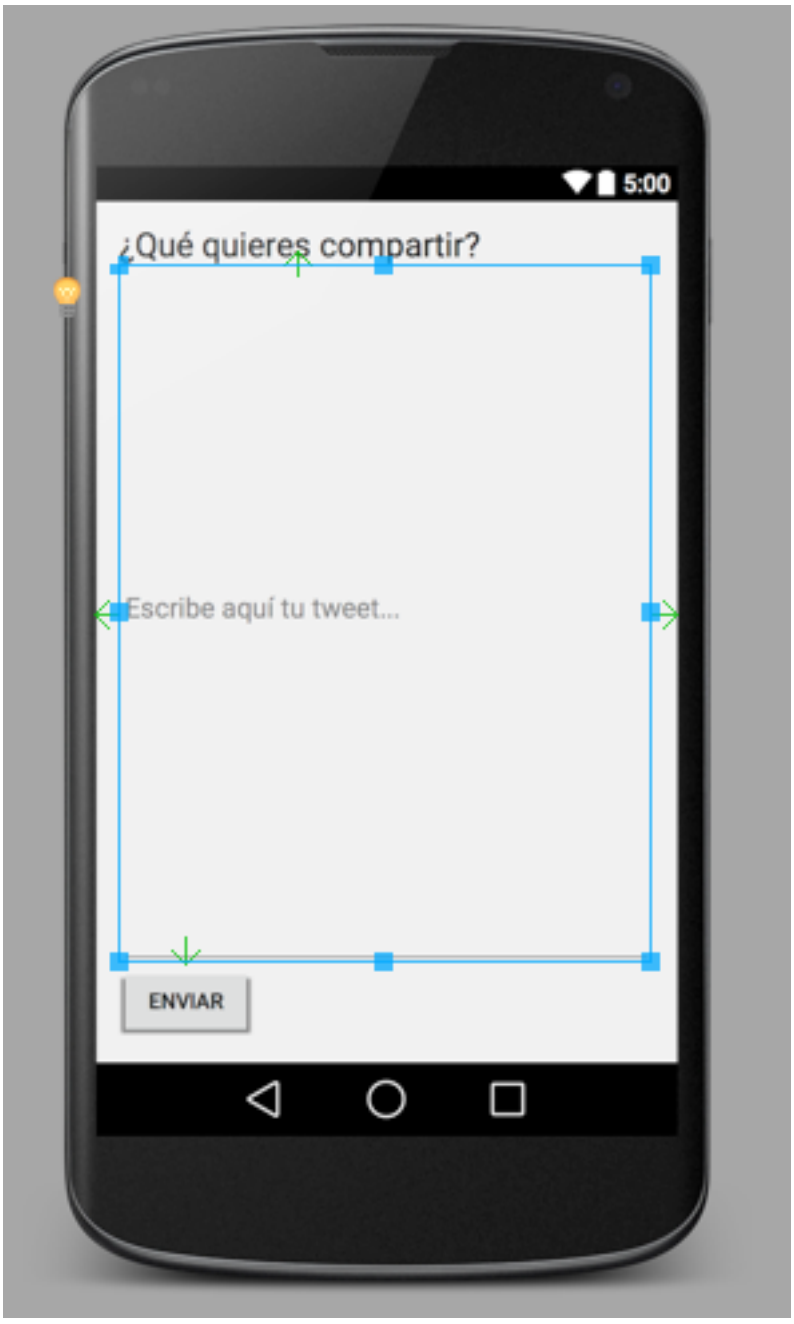

Figura 11: TweetActivity

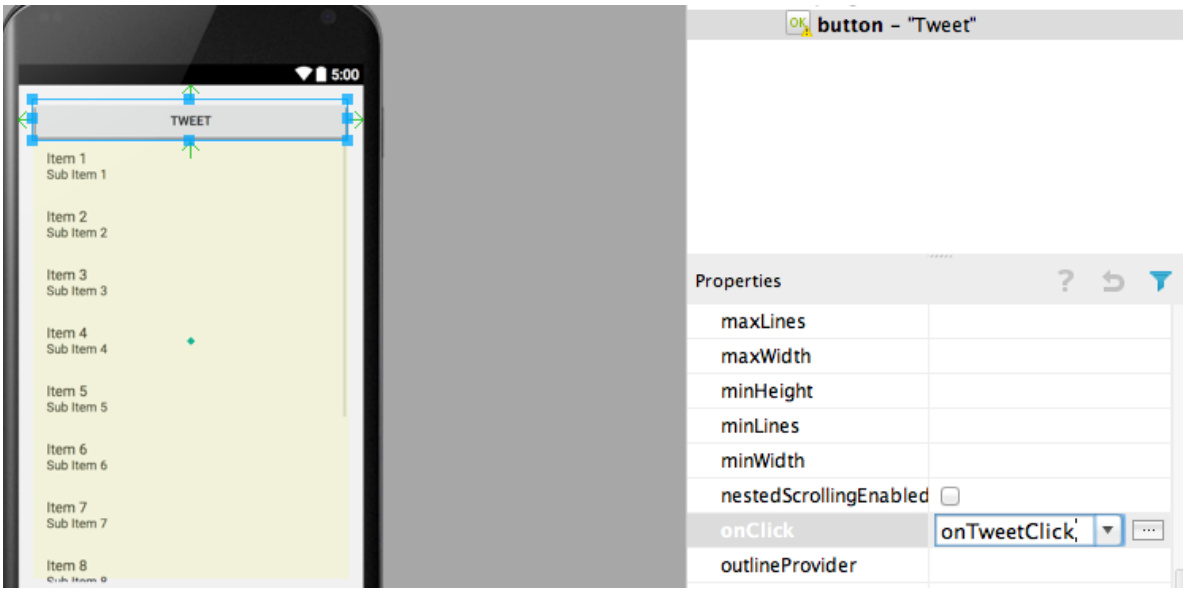

Figura 12: Campo onClock

```
        this.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                Toast.makeText(context, "Tweet publicado",
                        Toast.LENGTH_SHORT).show();
                context.finish();
            }
        });
    }

    private void onStatusException(TwitterException e) {
        final Activity context = this;
        Log.e("TwitterException",
                Integer.toString(e.getErrorCode()) + " :: " +
                   e.getErrorMessage());
        this.runOnUiThread(new Runnable() {
            @Override
            public void run() {
                Toast.makeText(context, "Imposible enviar tweet :"(",
                        Toast.LENGTH_SHORT).show();

            }
        });
    }
```

Hemos utilizado en los métodos auxiliares el constructor `Toast.makeText()` de Android, que nos permite mostrar mensajes en pantalla al usuario, así le informamos de que el tweet ha sido publicado, o bien de que se ha producido un error y no se ha podido publicar.

2.6.2. Vista detalle de un tweet

Para implementar las opciones retweet y responder vamos a crear una nueva actividad, **DetailedTweetActivaty** que reciba una tweet y nos permita mostrar el nick del usuario, el número de retweets, el número de favoritos, y además incluya botones para hacer *rewteet* y *responder*.

Creamos la activity **DetailedTweetActivity**, como siempre de tipo *BlankActivity*. La vista contendrá estos widgets (véase figura 13):

- *LargeText* con `id=nick`
- *LargeText* con `id=message`
- *MediumText* con `text=Favoritos`
- *MediumText* con `text=Retweets`
- *MediumText* con `text=0` e `id=favourites`
- *MediumText* con `text=0` e `id=retweets`
- *Button* con `id=retweetButton`
- *Button* con `id=replyButton`

La actividad **DetailedTweetActivity** necesita recibir parámetros en el Intent que la ejecuta. Concretamente necesita el nombre del usuario (NICK), el mensaje (MESSAGE), la cuenta de favoritos (FAVOURITES), la cuenta de retweets (RETWEETS) y el ID del tweet (TWEETID). Para ello, modificaremos el método `onCreate()`, que cogerá el Intent recibido y despempaquetará de allí la información que necesita. Primero añadimos unas constantes estáticas que nos serán útiles, junto a una propiedad `tweet_id` necesaria para hacer funcionar los botones.

```
    private long tweet_id;
    public final static String NICK = "NICK";
    public final static String MESSAGE = "MESSAGE";
    public final static String FAVOURITES = "FAVOURITES";
    public final static String RETWEETS = "RETWEETS";
    public final static String TWEETID = "TWEETID";
```

Y posteriormente modificamos el método `onCreate()`.

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_detailed_tweet);
        Intent intent = getIntent();
        TextView nick = (TextView)findViewById(R.id.nick);
        TextView message = (TextView)findViewById(R.id.message);
        TextView favourites = (TextView)findViewById(R.id.favourites);
        TextView retweets = (TextView)findViewById(R.id.retweets);
        nick.setText("@" + intent.getStringExtra(NICK));
        message.setText(intent.getStringExtra(MESSAGE));
        favourites.setText(String.valueOf(intent.getIntExtra(FAVOURITES,
            0)));
        retweets.setText(String.valueOf(intent.getIntExtra(RETWEETS,
            0)));
        tweet_id = intent.getLongExtra(TWEETID, -1);
        if (tweet_id < 0) {
```

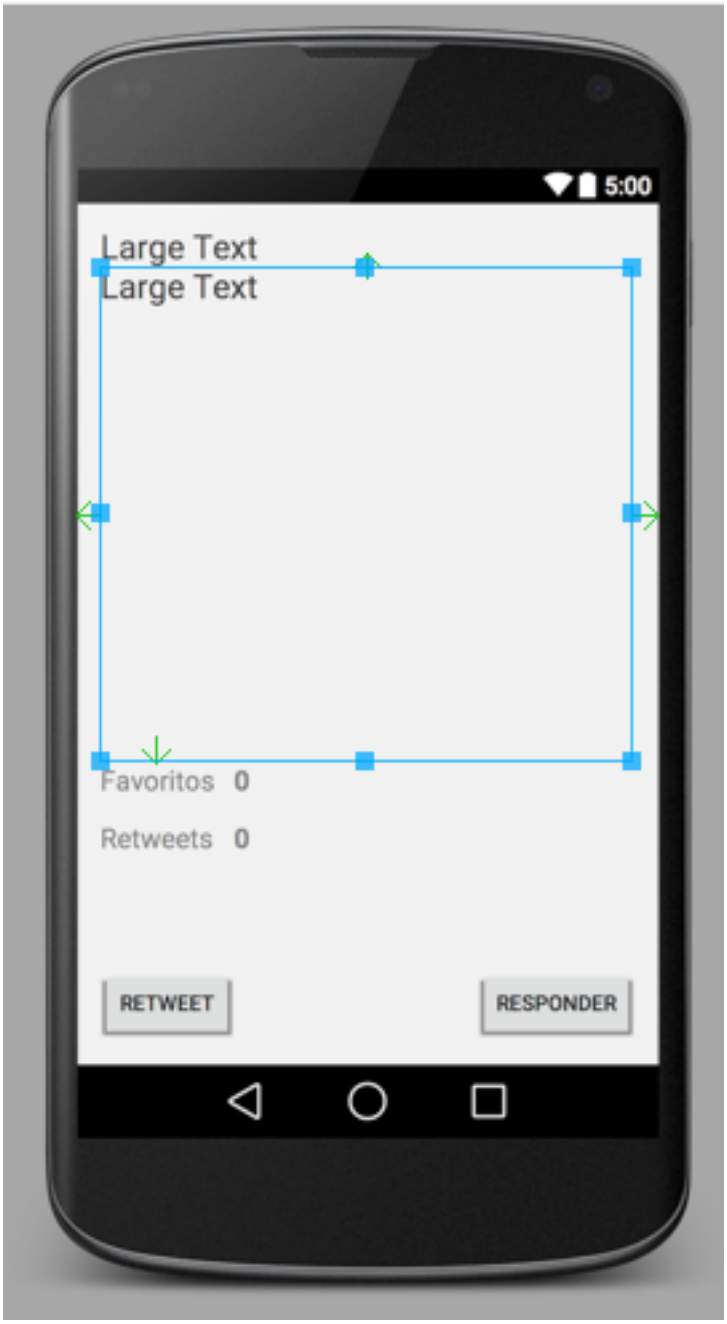

Figura 13: Vista detalle de un tweet

```
            Toast.makeText(this, "Imposible abrir vista detalle",
                Toast.LENGTH_SHORT);
            finish();
        }
    }
```

Para poder hacer referencia a los tweets y recuperar sus datos, vamos a refactorizar el código de **TimeLineActivity**, haciendo que el objeto `statuses` se guarde en un campo del objeto. También necesitaremos que la lista de tweets sea sensible y pueda pulsarse.

Procedemos de la siguiente manera, añadimos una propiedad privada a la clase **TimeLineActivity**, con la lista de tweets:

```
public class TimeLineActivity extends ActionBarActivity {

    private ResponseList<Status> time_line;
```

Inicializamos la variable `time_line = null` en el método `onCreate()`. Asignamos a la variable `time_line = statuses` cuando recibimos la respuesta de twitter.

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_time_line);
        final Context context = this;

        time_line = null;
```

```
    private void showTimeLine(ResponseList<Status> statuses) {
        time_line = statuses;
```

Añadimos a la clase una nueva propiedad que se encargará de seleccionar el tweet y lanzar la actividad detalle.

```
    private AdapterView.OnItemClickListener item_listener = new
        AdapterView.OnItemClickListener() {
        @Override
        public void onItemClick(AdapterView parent, View view, int
            position, long id) {
            Status st = time_line.get(position);
            Intent intent = new Intent(getBaseContext(),
                DetailedTweetActivity.class);
            // Le pasamos los datos al intent
            intent.putExtra(DetailedTweetActivity.NICK,
                st.getUser().getScreenName());
            intent.putExtra(DetailedTweetActivity.MESSAGE, st.getText());
            intent.putExtra(DetailedTweetActivity.RETWEETS,
                st.getRetweetCount());
            intent.putExtra(DetailedTweetActivity.FAVOURITES,
                st.getFavoriteCount());
            intent.putExtra(DetailedTweetActivity.TWEETID, st.getId());
            // y lo ejecutamos
            startActivity(intent);
```

```
        }
    };
```

Ahora necesitamos que la lista de tweets sea sensible. Para eso añadimos en el método **showTimeLine()**, justo cuando asignamos el `adapter` como a la lista, y añadismo a la lista un objeto tipo Listener que nos permite saber cuando se ha pulsado un tweet.

```
                final ArrayAdapter adapter = new ArrayAdapter(context,
                  android.R.layout.simple_list_item_1, array);
                list_view.setAdapter(adapter);
                list_view.setOnItemClickListener(item_listener);
```

2.6.3. Retweet

De manera similar a como ya hicimos antes, vamos a modificar la clase **TwitterConnection** para que tenga un métohod **retweetInBackground()**. En este caso, recibirá una variable de tipo `long` y un `Listener`, y tendrá este aspecto:

```
    public void retweetInBackground(Long id, TwitterListener listener) {
        AsyncTwitterFactory factory = new
            AsyncTwitterFactory((getConfBuilder().build()));
        AsyncTwitter async = factory.getInstance();
        async.addListener(listener);
        async.retweetStatus(id);
    }
```

Desde la clase **DetailedTweetActivity** modificaremos el botón `retweetButton` para que su campo `onClick` haga referencia a un nuevo método de la clase, que llamaremos `onRetweetClick()`. El método quedará así:

```
    public void onRetweetClick(View view) {
        final Activity context = this;
        TwitterConnection.getInstance().retweetInBackground(tweet_id,
                new TwitterAdapter() {
                    @Override
                    public void retweetedStatus(Status status) {
                        context.runOnUiThread(new Runnable() {
                            @Override
                            public void run() {
                                Toast.makeText(context, "Retweet
                                    conseguido",
                                        Toast.LENGTH_SHORT).show();
                            }
                        });
                    }

                    @Override
                    public void onException(TwitterException e,
                        TwitterMethod method) {
                        context.runOnUiThread(new Runnable() {
                            @Override
```

```
                            public void run() {
                                Toast.makeText(context, "Retweet
                                    fallido",
                                        Toast.LENGTH_SHORT).show();
                            }
                        });
                        Log.e("TwitterException",
                                String.valueOf(e.getErrorCode()) + " " +
                                    e.getErrorMessage());
                    }
                });
    }
```

2.6.4. Responder

Añadimos otro método a nuestro **TwitterConnection**, método **replyInBackgroud()**:

```
    public void replyInBackground(String message, Long id,
        TwitterListener listener) {
        AsyncTwitterFactory factory = new
            AsyncTwitterFactory((getConfBuilder().build()));
        AsyncTwitter async = factory.getInstance();
        async.addListener(listener);
        async.updateStatus(new
            StatusUpdate(message).inReplyToStatusId(id));
        async.retweetStatus(id);
    }
```

Para responder procederemos a otra refactorización del código. Vamos a generalizar la actividad **TweetActivity** para que permita recibir un `tweet_id` y un `nick`, de manera que si los recibe, actue para implementar una respuesta.

Ëntonces, añadimos dos propiedades:

```
public class TweetActivity extends ActionBarActivity {

    private long tweet_id;
    private String nick;
```

Y modificamos el método `onCreate()` para que quede así:

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_tweet);
        Intent intent = getIntent();
        tweet_id = intent.getLongExtra(DetailedTweetActivity.TWEETID,
            -1);
        nick = intent.getStringExtra(DetailedTweetActivity.NICK);
        if (tweet_id >= 0) {
            EditText message = (EditText)findViewById(R.id.editText);
            message.setText(nick + " ");
```

```
        }
    }
```

El método `onSendClick()` tendrá en cuenta los dos casos posibles, tweet y responder.

```
    public void onSendClick(View view) {
        EditText edit_text = (EditText) findViewById(R.id.editText);
        String tweet_text = edit_text.getText().toString();
        if (tweet_id < 0) {
            TwitterConnection.getInstance().sendTweetInBackground(tweet_text,
                    new TwitterAdapter() {
                        @Override
                        public void updatedStatus(Status status) {
                            onUpdatedStatus();
                        }

                        @Override
                        public void onException(TwitterException e,
                            TwitterMethod method) {
                            onStatusException(e);
                        }
                    });
        }
        else {
            TwitterConnection.getInstance().replyInBackground(tweet_text,
               tweet_id,
                    new TwitterAdapter() {
                        @Override
                        public void updatedStatus(Status status) {
                            onUpdatedStatus();
                        }

                        @Override
                        public void onException(TwitterException e,
                            TwitterMethod method) {
                            onStatusException(e);
                        }
                    });
        }
    }
```

Desde la actividad **DetailedTweetActivity**, añadiremos un método relacionado con el botón `replyButton`, de tal forma que dicho método ejecutará la actividad **TweetActivity** pasándole en el Intent la información del tweet id y el nick del usuario.

```
    public void onReplyClick(View view) {
        TextView nick = (TextView)findViewById(R.id.nick);
        Intent intent = new Intent(this, TweetActivity.class);
        intent.putExtra(DetailedTweetActivity.TWEETID, tweet_id);
        intent.putExtra(DetailedTweetActivity.NICK,
           nick.getText().toString());
        startActivity(intent);
```

```
    }
```

3. Posibles mejoras

¡Muchísimas! Esta aplicación es sólo un esbozo de lo que se puede hacer con Android Studio (basicamente todo lo que queráis, depende del tiempo que le dediquéis). La aplicación TwitDroid puede ser mejorada para que muestre las imágenes que la gente adjunta al tweet, para que muestre los nicks en el listado de tweets, que permita reducir el consumo de datos usando una pequeña caché con los últimos tweets, mejorar el estilo de las vistas utilizadas, ...

Os animamos a que le dediquéis un rato, intentando introducir las mejoras que se os ocurran. O incluso que rediseñéis la aplicación para hacerla acorde a vuestras necesidades.

3.1. Control de los botones y barras de progreso

Cuando se pulsa el botón de *Enviar* en la **TweetActivity**, se puede ocultar el botón, desactivar el texto del mensaje y mostrar una barra de progreso mientras el tweet es enviado. Sólo hay que poner las operaciones siguientes en el orden adecuado y en el lugar adecuado.

```
Button button = (Button) findViewById(R.id.sendButton);
ProgressBar progress = (ProgressBar) findViewById(R.id.sendProgress);
EditText edit_text = (EditText) findViewById(R.id.editText);
button.setVisibility(View.INVISIBLE);
progress.setVisibility(View.VISIBLE);
edit_text.setEnabled(false);
edit_text.setFocusable(false);
```

NOTA fijáos que la visibilidad del botón es `View.INVISIBLE`. Esto es importante ya que si se asigna a `View.GONE` es posible que la interfaz tenga problemas para localizar los widgets en pantalla.

En caso de que falle el envió del mensaje, hay que restaurar el estado anterior.

```
Button button = (Button) findViewById(R.id.sendButton);
ProgressBar progress = (ProgressBar) findViewById(R.id.sendProgress);
EditText edit_text = (EditText) findViewById(R.id.editText);
button.setVisibility(View.VISIBLE);
progress.setVisibility(View.GONE);
edit_text.setEnabled(true);
edit_text.setFocusable(true);
```

Lo mismo se puede hacer para la clase **DetailedTweetActivity**, eliminando el botón de *Tweet* una vez se haya producido el mismo.

```
ProgressBar progress = (ProgressBar) findViewById(R.id.retweetProgress);
Button button = (Button) findViewById(R.id.retweetButton);
progress.setVisibility(View.VISIBLE);
button.setVisibility(View.INVISIBLE);
```

En caso de que el tweet se realice correctamente, hay que eliminar la barra de progreso para evitar que se quede para siempre allí.

```
Button button = (Button) findViewById(R.id.retweetButton);
progress.setVisibility(View.GONE);
```

Y recuperar el estado en caso de error.

```
ProgressBar progress = (ProgressBar) findViewById(R.id.retweetProgress);
Button button = (Button) findViewById(R.id.retweetButton);
progress.setVisibility(View.GONE);
button.setVisibility(View.VISIBLE);
```

3.2. Recuperar el estado de retweet

Cuando se lanza la actividad **DetailedTweetActivity** se le puede pasar un parámetro que indique el estado de retweet de ese mensaje. Pondríamos algo así en el método **OnItemClickListener()** de la clase **TimeLineActivity**.

```
intent.putExtra(DetailedTweetActivity.RETWEETED, st.isRetweetedByMe());
```

Y recuperar dicho parámetro en **DetailedTweetActivity**, método **onCreate()**. Guardamos el valor en una nueva propiedad privada, de tipo `boolean`. Adicionalmente, eliminamos el botón de retweet en caso de que dicho parámetro sea cierto.

```
retweeted = intent.getBooleanExtra(RETWEETED, false);
if (retweeted) {
    Button retweet = (Button) findViewById(R.id.retweetButton);
    retweet.setVisibility(View.INVISIBLE);
}
```

Y en caso de que se pulse el botón de retweet, hay que forzar el valor de la propiedad.

```
retweeted = true;
```

3.3. Salvar el estado de la aplicación

La actividad **DetailedTweetActivity** tiene varias propiedades cuyo contenido se pierde cuando la actividad es destruida y vuelta a construir, por ejemplo, cuando se gira el dispositivo y cambia la orientación de la pantalla. Para evitar ese problema, podemos guardar el valor de dichas propiedades en el método **onSaveInstanceState(Bundle out_state)**.

```
    @Override
    public void onSaveInstanceState(Bundle out_state) {
        super.onSaveInstanceState(out_state);
        out_state.putBoolean(RETWEETED, retweeted);
    }
```

En el método **onCreate()**, la actividad recibe un *Bundle* que puede usarse para recuperar el estado guardado anteriormente. El estado de la propiedad `retweeted` será recuperado de dicho *Bundle* en caso de existir, o del *Intent* en caso contrario como ya hicimos antes.

```
        if (savedInstanceState != null) {
            retweeted = savedInstanceState.getBoolean(RETWEETED);
        }
        else {
            retweeted = intent.getBooleanExtra(RETWEETED, false);
        }
```

3.4. Añadir botón de *refresh* a la TimeLineActivity

Añadimos un botón que nos permita **refrescar** la lista de tweets, descargándose los 20 últimos. Para esto bastaría con refactorizar el código en un nuevo método `getTimeLine()` que sería así. Dicho método sería ejecutado desde el método `onCreate()`, y adicionalmente desde el método `onResume()` para permitir recuperar los tweets cada vez que la actividad es revivida. No mostramos el código de esta propuesta, quedaría como ejercicio. Cuidado con esta modificación, si superáis un número máximo de peticiones a Twitter, este os impedirá recibir el time line. Es posible reducir el impacto de esto simplemente usando un contador y sólo permitiendo el refresco cada 30 segundos, por ejemplo.

3.5. Caché de tweets

Una mejora muy interesante, de cara a reducir el ancho de anda y mejorar la respuesta de la aplicación, sería utiliar una lista con caché para guardar los tweets. Esta caché puede ser implementada mediante una base de datos en *SQLite* o cualquier otra capa de persistencia en el dispositivo. Esta mejora permitiría evitar superar la frecuencia de peticiones que tiene Twitter, y que comentábamos en la propuesta anterior. Esta propuesta de mejora requiere un rediseño de la aplicación teniendo en cuenta que la lista de tweets ha de ser un objeto con su estado propio, que se puede actualizar pidiéndo tweets nuevos, evitando los que ya han sido descargados, actualizando aquellos campos como número de retweets, y mucha más casuística que es necesaria ser tenida en cuenta.

3.6. Login con usuario y contraseña

Por supuesto, la mejora más importante sería añadir una actividad de login que solicite al usuario nombre y contraseña. Dicha actividad obtendría el token de acceso a partir de dicha información, y permitiría salvar de forma segura la contraseña para futuros usos de la aplicación. Esta mejora es más compleja que las anteriores, por lo que no será descrita en este documento.