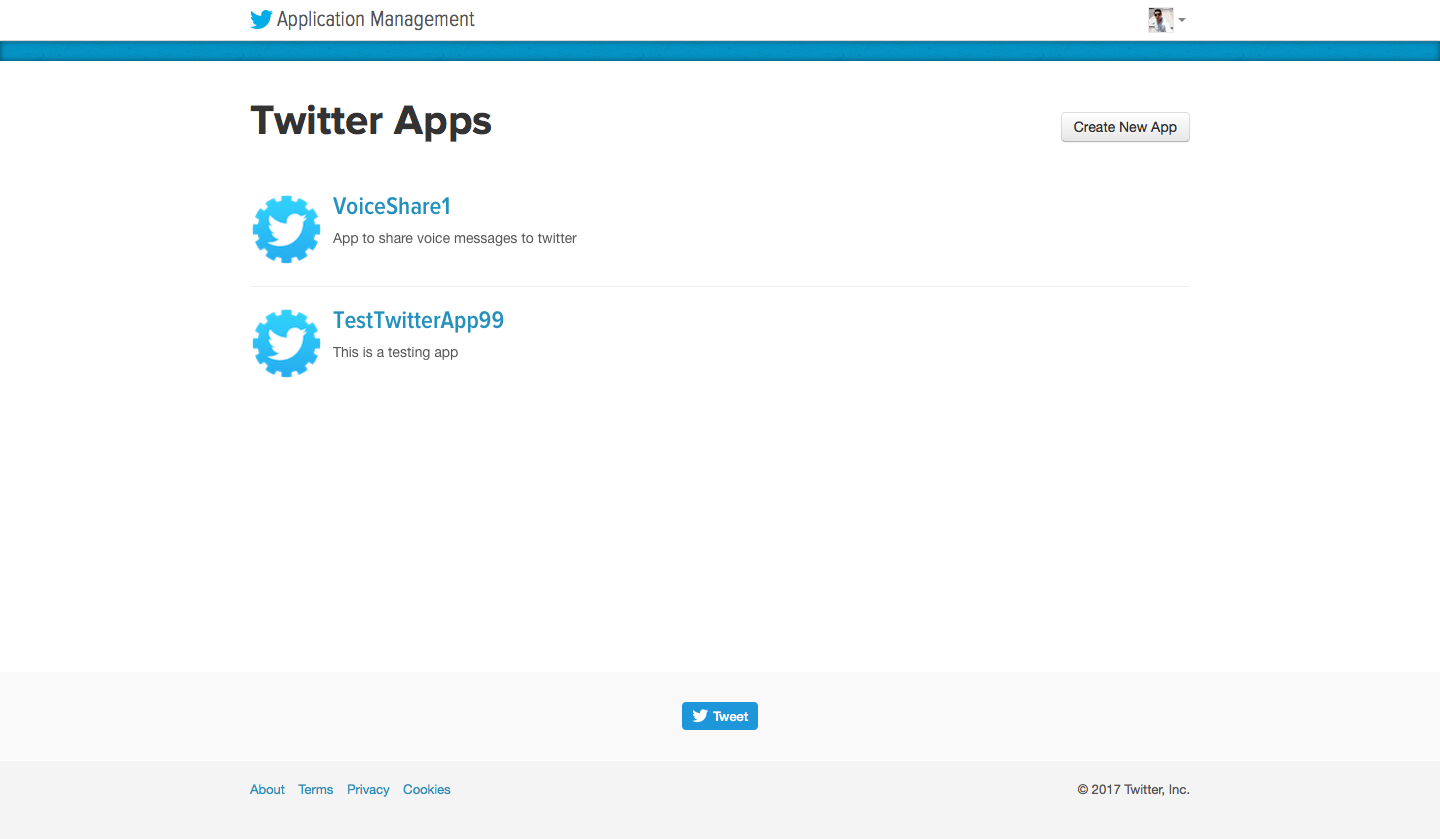
**Instructions for TestTwitterApp (Uses of Speech Recognition and Twitter REST Api)**

*\*\*This instructions do not include instructions for importing packages to Android, so, if you get an error message saying “Cannot Resolve symbol \_\_\_\_\_\_\_\_”, just click on the object name and click [option][enter] if MAC or [ALT][ENTER] if Windows to import the library that it needs.*

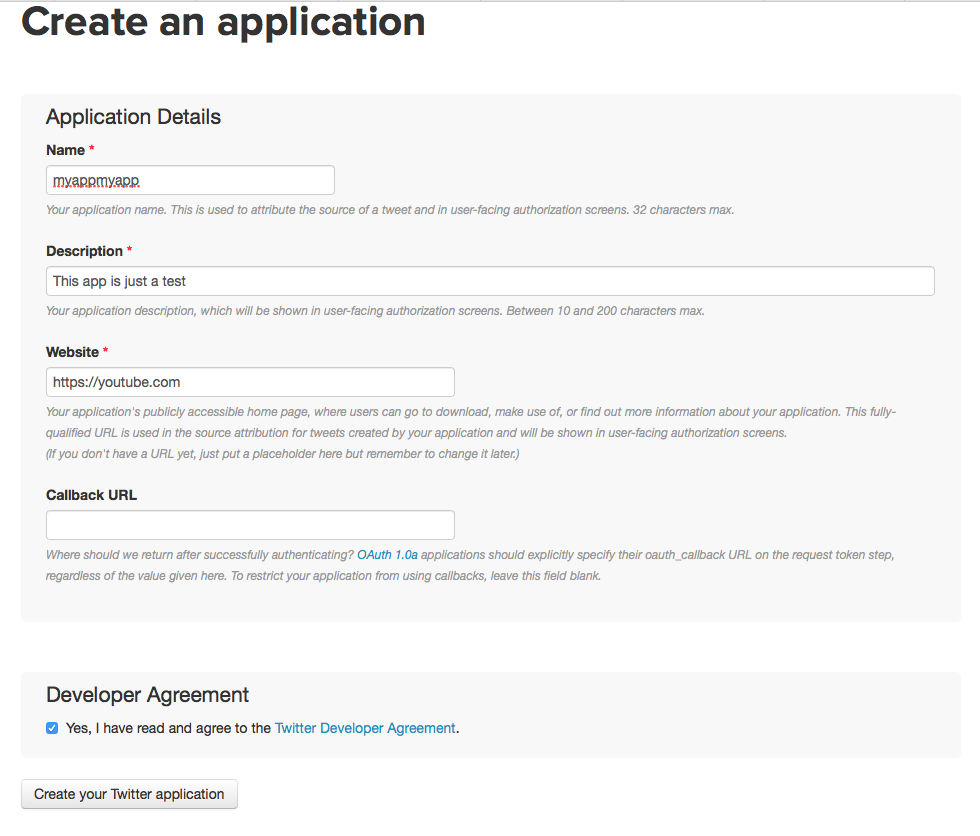
1.- The first step for creating our twitter app is register into Twitter’s website as developers to be able to utilize their services. To do that you need to go to <https://dev.twitter.com/apps/> . If you have already logged in to your Twitter account it will recognize it, otherwise just log in with your Twitter account until you see a page like the one shown in Figure 1.



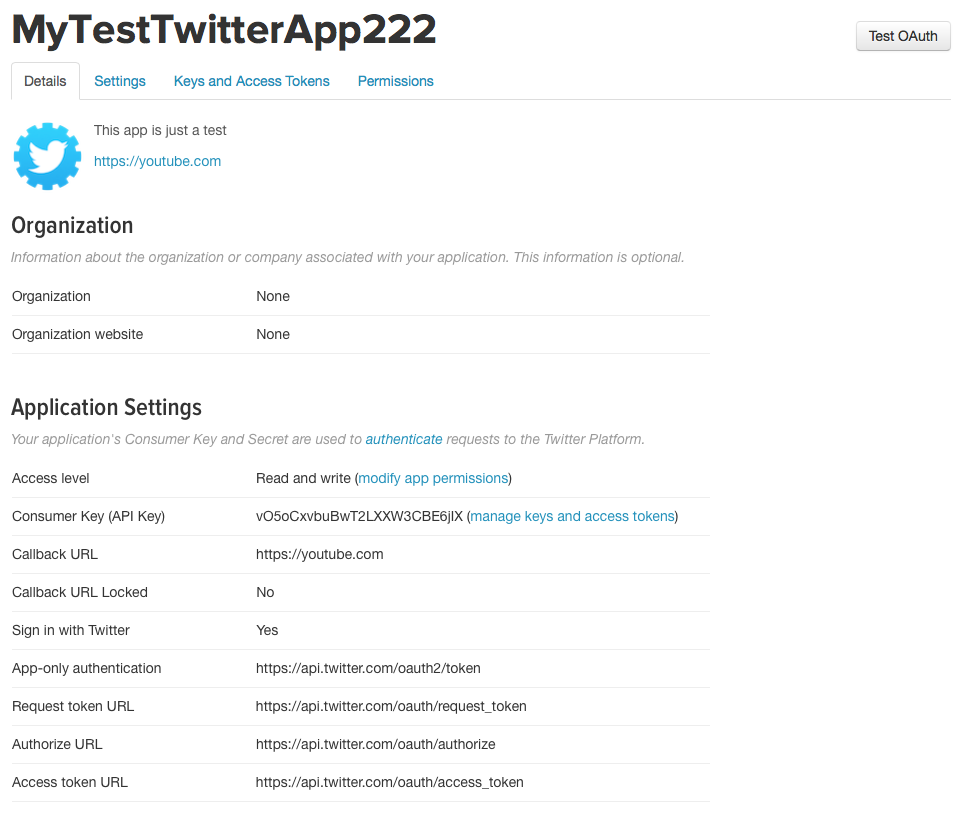
Figure

After that, you need to register your new twitter app by clicking on the “Create New App” button.

Figure



Fill out the spaces in there. The website and callback URL are just URLs that your app may use in case there is a need to test, so any website would work, in my case I just chose <https://youtube.com> as shown in Figure 2.

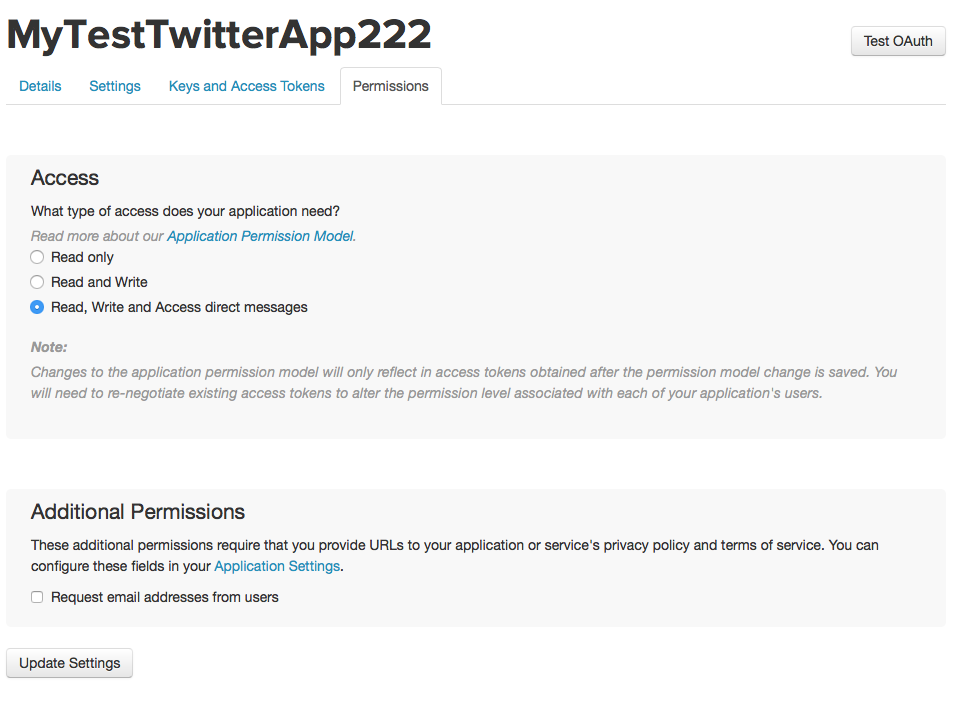


Figure

After you had created your app a page that looks like Figure 3 will appear. This contains all the credentials that you will eventually need for your application to communicate with the Twitter REST web service.

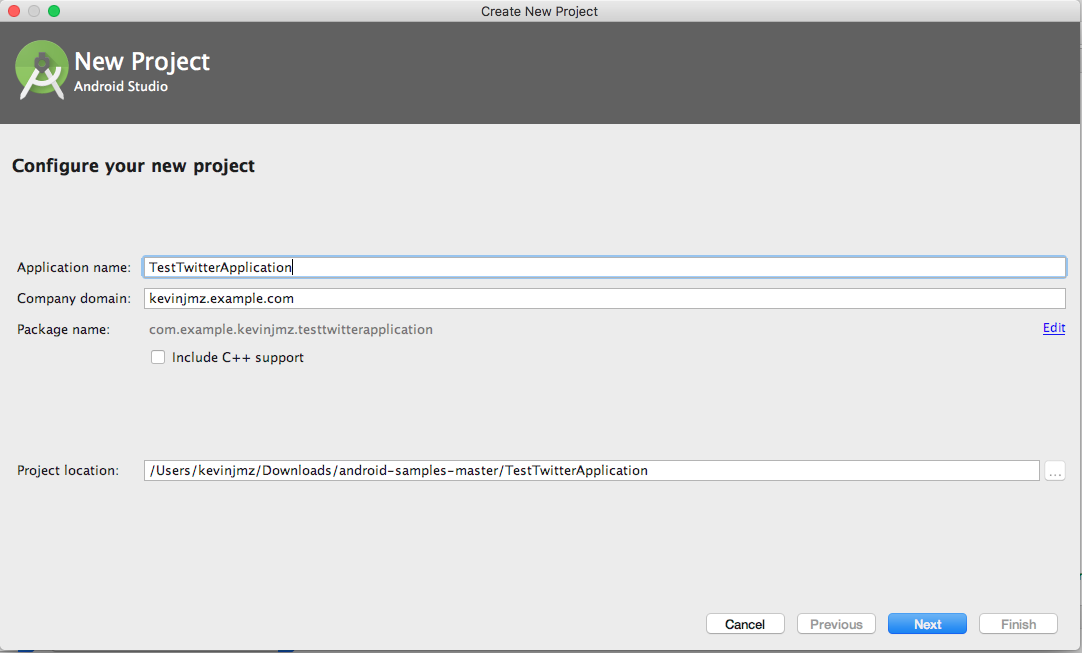
Then, you would need to click on Permissions as shown in Figure 4, and change your access permissions to “Read,Write and Access direct messages”, and then “update Settings” to allow your app to post into your twitter.

Figure

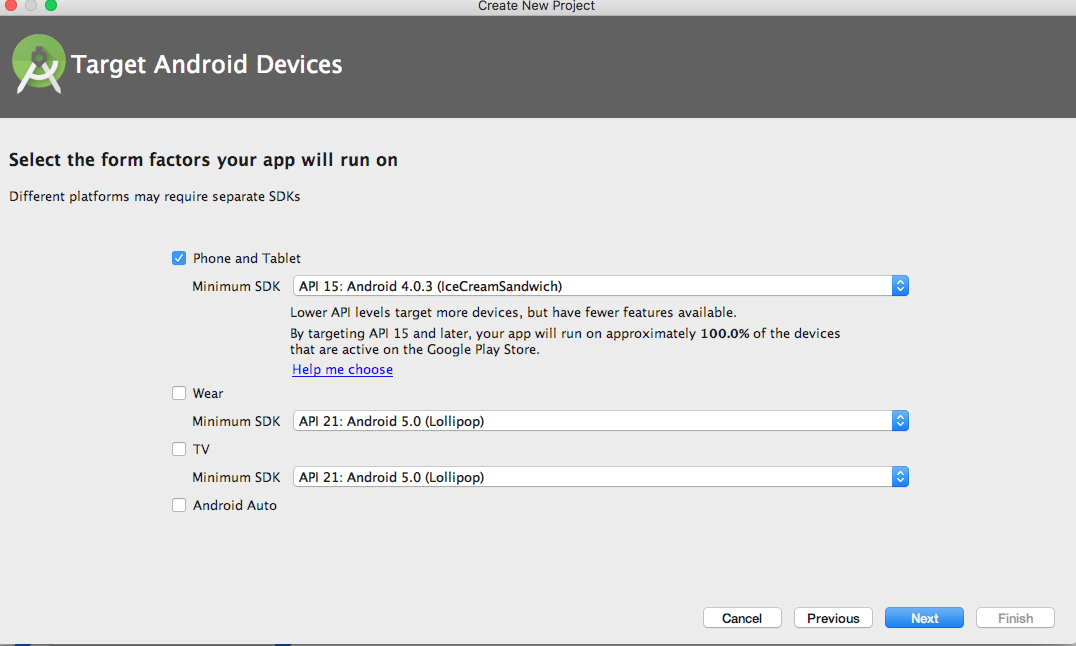


Now we go to Android Studio and create a new empty project as shown in Figures 5,6,7.

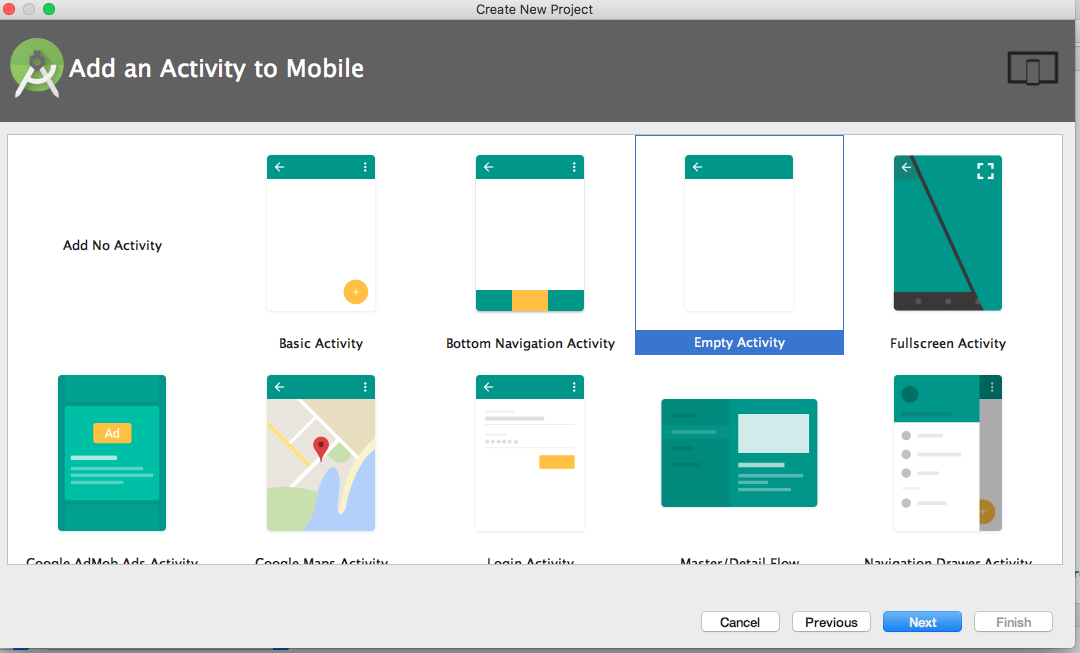
Figure



Figure

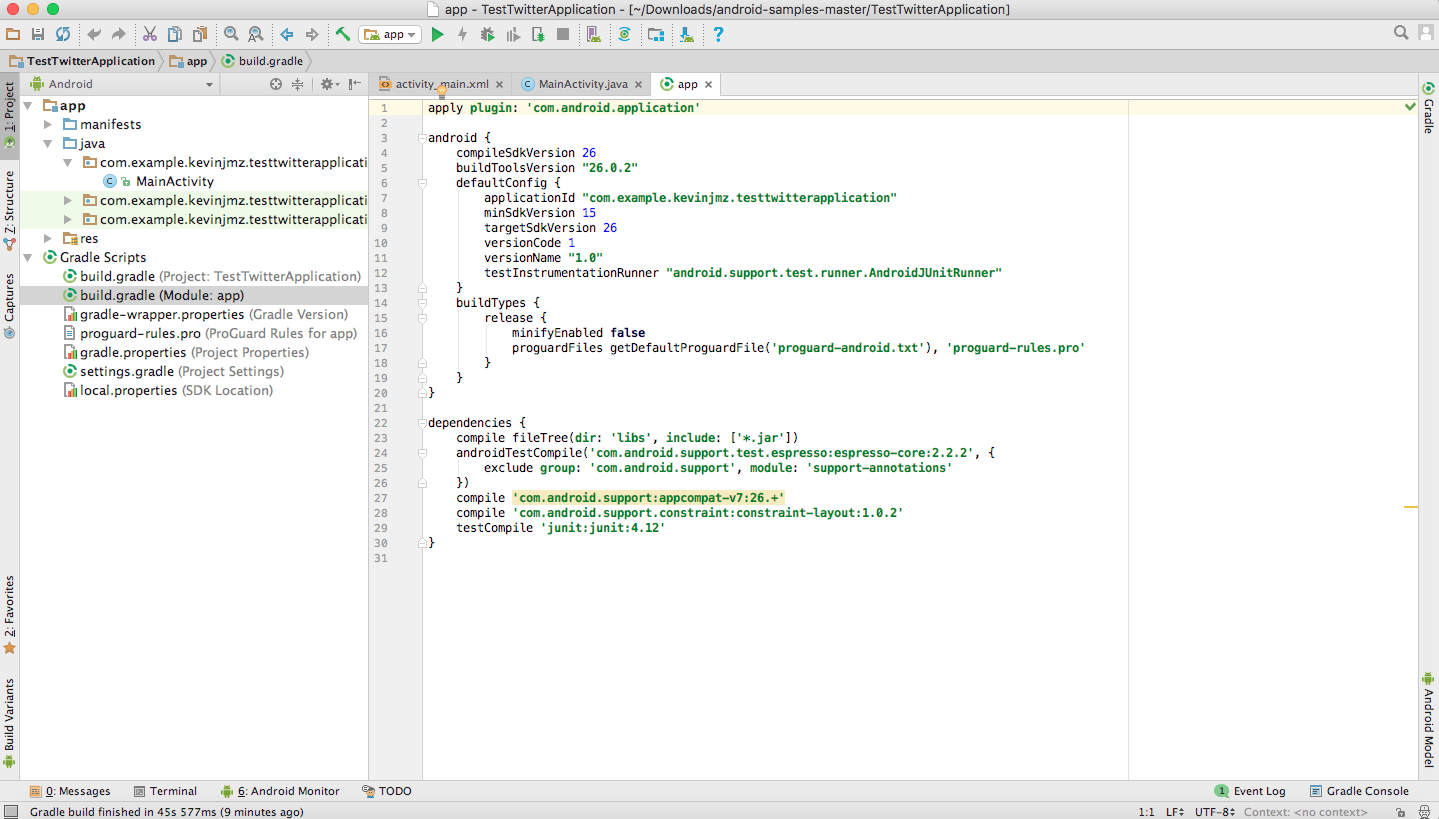


Figure



The first thing that we will be doing is importing the libraries that we will be using through the build.gradle, you can access it in the Gradle Scripts as shown in Figure 8.

Figure



Include this text inside of the dependencies block to import the Twitter functions that we will be using.

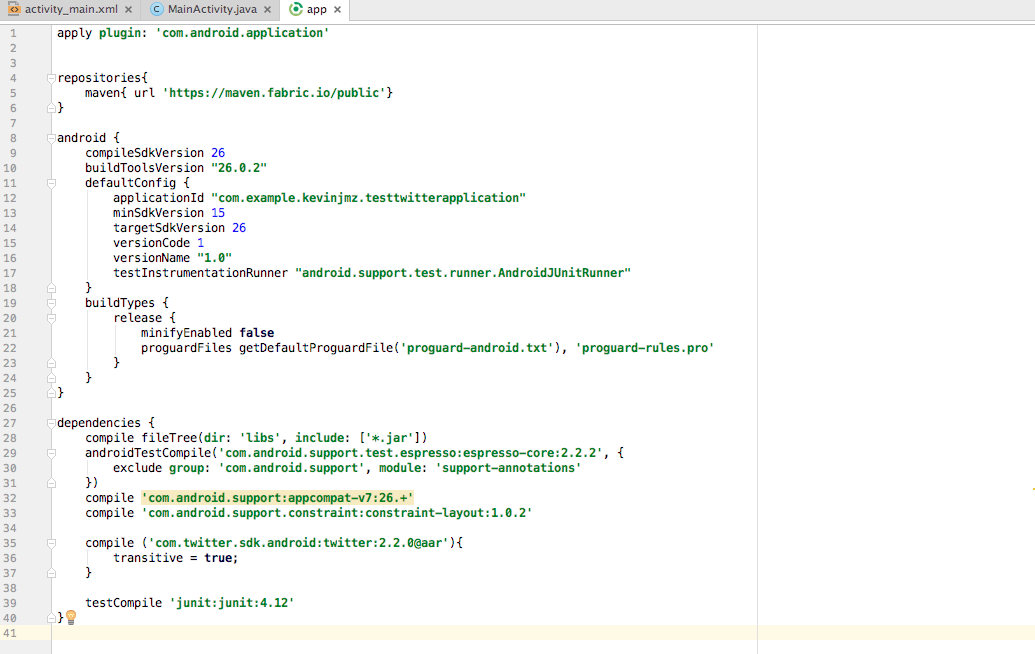
|  |  |
| --- | --- |
| compile ('com.twitter.sdk.android:twitter:2.2.0@aar'){ |  |
|  | transitive = true; |
|  | } |

You also need to write this text in top of the document to be able to import Twitter functions.

|  |  |
| --- | --- |
| repositories{ |  |
|  | maven{ url 'https://maven.fabric.io/public'} |
|  | } |

The final version of the app document is shown in figure 9.

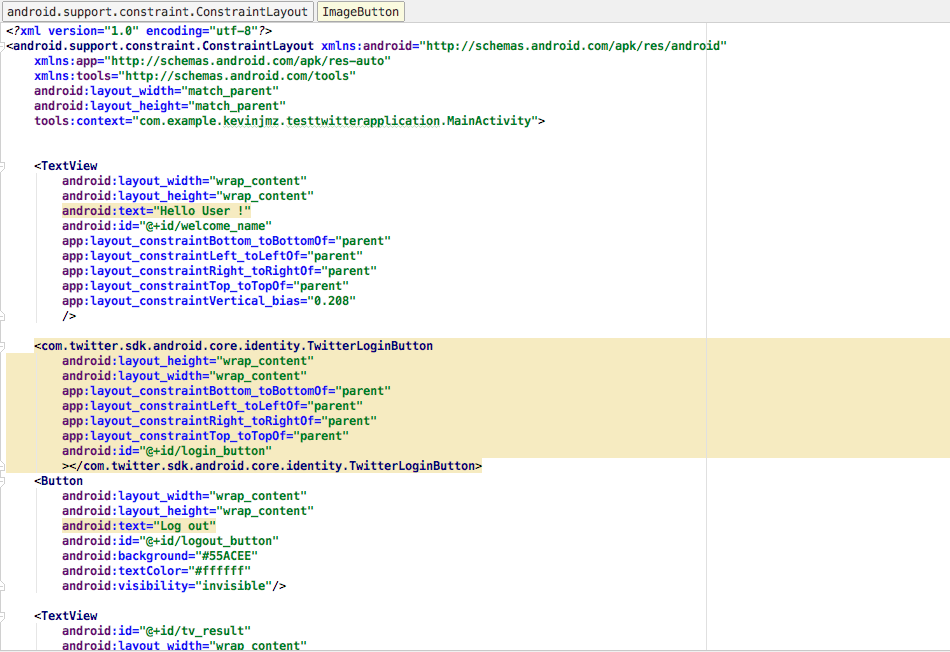
Figure



Don’t forget to “Sync” after you make the changes to your build.gradle file.

Once the document synced well, the next thing to do is modify the activity\_main.xml document to include a Textview to show a recording message, the initial Twitter log in, log out button, another TextView to show the username and an ImageButton. The final version of activity\_main.xml is shown in Figures 10 and 11.

Figure



Figure

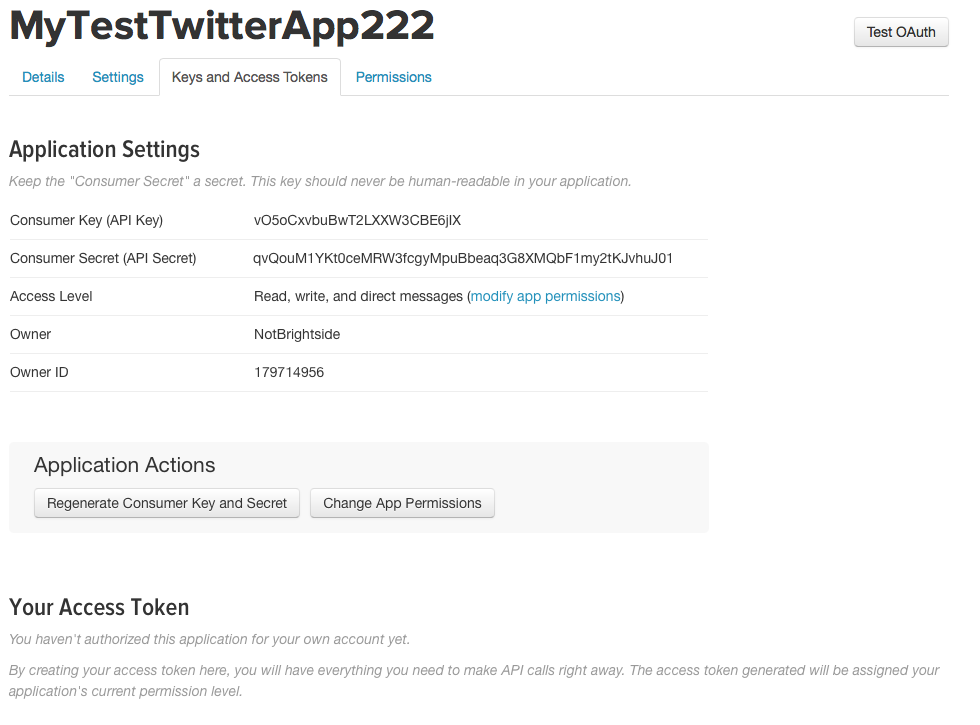


The next step is allowing our app to use the internet. To do that we need to go to

manifests ->AndoidManifest.xml and write <**uses-permission android:name="android.permission.INTERNET"** /> before the <application> block as shown in Figure 12.

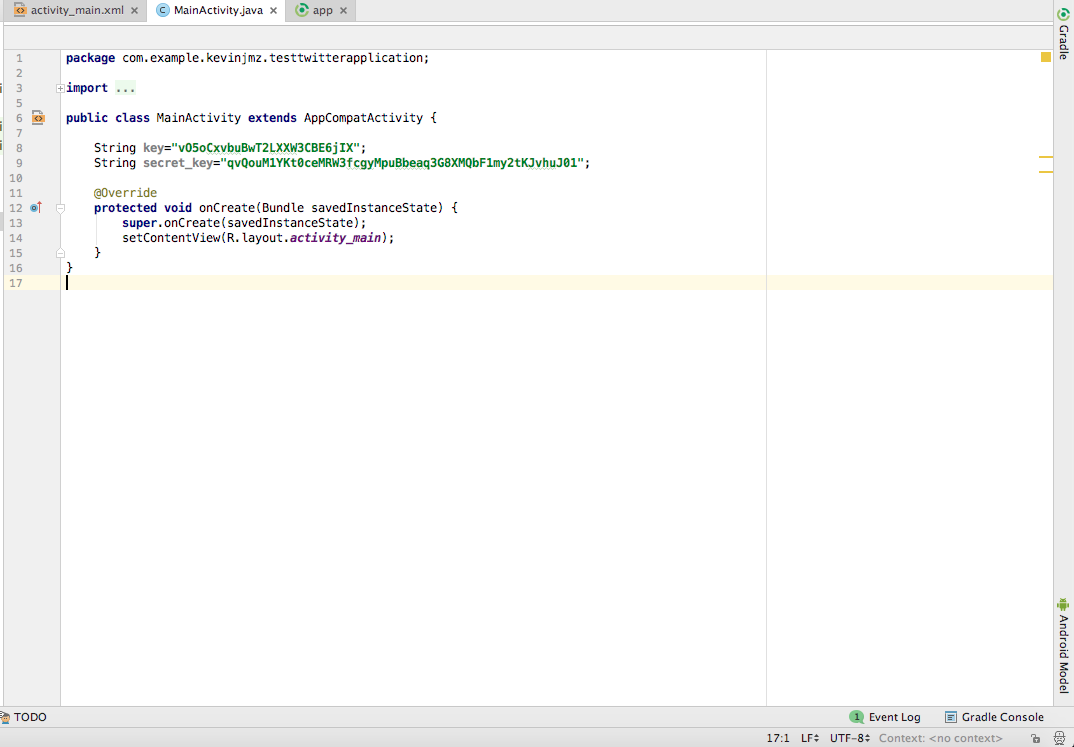
Figure



After you are done typing this or inserting the Internet Permissions, we will start coding our activity, the first step is saving our keys into Strings at our MainActivity.java Activity. To retrieve the keys we need to go to the webpage where we have our app registered as shown in Figure 13, click on “Keys and Access Tokens” and copy the customer Key and the Customer Secret as global string variables. These keys will be used to generate requests to the Twitter REST Service, in our case to post new Tweets or “UPDATE” as they call it on their API Documentation.

Figure

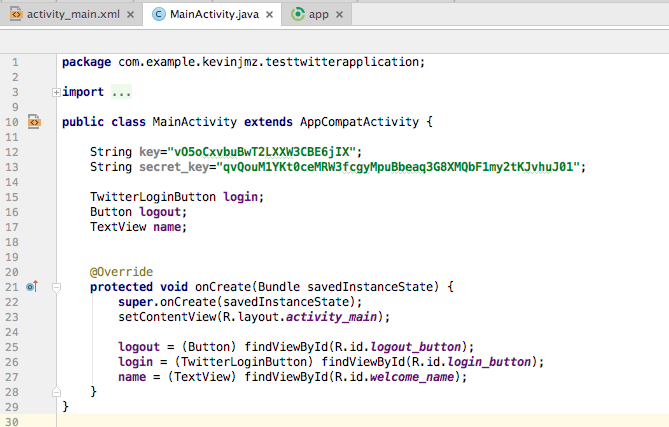
String key="**vO5oCxvbuBwT2LXXW3CBE6jIX**"; //use your own keys  
String secret\_key="**qvQouM1YKt0ceMRW3fcgyMpuBbeaq3G8XMQbF1my2tKJvhuJ01**";//use your own keys

In Figure 14 it is shown how the MainActivity.java should look like.

Figure

The next step is to create buttons and a label object that reflect the ones that we created in the activity\_main.xml and link them together as shown in Figure 15.

Figure



Next, you need to establish a connection with twitter, to do so you need to add these pieces of code before the setContentView(R.layout.activity.main); statement as shown in Figure 16. These two lines of code will create objects that we will need further on our code.

Figure



The next step is setting a callback method to the login button, for the authentication process. To do this we have to write login.setCallback(new Callback...); as shown in Figure 17, when you write new Callback a menu will appear that will take care of the methods that could be overridden automatically.

Figure



Inside the success method we will introduce the code that will happen after the user has successfully authenticated inside our app, the failure method will execute otherwise.

What we want the app to do once the app has successfully logged in is to show the name of the Twitter user, hide the login button, show the logout button and show the microphone button.

Once the user successfully logs in, it is needed to save the successful session to access and modify data inside that specific account like list of tweets or tweet something new. For this we need a TwitterAuthToken object. This object will be receiving a “password” or “key” to access this specific account data from Twitter to modify information.

First, we need to create a TwitterAuthToken and gather that token once the connection with the account has been established.

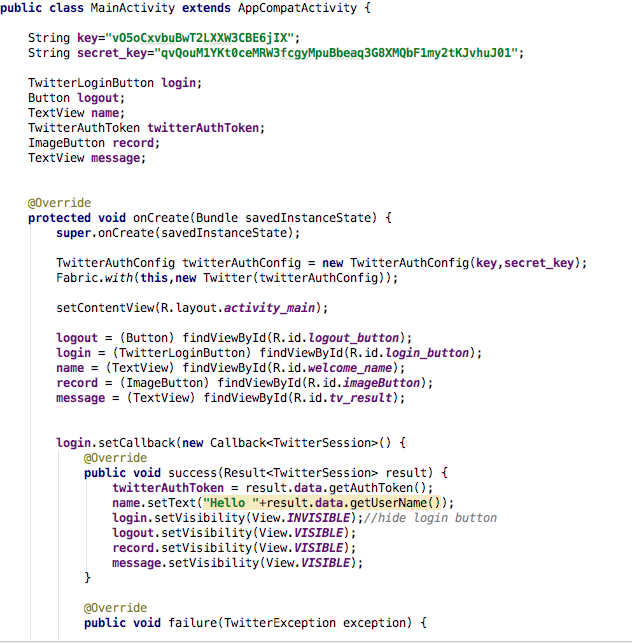
There are some other actions that would be needed once the user logs in.

1. Hide Login button
2. Show Logout button
3. Show the user’s username with a “Hello” message (mostly to confirm that we are connecting)
4. Show the microphone button

To hide or show buttons we are just adjusting the visibility of them, to avoid using another Activity. The code looks like this:

name.setText("Hello "+r.data.getUserName());  
 login.setVisibility(View.INVISIBLE);//hide login button  
 logout.setVisibility(View.VISIBLE);  
 record.setVisibility(View.VISIBLE);  
 result.setVisibility(View.VISIBLE);

The first line of code gathers the successful session and retrieves the user from it. The rest of the code is just hiding and showing buttons and TextViews. This is how the MainActivity.java looks so far.



Notice that I also inserted a “message” TextView object and the “record” imagebutton and instantiated them with the rest of the GUI elements.

Next we will insert a failure message inside the public void failure() method as a Toast in case the user was not able to login.

Toast.makeText(MainActivity.this, "Sorry couldnt identify you", Toast.LENGTH\_SHORT).show();

Then, we need to handle the LOGOUT button to let the user LOGOUT of the app if needed, to do so we will need an object called TwitterSession that represents the user’s active session and close all the active information. Of course we want the app to still be functional so we would enable the “login” button once again and disable the buttons that were enabled before. To do so we need to insert this code after the login callback.

//get session

final TwitterSession twitterSession = Twitter.getSessionManager().getActiveSession();

logout.setOnClickListener(new View.OnClickListener() { //enable click listener to logout button  
 @Override  
 public void onClick(View v) {  
 if(twitterSession!=null){

//gather cookies  
 CookieManager cookieManager = CookieManager.getInstance();   
 if(Build.VERSION.SDK\_INT>=Build.VERSION\_CODES.LOLLIPOP){  
 cookieManager.removeSessionCookies(null);  
 }  
 else{

//remove cookies  
 cookieManager.removeSessionCookie();  
 }  
 Twitter.getSessionManager().clearActiveSession(); //clear session  
 Twitter.logOut(); //logout from session

//enable and disable buttons  
 name.setText("End of session");  
 logout.setVisibility(View.INVISIBLE);  
 record.setVisibility(View.INVISIBLE);  
 message.setVisibility(View.INVISIBLE);  
 login.setVisibility(View.VISIBLE);  
 }  
 }  
 });

Figure 18 and 19 show how the activity would look after the changes:

Figure



Figure



In the following steps we will start working with the Google Speech Recognition API included with Android devices. To do so, we need to set a click listener to the imageButton “record” and call an intent to their activity, then the result will be given back to our activity at the OnActivityResult() method that is going to be overridden. (I know it sounds complicated but it is very simple, just two methods!)

First step is detect when the imageButton is pressed and tell it what to do. As we did before, all we need to do is create an OnClickListener(new OnClickListener() {...]); method as follows:

**record**.setOnClickListener(**new** View.OnClickListener() {

@Override

**public void** onClick(View v) {

promptSpeechInput();

}

});

This code is included after the logout.setOnClickListener code as shown in Figure 20.

Figure



Now, we need to specify what promptSpeechInput() will do. PromptSpeechInput will call an intent that will take charge of everything related with detecting what the user says. So, let’s declare this method and include the code necessary to call this intent. \*\*Notice that this method MUST be declared outside of the OnCreate() method.

**public void** promptSpeechInput(){

Intent i = **new** Intent (RecognizerIntent.***ACTION\_RECOGNIZE\_SPEECH***); *//initialize intent as a speech recognition intent*

i.putExtra(RecognizerIntent.***EXTRA\_LANGUAGE\_MODEL***, RecognizerIntent.***LANGUAGE\_MODEL\_FREE\_FORM***);*//add language configuration*

i.putExtra(RecognizerIntent.***EXTRA\_LANGUAGE***, Locale.*getDefault*());

i.putExtra(RecognizerIntent.***EXTRA\_PROMPT***, **"Say something!"**);*//Add custom message to the intent*

**try** {

startActivityForResult(i, 100);*//notice that the result of the intent will be sent to the key 100 !*

}

*//Toast a message in case device does not support speech recognition*

**catch** (ActivityNotFoundException e){

Toast.*makeText*(MainActivity.**this**,**"Sorry your device doesnt support speech language"**, Toast.***LENGTH\_LONG***);

}

}

Next step is retrieve what the user said to words. Notice we are calling startActivityForResult(i,100); which basically calls the intent and retrieves the result of it. The way to retrieve it is by overriding the onActivityResult() method as follows.

@Override

**protected void** onActivityResult(**int** requestCode, **int** resultCode, Intent data) {

**super**.onActivityResult(requestCode, resultCode, data);

**login**.onActivityResult(requestCode,resultCode,data);

**switch** (requestCode){

**case** 100:

**if** (resultCode == ***RESULT\_OK*** && data != **null**){

ArrayList<String> r = data.getStringArrayListExtra(RecognizerIntent.***EXTRA\_RESULTS***);

**message**.setText(r.get(0));

Intent i = **new** Intent(MainActivity.**this**, EditTweet.**class**);

Bundle e = **new** Bundle();

e.putSerializable(**"FROMSPEECH"**, r.get(0));

i.putExtras(e);

startActivityForResult(i, 200);*//intent, request code*

**break**;

}

**default**: **break**;

}

}

There are several things that are happening in this method, first we are detecting “case 100” which connects the following code with the result of the speech recognition Intent. Then we gather the message as an ArrayList<String>r and sets the TextView to the received message. Then we create another Intent to call another activity called EditTweet.class that we will be creating later on. After that it puts the message into a Bundle and adds the key “FROMSPEECH” to be able to retrieve it and calls the activity with the new key 200.

ActivityMain.java should look like Figures 21, 22 and 23.

Figure

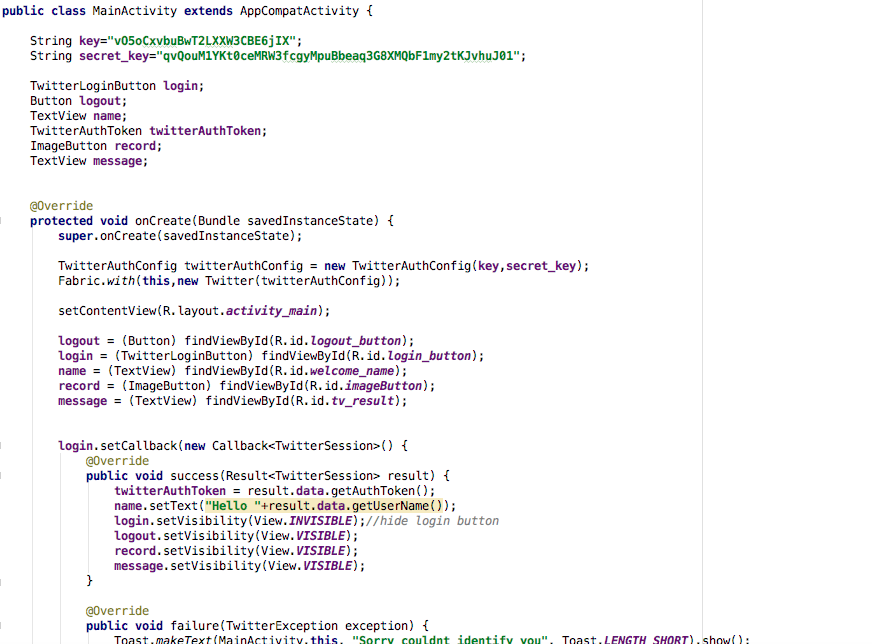
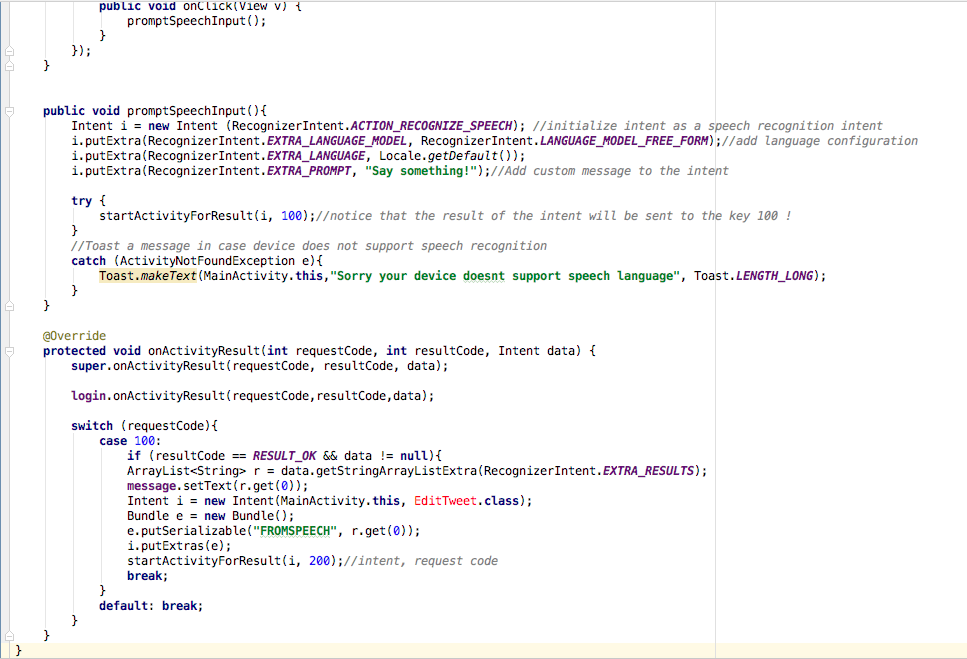


Figure 

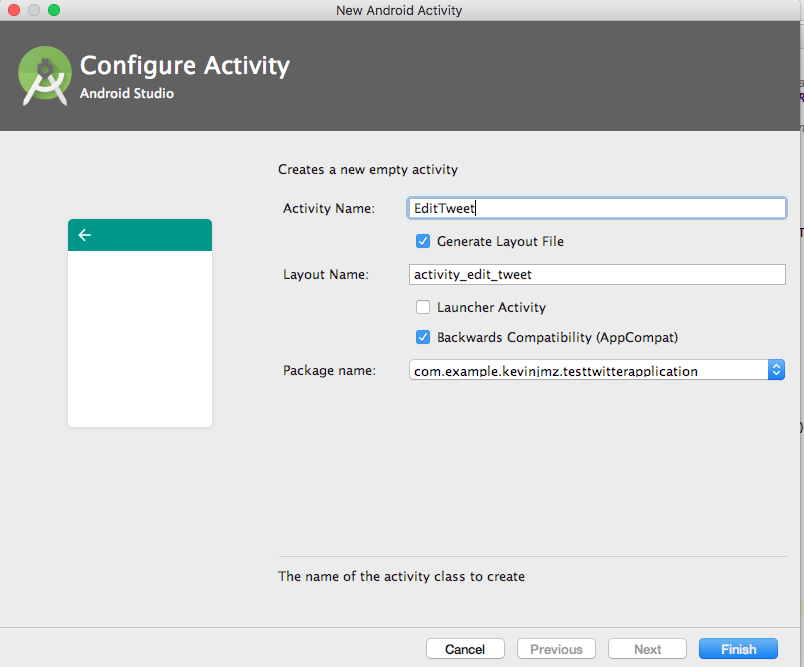
Figure



Next, we will need to create the EditTweet Activity that will allow the user to modify the message if the speech recognition did not captured the message completely or if they want to add new content to it as shown in Figure 24.

To do so, go to src -> main -> java -> testtwitterapp and right click the folder then go to New -> Activity -> EmptyActivity

Figure

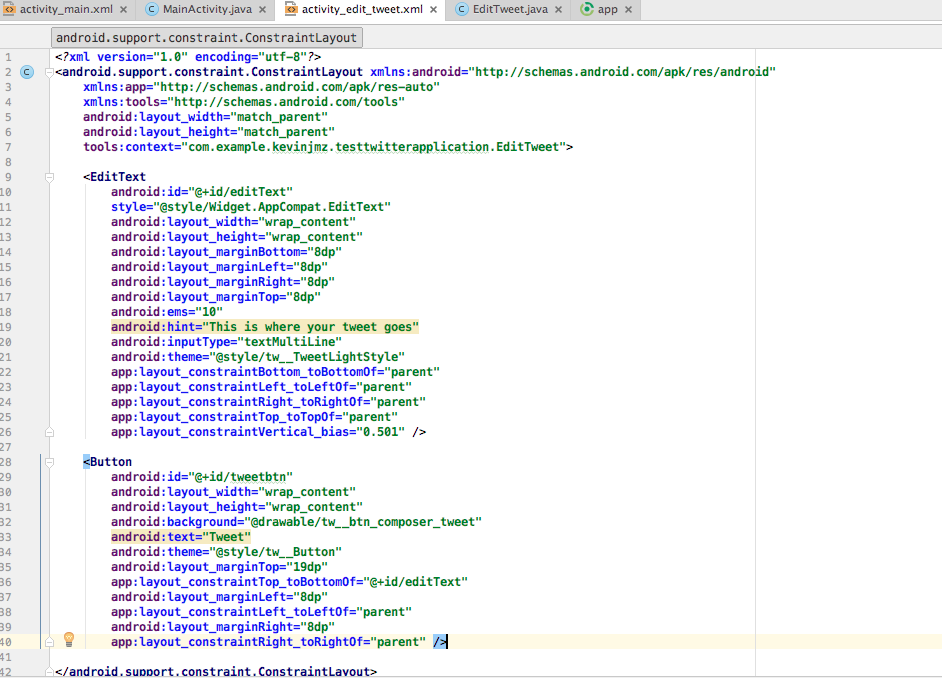


Once you click on Finish an EditTweet.java class will be created along with a correspoding xml file in the res->layout folder. Now, let’s modify the GUI elements to modify the message.

If activity\_layout\_tweet.xml did not open you navigate to res -> layout -> activity\_layout\_tweet.xml and open it.

Now, go ahead and add an EditText and a button as shown in Figure 25:

Figure



The next step is coding the behavior of this Activity. Remember that what we want to do with this activity is receive the message from the Intent to be able to modify it, then send the final message to the main activity for tweeting.

The code inside this class is very simple and is shown in Figure 26:

Figure



As shown in the picture, we first declare the EditText and the Button and connect them to the activity\_edit\_tweet.xml GUI elements. The we create a bundle and we gather the message received and stored in the “FROMSPEECH” key. After we gathered the message we set the text of the editText to show the message there. Finally we set a clickListener that packs the message into an Intent and finishes the activity.

\*\*\*\*\*\*THIS IS A GOOD TIME TO TEST YOUR APP\*\*\*\*\*\*\*\*

Your app should be able to do the following:

1. Allow you to login to a twitter account when clicking the Log in with Twitter Button
2. Show “Hello <<username>>” once connected
3. Show the speech recognition intent once clicking on the android button
4. Recognize your speech after the google speech window appears
5. Place the result from speech recognition in the editText
6. Allow you to modify the text if needed
7. Show the final version of the tweet in the main activity once you click on tweet on top of the android button.

We are still not done, we are missing the most important part, which is tweeting your message to your account, but we are getting closer.

First we need to declare a global variable String finaltweet; to store the information of the tweet. Go to the top of your document, locate where your other variables are and write the following statement:

String **finalTweet**;

The next step is going back to MainActivity.java and locate the OnActivityResult method that we overrode before to gather the final version of the tweet and tweet it. To do so, we will need another case inside our switch statement to gather the tweet message from the EditTweet.java activity.The result code is shown in Figure 27.

Figure



Notice that we have an error in “sendTweet” because we haven’t created this yet, but it will eventually disappear so don’t worry too much.

Next step is to create a method that writes a JSON message, which basically is a regular message encrypted in JSON format so Twitter can read it and understand it. The method is shown in Figure 28.

Figure



This is an external method, meaning you can place it right after the end of the OnActivityResult().

The next step is importing some tools that we will be using in the future to connect to the API. To install these tools you have to go to the build.grade(Module:app) file and modify it. All you have to modify is adding

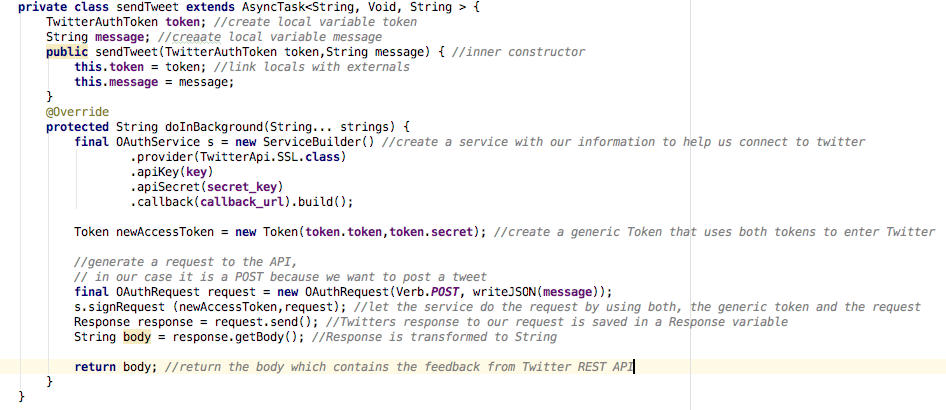
compile **'org.scribe:scribe:1.3.5'**

Inside dependencies and sync once again.

build.grade(Module:app) should look like this:

After that we will create an Async inner class that takes care of tweeting the message in a background thread inside MainActivity.java. The class looks like Figure 29

Figure



This class contains comments line by line to be able to understand its behavior. In the first lines we just create inner variables corresponding to the token sent and the message to tweet. After that, the doInBackground method gets overridden to run in the background thread. Inside this method we declare a service object that is going to help us create the connection between our code and Twitter REST Api. Then we just create a generic token using the secret and the regular token that we passed. Then we need to create a request object that will be sent to Twitter. In our case we used a Verb.POST since we are trying to post new tweets. (If you are interested in gathering tweets from timeline you can use Verb.GET to get the list tweets or other functions that the API offers.) Then we create a response variable to store the response from twitter and call the “request.send()” method. This method is the one that sends your message to Twitter and retrieves feedback information about it. Then we convert the response to String to be able to use it as needed and return that body variable.