# UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA

**FACULTAD DE INGENIERÍA ARQUITECTURA Y DISEÑO**

Cómputo móvil y ubicuo

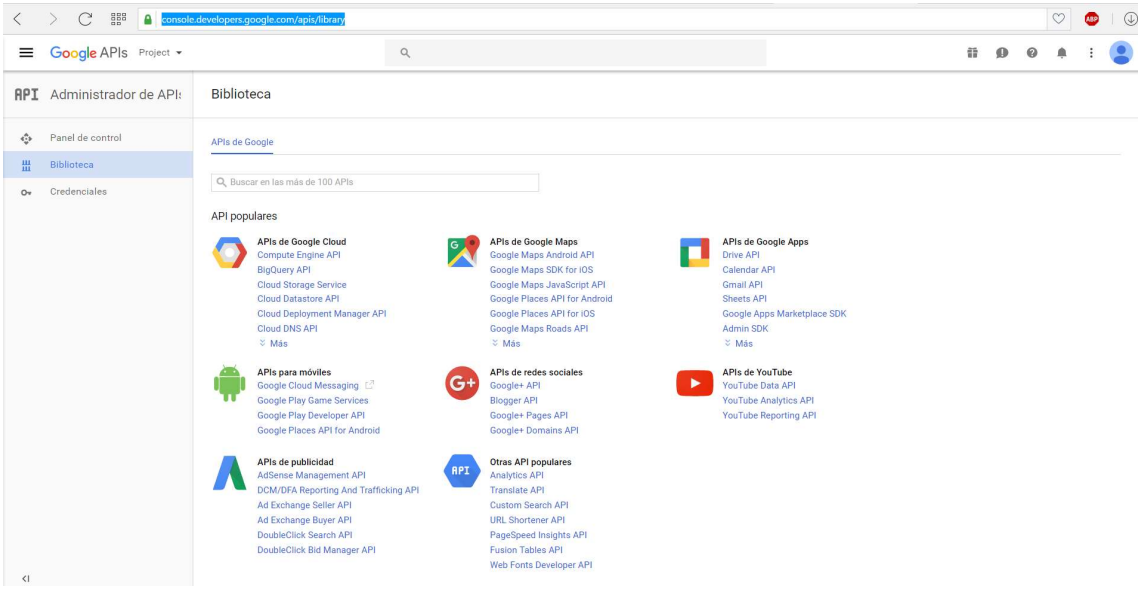
Manual de prácticas

Christian Xavier Navarro Cota

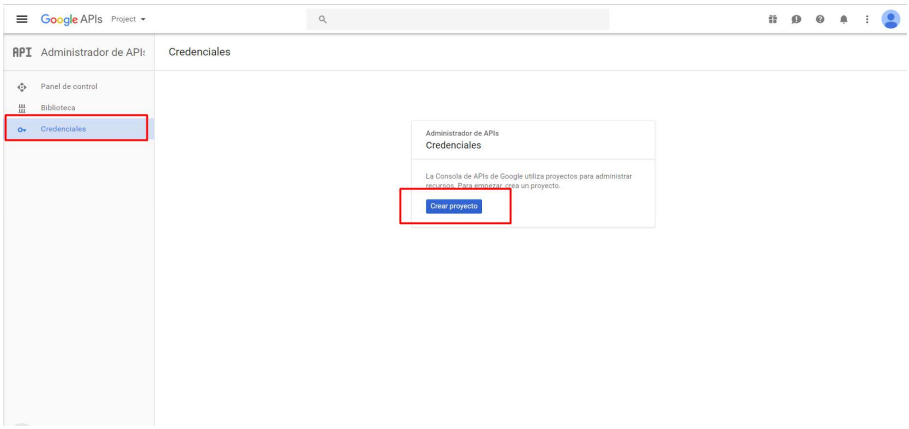
Google Maps

***<u>Cómo vincular nuestro proyecto con la API Console de google</u>***

- Primero debemos ir a la siguiente pagina: https://console.developers.google.com/apis/library

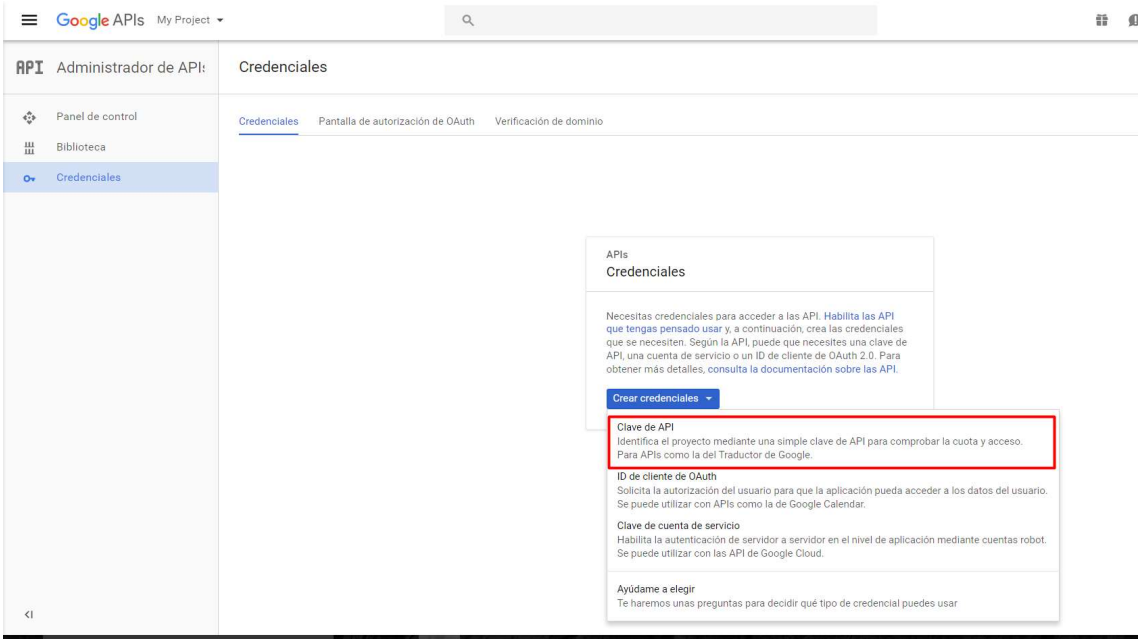



- Vamos a la parte de credenciales y damos clic en crear Proyecto

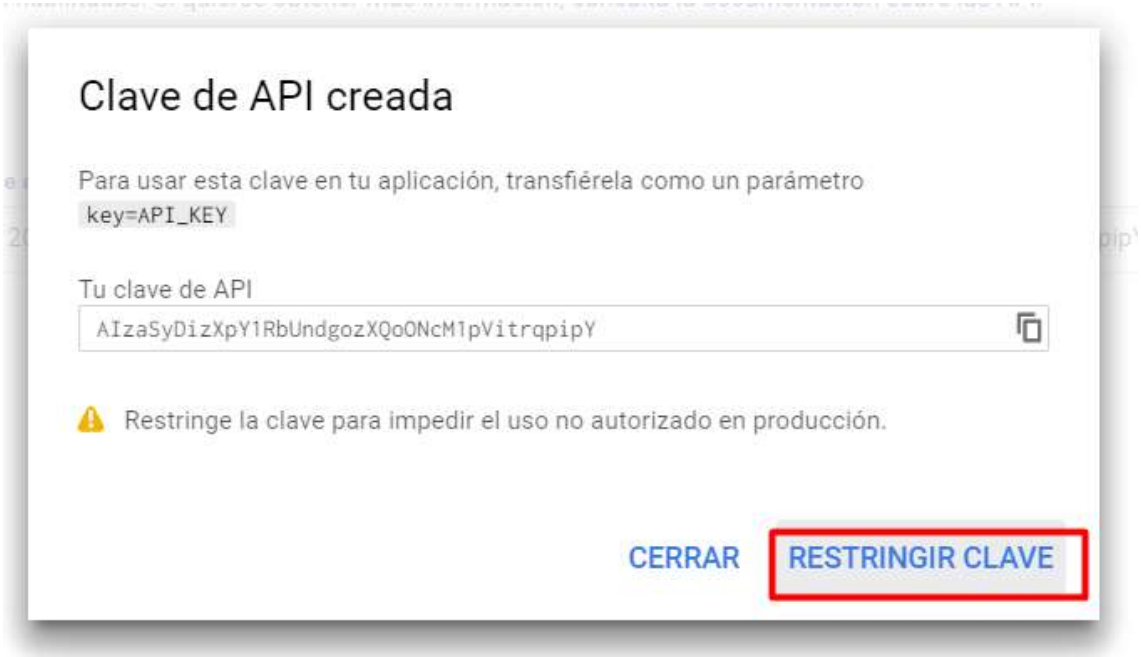



Damos nombre a nuestro proyecto y damos clic en crear.

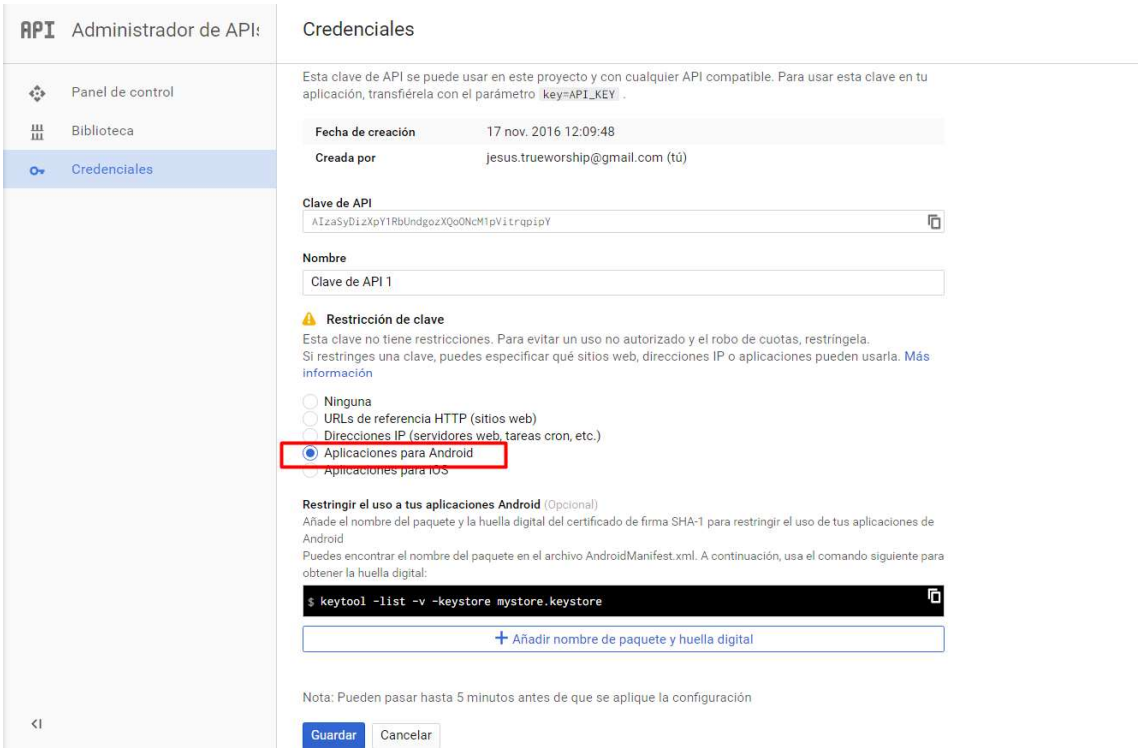
- Damos clic en crear credencial y seleccionamos la opción de “Clave de Api”



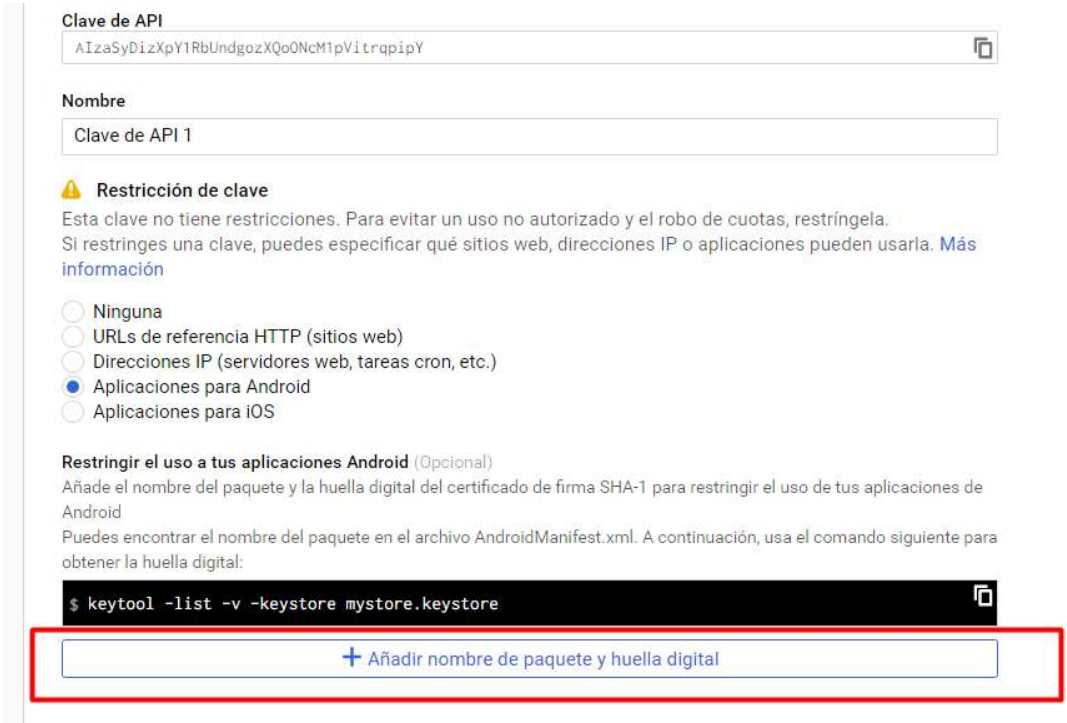

- Nos dará esta ventana y damos clic en restringir clave



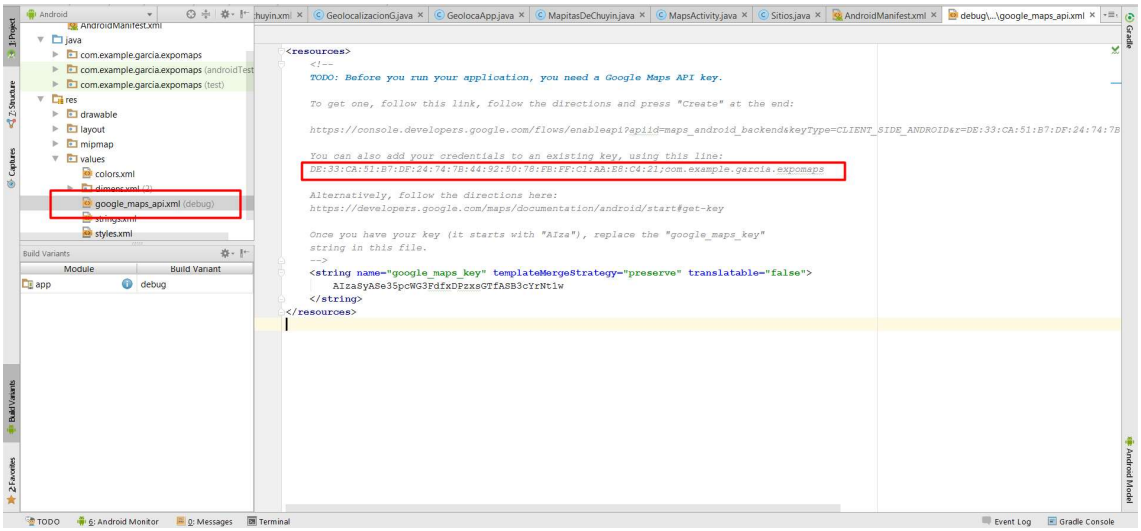
- En esta parte vamos a seleccionar “Aplicaciones para android”



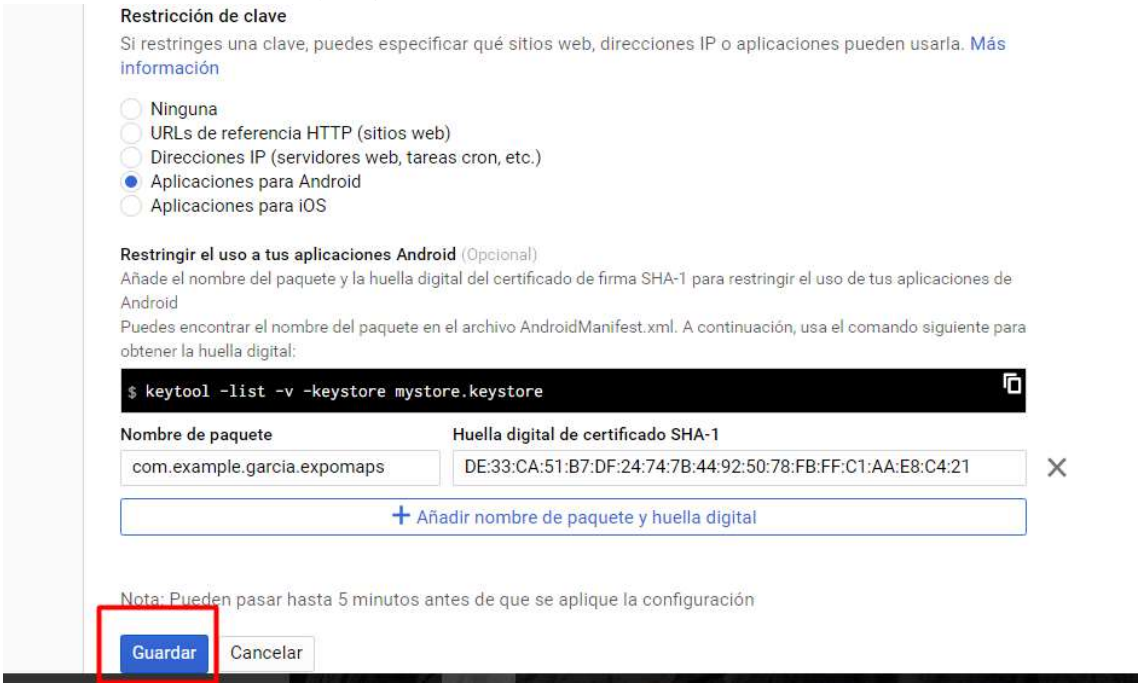

- Seleccionamos la opción siguiente.



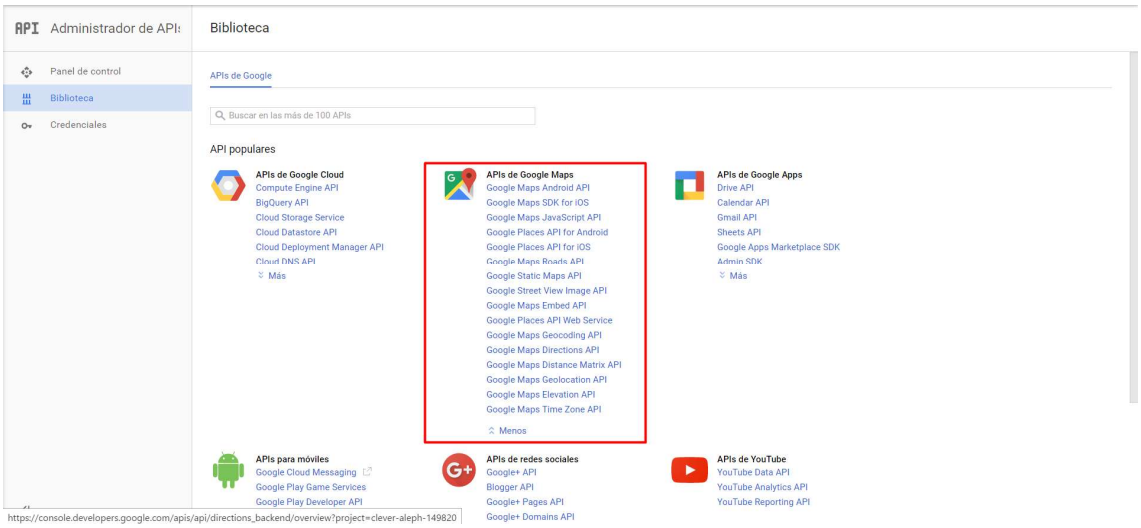

- La huella digital y el nombre los encontraremos en nuestro proyecto en android studio.




- una vez copiados y pegado, damos clic en Guardar

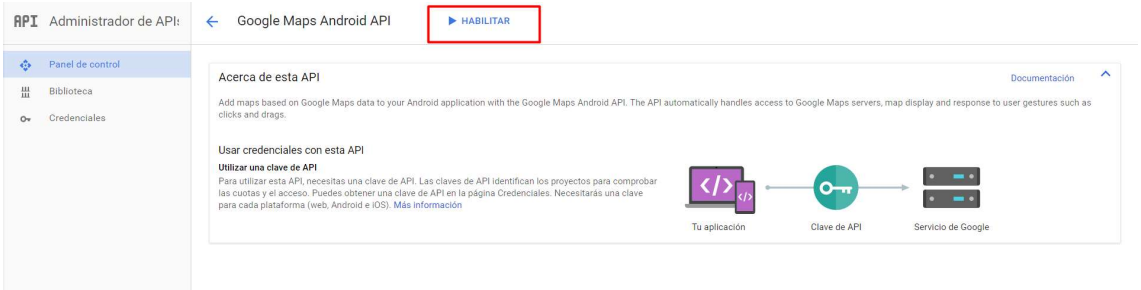



- Ahora nos dirigimos a la pestaña de “Bibliotecas” y localizamos las API’s de google

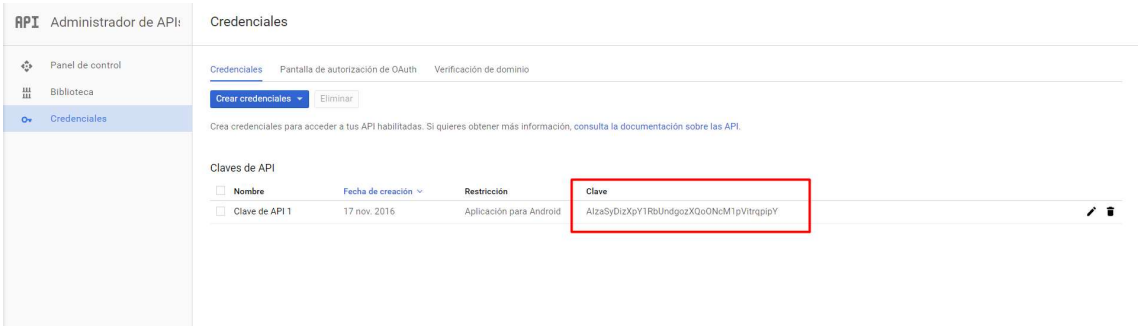



- En esta seleccionamos las API's que nuestro proyecto requiera de la siguiente manera:

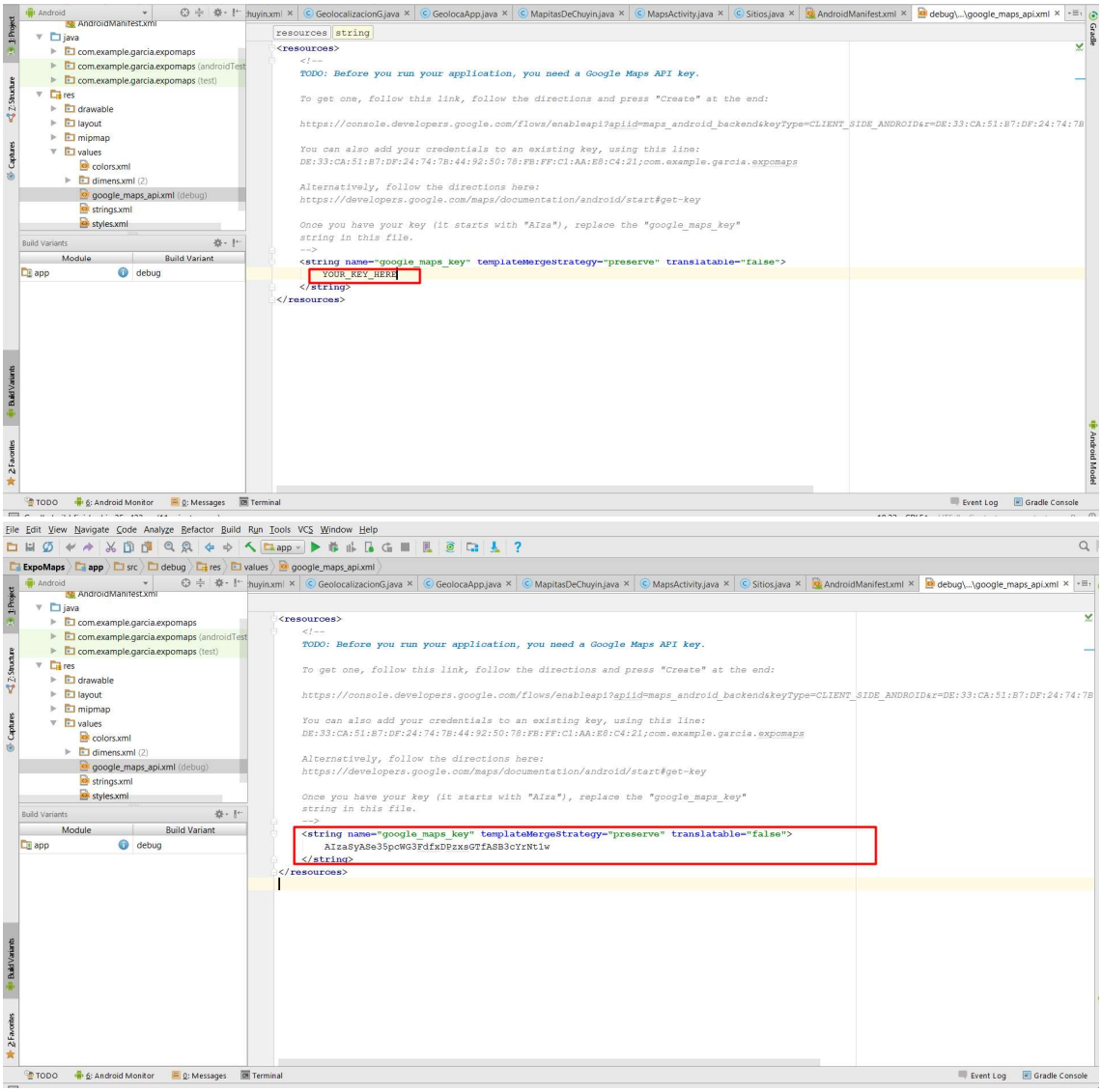

- Damos clic en habilitar, Este paso es el mismo para cualquier API que necesites.




- Por último, regresamos a la pestaña de "Credenciales" y copiamos la clave

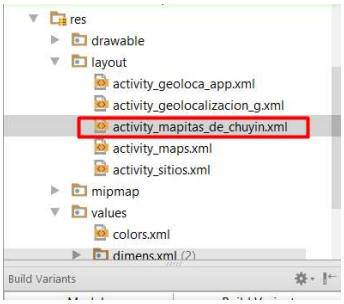



- Esta clave la pegaremos en nuestro proyecto de android en la pestaña de donde sacamos la huella digital y el nombre del proyecto.

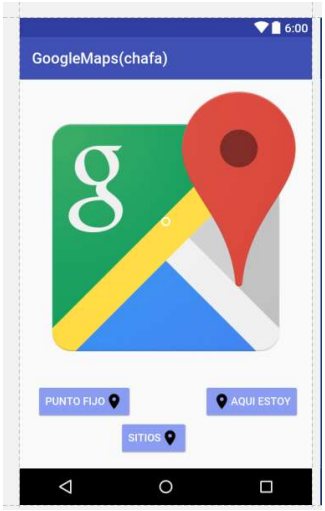



LISTO!! ya podemos comenzar nuestra App de Android utilizando las APIs de Google.

Para comenzar el programa lo primero que haremos será agregar un par de botones y un imagebutton donde colocaremos la imagen oficial de google maps. Esto lo hacemos en nuestro activity.




de forma que quedara de la siguiente manera, para darle formato a continuación veremos el código de los botones.




Código de los botones y del imagebutton:

En este código podemos observar las líneas que ayudan para darle color al botón que es la parte de `android:textColor="@android:color/white"` `app:backgroundTint="#8B9DF3"` la de arriba es para darle color al texto y la línea de abajo es para el fondo que tendrá el botón, también observamos que cada botón tiene un ID con el que haremos referencia en nuestro código para poder implementar el botón, esta acción la definimos con la línea de código siguiente:

`android:text="Sitios"` .

```
<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Aqui Estoy"
    android:id="@+id/geoloca"
    android:textColor="@android:color/white"
    app:backgroundTint="#8B9DF3"
    android:drawableLeft="@drawable/map_marker2"
    android:layout_alignBaseline="@+id/ubicarme"
    android:layout_alignBottom="@+id/ubicarme"
    android:layout_alignParentRight="true"
    android:layout_alignParentEnd="true" />


<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    app:srcCompat="@drawable/cast_abc_scrubber_control_off_mtrl_alpha"
    android:id="@+id/imageButton"
    android:background="@drawable/mapa"
    android:onClick="mapasDeGoogle"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true" />


<Button
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Sitios"
    android:id="@+id/sitios"
    android:textColor="@android:color/white"
    app:backgroundTint="#8B9DF3"
    android:drawableRight="@drawable/map_marker2"
    android:layout_alignParentBottom="true"
    android:layout_toLeftOf="@+id/geoloca"
    android:layout_toStartOf="@+id/geoloca"
    android:layout_marginRight="20dp"
    android:layout_marginEnd="20dp" />

<Button
android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Punto Fijo"
```

[I18N] Hardcoded string "Punto Fijo", should use `@string` resource more... (Ctrl+F1)

```
    app:backgroundTint="#8B9DF3"
    android:drawableRight="@drawable/map_marker2"
    android:layout_above="@+id/sitios"
    android:layout_alignLeft="@+id/imageButton"
    android:layout_alignStart="@+id/imageButton" />
```

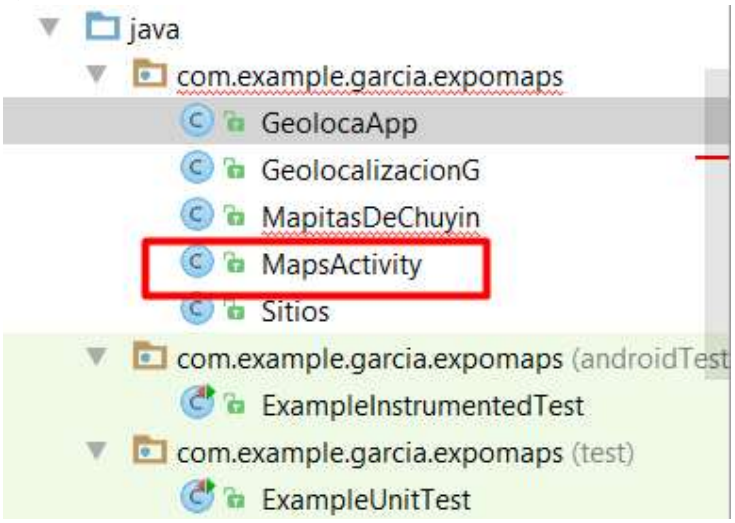
Para implementar los botones vamos a la parte:


y ahí es donde nuestro código para los mapas comenzará

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_mapitas_de_chuyin);

    ubicarme = (Button) findViewById(R.id.ubicarme);
    geoloca = (Button) findViewById(R.id.geoloca);
    sitio = (Button) findViewById(R.id.sitios);

    ubicarme.setOnClickListener((arg0) → {
            Intent inten = new Intent(MapitasDeChuyin.this, MapsActivity.class);
            startActivity(inten);
    });
    geoloca.setOnClickListener((arg0) → {
            Intent inten = new Intent(MapitasDeChuyin.this, GeolocaApp.class);
            startActivity(inten);
    });

    sitio = (Button) findViewById(R.id.sitios);
    sitio.setOnClickListener((view) → {
            PlacePicker.IntentBuilder builder = new PlacePicker.IntentBuilder();
            Intent intent;
            try {
                intent = builder.build((Activity) getApplicationContext());
                startActivityForResult(intent,PLACE_PICKER_REQUEST);
            } catch (GooglePlayServicesRepairableException e) {
                e.printStackTrace();
            } catch (GooglePlayServicesNotAvailableException e) {
                e.printStackTrace();
            }
```

En el método onCreate creamos nuestros botones que son los de ubicarme, geoloca y sitios y hacemos referencia a ellos a través del ID que le colocamos a nuestros botones para el momento en que damos clic el botón haga algo, entonces una vez colocado el ID necesitamos crear un evento en cada botón con ayuda del setOnClickListener.

ahora bien necesitamos crear el activity de ubicarme para que nuestra App nos de un punto fijo en el mapa que es el propósito de este botón, para eso vamos a pasar a:




y este es el código completo para agregar 2 puntos fijos en el mapa

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;

    mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
    UiSettings uiSettings = mMap.getUiSettings();
    uiSettings.setZoomControlsEnabled(true);
    uiSettings.setAllGesturesEnabled(true);
    uiSettings.setMapToolbarEnabled(true);

    // Add a marker in Sydney and move the camera
    LatLng uabc = new LatLng(31.864597, -116.666242);
    mMap.addMarker(new MarkerOptions()
            .position(uabc)
            .title("La Skul")
            .icon(BitmapDescriptorFactory
                    .defaultMarker(BitmapDescriptorFactory.HUE_GREEN)));

    LatLng casa = new LatLng(31.842647, -116.600640);
    mMap.addMarker(new MarkerOptions()
            .position(casa)
            .title("my home carnal")
            .icon(BitmapDescriptorFactory
                    .defaultMarker(BitmapDescriptorFactory.HUE_GREEN)));

    float zoomlevel=16;
    mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(uabc,zoomlevel));
}
```

lo primero que hacemos es agregar una variable llamada mMaps que es con la que trabajaremos y comenzamos diciendo que tipo de mapa es el que mostrará, en este caso es con el satélite `mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);` y de ahí agregamos las utilerías con las que podremos tener más control sobre nuestra app que son los permisos para el zoom, para los gestos y la barra de herramientas.

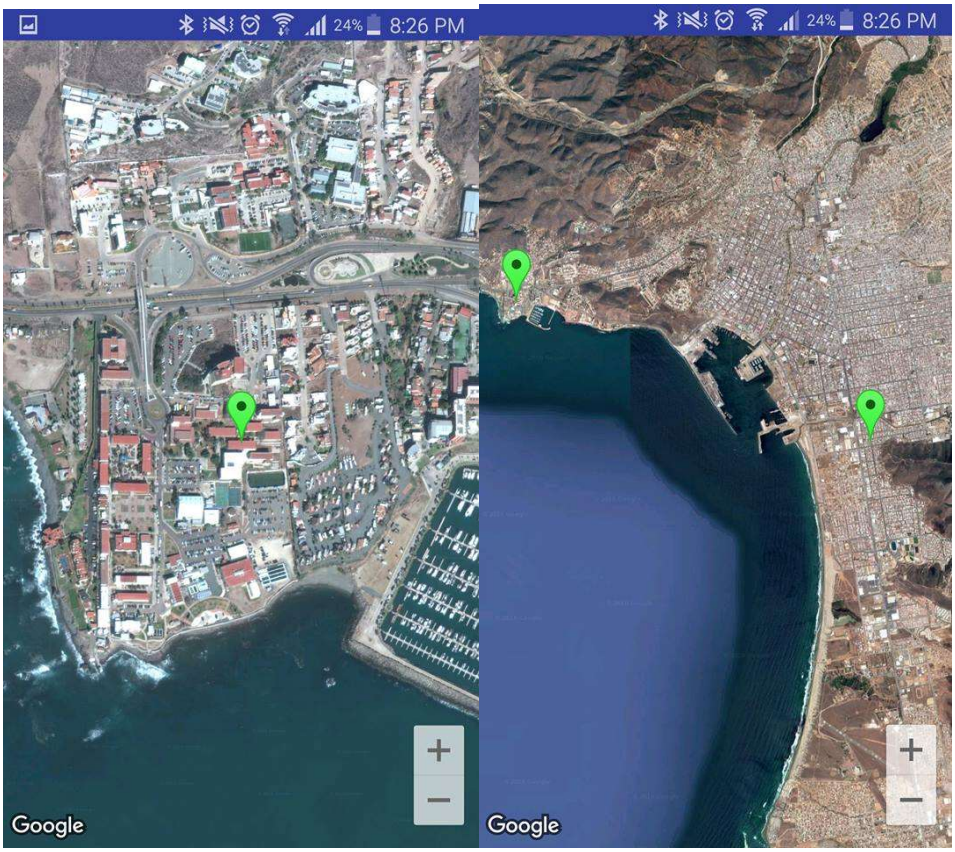
Enseguida de eso agregamos la primera ubicación en el mapa que será la de uabc y agregamos las coordenadas, el siguiente paso es agregar las características del marcador, la posición será la que guardamos en “uabc” el siguiente es el título que el marcador tendrá, el tercero es para escoger el icono del marcador y también su color.

```
LatLng uabc = new LatLng(31.864597, -116.666242);
mMap.addMarker(new MarkerOptions()
        .position(uabc)
        .title("La Skul")
        .icon(BitmapDescriptorFactory
                .defaultMarker(BitmapDescriptorFactory.HUE_GREEN)));
```

de la misma manera agregamos el segundo marcador.

La línea siguiente es para cuando abrimos nuestra app el satélite se enfoque en uno de los puntos que agregamos, en este caso es el de uabc y tiene un zoom de x16:

```
float zoomlevel=16;
mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(uabc,zoomlevel));
```

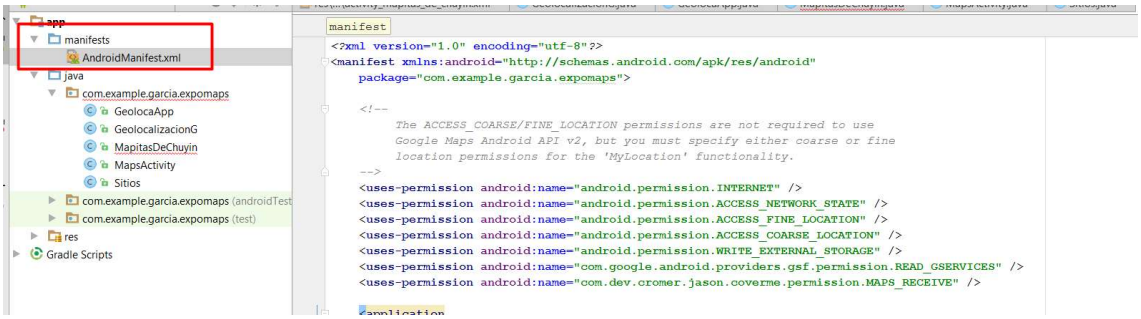


Para la parte de geolocalización que en mi proyecto se llama “ubicarme” ahí solo utilizaremos el gps de nuestro teléfono y lo vinculamos con nuestro maps. Este es el código necesario para esta parte:

```
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    miUbicacion();

    mMap.setMapType(GoogleMap.MAP_TYPE_SATELLITE);
    UiSettings uiSettings = mMap.getUiSettings();
    uiSettings.setZoomControlsEnabled(true);
    uiSettings.setScrollGesturesEnabled(true);
    uiSettings.setMyLocationButtonEnabled(true);
    uiSettings.setMapToolbarEnabled(true);

}



private void agregarMarcador(double lat, double lng) {
    LatLng coordenadas = new LatLng(lat, lng);
    CameraUpdate miUbicacion = CameraUpdateFactory.newLatLngZoom(coordenadas, 16);
    if (marcador != null) marcador.remove();
    marcador = mMap.addMarker(new MarkerOptions()
            .position(coordenadas)
            .title("Aqui estoy homs")
            .icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN)));
    mMap.animateCamera(miUbicacion);
}

private void actualizarUbicacion(Location location) {
    if (location != null) {
        lat = location.getLatitude();
        lng = location.getLongitude();
        agregarMarcador(lat, lng);
    }
```

Lo primero que hacemos es agregar los permisos al igual que en punto fijo, en el método agregarMarcador obtendremos las coordenadas a través del método actualizarUbicacion y las guardaremos en un marcador para que salga sobre nuestro mapa.

Por último pero no por eso menos importante, y la verdad creo que es la parte más importante porque si ellos la app no funciona y estoy hablando de los permisos.
Como se muestra en la imagen le estamos otorgando a la app los permisos para internet, para el gps, para que utilice la memoria del teléfono, para que utilice también la red.


app
manifests
AndroidManifest.xml
java
com.example.garcia.expomaps
GeolocaApp
GeolocalizacionG
MapitasDeChuyin
MapsActivity
Sitios
com.example.garcia.expomaps (androidTest
com.example.garcia.expomaps (test)
res
Gradle Scripts
manifest
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.example.garcia.expomaps">
<!--
The ACCESS_COARSE/FINE_LOCATION permissions are not required to use
Google Maps Android API v2, but you must specify either coarse or fine
location permissions for the 'MyLocation' functionality.
-->
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
<uses-permission android:name="com.dev.cromer.jason.coverme.permission.MAPS_RECEIVE" />
<application

# Base de datos Remotas

# Bases de datos remotas

## Android

**Cómo funciona el acceso a datos remoto**

El acceso a datos remotos proporciona un método sencillo para que una aplicación obtenga acceso a los datos que se encuentran en una base de datos remota de SQL Server. La propagación de datos se inicia primero en el cliente. Los datos de una tabla se extraen del servidor al cliente.

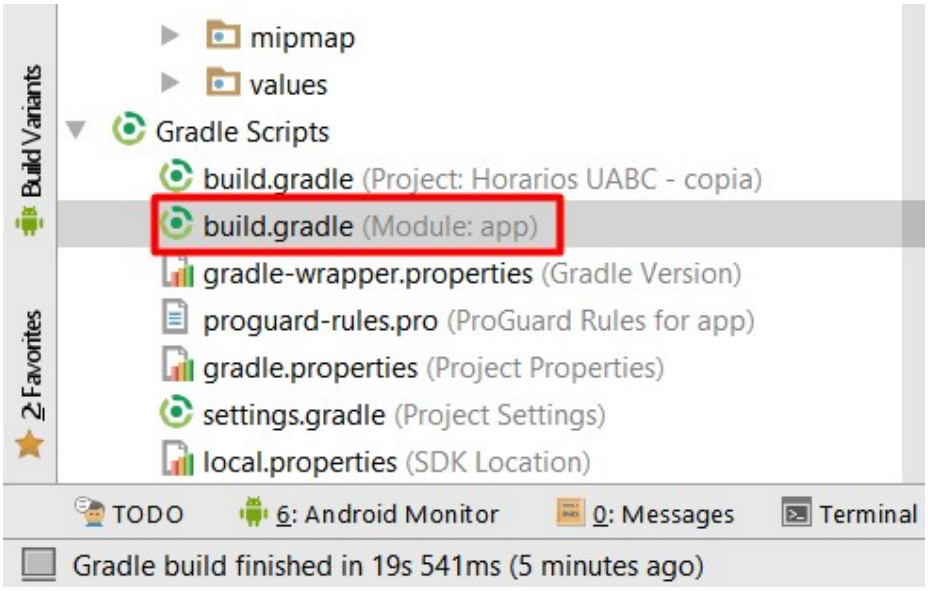
## Obtención de acceso remoto a una base de datos mediante Android

Para lograr una conexión a alguna base de datos remota mediante Android necesitaremos alguna herramienta que logra realizar peticiones https dentro de la aplicación.

**Volley:** es una librería desarrollada por Google para optimizar el envío de peticiones Http desde las aplicaciones Android hacia servidores externos.

La línea de código para agregar esta librería es "**com.github.clans:fab:1.6.2"**

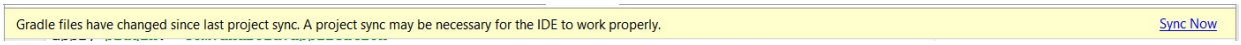
Esta línea será introducida a nuestra aplicación en el archivo build.gradle de la siguiente manera




Una vez abierto el archivo nos dirigiremos a la sección de dependencias y se agregara ahí mismo

```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    testCompile 'junit:junit:4.12'
    compile 'com.android.support:appcompat-v7:25.0.0'
    compile 'com.android.support:support-v4:25.0.0'
    compile 'com.android.support:design:25.0.0'
    compile 'com.google.android.gms:play-services-appindexing:8.4.0'
    compile 'com.android.volley:volley:1.0.0'
    compile 'com.github.clans:fab:1.6.2'

}
```

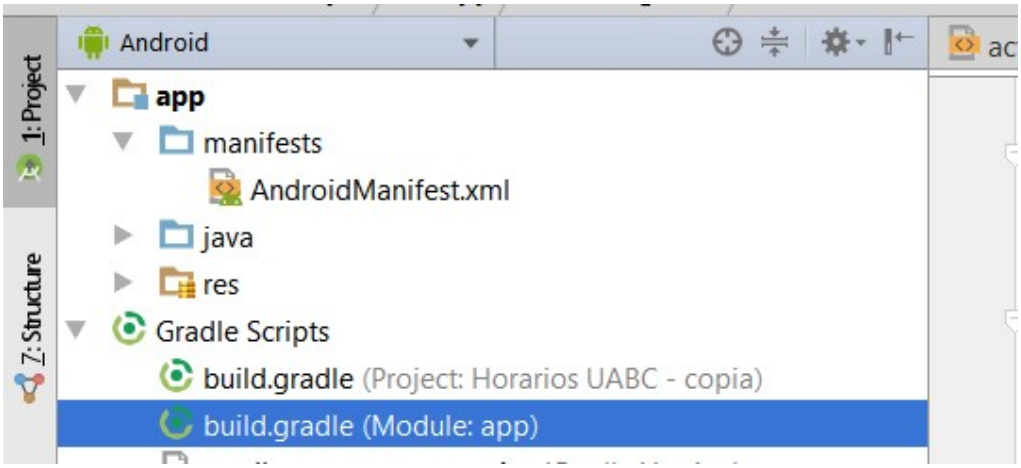
Después de realizar esto la aplicación de Android Studio nos dirá que tendremos que sincronizar el proyecto para obtener las clases y métodos de la librería que acabamos de agregar




Ya que el proyecto haya sincronizado los datos, procederemos a otorgarle a la aplicación los permisos de acceso a internet y acceso a el estado de red.

Estos permisos son primordiales para que el acceso pueda cumpliere ya que sin ellos Android no podría establecer la comunicación

Estos permisos son agregados en el archivo manifest de la siguiente manera



AndroidManifest.xml | LoginActivity.java | LoginRequest.java

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.samue.menulateral">

    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

## Layout de acceso

Es primordial crear un layout que nos proporcione el acceso a la base de datos mediante un nombre de usuario y una contraseña

Este layout contendrá los siguientes campos

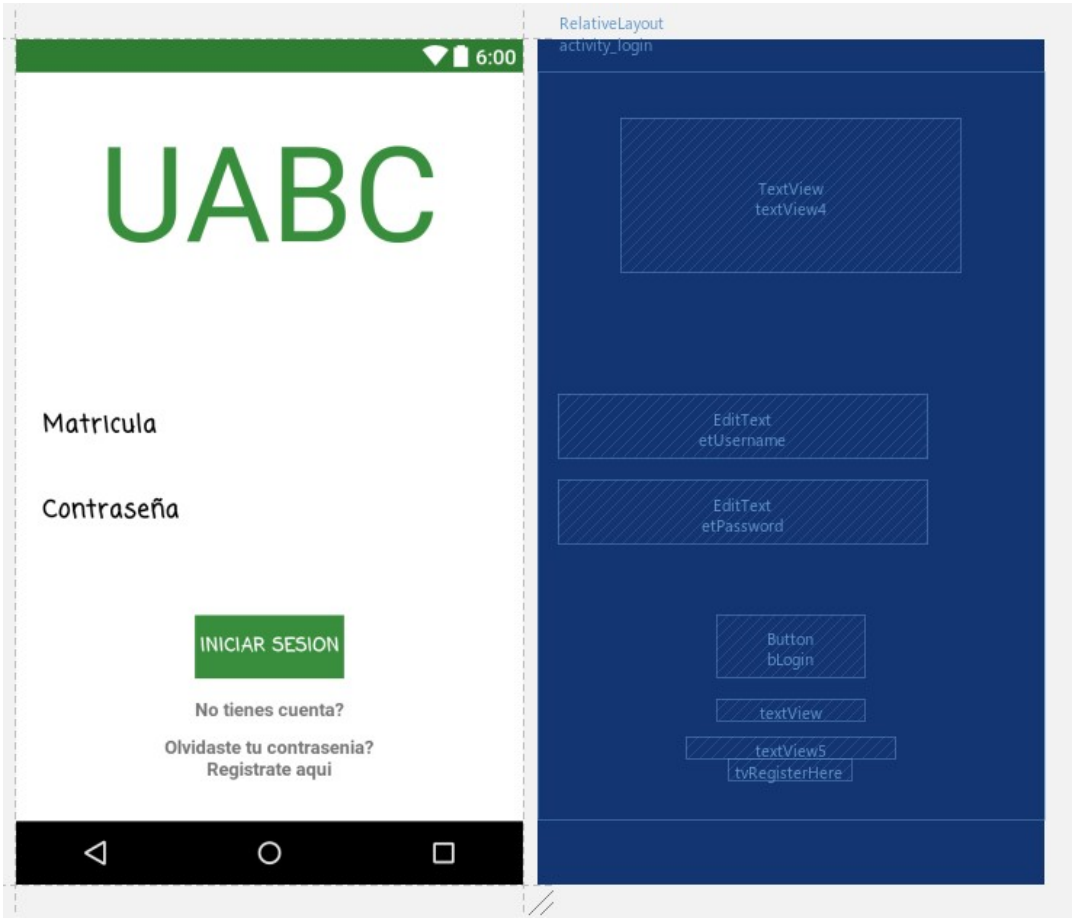
```xml
<EditText

    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="textPersonName"
    android:ems="10"
    android:id="@+id/etUsername"
    android:hint="Matricula"
    android:textStyle="bold"
    tools:textColorLink="@android:color/darker_gray"
    android:textColorLink="@color/colorPrimaryDark"
    android:fontFamily="casual"
    android:textColorHighlight="@android:color/background_dark"
    android:textColor="@android:color/background_dark"
    android:textColorHint="@android:color/black"
    android:layout_marginBottom="16dp"

    android:layout_above="@+id/etPassword"
    android:layout_alignParentStart="true" />
```

```
<EditText
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:inputType="textPassword"
    android:ems="10"
    android:id="@+id/etPassword"
    android:hint="Contraseña"
    android:fontFamily="casual"
    android:textColorHighlight="@android:color/background_dark"
    android:textColor="@android:color/background_dark"
    android:textColorHint="@android:color/black"
    android:layout_marginBottom="54dp"
    android:textStyle="bold"

    android:layout_above="@+id/bLogin"
    android:layout_alignParentStart="true" />
```

```
<Button
    android:textStyle="bold"
    android:text="Iniciar Sesion"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/bLogin"
    android:textColor="?attr/colorBackgroundFloating"
    android:background="@color/verdeCimarron"
    android:fontFamily="casual"
    android:layout_marginBottom="16dp"
    android:textColorLink="?attr/colorPrimary"
    android:layout_above="@+id/textView"
    android:layout_alignStart="@+id/textView" />
```

Con dos widgets de edición de textos que incluirán las credenciales que el usuario proporcionara y que la aplicación compara con la base para otorgar el acceso y otro que contendrá un botón que es el que marcara la acción para realizar este acceso.

El layout puede quedar de la siguiente manera



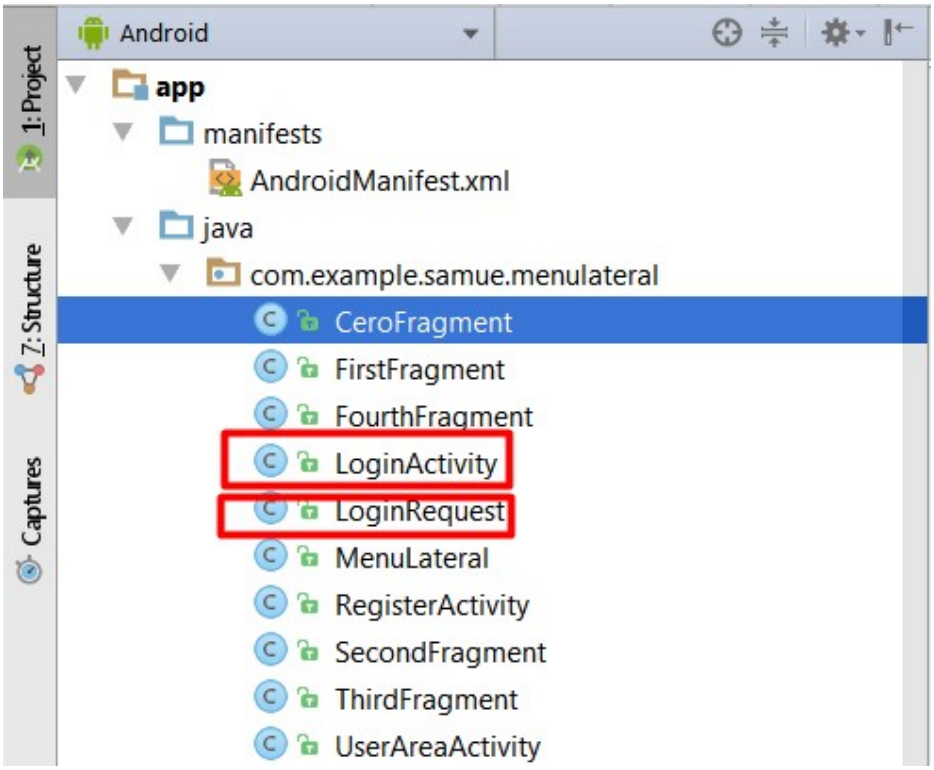
## LoginActivity y LoginRequest

Una vez creada la interfaz en la cual trabajaremos, creamos dos clases nuevas con el nombre de LoginActivity y LoginRequest

Estas dos clases trabajan simultáneamente para poder realizar la conexión a la base de datos que previamente te tuvo que haber realizado. Se trabajarán con las clases, métodos y objetos que nos proporciona la librería volley que son los siguientes

- **REQUEST:** Este METODO se encarga de gestionar automáticamente el envió de las peticiones, la administración de los hilos, la creación de la caché y la publicación de resultados en la UI.
- **RESPONSE:** Es una clase la cual contiene los métodos necesarios que requiere JSON
- **JSON:** Este objeto contiene un array de objetos con los atributos: **título**, **descripción** etc. , que viene desde el servidor. Todos los atributos son de tipo Sting y estos son traducíos por JSON de manera que se logra crear una comunicación entre el servidor y la aplicación

Creación de las clases:




## Estructura LoginRequest

### Llamada a la actividad y obtención de datos desde el layout

Después de haber inicializado la clase se procederá a llamar a la actividad que creamos para iniciar sesión en la base y obtener de ella lo que se escriba en los editText al igual que también obtener el momento en el que se realiza el clic en el y así realizar las acciones correspondientes. El código quedaría de la siguiente manera:

```
setContentView(R.layout.activity_login);
final EditText etUsername = (EditText) findViewById(R.id.etUsername);
final EditText etPassword = (EditText) findViewById(R.id.etPassword);
final Button bLogin = (Button) findViewById(R.id.bLogin);
```

Usamos el método **setContentView**, para mandar llamar el layout de inicio de sesión, después creamos dos variables de tipo **EditText** que son inicializadas con lo que contienen los **EditText** del layout pasándole como parámetro los **id** que contiene estos mismos, continuamos con una varible de tipo **Button** que se inicializa de igual manera con el **id** que contiene el botón que creamos en el layout

### Inicio del clic al botón

Se utiliza la variable que creamos para capturar el momento del clic del botón y le asignamos el método **SetOnClickListener** cargado en forma de parámetro con un objeto **View** que a su vez este carga consigo el método **onClickListener**, este método nos indicara cuando el botón reciba un clic y así poder crear en ese momento una cierta acción, que en este caso crear un método **onClick** con un objeto de tipo **View**, dentro de este método se crearan dos variables de tipo **String** para la obtención de los valores de los **EditText** y su conversión de estos mismo a variables de tipo **String**, la estructura del código quedara de la siguiente manera

```
bLogin.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        final String username = etUsername.getText().toString();
        final String password = etPassword.getText().toString();
```

### Creación de la nueva actividad y conectando con la base de datos utilizado la clase LoginRequest

Para conectarnos a la base de datos y comparar los valores que hemos obtenido de la aplicación se deberá hacer lo siguiente.

Se crea un objeto de tipo **Response.Listener** que contendrá valores de tipo **String** y así poder utilizar los métodos que contiene esta clase

```
Response.Listener<String> responseListener = new Response.Listener<String>()
```

Se sobrecargará el método onResponse y contendrá como parámetro una variable de tipo String

```
    @Override
    public void onResponse(String response) {
```

Después se creara un objeto de la clase que creamos LoginRequest la cual inicializaremos con las variables que contiene los editText ya convertidos a tipo String

Se creara una cola que de tipo Request que se inicializara con los intentos de solicitud de esta clase y se agrega a la cola el objeto que creamos de tipo LoginRequest

```
LoginRequest loginRequest = new LoginRequest(username,password,responseListener);
RequestQueue queue = Volley.newRequestQueue(LoginActivity.this);
queue.add(loginRequest);
```

LoginRequest

Esta clase contendrá los valores obtenidos con la aplicación y el servidor al cual nos queremos conectar y hará la comparación de estos valores con la de la base de datos para indicarnos si nos iguales y así continuar con el acceso a la aplicación

Su estructura es de la siguiente manera

```
package com.example.samue.menulateral;

import ...



public class LoginRequest extends StringRequest {
    private static final String LOGIN_REQUEST_URL = "http://al331386.net23.net/Login.php" ;//direccion del servidor de la base de dato
    private Map<String,String> params;

    public LoginRequest(String username, String password, Response.Listener<String> listener){
        super(Request.Method.POST,LOGIN_REQUEST_URL,listener,null);
        params = new HashMap<>();
        params.put("matricula",username);
        params.put("password",password);

    }

    @Override
    public Map<String, String> getParams() { return params; }
}
```

Al momento de inicializar la variable de tipo loginRequest que habíamos declarado anterior mente se hará uso de esta clase con los parámetros que se establecieron que son las variables username, password y responseListener, estas las recibirá el constructor y las inicializara. La primeras dos varaibles las meterá inicializadas con el nombre de matricula y password dentro de un hashMap

llamado params y la variable listener la usara para hacer la comunicación con el servidor que contiene la base de datos SQL, una ves que el constructor haya hecho su trabajo se creara una función llamada map la cual nos ayudara a retornar los el hashMap con los valores que se obtuvieron del servidor.

Una ves terminado el proceso de la clase LoginRequest se utilizara un try catch dentro del método onResponse para comparar los valores obtenidos del servidor y los valores que el usuario introdujo en la aplicación.

Se crear un objeto de tipo JSONObject el cual se inicializa con nuestra variable String que contiene el método onResponse llamado response

```
JSONObject jsonResponse = new JSONObject(response);
```

Después crearemos una variable de tipo booleano llamada success la cual la inicializamos con la variable jsonResponse con el método getBoolean y le pasamos como parámetro la palabra “succes”, esto nos indicara que si los datos que introdujimos coincidieron con los datos de la base de datos se inicializara con un valor verdadero

```
boolean success = jsonResponse.getBoolean("success");
```

Construiremos un if donde meteremos la variable succes la cual si contiene un valor verdadero entrara y mandara llamar la clase que contenga la actividad a desear que en este caso es un menú que fue creado como prueba

```
Intent intent = new Intent(LoginActivity.this,MenuLateral.class);
```

Si el valor que contiene la variable boleada es negativo entrara a un else que contiene una objeto de tipo AlertDialog llamado builder y a este se le asignara el mensaje de “usuario o contraseña incorrecta”, a su vez también contendrá un botón que diga reintentar para finalizar le mensaje. Se le asigna el método .créate() para que se cree y el método .show() para mostrarlo

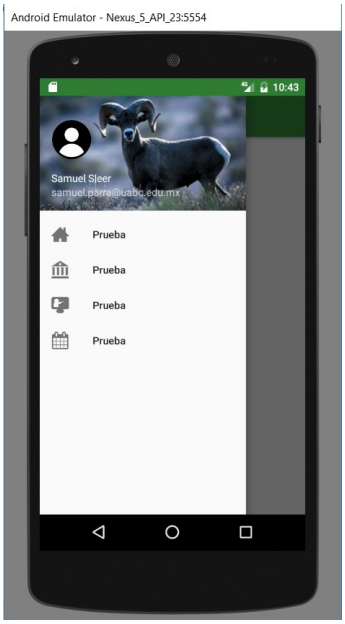
```
}else{
    AlertDialog.Builder builder = new AlertDialog.Builder(LoginActivity.this);
    builder.setMessage("usuario o contraseña incorrecta")
            .setNegativeButton("Reintentar",null)
            .create()
            .show();
}
```

La estructura completa del método quedaría de la siguiente manera

```
public void onClick(View v) {
    final String username = etUsername.getText().toString();
    final String password = etPassword.getText().toString();

    Response.Listener<String> responseListener = new Response.Listener<String>() {
        @Override
        public void onResponse(String response) {
            try {
                JSONObject jsonResponse = new JSONObject(response);
                boolean success = jsonResponse.getBoolean("success");
                if(success){
                    String name = jsonResponse.getString("name");
                   // int age = jsonResponse.getInt("age");

                    Intent intent = new Intent(LoginActivity.this,MenuLateral.class);
                    intent.putExtra("name",name);
                    intent.putExtra("matricula",username);
                  //  intent.putExtra("age",age);

                    LoginActivity.this.startActivity(intent);

                }else{
                    AlertDialog.Builder builder = new AlertDialog.Builder(LoginActivity.this);
                    builder.setMessage("usuario o contraseña incorrecta")
                            .setNegativeButton("Reintentar",null)
                            .create()
                            .show();
                }
            } catch (JSONException e) {
                e.printStackTrace();
            }
        }
    };
    LoginRequest loginRequest = new LoginRequest(username,password,responseListener);
    RequestQueue queue = Volley.newRequestQueue(LoginActivity.this);
    queue.add(loginRequest);
}
```

Con esto terminaríamos la aplicación ya 100% funcional, solo cabe recalcar que para que funcione se debe se haber creado previamente una base de datos en SQL y que los campos que contenga la tabla a donde se quiera acceder a cierto usuario coincidan con las variables que usamos en este código, así como también las los archivos query SQL que contiene el código PHP que utiliza la base de datos para las validaciones.



# Sensores

## 6.0 Aplicación de uso de sensores

Antes de la introducción de los teléfonos inteligentes, la gente interactuaba con una gama de sensores en la vida diaria. Cada sensor habitualmente residía en un solo dispositivo y usualmente diseñado para un solo propósito (sensores de temperatura del horno, sensores de presión de neumáticos, sistemas de control remoto, etc.). La introducción de los teléfonos inteligentes pone sensores en manos de usuarios y desarrolladores. Anteriormente, los sensores rara vez existían en tales cantidades o en una proximidad tan estrecha y continua con el usuario. La disponibilidad de los sensores múltiples en un solo dispositivo añade una amplia gama de usos para el dispositivo.

## 6.1 MEMS

Los sensores microelectromecánicos (MEMS) son sensores que se han hecho en una escala minúscula, usualmente en chips de silicio usando técnicas tomadas de la fabricación de chips de computadora. Todas los sensores de Android se hacen usando estas técnicas, pero técnicamente, el término sensor de MEMS se refiere a los que incorporan alguna parte de su diseño que físicamente se mueve o vibra: el sensor de presión, acelerómetro, giroscopio, y posiblemente la brújula son verdaderos sensores MEMS.

## 6.12 API

A partir de Android 1.5 (API nivel 3), un conjunto estándar de sensores y las librerías API se ha puesto a disposición. En Android 2.3 (API nivel 9), se agregaron nuevos sensores y herramientas a la caja de utilerías del desarrollador de Android. Los sensores estándar ahora incluyen el acelerómetro, giroscopio, magnetómetro (brújula), sensor de luz, sensor de proximidad, sensor de humedad relativa Y sensor de presión. Las herramientas añadidas en el nivel 9 de API incluyen métodos para obtener matrices de rotación, y sensores "sintéticos". Estos proporcionan a desarrolladores una amplia gama de opciones para navegación física, control de juegos, realidad, y muchos otros usos.

## 6.2.2 CLASES DEL API

El Android Sensor API consta de clases para solicitar y procesar la información de los sensores desde el hardware de un dispositivo. Esta sección describe las clases dentro de la API de Sensor de Android e ilustra cómo utilizar las clases. El punto de entrada a la API es la clase SensorManager, que permite a una aplicación solicitar un sensor información y registro para recibir los datos del sensor. Cuando se registran, los valores de los datos del sensor se envían a una SensorEventListener en la forma de un SensorEvent que contiene información producida a partir de un Sensor dado. Esto se retomará más adelante con un ejemplo

## 6.3 Tipos de sensores

Los sensores a los que se hace referencia a través de la clase Sensor pueden ser de dos tipos.

Sintético (o compuesto o virtual). Los sensores sin procesar proporcionan datos brutos de un sensor y un sensor sin procesar corresponde a un componente físico real dentro del dispositivo Android.

Los sensores sintéticos proporcionan una capa de abstracción entre el código de aplicación y el dispositivo de bajo nivel componentes combinando los datos brutos de varios sensores sin procesar, o bien modificando los datos de sensor sin procesar para que sea más fácil de consumir. Pueden reportar una cantidad física refiriéndose a dos o tres sensores (tales como informar la orientación haciendo referencia a la brújula, que da un rumbo norte-sur-este-oeste y el acelerómetro, que da la inclinación).

Los sensores sintéticos pueden manipular la lectura del sensor antes de reportarlo; Por ejemplo, para integrar los datos del giroscopio ,se hace una comparación interna de datos con el giroscopio.

### 6.4 Sensor Manager

SensorManager es el servicio de sistema Android que le da a una aplicación acceso a sensores de hardware. Otros servicios del sistema, permite que las aplicaciones registren y cancelen el registro de eventos relacionados con los sensores. Una vez registrado, una aplicación recibirá datos de los sensores del hardware. Además de permitir que una aplicación registre datos de sensores, SensorManager también proporciona métodos que procesan los datos del sensor.

SensorManager.getOrientation () es un ejemplo de un método que Utiliza datos de sensores para generar información de orientación del dispositivo

### 6.5 Sensor Rates

Es una variable que indica el retraso en la tasa de escucha de los sensores , esto se ajusta según a la finalidad que se requiera ya sea solo tener noción de los datos del sensor o tener mucha precisión. Las constantes a usar son las siguientes:

- SENSOR_DELAY_FASTEST
- SENSOR_DELAY_GAME
- SENSOR_DELAY_UI
- SENSOR_DELAY_NORMAL

### 6.6 Giroscopio

Los giroscopios MEMS son también pequeñas masas en pequeños resortes, pero en lugar de medir la aceleración, están diseñados para medir una fuerza diferente - la llamada fuerza de Coriolis debido a la rotación. El Coriolis es una fuerza en la tendencia de un objeto libre a viraje fuera de curso cuando se ve desde una referencia giratoria , por ejemplo, cuando usted está sentado en un carrusel y tirar un balón lejos de usted, el la bola parece alejarse de una línea recta como si hubiera una fuerza que actúa sobre ella.

Esta fuerza es llamada la fuerza de Coriolis. Es una "fuerza ficticia" porque cuando se ve de alguien de pie al lado al carrusel, ninguna fuerza actúa sobre la pelota simplemente está rodando en línea recta como lo haría esperan de la física newtoniana. En el mundo de MEMS, el giroscopio trabaja empujando una masa minúscula hacia adelante y hacia atrás a lo largo de un eje.

Cuando el giroscopio se gira, la fuerza de Coriolis hace que la masa se aleje de la dirección vibrando, y comenzando a moverse a lo largo de un eje diferente. Se percibe el movimiento a lo largo de este nuevo eje eléctricamente, usando placas del condensador una placa del condensador se fija al marco y una se fija a la masa del móvil.

La fuerza de Coriolis actúa sólo cuando el dispositivo está girando, por lo tanto los giroscopios miden sólo ángulos de velocidad, o, la velocidad a la cual el dispositivo está girando. Cuando el dispositivo está parado, independientemente de dirección a la que apunta el dispositivo, los tres ejes del giroscopio medirán cero

### 6.7 Unidades de sensor

Android informa los valores en radianes por segundo alrededor de los ejes estándar x, y, z. Se sigue la convención matemática estándar: si el eje en cuestión está apuntando hacia usted, los valores positivos indican rotaciones en sentido contrario a las agujas del reloj. Esto es dado por la regla de la mano derecha

Un rango máximo típico a esperar es alrededor de 35 grados / segundo (0.61 rad / s), y una resolución típica es alrededor de 0,001 grados / segundo.

### 6.8 Magnetómetro

Los sensores de campo magnético pueden operar bajo una variedad de métodos diferentes según el fabricante y arquitectura pueden utilizar el efecto Hall, materiales magnetorresistivos, o las fuerzas de Lorentz.

Los sensores de efecto Hall comprenden actualmente la mayor cuota de mercado de magnetómetros y simplemente pasando una corriente a través de un alambre. Un componente de campo magnético perpendicular a ese alambre hace que los electrones tengan mayor densidad en un lado del cable comparado con el otro, lo que da como resultado una tensión a través de la anchura del alambre que es proporcional al campo magnético. Lorentz los sensores de fuerza son similares pero

miden una deflección mecánica del cable en lugar de una tensión a través de el ancho del alambre.
Independientemente del mecanismo físico, los sensores de campo magnético informan sobre el campo magnético en x, y, z (por tener tres sensores separados, uno alineado a lo largo de cada eje).

### 6.8.1 Unidades de sensor, rango y resolución

Android informa campos magnéticos en microtesla. Una gama dinámica típica es alrededor 2000 micro tesla.

La resolución para el sensor de campo magnético es de 0,1 microtesla. El campo magnético de la Tierra puede andar entre 30 microtesla a 60 microtesla, y sobre los EE.UU. el valor varía desde alrededor de 58 microtesla en Dakota del Norte está alrededor de 48 microtesla en el sur de Texas, y estos valores a la deriva con el tiempo. Sin embargo, el valor absoluto no importa mucho, y éstos deben ser tomados solamente para tener referencia de los datos con estimaciones que los magnetómetros basados en MEMS tienen.

En el entorno local (la presencia de metal cercano, incluso muchos no magnéticos Metales), efectos histéricos (el efecto de la historia ambiental, por ejemplo, si un cuerpo metálico o imán estaba cerca del sensor y después se quitaba, puede haber cambiado la lectura en el sensor), esto hace que los valores medidos cambien con el tiempo.
Si se desea una mayor precisión para la magnitud de la medición, la clase android.hardware Geomagnetic Field calculará la magnitud y la dirección del campo magnético en un punto de la Tierra dando valores más precisos

### 6.9 App de Brújula usando acelerómetro , giroscopio y magnetómetro.

Para iniciar la clase main que tenemos le tenemos que implementar la interfaz SensorListener , la cual es una interfaz. Implementamos los métodos que nos da, los cuales explicaremos más adelante.

Variables:

```
private SensorManager sensorManager;
private Sensor gsensor;
private Sensor msensor;
```

SensorManager como se había comentado antes es el manejador de los sensores y es la clase padre de todos los sensores.
Posteriormente creamos dos variables , una de ellas es para manejar el magnetómetro y el otro es para manejar el giroscopio.

```
public ImageView arrowView = null;
```

Creamos un objeto de tipo ImageView que nos ayudará a girar la imagen que nos indicará la posición de la imagen, rotando por ángulos, como se verá a continuación.

```
public Compass(Context context) {
    sensorManager = (SensorManager) context
            .getSystemService(Context.SENSOR_SERVICE);
    gsensor = sensorManager.getDefaultSensor(Sensor.TYPE_GYROSCOPE);
    msensor = sensorManager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);
}
```

En el constructor inicializamos los sensores con su respectiva variable. Primeramente instanciamos el manejador de sensores y con este podemos designar cada variable a su respectivo tipo.

```
public void start() {
    sensorManager.registerListener(this, gsensor,
            SensorManager.SENSOR_DELAY_GAME);
    sensorManager.registerListener(this, msensor,
            SensorManager.SENSOR_DELAY_GAME);
}
```

Dentro del ciclo de vida de una aplicación encontramos el método start. En este colocaremos los listener , que estarán registrando cada cierto tiempo según los inicializamos como se muestra, en este caso es con la constante DELAY_GAME, que es aproximadamente 30 ms.

```
private void adjustArrow() {
    if (arrowView == null) {
        Log.i(TAG, "arrow view is not set");
        return;
    }

    Log.i(TAG, "will set rotation from " + currectAzimuth + " to "
            + azimuth);

    Animation an = new RotateAnimation(-currectAzimuth, -azimuth,
            Animation.RELATIVE_TO_SELF, 0.5f, Animation.RELATIVE_TO_SELF,
            0.5f);
    currectAzimuth = azimuth;

    an.setDuration(500);
    an.setRepeatCount(0);
    an.setFillAfter(true);

    arrowView.startAnimation(an);
}
```

```
public void onSensorChanged(SensorEvent event) {
    final float alpha = 0.97f;

    synchronized (this) {
        if (event.sensor.getType() == Sensor.TYPE_ACCELEROMETER) {

            mGravity[0] = alpha * mGravity[0] + (1 - alpha)
                    * event.values[0];
            mGravity[1] = alpha * mGravity[1] + (1 - alpha)
                    * event.values[1];
            mGravity[2] = alpha * mGravity[2] + (1 - alpha)
                    * event.values[2];

            // mGravity = event.values;

            // Log.e(TAG, Float.toString(mGravity[0]));
        }

        if (event.sensor.getType() == Sensor.TYPE_MAGNETIC_FIELD) {
            // mGeomagnetic = event.values;

            mGeomagnetic[0] = alpha * mGeomagnetic[0] + (1 - alpha)
                    * event.values[0];
            mGeomagnetic[1] = alpha * mGeomagnetic[1] + (1 - alpha)
                    * event.values[1];
            mGeomagnetic[2] = alpha * mGeomagnetic[2] + (1 - alpha)
                    * event.values[2];
            // Log.e(TAG, Float.toString(event.values[0]));

        }
```

Este es uno de los métodos  que se instancian al implementar la interfaz SensorListener. Este método se activará cada vez que se encuentre un cambio en los sensores. Como esta función se activa en un hilo independiente al de la aplicación, se sincroniza para que solo una instancia pueda acceder a ella.

Vemos de qué tipo de sensor es el que está ocurriendo el evento. Tenemos un vector inicializado en ceros el cual guardará el cambio de posición según lo que indique el respectivo sensor.

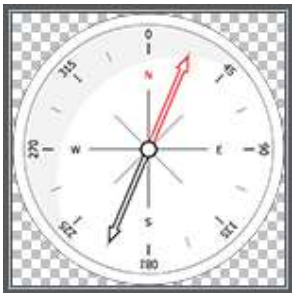
Para realizar el cálculo de posición se realiza la siguiente operación, que es una constante de cambio muy pequeña por el valor anterior de esa posición sobre uno de los ejes más  la constante menos uno,  esto hace que la variable se haga más pequeña, esto más el valor indicado por el sensor. Como se muestra en la imagen.

```
            float R[] = new float[9];
            float I[] = new float[9];
            boolean success = SensorManager.getRotationMatrix(R, I, mGravity,
                    mGeomagnetic);
            if (success) {
                float orientation[] = new float[3];
                SensorManager.getOrientation(R, orientation);
                // Log.d(TAG, "azimuth (rad): " + azimuth);
                azimuth = (float) Math.toDegrees(orientation[0]); // orientation
                azimuth = (azimuth + 360) % 360;
                // Log.d(TAG, "azimuth (deg): " + azimuth);
                adjustArrow();
            }
        }
    }
```

Finalmente para hacer la rotación de la imagen lo hacemos por medio de una animación, para que el cambio no se vea brusco y se vea correspondiente al movimiento realizado. Primeramente se calcula el ángulo de movimiento con una regla de tres y se llama a la función adjust Arrow que ajusta la imagen en base a las animaciones predefinidas en android.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent" android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp" tools:context=".CompassActivity">

    <ImageView
        android:id="@+id/main_image_dial"
        android:layout_width="320dp"
        android:layout_height="320dp"
        android:layout_centerInParent="true"
        android:contentDescription="Compass Dial"
        android:src="@drawable/dial" />

    <ImageView
        android:id="@+id/main_image_hands"
        android:layout_width="20dp"
        android:layout_height="240dp"
        android:layout_centerInParent="true"
        android:contentDescription="Compass Hands"
        android:scaleType="fitXY"
        android:src="@drawable/hands" />

</RelativeLayout>
```

Finalmente creamos un XML, donde se coloca la imagen de la brújula y la imagen de una flecha que se irá ajustando con el movimiento y los cálculos antes definidos.



Esta la imagen de la brújula que se colocó en el imageview que se controlara por medio del código, como se explicó anteriormente

# Face Detection

# Manual para Face Detection en Android

Este documento contendrá una introducción a la librería de Vision en Play Services 8.1, llamada Face Detection esta es una herramienta bastante útil para los desarrolladores que se centran en la localización de Caras humanas. Una vez que tienes la lista de caras detectadas en una imagen, puedes obtener a la información de dicha cara, así como su orientación, la forma en que sonríe, o saber si una persona esta con los ojos abiertos o cerrados sea cual sea el caso, además de especificar si tiene marcas especificas en su cara.

Esta información que nos proporciona puede ser muy útil para múltiples funciones, así como la app de la cámara que automáticamente toma una fotográfica cuando todos tienen una sonrisa en su rostro y además los ojos abiertos, también nos puede servir para agregar animaciones o caricaturas a nuestra imagen ya que es capaz de ubicar en donde se encuentran nuestros ojos o boca, etc. Es importante mencionar que la librería de Face Detection no es para reconocimiento facial.
La información que nos proporciona acerca de la cara, es información que no es utilizable para la librería de Visión que pueda determinar si dos caras provienen la una misma persona.

En este manual daremos un ejemplo de cómo podemos utilizar la Face Detection API para obtener información acerca de una fotografía donde este ilustrada alguna persona.

# 1. Preparación del Proyecto

Para añadir la librería Vision a nuestro proyecto, primero debemos importar Play Services 8.1 o una versión mayor en nuestro proyecto.

Solo vamos a importar la librería de Play Services Vision en nuestra pestaña de **build.gradle** y agregamos la siguiente línea de compilación al nodo de "dependencies".

```
compile 'com.google.android.gms:play-services-vision:8.1.0'
```

Una vez que tenemos nuestro Play Services en nuestro proyecto, podemos cerrar la pestaña de **build.gradle** y abrir el **AndroidManifest.xml.** Necesitamos agregar un objeto "meta-data" definiendo que nuestra aplicación trabajar con este nodo. Esto permite que la librería de Vision sepa que nuestro plan es detectar las caras dentro de nuestra aplicación.

```
<meta-data android:name="com.google.android.gms.vision.DEPENDENCIES" android:value="face"/>
```

Una vez que tenemos el **AndroidManisfest.xml** listo, podemos cerrarlo.

Ahora vamos a crear una clase llamada **FaceOverlayView.java,** esta clase la extendemos de View y contiene la lógica para detectar las caras, desplegando un **Bitmap** que fue analizado y dibujado en la parte superior de la imagen indicando e ilustrando los puntos.

Por ahora empezaremos añadiendo las variables de nuestra clase y definiendo los constructores. El objeto **Bitmap** lo utilizaremos para guardar y analizar el

**SparseArray** de Caras que estaremos guardando y cada cara se podrá encontrar dentro del **Bitmap**.

```
public class FaceOverlayView extends View {

    private Bitmap mBitmap;
    private SparseArray<Face> mFaces;

    public FaceOverlayView(Context context) {
        this(context, null);
    }

    public FaceOverlayView(Context context, AttributeSet attrs) {
        this(context, attrs, 0);
    }

    public FaceOverlayView(Context context, AttributeSet attrs, int defStyleAttr)
        super(context, attrs, defStyleAttr);
    }
}
```

Después,  agregamos un nuevo método dentro de FaceOverlayView llamado setBitmap(Bitmap bitmap). Por ahora solo le pasaremos simple el bitmap.

Cuando tengamos nuestra imagen seleccionada la agregamos a nuestro directorio res/raw.

Abrimos MainActivity y creamos un obejto de FaceOverlayView desde onCreate(). Después le damos la referencia hacia View, leyendo la imagen que seleccionamos.

```
public class MainActivity extends AppCompatActivity {

    private FaceOverlayView mFaceOverlayView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mFaceOverlayView = (FaceOverlayView) findViewById( R.id.face_overla

        InputStream stream = getResources().openRawResource( R.raw.face );
        Bitmap bitmap = BitmapFactory.decodeStream(stream);

        mFaceOverlayView.setBitmap(bitmap);

    }
}
```

## 2. Detectando las Caras

Ya que tenemos nuestro proyecto listo, ahora empezaremos a hacer la parte de detección de caras. Vamos a crear un objeto que se llama FaceDetector. Para hacer estoy necesitamos importar FaceDetector.Builder, esto nos ayudara a definir los diferentes parámetros que afectan que tan rápido va a detectar nuestro FaceDetector.

```
FaceDetector detector = new FaceDetector.Builder( getContext() )
        .setTrackingEnabled(false)
        .setLandmarkType(FaceDetector.ALL_LANDMARKS)
        .setMode(FaceDetector.FAST_MODE)
        .build();
```

Ahora que tenemos nuestro detector de caras en nuestra imagen, podemos utilizarlo.

Aquí vamos a agregar nuestras caras que tenemos en el Bitmap a un Canvas para que esta función dibuje lo que tiene dentro. Este método primero lo dibuja en el Bitmap y después en el Canvas.

```
@Override
protected void onDraw(Canvas canvas) {
    super.onDraw(canvas);

    if ((mBitmap != null) && (mFaces != null)) {
        double scale = drawBitmap(canvas);
        drawFaceBox(canvas, scale);
    }
}
```

El drawBitmap(Canvas nameCanvas) dibuja el Bitmap en el canvas y lo escala para que nuestras dimensiones sean las correctas.

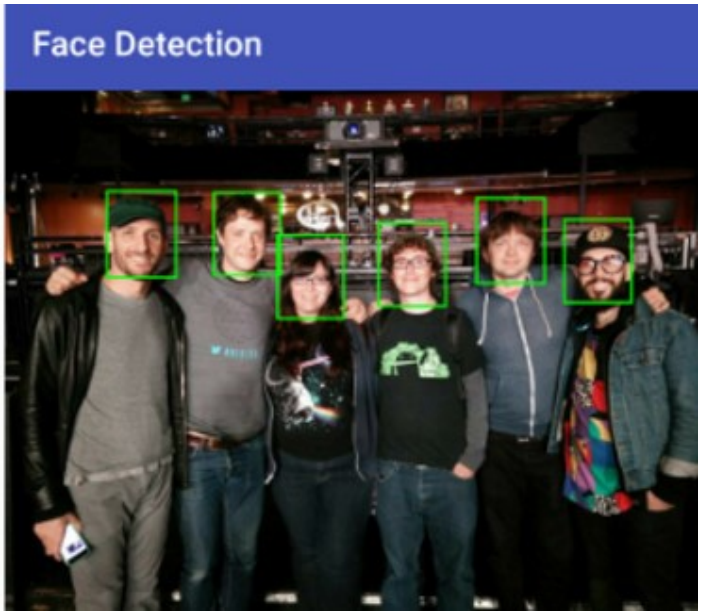
```
private double drawBitmap( Canvas canvas ) {
    double viewWidth = canvas.getWidth();
    double viewHeight = canvas.getHeight();
    double imageWidth = mBitmap.getWidth();
    double imageHeight = mBitmap.getHeight();
    double scale = Math.min( viewWidth / imageWidth, viewHeight / imageHeight );

    Rect destBounds = new Rect( 0, 0, (int) ( imageWidth * scale ), (int) ( imageH
    canvas.drawBitmap( mBitmap, null, destBounds, null );
    return scale;
}
```

El método de drawFaceBox es un poco mas extenso, este es el que se va ganar el sueldo. Ya que este método nos va detectar cada cara y va a guardar la posición exacta de los valores de arriba y de la izquierda de cada cara. Este método tomara la posición de la cara y dibujara un Recuadro verde alrededor de la misma.

```
private void drawFaceBox(Canvas canvas, double scale) {
    //paint should be defined as a member variable rather than
    //being created on each onDraw request, but left here for
    //emphasis.
    Paint paint = new Paint();
    paint.setColor(Color.GREEN);
    paint.setStyle(Paint.Style.STROKE);
    paint.setStrokeWidth(5);

    float left = 0;
    float top = 0;
    float right = 0;
    float bottom = 0;

    for( int i = 0; i < mFaces.size(); i++ ) {
        Face face = mFaces.valueAt(i);

        left = (float) ( face.getPosition().x * scale );
        top = (float) ( face.getPosition().y * scale );
        right = (float) scale * ( face.getPosition().x + face.getWidth() );
        bottom = (float) scale * ( face.getPosition().y + face.getHeight()

        canvas.drawRect( left, top, right, bottom, paint );
    }
}
```

Utilizaremos un ciclo For para encontrar las posiciones de cada cara.

Y en este punto ya deberíamos se capaces de correr nuestras aplicación y detectando las caras dentro de rectángulos de color verde.




Y por último vamos a agregarle una función más para la detección de los rasgos de la cara, así como ojos, boca, nariz.

```
private void drawFaceLandmarks( Canvas canvas, double scale ) {
    Paint paint = new Paint();
    paint.setColor( Color.GREEN );
    paint.setStyle( Paint.Style.STROKE );
    paint.setStrokeWidth( 5 );

    for( int i = 0; i < mFaces.size(); i++ ) {
        Face face = mFaces.valueAt(i);
```

# BlueTooth

**Introducción:**

Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia. Los principales objetivos que se pretenden conseguir con esta norma son:

- Facilitar las comunicaciones entre equipos móviles.
- Eliminar los cables y conectores entre estos.
- Ofrecer la posibilidad de crear pequeñas redes inalámbricas y facilitar la sincronización de datos entre equipos personales.

**Introducción al entorno de programación:**

La plataforma de Android incluye compatibilidad con la pila de red Bluetooth, la cual permite que un dispositivo intercambie datos de manera inalámbrica con otros dispositivos Bluetooth. El framework de la aplicación proporciona acceso a la funcionalidad Bluetooth mediante las Android Bluetooth API. Estas API permiten a las aplicaciones conectarse de manera inalámbrica con otros dispositivos Bluetooth y habilitan las funciones inalámbricas punto a punto y de multipunto.
Con las Bluetooth API, una aplicación de Android puede realizar lo siguiente:

- buscar otros dispositivos Bluetooth;
- consultar el adaptador local de Bluetooth en busca de dispositivos Bluetooth sincronizados;
- establecer canales RFCOMM;
- conectarse con otros dispositivos mediante el descubrimiento de servicios;
- transferir datos hacia otros dispositivos y desde estos;
- administrar varias conexiones.

**Conceptos básicos:**
En este documento, se describe la manera de usar las Android Bluetooth API a fin de llevar a cabo las cuatro tareas principales necesarias para establecer la comunicación a través de Bluetooth: configuración de Bluetooth, búsqueda de dispositivos sincronizados o disponibles en el área local, conexión con dispositivos y transferencia de datos entre dispositivos.

Todas las Bluetooth API están disponibles en el paquete de android.bluetooth. A continuación, se ofrece un resumen de las clases e interfaces que necesitarás para crear conexiones Bluetooth:
BluetoothAdapter
Representa el adaptador local de Bluetooth (radio Bluetooth). El BluetoothAdapter es el punto de entrada de toda interacción de Bluetooth. Gracias a esto, puedes ver otros dispositivos Bluetooth, consultar una lista de los dispositivos conectados (sincronizados), crear una instancia de BluetoothDevice mediante una dirección MAC conocida y crear un BluetoothServerSocket para oír comunicaciones de otros dispositivos.
BluetoothDevice
Representa un dispositivo Bluetooth remoto. Usa esto para solicitar una conexión con un dispositivo remoto mediante un BluetoothSocket o consultar información sobre el dispositivo, como su nombre, dirección, clase y estado de conexión.
BluetoothSocket
Representa la interfaz de un socket de Bluetooth (similar a un Socket de TCP). Este es el punto de conexión que permite que una aplicación intercambie datos con otro dispositivo Bluetooth a través de InputStream y OutputStream.
BluetoothServerSocket
Representa un socket de servidor abierto que recibe solicitudes entrantes (similar a un ServerSocket de TCP). A fin de conectar dos dispositivos Android, un dispositivo debe abrir un socket de servidor con esta clase. Cuando un dispositivo Bluetooth

remoto realice una solicitud de conexión a este dispositivo, el BluetoothServerSocket mostrará un BluetoothSocket conectado una vez que la conexión se acepte.

BluetoothClass

Describe las características y capacidades generales de un dispositivo Bluetooth. Se trata de un conjunto de propiedades de solo lectura que define las clases del dispositivo mayor y menor y sus servicios. Sin embargo, esto no describe de manera confiable todos los perfiles de Bluetooth y los servicios compatibles con el dispositivo, pero resulta útil como sugerencia para el tipo de dispositivo.

BluetoothProfile

Interfaz que representa un perfil de Bluetooth. Un perfil de Bluetooth es una especificación de interfaz inalámbrica para la comunicación entre dispositivos basada en Bluetooth. Un ejemplo es el perfil de manos libres. Para obtener más información sobre los perfiles, consulta Trabajo con perfiles

BluetoothHeadset

Brinda compatibilidad para que el uso de auriculares Bluetooth con teléfonos móviles. Esto incluye los perfiles de manos libres (v1.5) y de auriculares Bluetooth.

BluetoothA2dp

Define la manera en que puede transmitirse audio de alta calidad de un dispositivo a otro a través de la conexión Bluetooth. “A2DP” significa “perfil de distribución de audio avanzada”.

BluetoothHealth

Representa un proxy de perfil de dispositivos de salud que controla el servicio Bluetooth.

BluetoothHealthCallback

Clase abstracta que se usa para implementar callbacks de BluetoothHealth. Debes extender esta clase e implementar los métodos de callback para recibir actualizaciones sobre los cambios en el estado de registro de la aplicación y el estado de canal de Bluetooth.

BluetoothHealthAppConfiguration

Representa una configuración de aplicación que la aplicación de salud de terceros de Bluetooth registra para comunicarse con un dispositivo de salud Bluetooth remoto.

BluetoothProfile.ServiceListener

Interfaz que notifica a los clientes de IPC del BluetoothProfile cuando se conectan al servicio o se desconectan de este (es decir, el servicio interno que ejecuta un perfil en particular).

**Permisos de Bluetooth:**

A fin de usar las funciones de Bluetooth en tu aplicación, debes declarar el permiso de Bluetooth BLUETOOTH. Necesitas este permiso para establecer cualquier comunicación de Bluetooth, como solicitar o aceptar una conexión y transferir datos. Si deseas que tu app inicie la detección de dispositivos o controle los ajustes de Bluetooth, también debes declarar el permiso BLUETOOTH_ADMIN . La mayoría de

las aplicaciones necesitan este permiso solamente para poder ver dispositivos Bluetooth locales.
Declara los permisos de Bluetooth del archivo de manifiesto de tu aplicación. Por ejemplo:

```
<uses-permission android:name="android.permission.BLUETOOTH"/>
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
```

**Conjuración de adaptador:**

Para que tu aplicación pueda comunicarse a través de Bluetooth, debes verificar que Bluetooth sea compatible con el dispositivo y, si es así, asegurarte de que esté habilitado.
El BluetoothAdapter es obligatorio para toda actividad de Bluetooth. Para obtener el BluetoothAdapter, llama al método estático del getDefaultAdapter(). Esto muestra un BluetoothAdapter que representa el propio adaptador de Bluetooth del dispositivo (la radio Bluetooth). Existe un adaptador de Bluetooth para todo el sistema y tu aplicación puede interactuar con él usando este objeto. Si getDefaultAdapter() muestra null, significa que el dispositivo no es compatible con Bluetooth y que tu tarea finaliza aquí. Por ejemplo:

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
    // Device does not support Bluetooth
}
```

Importaciones:
Es importante importar a nuestros archivos java las clases que vamos a usar, de lo contrario nos marcara un error en el nombre de las variables, lo único que tenemos que hacer es poner esto en los importes de nuestro respectivo archivo java:

```
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
```

**Habilita Bluetooth:**
A continuación, debes asegurarte de que Bluetooth esté habilitado. Llama al isEnabled() para verificar si Bluetooth se encuentra actualmente habilitado. Si este método muestra false, Bluetooth no estará habilitado.

```
if (!mBluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

**Consulta de dispositivos sincronizados**

Antes de llevar a cabo la detección de dispositivos, es importante consultar el conjunto de dispositivos sincronizados a fin de ver si el dispositivo deseado ya es conocido. Para ello, llama a getBondedDevices(). Esto mostrará un conjunto de BluetoothDevice que representa los dispositivos sincronizados. Por ejemplo, puedes consultar todos los dispositivos sincronizados y luego mostrar el nombre de cada uno al usuario mediante ArrayAdapter:

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();
// If there are paired devices
if (pairedDevices.size() > 0) {
    // Loop through paired devices
    for (BluetoothDevice device : pairedDevices) {
        // Add the name and address to an array adapter to show in a ListView
        mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
    }
}
```

**Detección de dispositivos:**
Para comenzar a detectar dispositivos, simplemente, llama a startDiscovery(). El proceso es asincrónico y el método mostrará inmediatamente un booleano que indica si la visibilidad se inició con éxito. Por lo general, el proceso de visibilidad incluye un escaneo de consulta de 12 segundos aproximadamente, seguido de un escaneo de la página de cada dispositivo encontrado a fin de recuperar su nombre de Bluetooth.

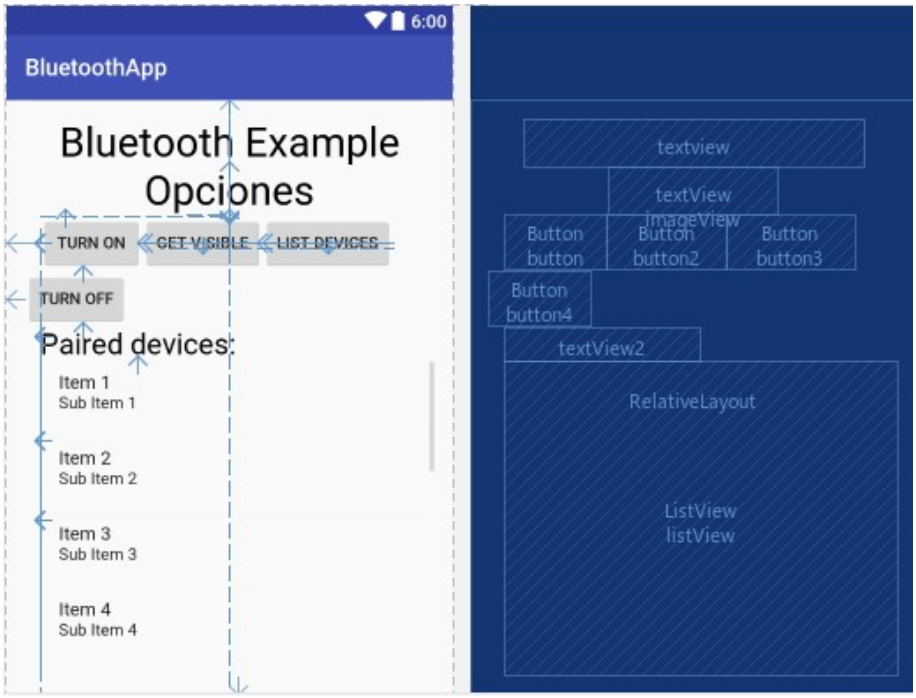
Tu aplicación debe registrar un BroadcastReceiver para la intent ACTION_FOUND a fin de recibir información sobre cada dispositivo detectado. Para cada dispositivo, el sistema transmitirá la intent ACTION_FOUND. Esta intent incluye los campos extra EXTRA_DEVICE y EXTRA_CLASS, que contienen un BluetoothDevice y una BluetoothClass, respectivamente. Por ejemplo, a continuación se describe cómo puedes registrarte para manejar la transmisión cuando se detectan los dispositivos:

```
// Create a BroadcastReceiver for ACTION_FOUND
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        // When discovery finds a device
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            // Get the BluetoothDevice object from the Intent
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            // Add the name and address to an array adapter to show in a ListView
            mArrayAdapter.add(device.getName() + "\n" + device.getAddress());
        }
    }
};
// Register the BroadcastReceiver
IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
registerReceiver(mReceiver, filter); // Don't forget to unregister during onDestroy
```

**Programa:**

Ahora vamos a  hacer un breve programa:, dicho programa mostrara una interface donde se podrá activar el bluetooth, desactivar, obtener los dispositivos sincronizados el, hacer visible nuestro dispositivo para que otros dispositivos puedan verlo.

Creamos una interfaz que contenga 4 botones, y un listView. La interface principal es como se muestra a continuación:




Ahora en la parte de código lo que tenemos que programar es lo siguiente:
Declaramos nuestros atributos de la clase:

```
public class MainActivity extends Activity  {
    Button b1,b2,b3,b4;
    private BluetoothAdapter BA;
    private Set<BluetoothDevice>pairedDevices;
    ListView lv;

    @Override
```

Asociamos los objetos de nuestro archivo .Xml con variables que vamos a poder modificar:

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    b1 = (Button) findViewById(R.id.button);
    b2=(Button)findViewById(R.id.button2);
    b3=(Button)findViewById(R.id.button3);
    b4=(Button)findViewById(R.id.button4);

    BA = BluetoothAdapter.getDefaultAdapter();
    lv = (ListView)findViewById(R.id.listView);
}
```

Programamos nuestros primeros dos botones de encendido y apagado de bluetooth:

```
public void on(View v){
    if (!BA.isEnabled()) {
        Intent turnOn = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
        startActivityForResult(turnOn, 0);
        Toast.makeText(getApplicationContext(), "Turned on",Toast.LENGTH_LONG).show();
    } else {
        Toast.makeText(getApplicationContext(), "Already on", Toast.LENGTH_LONG).show();
    }
}

public void off(View v){
    BA.disable();
    Toast.makeText(getApplicationContext(), "Turned off" ,Toast.LENGTH_LONG).show();
}
```

Después los siguientes dos: el de hacer visible nuestro dispositivo y el de mostrar los dispositivos sincronizados.

Manejo de imágenes

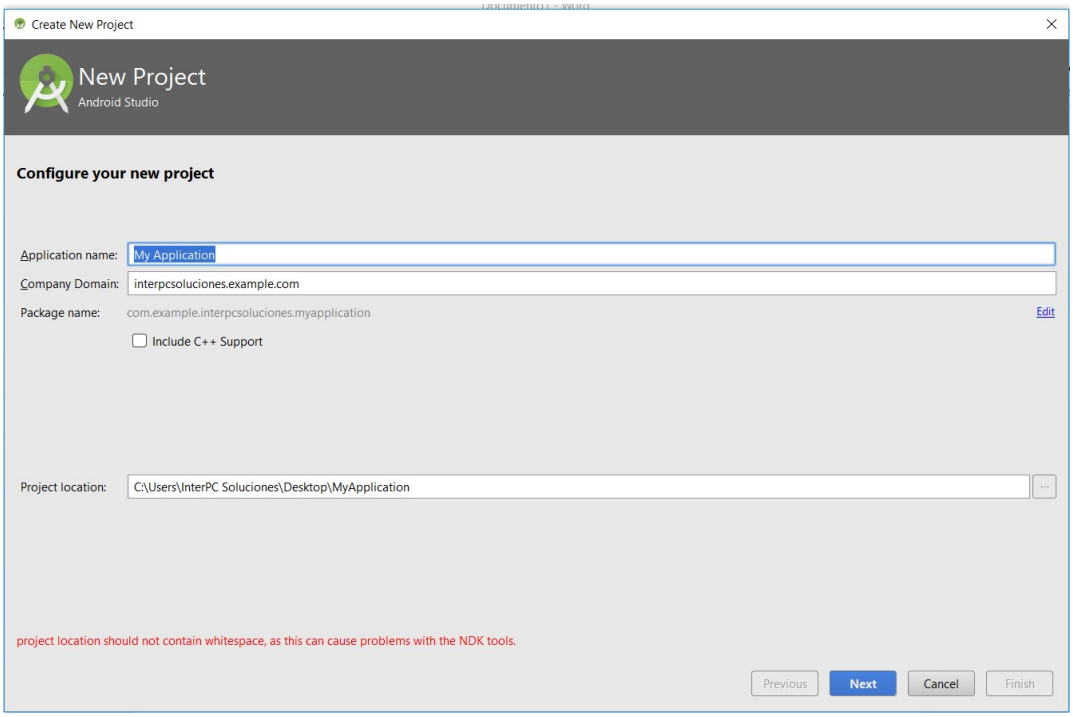
## MANEJO DE IMÁGENES Y GALERIA EN ANDROID

Este manual estará dedicado al manejo de imágenes en el sistema operativo móvil android, se estará utilizando lo que es la cámara del

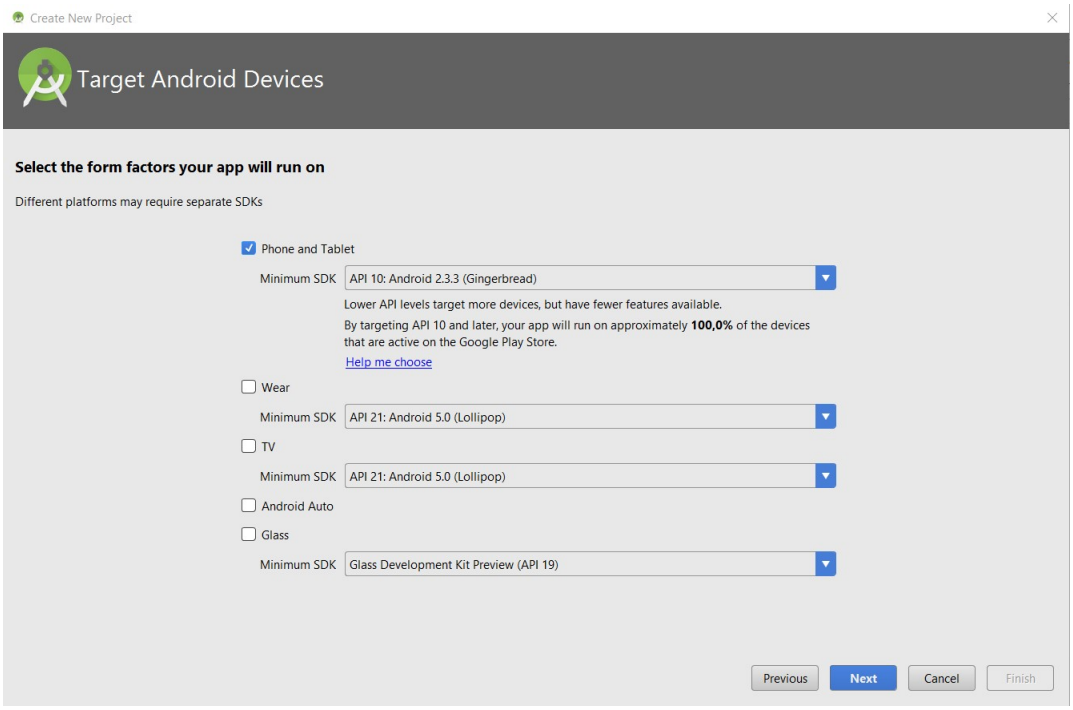
dispositivo móvil, otro de los factores que se utilizara es el almacenamiento del dispositivo, también se enseñara como trabajar con la galería de fotos de android.

Explicaremos paso a paso como se fue creando la aplicación y las funciones que se usaron, los permisos que se utilizaron y algunos de los métodos requeridos para esta aplicación.

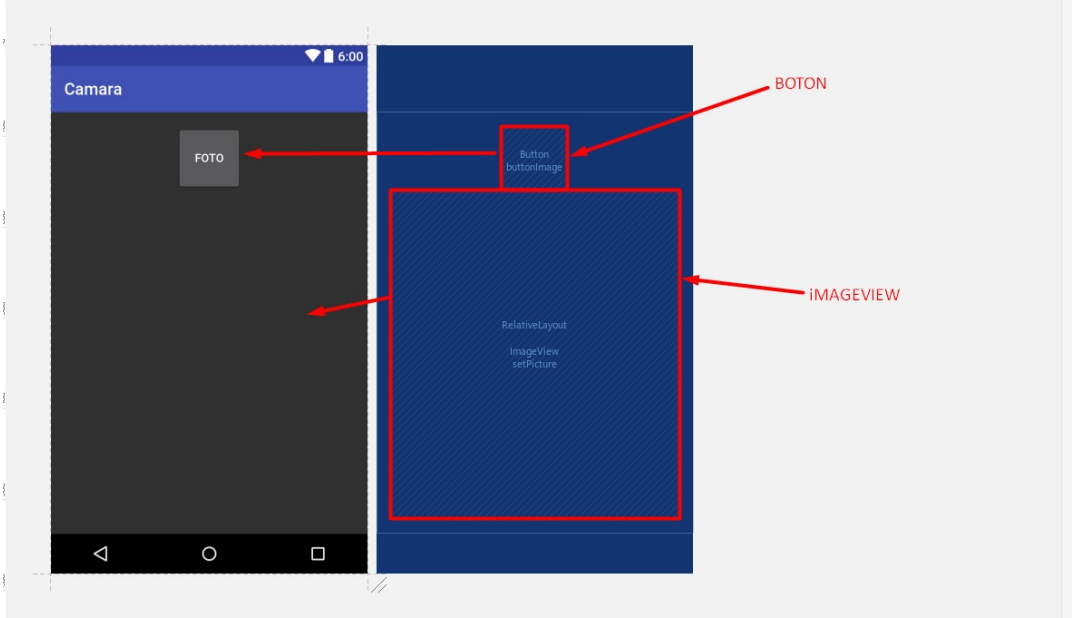
1.- El primero crearemos un nuevo proyecto en android studio




Al dar next seguirá una pantalla donde te dará varias opciones para la configuración de la aplicación, como por ejemplo la versión de android que va hacer nuestra aplicación en este caso sería la versión 2.3.3.



2. Una vez creado nuestro nuevo proyecto en android studio, vamos hacer una pequeña interfaz para esta aplicación de manejo de imágenes, esta interfaz solo contara con un botón para las acciones de la aplicación y un ImageView.




Así se vería el diseño de nuestra aplicación las siguientes líneas se escribirán en el "activity_main.xml".

Estas líneas de código solo se definirán el tamaño del botón

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools" android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    tools:context="com.example.interpcsoluciones.camara.MainActivity">
```

```
  <Button
      android:id="@+id/buttonImage"
      android:layout_width="80dp"
      android:layout_height="80dp"
      android:text="FOTO"
      android:layout_centerHorizontal="true"/>
```

Estas líneas de código son las características del botón como el tamaño

```
  <ImageView
      android:id="@+id/setPicture"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      android:contentDescription="This is where the picture is going to assign"
      android:layout_below="@+id/buttonImage"/>
```

y el texto que tendrá el botón.

Estas líneas son para el ImageView y sus características.

3. Una vez creada la interfaz que tendrá la aplicación lo siguiente que haremos es implementar los permisos necesarios para que podamos acceder a la cámara del dispositivo y también implementaremos los permisos para guarda leer y poder escribir en lo que es la memoria del dispositivo.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.interpcsoluciones.camara">

    <uses-permission android:name="android.permission.CAMERA"/>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
    <protected-broadcast android:name="android.intent.action.MEDIA_MOUNTED"/>
```

La implementación de los permisos se deberá poner en el "AndroidManifest.xml".

```
<uses-permission android:name="android.permission.CAMERA"/>
```

Este es la implementación de la cámara la cual nos podrá dar acceso a lo que es la cámara que android por default.

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
```

Esta es la implementación son para los permisos para guardar los o leer archivos de la memoria externa así podremos guardar las fotos y leerlas de la galería de imágenes.

4. Ahora se implementara las funciones para hacer funcionar la aplicación donde mandaremos llamar la cámara, haremos un AlertDialog para presentar las opciones de la aplicación y mostramos las imágenes que se capturen en el ImageView y las que elijamos de la galería

Esta será la función para poder abrir la cámara y capturarla.

- Estas líneas de código se usan para indicarle a la aplicación que vamos a guardad las fotos en la memoria externa:

```
File file  = new File(Environment.getExternalStorageDirectory(), MEDIA_DIRECTORY);
file.mkdirs();
```

- Esto indicara la ruta donde se guardó la imagen que capturamos :

```
String path = Environment.getExternalStorageDirectory() + File.separator
        + MEDIA_DIRECTORY + File.separator + TEMPORAL_PICTURE_NAME;

File newFile = new File(path);
```

- Las siguientes línea s son para poder abrir la cámara.

```
Intent intent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
intent.putExtra(MediaStore.EXTRA_OUTPUT, Uri.fromFile(newFile));
startActivityForResult(intent, PHOTO_CODE);
```

Estas es la función que utilizaremos para saber si la imagen que se está mostrar en el ImageView viene de la cámara (la que hemos capturado) o que es una de las que tenemos en la galería:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    switch (requestCode){
        case PHOTO_CODE:
            if (resultCode == RESULT_OK){
                String dir = Environment.getExternalStorageDirectory() + File.separator
                        + MEDIA_DIRECTORY + File.separator + TEMPORAL_PICTURE_NAME;
                decodeBitmap(dir);
            }
            break;

        case SELECT_PICTURE:
            if (resultCode == RESULT_OK){
                Uri path = data.getData();
                imageView.setImageURI(path);
            }
    }

}
```

Y para ver la imagen en el ImageView implementaremos estas líneas de código:

```
private void decodeBitmap(String dir){
    Bitmap bitmap;
    bitmap = BitmapFactory.decodeFile(dir);

    imageView.setImageBitmap(bitmap);
}
```

5. En las siguientes líneas implementaremos la función que va a tener el botón el cual desplegara un AlertDialog el cual nos mostrara las diferentes opciones que podremos elegir.
Las opciones que tendrá el botón son: Tomar Foto, Galería o Cancelar

La de captura mandara a llamar la función que realizamos para poder abrir la cámara y tomar una fotografía.

```
final CharSequence[] options = {"Tomar Foto","Elegir de Galeria", "Cancelar"};
final AlertDialog.Builder ventana = new AlertDialog.Builder(MainActivity.this);
ventana.setTitle("Elige un Opcion");
ventana.setItems(options, (dialog, seleccion) → {
        if(options[seleccion] == "Tomar Foto"){
            openCamara();
```

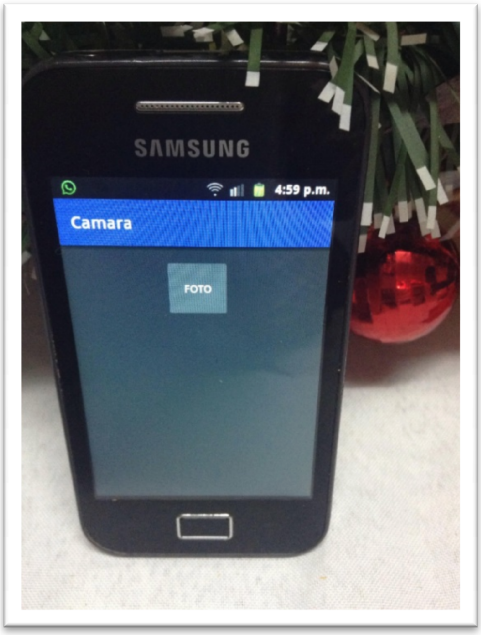
La segunda llamaremos la función para mostrar una imagen de la galería en el ImageView.

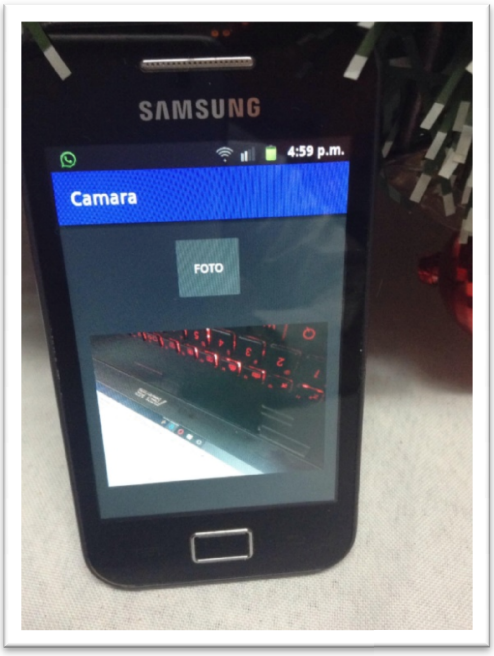
```
}else if (options[seleccion] == "Elegir de Galeria"){
    Intent intent = new Intent(Intent.ACTION_PICK, MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
    intent.setType("image/*");
    startActivityForResult(intent.createChooser(intent, "Seleccina app de imagen"), SELECT_PICTURE);
```

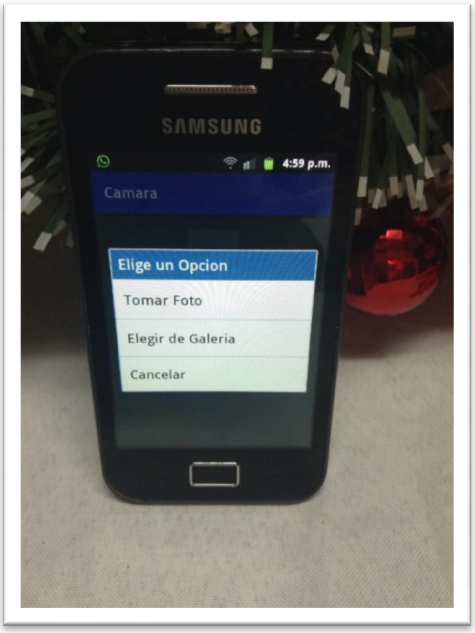
Por último la opción de Cancelar solo cerrar el AlertDialog

```
}else if (options[seleccion] == "Cancelar"){
    dialog.dismiss();
```

RESULTADO EN DISPOSITIVO MOVIL



Archivos en la nube

## Registro de proyecto en la consola de desarrolladores de google:

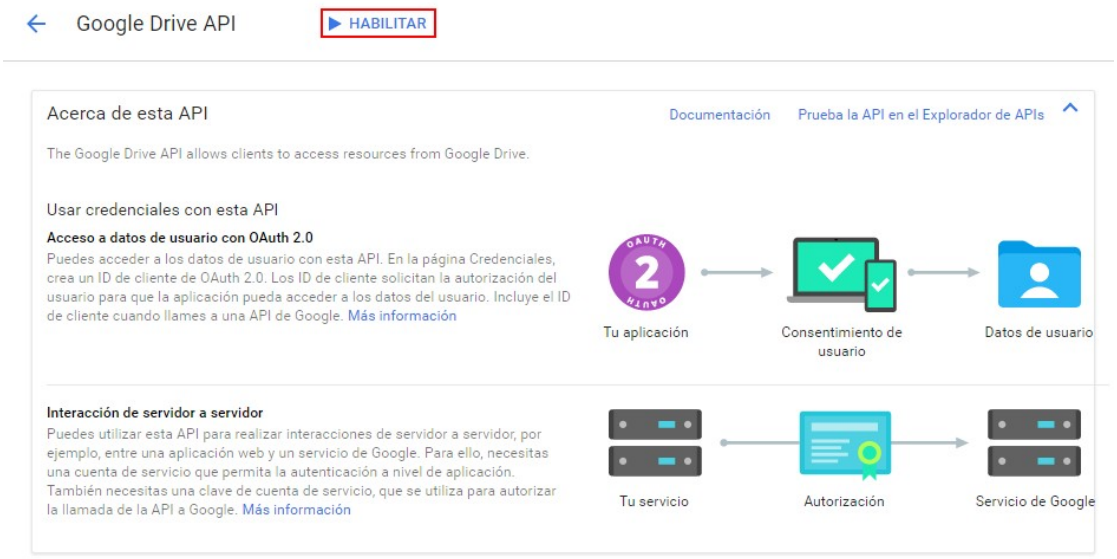
Antes de comenzar a implementar nuestra aplicación, tendremos que registrar un nuevo proyecto en la consola de desarrolladores de Google, ya que para hacer uso de los servicios de Google Drive es necesario que previamente generemos una *Clave de API* (o *API key*) asociada a nuestra aplicación. Esto se realiza accediendo a la Consola de Desarrolladores de Google.

Una vez hemos accedido, tendremos que crear un nuevo proyecto desde la lista desplegable que aparece en la parte superior derecha y seleccionando la opción “Crear proyecto…”.

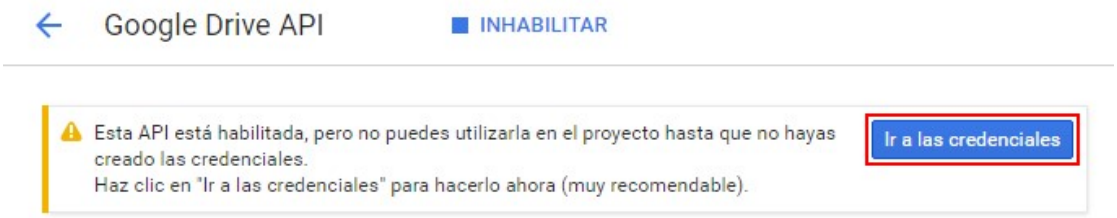

Aparecerá entonces una ventana que nos solicitará el nombre del proyecto. Introducimos algún nombre descriptivo, se generará automáticamente un ID único (que podemos editar aunque no es necesario), y aceptamos pulsando “Crear”.


Una vez creado el proyecto llegamos a una página donde se nos permite seleccionar las APIs de Google que vamos a utilizar. Hay que seleccionar “*Google Drive Android API*“.


Aparecerá entonces una ventana informativa con una breve descripción de la API y una advertencia indicando que se necesitará una clave de API para poder utilizarla. Activar la API haciendo click sobre la opción “HABILITAR” que aparece en la parte superior.

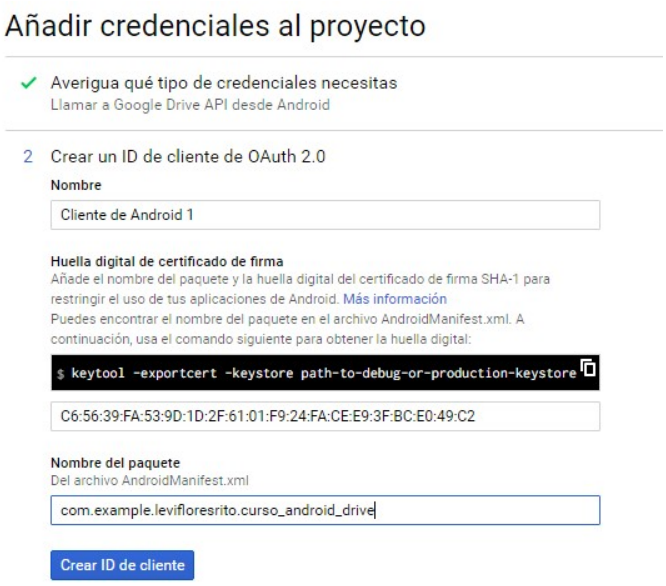



Aparece la advertencia sobre la necesidad de obtener credenciales para el uso de la API por lo que tendremos que iniciar el proceso de obtención de la clave.
Pulsamos en “Ir a credenciales” para crear iniciar el proceso de creación de nuestro *ID de Cliente (Client ID)*.




Esto iniciará un pequeño asistente. En el primer paso indicamos la API que utilizaremos (Google Drive API), la plataforma desde la que vamos a utilizar el servicio (Android) y el tipo de datos a los que accederemos. Para este último dato seleccionaremos la opción “Datos de usuario” (la única opción válida disponible por el momento para la plataforma Android).


Daremos un nombre al ID de Cliente que estamos creando, obtendremos la huella digital SHA-1 del certificado de firma, e indicaremos el paquete java principal de nuestra aplicación Android (ubicado en *AndroidManifest.xml* en el atributo package).



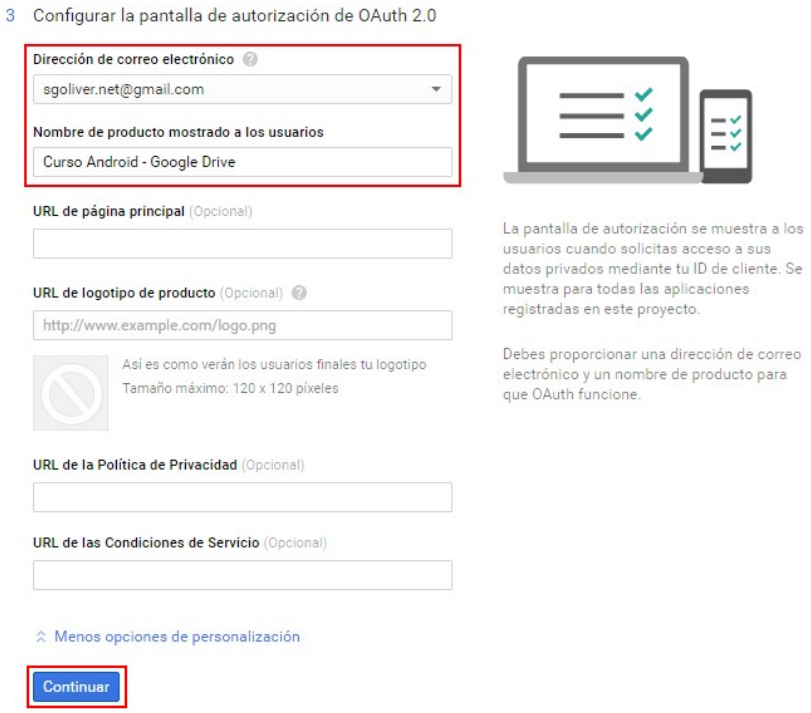

Para obtener la huella digital SHA-1 del certificado de firma hay que ingresar la siguiente ruta desde la línea de comandos de windows:

**C:\Program Files\Java\jdk1.8.0_91\bin\keytool" -list -v -keystore "%USERPROFILE%\.android\debug.keystore" -alias androiddebugkey -storepass android -keypass android**

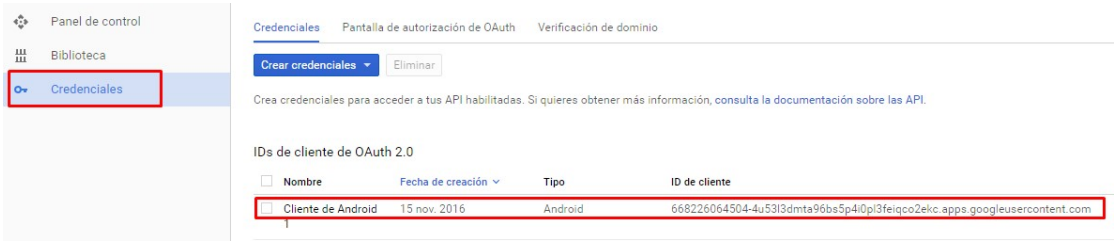
Y nos aparecerá lo siguiente en ventana:

```
C:\Users\Levi Flores Rito>"C:\Program Files\Java\jre1.8.0_111\bin\keytool" -list -v -keystore "%USERPROFILE%\.
android\debug.keystore" -alias androiddebugkey -storepass android -keypass android
Nombre de Alias: androiddebugkey
Fecha de Creación: 23/10/2016
Tipo de Entrada: PrivateKeyEntry
Longitud de la Cadena de Certificado: 1
Certificado[1]:
Propietario: C=US, O=Android, CN=Android Debug
Emisor: C=US, O=Android, CN=Android Debug
Número de serie: 1
Válido desde: Sun Oct 23 19:24:14 PDT 2016 hasta: Tue Oct 16 19:24:14 PDT 2046
Huellas digitales del Certificado:
         MD5: DE:F4:3B:16:C1:70:7B:69:69:6A:0E:D4:DF:E4:99:E4
         SHA1: C6:56:39:FA:53:9D:1D:2F:61:01:F9:24:FA:CE:E9:3F:BC:E0:49:C2
         SHA256: D0:49:D1:6F:16:CA:5E:90:E5:DB:8A:FB:91:79:0D:3C:82:5A:BC:DF:DB:C5:03:ED:F3:EB:0E:8A:26:6D:69:
84
         Nombre del Algoritmo de Firma: SHA1withRSA
         Versión: 1

C:\Users\Levi Flores Rito>
```

En el tercer paso que nos muestra el asistente tendremos que indicar una dirección de correo electrónico y un nombre de producto (nuestra aplicación), además de otros datos opcionales. Esta información se mostrará al usuario de la aplicación cuando se les solicite acceso a sus datos privados de Google Drive.




Tras aceptar este paso, y si todo ha ido bien, ya tendremos generado nuestro nuevo ID de Cliente y habremos concluido los preparativos necesarios en la Consola de Desarrolladores. No es necesario descargar el fichero ya que en principio no necesitaremos este dato en nuestra aplicación Android (es suficiente con el registro realizado), de cualquier forma, siempre podremos volver a la consola para consultarlo más adelante.



# Inicio del proyecto:

Hecho esto, podemos dirigirnos ya a Android Studio para empezar a preparar nuestro proyecto. Lo primero que haremos será añadir a nuestro fichero build.gradle la referencia a la librería específica de Google Drive perteneciente a los Google Play Services:

```
dependencies {
    //...
    compile 'com.google.android.gms:play-services-drive:9.6.1'
}
```

El siguiente paso, como ya hemos hecho para otros servicios de Google Play será crear en nuestra actividad principal (por ejemplo dentro del método `onCreate`()) un objeto de tipo `GoogleApiClient`, que utilizaremos como referencia en todas las operaciones sobre los servicios. Volveremos a usar una vez más la opción de *Auto Manage* (que gestionará automáticamente la conexión con los servicios dentro del ciclo de vida de la actividad), implementaremos en la clase principal la interfaz `OnConnectionFailedListener` y su método `onConnectionFailed()`, incluiremos la api de Google Drive añadiendo `Drive.Api`, añadiremos también un *scope* determinado para los servicios. El scope utilizado determinará el nivel de visibilidad que tendrá nuestra aplicación sobre los elementos almacenados en la cuenta de Drive del usuario. En principio utilizaremos tan `Drive.SCOPE_FILE` (más adelante comentaremos otra alternativa), que nos dará acceso a todos los ficheros y carpetas que el usuario haya abierto o creado con nuestra aplicación. Es importante entender esto último, ya que es la fuente de muchas de las dudas/problemas que surgen con la utilización de esta API. Utilizando la API de Drive para Android proporcionada por los Servicios de Google Play no tendremos acceso directo a la totalidad de los elementos almacenados en la cuenta de Drive del usuario logueado, sino tan solo a aquellos que el usuario haya abierto o creado con nuestra aplicación.

La creación del cliente API quedaría por tanto de la siguiente forma:

```
private GoogleApiClient apiClient;

public class MainActivity extends AppCompatActivity
        implements GoogleApiClient.OnConnectionFailedListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        apiClient = new GoogleApiClient.Builder(this)
            .enableAutoManage(this, this)
            .addApi(Drive.API)
            .addScope(Drive.SCOPE_FILE)
            .build();
    }

    @Override
    public void onConnectionFailed(ConnectionResult connectionResult) {
        Toast.makeText(this, "Error de conexion!", Toast.LENGTH_SHORT).show();
        Log.e(LOGTAG, "OnConnectionFailed: " + connectionResult);
    }
}
```

Hecho esto ya podríamos ejecutar por primera vez la aplicación para ver si todo está correctamente configurado. La primera vez que ejecutemos la aplicación, se mostrará al usuario un listado de todas las cuentas de Google del dispositivo, de forma que pueda seleccionar aquella con la que quiera acceder.
Una vez seleccionada, se mostrará también la pantalla de permisos, donde se informa al usuario de los recursos a los que va a acceder la aplicación.
Dada la naturaleza de este servicio, las entidades principales que manejaremos serán por supuesto ficheros y carpetas. Estos elementos estarán representados por las clases DriveFile y DriveFolder respectivamente (ambas implementan la interfaz DriveResource). Tanto ficheros como carpetas poseen atributos comunes importantes, entre los que destacan el identificador único del elemento (representado por la clase DriveId) y un conjunto de metadatos (clase Metadata) que contendrán información relevante sobre la entidad, como por ejemplo su nombre o título, su tipo de contenido (o MIME type), su tamaño, sus fechas de creación y modificación, … En el caso de los ficheros, la entidad también contendrá otro elemento importante, clase DriveContents, que encapsulará el contenido del fichero.

Repasado el modelo de objetos principales que vamos a utilizar empecemos ya con la funcionalidad disponible.
Las operaciones realizadas sobre Google Drive seguirán casi siempre el mismo esquema:

1. (Opcional) Definir los metadatos (`MetadataChangeSet`) a asociar al elemento sobre el que va a actuar la operación realizada.
2. Llamar al método correspondiente a la operación (p.ej. abrir, crear, actualizar, eliminar, …)

3. Definir y asignar a la operación un *callback* (de tipo `ResultCallback`) que será llamado automáticamente cuando ésta finalice.
4. Actuar según el resultado de la operación dentro del método `onResult()` del callback definido.

# Creación de carpetas:

Vamos a empezar por una de las operaciones más sencillas, la creación de carpetas. Para crear una nueva carpeta en Google Drive seguiremos los pasos generales que acabamos de comentar. En primer lugar, definiremos los metadatos de la nueva carpeta, que en este caso se limitarán a su título o nombre. Por tanto, lo que haremos será crear un objeto `MetadataChangeSet` mediante su `Builder`, y asignaremos el título llamando a su método `setTitle()`.

```
MetadataChangeSet changeSet =
    new MetadataChangeSet.Builder()
        .setTitle(foldername)
        .build();
```

En segundo lugar, obtendremos una referencia a la localización donde queremos crear la nueva carpeta. Esta localización podrá ser la carpeta raíz de Drive o alguna otra carpeta ya existente. Para obtener una referencia a la carpeta raíz podemos utilizar el método `getRootFolder()` de la API, que nos devolverá directamente su objeto `DriveFolder` correspondiente.

```
DriveFolder folder = Drive.DriveApi.getRootFolder(apiClient);
```

Una vez tenemos los metadatos y la carpeta donde vamos a crear el nuevo elemento, ya tan sólo nos queda llamar al método de creación correspondiente, que en este caso será `createFolder`(). Llamaremos a este método sobre el `DriveFolder` base y le pasaremos como parámetros, además del omnipresente Cliente API, los metadatos definidos anteriormente. Como ya indicamos, sobre esta llamada asignaremos un callback de tipo `ResultCallback` cuyo método `onResult()` será llamado automáticamente cuando finalice la operación. Dentro de este método podremos saber si la finalización ha sido correcta a través del método `isSuccess()`. En nuestro caso de ejemplo, simplemente mostraremos un mensaje de log con el resultado de la operación, y en caso de haber sido correcta mostraremos adicionalmente el `DriveId` del nuevo elemento creado mediante los métodos `getDriveFolder()` y `getDriveId()`.

código completo de esta llamada:

```
folder.createFolder(apiClient, changeSet).setResultCallback(
    new ResultCallback<DriveFolder.DriveFolderResult>() {
        @Override
        public void onResult(DriveFolder.DriveFolderResult result) {
            if (result.getStatus().isSuccess())
                Log.i(LOGTAG, "Carpeta creada con ID = " + result.getDriveFolder().getDriveId());
            else
                Log.e(LOGTAG, "Error al crear carpeta");
        }
    });
```

método completo de creación de una nueva carpeta en Google Drive:

```
private void createFolder(final String foldername) {

    MetadataChangeSet changeSet =
        new MetadataChangeSet.Builder()
            .setTitle(foldername)
            .build();

    //Opción 1: Directorio raíz
    DriveFolder folder = Drive.DriveApi.getRootFolder(apiClient);

    //Opción 2: Otra carpeta distinta al directorio raiz
    //DriveFolder folder =
    //        DriveId.decodeFromString("DriveId:CAESABjKGSD6wKnM71QoAQ==").asDriveFolder();

    folder.createFolder(apiClient, changeSet).setResultCallback(
        new ResultCallback<DriveFolder.DriveFolderResult>() {
            @Override
            public void onResult(DriveFolder.DriveFolderResult result) {
                if (result.getStatus().isSuccess())
                    Log.i(LOGTAG, "Carpeta creada con ID = " + result.getDriveFolder().getDriveId());
                else
                    Log.e(LOGTAG, "Error al crear carpeta");
            }
        });
}
```

no es recomendable realizar operaciones de I/O en el hilo principal de una aplicación Android, por lo que para llamar al método anterior deberíamos usar un hilo independiente.

Así, por ejemplo, podríamos llamar al método desde el evento `onClick()` de un botón de la siguiente forma:

```
new Thread() {
    @Override
    public void run() {
        createFolder("Pruebas");
    }
}.start();
```

Si ejecutamos ahora la aplicación y lanzamos la operación de creación de la nueva carpeta, podremos comprobar en el log de Android el resultado esperado. Y por supuesto podremos dirigirnos a nuestra cuenta de Google Drive a verificar que la carpeta se ha creado correctamente.

# Creación de ficheros en Android:

La lógica para crear un fichero va a ser bastante similar a la ya vista para carpetas, aunque con la particularidad de que serán necesarias dos operaciones encadenadas para crear el fichero final, la primera de ellas construirá el contenido del fichero, objeto `DriveContents`, y la segunda creará el fichero como tal, objeto `DriveFile`, incluyendo dicho contenido. Empezaremos creando el objeto DriveContents llamando al método `newDriveContents()` de la API de Google Drive. Como siempre, asignaremos un callback de tipo `ResultCallback` a dicha operación para ser notificados cuando ésta finalice. Si la operación ha finalizado con éxito (dato obtenido con `isSuccess()`) obtendremos una referencia al recién creado `DriveContents` mediante `getDriveContents()`, escribiremos el contenido del fichero sobre dicho `DriveContents`, definiremos los metadatos del nuevo fichero, obtendremos la carpeta base donde queremos crearlo (carpeta raíz u otra carpeta), y por

último crearemos el fichero final a partir de toda la información anterior utilizando el método `createFile()`.
Una vez disponemos del nuevo `DriveContents`, para escribir sobre él podremos utilizar la API estandar de java para escritura sobre ficheros (como ejemplo será un fichero de texto, pero en realidad podría escribirse cualquier tipo de contenido). Para ello, a partir del objeto `DriveContents` obtendremos su *stream* de salida asociado en forma de *OutputStream*, crearemos a partir de éste un objeto `Writer`, y finalmente escribiremos el texto necesario utilizando el método `write`(). Finalmente cerraremos el stream con `close()`. Esta lógica no debería plantear ningún problema, veamos cómo quedaría el método auxiliar que emplearemos para escribir el contenido del fichero a partir del nuevo `DriveContents`:

```
private void writeSampleText(DriveContents driveContents) {
    OutputStream outputStream = driveContents.getOutputStream();
    Writer writer = new OutputStreamWriter(outputStream);

    try {
        writer.write("Esto es un texto de prueba!");
        writer.close();
    } catch (IOException e) {
        Log.e(LOGTAG, "Error al escribir en el fichero: " + e.getMessage());
    }
}
```

Para definir los metadatos utilizaremos el mismo método que para las carpetas, con la diferencia de que en esta ocasión asignaremos algún dato adicional además del nombre o título, por ejemplo el MIME Type mediante `setMimeType()`, que indicará el tipo de contenido de nuestro fichero. Como hemos indicado, para el ejemplo crearemos un fichero de texto por lo que nuestro MIME Type será “text/plain” (se puede consultar una lista bastante extensa con otros valores posibles en este enlace).
Como carpeta base utilizaremos una vez más la carpeta raíz de Drive, obtenida mediante `getRootFolder`(), aunque podríamos utilizar cualquier otra, a través de su `DriveId`.

Por último, crearemos el fichero final con todos estos datos llamando a `createFile()` sobre la carpeta base, al que pasaremos como parámetros el cliente API, el conjunto de metadatos, y el objeto `DriveContents` que creamos y escribimos anteriormente. Como siempre, asignaremos un callback sobre esta llamada para ser notificados cuando finalice.
código completo del método de creación de ficheros:

```
private void createFile(final String filename) {

    Drive.DriveApi.newDriveContents(apiClient)
        .setResultCallback(new ResultCallback<DriveApi.DriveContentsResult>() {
            @Override
            public void onResult(DriveApi.DriveContentsResult result) {
                if (result.getStatus().isSuccess()) {

                    writeSampleText(result.getDriveContents());

                    MetadataChangeSet changeSet =
                        new MetadataChangeSet.Builder()
                            .setTitle(filename)
                            .setMimeType("text/plain")
                            .build();

                    //Opción 1: Directorio raiz
                    DriveFolder folder = Drive.DriveApi.getRootFolder(apiClient);

                    //Opción 2: Otra carpeta distinta al directorio raiz
                    //DriveFolder folder =
                    //    DriveId.decodeFromString("DriveId:CAESABjKGSD6wKnM71QoAQ==").asDriveFolder();

                    folder.createFile(apiClient, changeSet, result.getDriveContents())
                        .setResultCallback(new ResultCallback<DriveFolder.DriveFileResult>() {
                            @Override
                            public void onResult(DriveFolder.DriveFileResult result) {
                                if (result.getStatus().isSuccess()) {
                                    Log.i(LOGTAG, "Fichero creado con ID = " + result.getDriveFile().getDriveId());
                                } else {
                                    Log.e(LOGTAG, "Error al crear el fichero");
                                }
                            }
                        });
                } else {
                    Log.e(LOGTAG, "Error al crear DriveContents");
                }
            }
        });
}
```

Este método, al igual que el anteriormente creado `createFolder()`, debemos ejecutarlo en un hilo de ejecución independiente, por ejemplo de la siguiente forma:

```
new Thread() {
    @Override
    public void run() {
        createFile("prueba1.txt");
    }
}.start();
```

Además de esta forma de crear ficheros totalmente programática, también tenemos la posibilidad de crearlos utilizando una actividad predefinida que permitirá al usuario seleccionar una ubicación y un nombre para el fichero.

Para hacer esto el proceso será muy similar al descrito anteriormente. Creamos un nuevo `DriveContents`mediante `newDriveContents()`, asignamos el callback correspondiente, y en su método `onResult()` definimos los metadatos (esta vez no asignamos ningún título), escribimos el texto de ejemplo sobre el `DriveContents`, y por último, como única novedad, en vez de crear directamente el fichero final con `createFile`(), creamos un `IntentSender` para lanzar la actividad de creación de ficheros en Drive. Lo crearemos llamando al método `newCreateFileActivityBuilder()` de la API de Google Drive. Sobre éste asignaremos nuestros metadatos con `setInitialMetadata`(), asignaremos el contenido inicial del fichero con `setInitialDriveContents()` y por último llamaremos al método `build()`.

Construido el `IntentSender` ya solo nos queda lanzarlo para iniciar la actividad mediante `startIntentSenderForResult`(), al que pasaremos nuestro `IntentSender` y una constante arbitraria (en mi caso `REQ_CREATE_FILE`, definida con cualquier valor único) que nos ayudará a conocer el resultado de la actividad (es decir, lo que haya realizado el usuario sobre la actividad mostrada).

```
private void createFileWithActivity() {

    Drive.DriveApi.newDriveContents(apiClient)
        .setResultCallback(new ResultCallback<DriveApi.DriveContentsResult>() {
            @Override
            public void onResult(DriveApi.DriveContentsResult result) {
                MetadataChangeSet changeSet =
                    new MetadataChangeSet.Builder()
                        .setMimeType("text/plain")
                        .build();

                writeSampleText(result.getDriveContents());

                IntentSender intentSender = Drive.DriveApi
                    .newCreateFileActivityBuilder()
                    .setInitialMetadata(changeSet)
                    .setInitialDriveContents(result.getDriveContents())
                    .build(apiClient);

                try {
                    startIntentSenderForResult(
                        intentSender, REQ_CREATE_FILE, null, 0, 0, 0);
                } catch (IntentSender.SendIntentException e) {
                    Log.e(LOGTAG, "Error al iniciar actividad: Create File", e);
                }
            }
        });
}
```

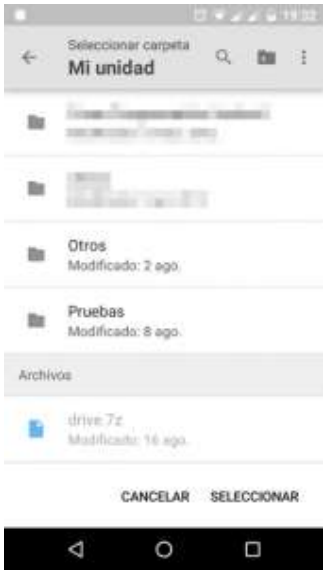
Para conocer este resultado debemos sobrescribir el método `onActivityResult()` de nuestra actividad principal. En éste, filtraremos por la constante que hemos utilizado anteriormente (`reqCode == REQ_CREATE_FILE`) y revisaremos si el usuario ha finalizado correctamente el proceso (`resultCode == RESULT_OK`). En este caso ya podremos obtener el `DriveId` del nuevo fichero creado a partir de los datos “extra” contenidos en el intent que recibimos como parámetro. Concretamente lo obtendremos obteniendo el dato de tipo `Parcelable` con
nombre `OpenFileActivityBuilder.EXTRA_RESPONSE_DRIVE_ID`.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
        case REQ_CREATE_FILE:
            if (resultCode == RESULT_OK) {
                DriveId driveId = data.getParcelableExtra(
                    OpenFileActivityBuilder.EXTRA_RESPONSE_DRIVE_ID);

                Log.i(LOGTAG, "Fichero creado con ID = " + driveId);
            }
            break;
        default:
            super.onActivityResult(requestCode, resultCode, data);
            break;
    }
}
```

**Ejemplo:**

Si ejecutamos ahora la aplicación y el método `createFileWithActivity()` veremos en nuestro dispositivo una pantalla como la siguiente:


Como puede comprobarse, al usuario se le da la posibilidad de escribir un nombre para el fichero, y de seleccionar la carpeta donde guardarlo. Si dejamos seleccionada la opción “Mi unidad” el fichero se creará en la carpeta raíz de Drive. Si por el contrario pulsamos sobre la lista desplegable nos aparecerá una nueva pantalla donde podemos ver las carpetas y ficheros almacenados actualmente en la cuenta de Drive, de forma que se permite seleccionar una carpeta distinta a la carpeta raíz.




Si pulsamos aquí el botón SELECCIONAR sobre la carpeta elegida y posteriormente el botón GUARDAR en la pantalla anterior, el fichero se creará con los datos indicados y aparecerá un mensaje de log con el DriveId del nuevo elemento.

# Operaciones extras para folders y ficheros de google drive:

# Eliminar elemento:

La eliminación de elementos se realiza mediante los métodos `trash()` o `delete()` de la interfaz DriveResource, por lo que aplica tanto a ficheros como carpetas. La diferencia entre ambos radica en que el primero envía el fichero o carpeta a la Papelera de Drive (de forma que podría recuperarse posteriormente si es necesario, utilizando el método `untrash()`) y el segundo elimina por completo el elemento sin pasar por la Papelera.

El esquema a seguir será el ya habitual, llamar al método de borrado correspondiente (trash o delete), asignar el callback, y actuar según el resultado dentro del método onResult(). Veamos por ejemplo cómo quedaría un método para eliminar ficheros a partir de su DriveId:

```
private void deleteFile(DriveId fileDriveId) {
    DriveFile file = fileDriveId.asDriveFile();

    //Opción 1: Enviar a la papelera
    file.trash(apiClient).setResultCallback(new ResultCallback<Status>() {
        @Override
        public void onResult(Status status) {
            if(status.isSuccess())
                Log.i(LOGTAG, "Fichero eliminado correctamente.");
            else
                Log.e(LOGTAG, "Error al eliminar el fichero");
        }
    });

    //Opción 2: Eliminar
    //file.delete(apiClient).setResultCallback(...)
}
```

# Consultar elemento:

La consulta de los metadatos de un fichero o carpeta seguirá una vez más el mismo esquema que las operaciones que ya conocemos. En primer lugar llamaremos al método `getMetadata()` sobre el objeto `DriveFile` o `DriveFolder` que queramos consultar. Asignaremos un callback (`ResultCallback`) e implementaremos su método `onResult()` para ser notificado cuando finalice la operación. Si el resultado es correcto (`getStatus().isSuccess()`) podremos obtener los metadatos mediante el método `getMetadata()` del objeto `MetadataResult` recibido como parámetro, y a partir de éste cualquier dato concreto que necesitemos, por ejemplo su nombre (con `getTitle`()) o su fecha de última actualización (con `getModifiedDate()`).

Método que recupere los metadatos de un fichero a partir de su DriveId:

```
private void getMetadata(DriveId fileDriveId) {
    DriveFile file = fileDriveId.asDriveFile();

    file.getMetadata(apiClient).setResultCallback(
        new ResultCallback<DriveResource.MetadataResult>() {
            @Override
            public void onResult(DriveResource.MetadataResult metadataResult) {
                if (metadataResult.getStatus().isSuccess()) {
                    Metadata metadata = metadataResult.getMetadata();
                    Log.i(LOGTAG, "Metadatos obtenidos correctamente." +
                            " Title: " + metadata.getTitle() +
                            " LastUpdated: " + metadata.getModifiedDate());
                }
                else {
                    Log.e(LOGTAG, "Error al obtener metadatos");
                }
            }
        });
}
```

# Modificar elemento:

Modificar los metadatos de un fichero/carpeta es tan sencillo como consultarlos. Crearemos en primer lugar un objeto `MetadataChangeSet` con los cambios que queramos realizar en

los metadatos, igual que hacíamos para crear un fichero nuevo. Una vez definidos los metadatos que queremos actualizar, llamaremos al método `updateMetadata()` pasándole como parámetros el conjunto de metadatos creados, asignaremos el callback habitual y revisaremos dentro del método `onResult()` que la operación ha finalizado correctamente.
Un método que actualizará por ejemplo el título de un fichero a partir de su DriveId quedaría de la siguiente forma:

```
private void updateMetadata(DriveId fileDriveId) {
    DriveFile file = fileDriveId.asDriveFile();

    MetadataChangeSet changeSet =
        new MetadataChangeSet.Builder()
            .setTitle("TituloModificado.txt")
            .build();

    file.updateMetadata(apiClient, changeSet).setResultCallback(
        new ResultCallback<DriveResource.MetadataResult>() {
            @Override
            public void onResult(DriveResource.MetadataResult metadataResult) {
                if (metadataResult.getStatus().isSuccess()) {
                    Metadata metadata = metadataResult.getMetadata();
                    Log.i(LOGTAG, "Metadatos actualizados correctamente.");
                }
                else {
                    Log.e(LOGTAG, "Error al actualizar metadatos");
                }
            }
        });
}
```

# Acelerómetro

**Introducción.**

El acelerómetro**,** como su propio nombre indica, un acelerómetro es un dispositivo capaz de medir aceleraciones, es decir, la variación en la velocidad por unidad de tiempo. Existen diferentes tipos de acelerómetros en función del tipo de tecnología que utilicen para medir esa magnitud: mecánicos, piezoeléctricos, de condensador, etc.

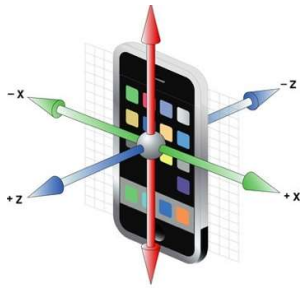
Como no podemos medir las aceleraciones por observación directa, se deben calcular a partir de otras variables que sí se pueden medir y el conocimiento de las leyes que rigen sus efectos. Para comprenderlo más claramente vamos a ver el caso del acelerómetro mecánico. En él obtenemos la aceleración del sistema a partir de la masa y la observación de su desplazamiento.

**Conceptos básicos *(acelerómetro)*.**

- **Ley de Hooke:** El alargamiento unitario que experimenta un material elástico es directamente proporcional a la fuerza aplicada. Esto es **F = k·x.** (donde F es la fuerza aplicada, k la constante de elasticidad del muelle y x el desplazamiento de la masa sísmica).
- **Segunda ley de Newton:** El cambio de movimiento es proporcional a la fuerza motriz impresa y ocurre según la línea recta a lo largo de la cual aquella fuerza se imprime. Esta es representada mediante la famosa ecuación **F = m·a.** (F: fuerza; m: masa, a: aceleración).

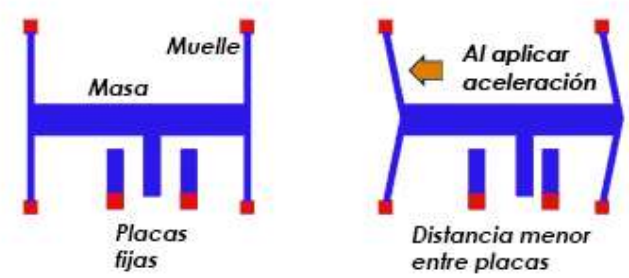
**Funcionamiento Acelerómetro Mecánico.**

A partir de las ecuaciones podemos establecer que: **m·a = k·x**, con lo que, finalmente, **a = (k/m)·x**. Por lo tanto, se puede obtener el valor de la aceleración, ya que k **es la constante de elasticidad** del muelle (y que conocemos, ya que lo hemos puesto nosotros), **m es la masa desplazada** (la masa sísmica) y **x la distancia desplazada**, que podemos medir.




**Acelerómetros en dispositivos.**

Los acelerómetros capacitivos, al ser electrónicos, pueden ser extremadamente pequeños y se pueden fabricar integrados en chips para soldar en placas de silicio, por ejemplo en tu Smartphone. Adicionalmente es necesario disponer de un software que sea capaz de interpretar las señales generadas por el acelerómetro. De esto se encarga Android.




Como la fuerza de la gravedad actúa en todo momento y conocemos su valor (9,8 m/s2), es fácil utilizar los valores de cada eje del acelerómetro para determinar el ángulo de inclinación y, por tanto, la posición del dispositivo. De esta forma se puede mostrar el contenido con la orientación correcta y rotarla cuando ésta cambie. De la misma forma, las variaciones en sus valores pueden ser utilizadas como señal de entrada para aplicaciones, por ejemplo para simular un volante en los juegos de conducción.

**Desarrollo.**

En el *activity_main.xml* agregaremos 3 etiquetas **TextView**, en los que mostraremos los valores de X, Y y Z del acelerómetro.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingBottom="16dp"
    android:paddingLeft="16dp"
    android:paddingRight="16dp"
    android:paddingTop="16dp"
    tools:context="com.example.angel.acelerometro.MainActivity">

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Valor de X"
        android:id="@+id/txtAccX"
        />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Valor de Z"
        android:id="@+id/txtAccZ"
        android:layout_marginTop="15dp"
        android:layout_below="@+id/txtAccY"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Valor de Y"
        android:id="@+id/txtAccY"
        android:layout_below="@+id/txtAccX"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_marginTop="11dp" />
</RelativeLayout>
```

*(Código en activity_main.xml).*

En el *MainActivity.java* implementaremos para nuestra aplicación el *SensorEventListener* esto es requerido para el manejo de las actualizaciones de los sensores, en nuestro caso el acelerómetro.

```
public class MainActivity extends AppCompatActivity implements SensorEventListener {
```

Crearemos las siguientes variables para el control de los valores de los 3 ejes que mide el acelerómetro y las otras para guardar el *timestamp* de la última actualización, y la última vez que se detectó movimiento.

```
private long last_update = 0, last_movement = 0;
private float prevX = 0, prevY = 0, prevZ = 0;
private float curX = 0, curY = 0, curZ = 0;
```

Al utilizar esta interfaz debemos sobrecargar dos métodos:

```
@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {}

@Override
public void onSensorChanged(SensorEvent event) {
```

Vamos a estar usando *onSensorChanged* nuestro código será un synchronized statement para evitar problemas de concurrencia al estar trabajando con los sensores.

```
synchronized (this) {
```

Basándonos en el evento recibido como parámetro vamos a obtener el timestamp de la fecha/hora actual y los valores para los 3 ejes del acelerómetro (X, Y, Z).

```
long current_time = event.timestamp;

curX = event.values[0];
curY = event.values[1];
curZ = event.values[2];
```

Después, revisamos si esta parte del código ya se ha ejecutado alguna vez, es decir, si las variables que guardan los valores anteriores de X, Y y Z tiene valores diferentes de cero. Esto tendría ejecutarse sólo la primera vez.

```
if (prevX == 0 && prevY == 0 && prevZ == 0) {
    last_update = current_time;
    last_movement = current_time;
    prevX = curX;
    prevY = curY;
    prevZ = curZ;
}
```

Obtenemos la diferencia entre la última actualización y el timestamp actual, esto nos servirá no solo para ver que si existe una diferencia de tiempos en las mediciones sino también para calcular el movimiento.

Para eso, tomamos la posición actual (con las 3 coordenadas) y la restamos a la posición anterior, puede ser que el movimiento sea en distintas direcciones por eso nos es útil el **valor absoluto**.

```
long time_difference = current_time - last_update;
if (time_difference > 0) {
    float movement = Math.abs((curX + curY + curZ) - (prevX - prevY - prevZ)) / time_difference;
```

Para decidir en qué momento mostramos un aviso Toast indicando el movimiento vamos a usar como valor de frontera de movimiento mínimo 2×10^-6; este valor es a la necesidad de la aplicación.
Es importante tener claro que: **Mientras mayor sea, se necesitará más movimiento y mientras menor sea más sensible será el display del aviso**.

También utilizamos 2 variables para el control de tiempo, una para saber la última actualización *(last_update)* y otra para conocer la última vez que se registró movimiento (*last_movement)* y en esta parte del código actualizamos su valor según sea conveniente.

```
int limit = 1000;
float min_movement = 2E-6f;
if (movement > min_movement) {
    if (current_time - last_movement >= limit) {
        Toast.makeText(getApplicationContext(), "Hay movimiento de " + movement, Toast.LENGTH_SHORT).show();
    }
    last_movement = current_time;
}
```

Al final actualizamos los valores de X, Y y Z anteriores la próxima vez que se registre cambio en los sensores.

```
prevX = curX;
prevY = curY;
prevZ = curZ;
last_update = current_time;
```

En esta parte actualizamos los valores de los 3 ejes del acelerómetro en las etiquetas para visualizarlo en pantalla.

```
((TextView) findViewById(R.id.txtAccX)).setText("Aceler—metro X: " + curX);
((TextView) findViewById(R.id.txtAccY)).setText("Aceler—metro Y: " + curY);
((TextView) findViewById(R.id.txtAccZ)).setText("Aceler—metro Z: " + curZ);
```

Teniendo todo lo anterior listo, ahora es necesario registrar y anular el registro del *Listener* para el sensor según corresponda, para hacer eso utilizaremos los **métodos onResume y onStop** del ciclo de vida de las Actividades.
En el registro obtenemos primero el servicio del sistema de sensores para asignarlo en un *SensorManager* y a partir de él obtenemos el **acceso al acelerómetro**.
Al realizar el registro del acelerómetro es necesario indicar una **tasa de lectura de datos**, en nuestro caso vamos a utilizar *SensorManager.SENSOR_DELAY_GAME* que es la **velocidad mínima para que el acelerómetro pueda usarse en un juego**.

```
@Override
protected void onResume() {
    super.onResume();
    SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);
    List<Sensor> sensors = sm.getSensorList(Sensor.TYPE_ACCELEROMETER);
    if (sensors.size() > 0) {
        sm.registerListener(this, sensors.get(0), SensorManager.SENSOR_DELAY_GAME);
    }
}
```

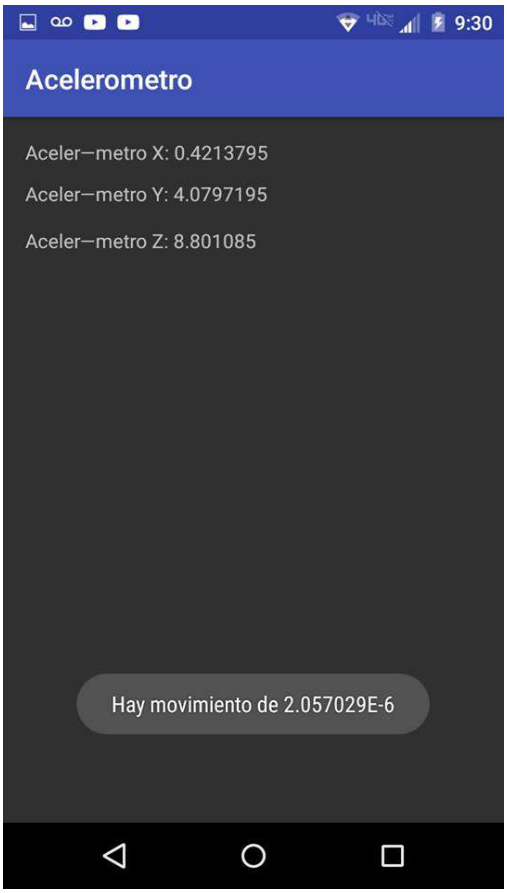
Para anular el registro debemos de indicar que la clase actual *(que implementa a SensorEventListener)* ya no está interesada en recibir actualizaciones de sensores.

```
@Override
protected void onStop() {
    SensorManager sm = (SensorManager) getSystemService(SENSOR_SERVICE);
    sm.unregisterListener(this);
    super.onStop();
}
```

Para terminar, en el método *onCreate de la Activity* vamos a bloquear la orientación para que al mover el teléfono la **pantalla no se mueva y la interfaz permanezca tal como la vemos.**

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    this.setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
}
```

**Resultados de la aplicación.**


**Conclusión.**

Este manual nos sirve para conocer cómo se implementan los sensores del acelerómetros y, aunque el código sea un poco sencillo, tiene un gran potencial y nos ayuda a tener en cuenta los datos que captura, como podemos capturar datos un tanto sensibles, o quitarle sensibilidad de respuesta a nuestros resultados y gracias a esto podemos pensar en la gran cantidad de aplicaciones que los utilizan y cómo implementarlos en nuestros proyectos personales.

# Trabajo con APIs

# Trabajo con APIs: Facebook y Twitter

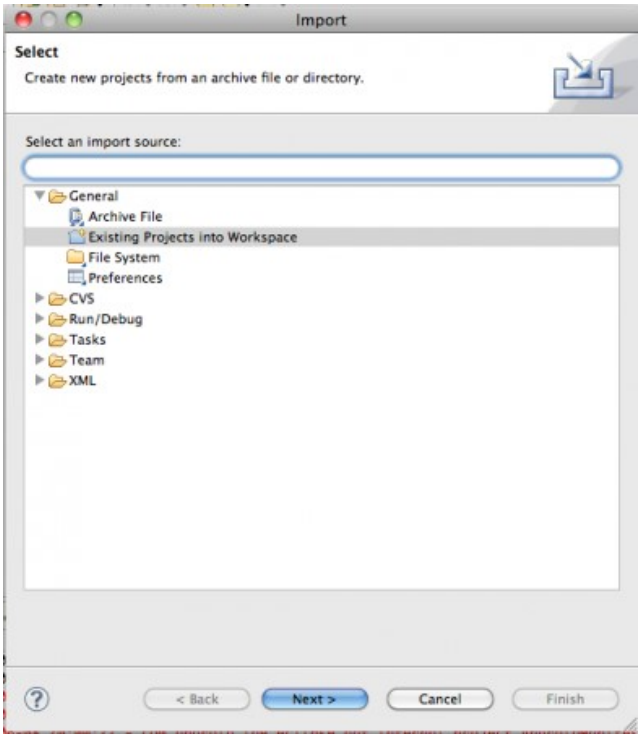
## Introducción

Para realizar la autenticación es necesario solicitar una aplicación en Facebook y una aplicación en Twitter. En este trabajo se utiliza el SDK de Facebook para Android que se descarga desde GitHub. Se tomará no sólo el SDK si no también algunos archivos del ejemplo simple que se incluye en el mismo SDK (El botón de inicio de sesión, el manejo y almacenamiento de la sesión y el listener).

Link al SDK de Facebook: http://www.maestrosdelweb.com/images/2011/06/facebook.zip

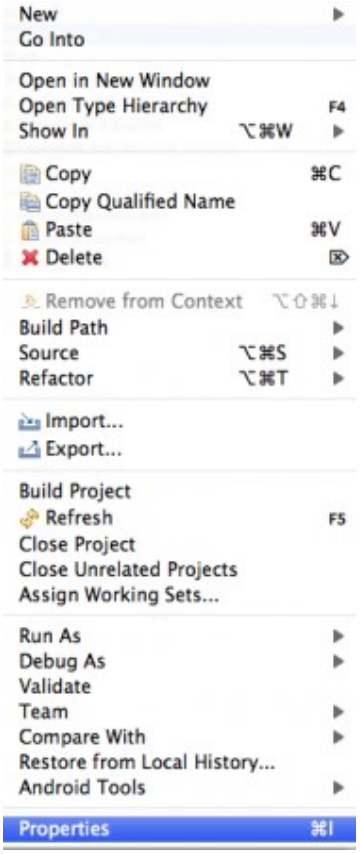
## Configuración

Primero, se importa el proyecto al workspace:

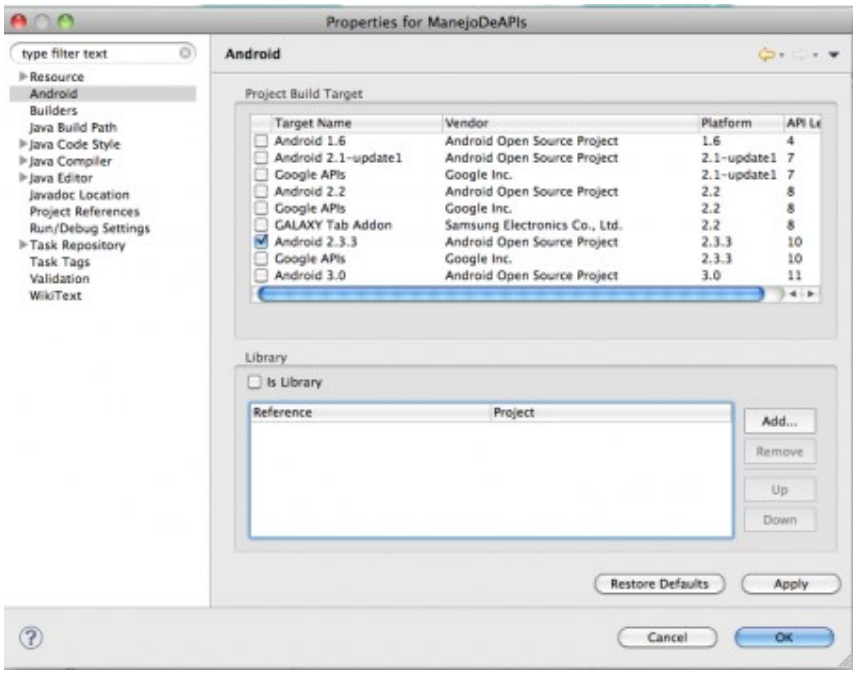



Después se crea un proyecto nuevo para iniciar con la nueva aplicación.

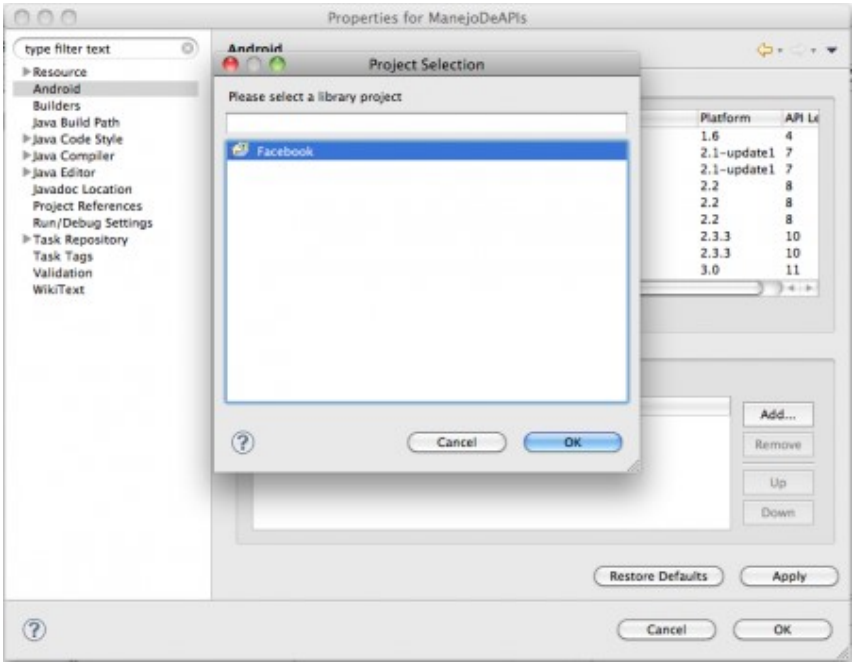
Luego de que está listo el proyecto indicamos que tome la librería del SDK desde otro proyecto, hacemos click derecho sobre el proyecto y luego en propiedades.



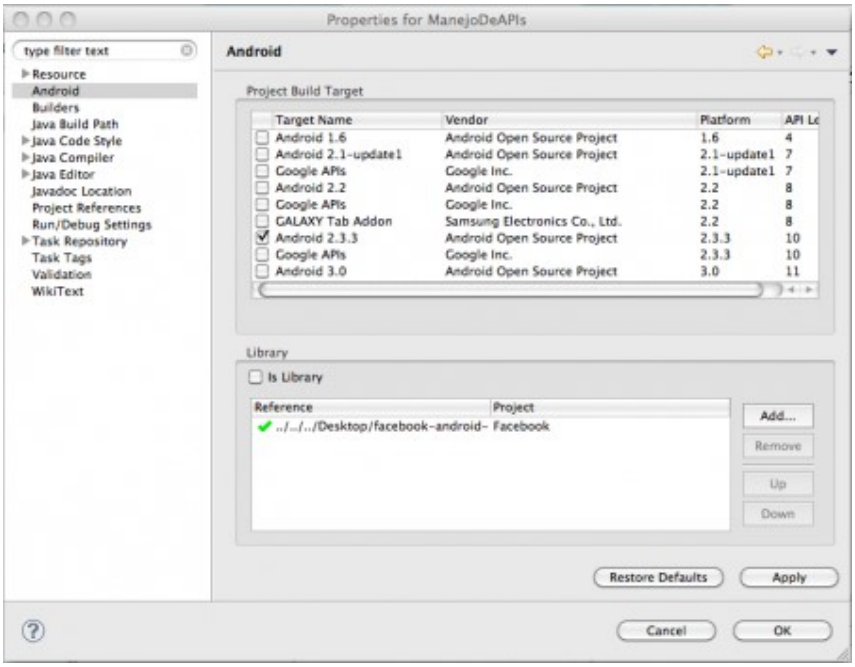

En la pantalla de las opciones buscamos la parte de Android:



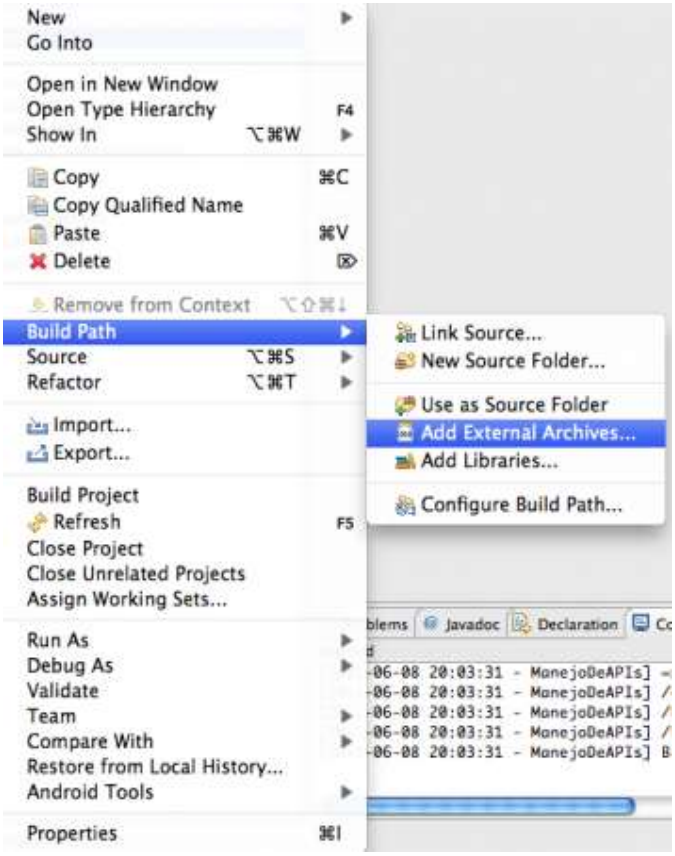

Hacemos click sobre “Add” en la parte de Library:




Deberá aparecernos el proyecto llamado “Facebook”, lo seleccionamos y estará lista la referencia:




Para la autenticación con Twitter hay varias opciones en nuestro caso estaremos utilizando OAuth SignPost y para hacerlo parte de nuestro proyecto necesitamos las librerías de Core y Apache Common, ambos son archivos JAR descargables, la forma de incluirlas es haciendo click derecho sobre el proyecto, luego “Build Path” y por último “Add External Archives”.




## Aplicación

Permisos para internet:

```
 </application>
 <uses-permission android:name="android.permission.INTERNET" />
anifest>
```

Utilizo un intent filter que permite el callback luego de la autorización de mi aplicación:

```
<intent-filter>
    <action android:name="android.intent.action.VIEW"></action>
    <category android:name="android.intent.category.DEFAULT"></category>
    <category android:name="android.intent.category.BROWSABLE"></category>
    <data android:scheme="mdw" android:host="twitter"></data>
</intent-filter>
```

El diseño tiene 2 botones para sesión y 2 etiquetas para el status. En el caso de Facebook, utilizo el botón incluido en el ejemplo del SDK implementado en una clase llamada LoginButton.

```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <com.facebook.android.LoginButton
        android:id="@+id/btnFbLogin"
        android:src="@drawable/login_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        />
    <TextView
        android:layout_height="wrap_content"
        android:text="Facebook status: sesión no iniciada"
        android:id="@+id/txtFbStatus"
        android:layout_width="fill_parent"></TextView>
    <View
        android:layout_height="1dip"
        android:layout_width="fill_parent"
        android:background="#FFFFFF"
    />
    <Button
        android:text="Twitter"
        android:id="@+id/btnTwLogin"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"></Button>
    <TextView
        android:layout_height="wrap_content"
        android:text="Twitter status: sesión no iniciada"
        android:id="@+id/txtTwStatus"
        android:layout_width="fill_parent"></TextView>
</LinearLayout>
```

Defino algunas variables globales dentro del Activity. Se hace el botón de Twitter, el de Facebook lo tomo del SDK. Declaro también los Listeners.

```
private Button btnTwLogin;
private OnClickListener twitter_auth, twitter_clearauth;
```

Defino banderas para checar el status de autenticación y etiquetas para mostrar el mismo.

```
private TextView txtFbStatus, txtTwStatus;
private boolean twitter_active = false, facebook_active = false;
```

Las peticiones para interactuar con el Graph API de Facebook son asincrónicas y las realizo a través de una instancia de la clase AsyncFacebookRunner.

```
private AsyncFacebookRunner mAsyncRunner;
```

Para la parte de autenticación con Twitter, necesito un provider y un consumer configurados con los URLs, el consumer key y el consumer secret.

```
private static CommonsHttpOAuthProvider provider =
        new CommonsHttpOAuthProvider(
        "https://api.twitter.com/oauth/request_token",
        "https://api.twitter.com/oauth/access_token",
        "https://api.twitter.com/oauth/authorize");

private static CommonsHttpOAuthConsumer consumer =
        new CommonsHttpOAuthConsumer(
            "7iEjG84wItGvXaIZFXAyZg",
            "sZKCJaUN8BgmYy4r9Z7h1I4BEHV8aAd6Ujw3hofQ4k");
```

Una vez realizada la autorización del usuario con su cuenta de Twitter, recibo de vuelta un key y un secret de acceso y es necesario guardarlos para cualquier futura interacción.

```
private static String ACCESS_KEY = null;
private static String ACCESS_SECRET = null;
```

Cuando el usuario se autentique con sus credenciales de Facebook, haremos una petición al Graph API para obtener sus datos (identificador y nombre), esta requisición es asincrónica y requiere un RequestListener (parte del SDK de Facebook) para realizar alguna acción una vez se ha recibido respuesta.
Encapsulo en un método esta llamada y en concreto me concentro en el método onComplete del Listener para realizar lo que quiero La respuesta recibida en JSON es necesario reconocerla (parsing) y luego enviar los valores que me interesan (ID y nombre) al hilo de ejecución(thread) principal para que modifique la vista(únicamente él, el thread principal, puede hacer este cambio en el contenido del TextView) .

```
private void updateFbStatus(){
    mAsyncRunner.request("me", new RequestListener() {
        @Override
        public void onMalformedURLException(MalformedURLException e, Object state) {}

        @Override
        public void onIOException(IOException e, Object state) {}

        @Override
        public void onFileNotFoundException(FileNotFoundException e, Object state) {}

        @Override
        public void onFacebookError(FacebookError e, Object state) {}
```

Además de este método, trabajo en onCreate. Primero obtengo botones y etiquetas (TextView) de status tanto de Facebook como de Twitter.

```
LoginButton mLoginButton = (LoginButton) findViewById(R.id.btnFbLogin);
btnTwLogin = (Button)findViewById(R.id.btnTwLogin);

txtTwStatus = (TextView) this.findViewById(R.id.txtTwStatus);
txtFbStatus = (TextView) this.findViewById(R.id.txtFbStatus);
```

Luego, construyo mi objeto Facebook (parte del SDK descargado) y a partir de él un objeto AsyncFacebookRunner que utilizo para las peticiones.

```
Facebook mFacebook = new Facebook(APP_ID);
mAsyncRunner = new AsyncFacebookRunner(mFacebook);
```

Guardo las credenciales de Twitter como preferencias de la aplicación (key/value), en onCreate las recupero y si no existen tengo un valor por defecto que es un String vacío. Si existen las credenciales entonces habilito la variable boolean twitter_active.

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
String stored_keys = prefs.getString("KEY", "");
String stored_secret = prefs.getString("SECRET", "");

if (!stored_keys.equals("") && !stored_secret.equals("")) {
    twitter_active = true;
}
```

Hago un proceso similar para el caso de Facebook, aquí es más sencillo porque me apoyo en el código del ejemplo del SDK. Restauro la sesión y luego valido la misma, en base a esto habilito la bandera facebook_active y si la sesión está activa entonces muestro los datos del usuario llamando a updateFbStatus().

```
SessionStore.restore(mFacebook, this);
facebook_active = mFacebook.isSessionValid();
if (facebook_active) {
    updateFbStatus();
}
```

Para el inicio de sesión de Facebook es necesario un Listener para autenticación y otro para cuando se finaliza sesión. Para el primero es necesario sobrecargar métodos dependiendo si tuvo éxito o no, en caso de tener éxito llamo a updateFbStatus y en caso de fallar reporto el error a través del TextView de status.

```
SessionEvents.addAuthListener(new AuthListener() {
    @Override
    public void onAuthSucceed() { updateFbStatus(); }

    @Override
    public void onAuthFail(String error) {
        txtFbStatus.setText("Facebook status: imposible iniciar sesion " + error);
    }
});
```

En el caso del listener para finalizar la sesión, necesito sobrecargar métodos para el inicio y finalización del proceso de cierre de sesión, le reporto al usuario estos eventos a través del TextView de status.

```
SessionEvents.addLogoutListener(new LogoutListener() {

    @Override
    public void onLogoutFinish() {
        txtFbStatus.setText("Facebook status: sesion no iniciada");
    }

    @Override
    public void onLogoutBegin() { txtFbStatus.setText("Facebook status: cerrando sesion..."); }
});
```

Para la parte de Twitter manejo 2 Listeners para el evento del click; el primer listener, el utilizado cuando aún no se inicia sesión, requiere que al darle click al botón obtenga el requestToken y a través de un intent con el URL recibido levante una actividad (el navegador) y muestre al usuario la pantalla de Twitter solicitando su aprobación.

```
twitter_auth = new OnClickListener() {
    @Override
    public void onClick(View v) {
        txtTwStatus.setText("Twitter status: iniciando sesion");

        try {
            String authUrl = provider.retrieveRequestToken(consumer, "mdw://twitter");
            startActivity(new Intent(Intent.ACTION_VIEW, Uri.parse(authUrl)));
        } catch (OAuthMessageSignerException e) {
            e.printStackTrace();
        } catch (OAuthNotAuthorizedException e) {
            e.printStackTrace();
        } catch (OAuthExpectationFailedException e) {
            e.printStackTrace();
        } catch (OAuthCommunicationException e) {
            e.printStackTrace();
        }

    }
};
```

El segundo listener, una vez ya ha ocurrido la autenticación, borrará las credenciales guardadas en los SharedPreferences.

```
twitter_clearauth = new OnClickListener() {
    @Override
    public void onClick(View v) {
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(getApplicationContext());
        SharedPreferences.Editor editor = prefs.edit();
        editor.putString("KEY", null);
        editor.putString("SECRET", null);
        editor.commit();
        btnTwLogin.setText("Autorizar twitter");
        txtTwStatus.setText("Twitter status: sesi\uFFFDn no iniciada ");
        btnTwLogin.setOnClickListener(twitter_auth);

    }
};
```

Es importante notar que las credenciales recibidas de Twitter (key y secret) no expiran y son válidas mientras el usuario no revoque los permisos de la aplicación, lo que hacemos al desautorizar es borrar las credenciales guardadas por lo que la siguiente vez que presionemos el botón solicitaremos nuevas.

Para terminar el método onCreate reviso el valor de twitter_active y actualizo el texto del botón y del TextView de status.

```
if (twitter_active) {
    txtTwStatus.setText("Twitter status: sesion iniciada ");
    btnTwLogin.setText("Deautorizar twitter");
    btnTwLogin.setOnClickListener(twitter_clearauth);
} else {
    btnTwLogin.setText("Autorizar twitter");
    btnTwLogin.setOnClickListener(twitter_auth);
}
```

Además del trabajo realizado en OnCreate, necesito modificar OnResume, una vez autorizado el uso de la aplicación de Twitter por parte del usuario, al recibir el redirect al callback mi aplicación lo recibirá por el intent-filter colocado previamente y se ejecutará el método OnResume.

Para almacenar las credenciales de forma persistente utilizo SharedPreferences es necesario obtener un editor, guardar los datos y luego realizar commit.

```
SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
SharedPreferences.Editor editor = prefs.edit();
editor.putString("KEY", ACCESS_KEY);
editor.putString("SECRET", ACCESS_SECRET);
editor.commit();
```

Luego actualizo el TextView del estado, el texto del botón y la acción asociada al mismo.

```
TextView txtTwStatus = (TextView) this.findViewById(R.id.txtTwStatus);
txtTwStatus.setText("Twitter status: sesion iniciada ");

btnTwLogin.setText("Deautorizar twitter");
btnTwLogin.setOnClickListener(twitter_clearauth);
```

Adicional a esto y fuera de la condición de que el intent tenga datos, es posible que el método onResume sea llamado y los valores del TextView de status de Facebook se hayan perdido, entonces es necesario revisarlo.

```
if (facebook_active) {
    updateFbStatus();
}
```

Entonces el método onResume() queda de la siguiente forma:

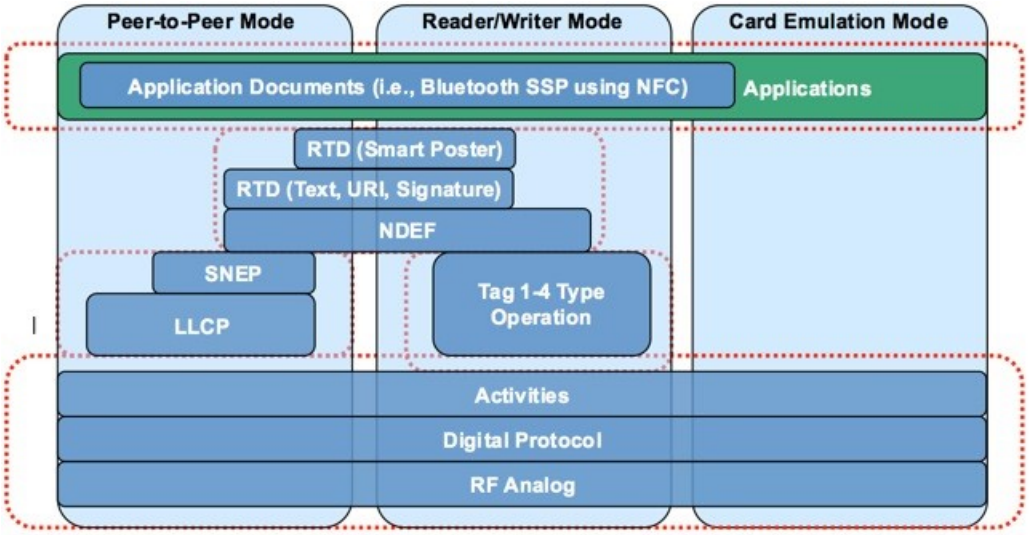
```
@Override
public void onResume() {
    super.onResume();
    Uri uri = this.getIntent().getData();
    if (uri != null && uri.toString().startsWith("mdw://twitter")) {
        String verifier = uri.getQueryParameter(OAuth.OAUTH_VERIFIER);
        try {
            provider.retrieveAccessToken(consumer,verifier);
            ACCESS_KEY = consumer.getToken();
            ACCESS_SECRET = consumer.getTokenSecret();

            SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(this);
            SharedPreferences.Editor editor = prefs.edit();
            editor.putString("KEY", ACCESS_KEY);
            editor.putString("SECRET", ACCESS_SECRET);
            editor.commit();

            TextView txtTwStatus = (TextView) this.findViewById(R.id.txtTwStatus);
            txtTwStatus.setText("Twitter status: sesion iniciada ");

            btnTwLogin.setText("Deautorizar twitter");
            btnTwLogin.setOnClickListener(twitter_clearauth);
```

```
        } catch (OAuthMessageSignerException e) {
            e.printStackTrace();
        } catch (OAuthNotAuthorizedException e) {
            e.printStackTrace();
        } catch (OAuthExpectationFailedException e) {
            e.printStackTrace();
        } catch (OAuthCommunicationException e) {
            e.printStackTrace();
        }

    }

    if (facebook_active) {
        updateFbStatus();
    }
}
```

Finalmente, la aplicación debería verse como en la imagen a continuación:

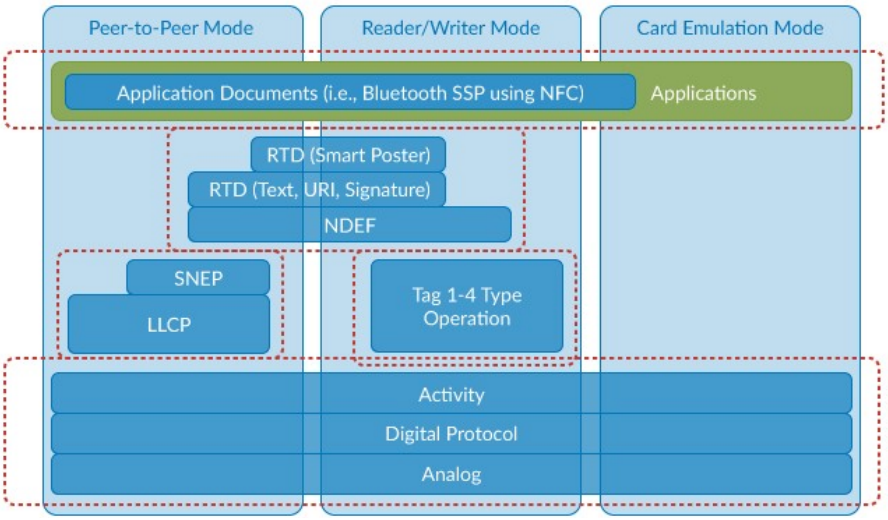
# Aplicación NFC

# Introducción




Utilizando los elementos clave en los estándares existentes y reconocidos como ISO/IEC 18092 y ISO/IEC 14443-2,3,4, así como el JIS X6319-4, el Foro de especificaciones de NFC formado por un estándar de tecnología que armoniza y extiende estándares sin-contacto existentes, desbloqueando todas las capacidades de la tecnología NFC a través de los diferentes modos de operación sin-contacto, modo peer-to-peer, modo lector/escritor, modo de emulación de tarjeta.
Para descubrir los diferentes tipos de especificaciones del foro NFC que habilitan estos modos de operación, serán descritos a continuación:

**Tipo de etiqueta:**




Especificación de operación de etiqueta tipo 1 de fórum NFC
Basada en ISO/IEC 14443. Las etiquetas son capaces de leer y reescribir; usuarios pueden configurarlas para que se vuelvan de solo lectura. Disponibilidad de memoria 96bytes expandible a 2kb

Especificación de operación de etiqueta tipo 2 de fórum NFC
Mismas características anteriores. Disponibilidad de memoria 48bytes expandible a 2kb

Especificación de operación de etiqueta tipo 3 de fórum NFC
Basada en el estándar industrial japonés(JIS) X 6319-4, también conocido como FeliCa.Las etiquetas están pre configuradas en la fabricación para ser ya sea de lectura, reescritura o solo lectura. Disponibilidad de memoria variable, limite 1MB por servicio.

Especificación de operación de etiqueta tipo 4 de fórum NFC
Compatible con las series de estándar ISO/IEC 14443. Las etiquetas son pre configuradas en la fabricación para ser ya sea de lectura, reescritura o solo lectura. Disponibilidad de memoria variable, hasta 32KB por servicio, la interfaz de comunicación puede ser Tipo A o tipo B compatible

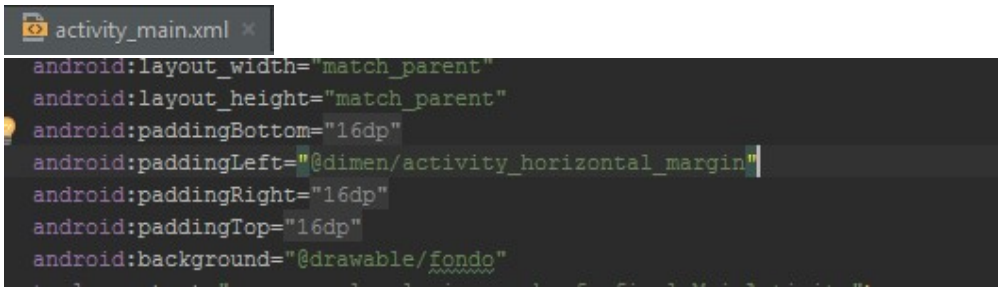
Para el desarrollo de la aplicación utilizando etiquetas NFC se utilizó un dispositivo Nexus S al igual que fueron proporcionadas distintas etiquetas NFC.

Al llevar a cabo investigaciones respecto a la forma de implementar estas aplicaciones, se optó por utilizar Android Studio.

Al principio del desarrollo de la aplicación se fueron haciendo pruebas las cuales tenían el propósito de poder familiarizarse con la plataforma y su funcionamiento, al igual que el código y la interfaz.

Al momento de tener un conocimiento básico sobre el desarrollo de una aplicación básica se empezó a desarrollar la aplicación PruebaNFC en donde ya se utilizarán las características que se relacionan con las etiquetas NFC.
Esta aplicación desarrollada toma un texto escrito por el usuario dentro de la aplicación y la escribe en la etiqueta utilizando el NFC, al momento de escribir el texto en la etiqueta, se selecciona la opción de leer, en donde al momento de acercar la etiqueta al teléfono, este lee lo escrito en la etiqueta.

Igualmente cabe mencionar que, al momento del desarrollo de esta aplicación, se modificó la interfaz de modo que sea un poco más amigable y tenga un aspecto atractivo. De esta forma, se agregó un fondo de pantalla personalizado al igual que se cambiaron los colores de los distintos puntos de la aplicación.



```
<EditText
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:id="@+id/txtTagContent"
    android:layout_below="@+id/tglReadWrite"
    android:backgroundTint="@android:color/background_dark" />
```

```
    <ToggleButton
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/tglReadWrite"
        android:textOn="Leer Etiqueta"
        android:textOff="Escribir Etiqueta"
        android:checked="true"
        android:onClick="tglReadWriteOnClick"
        android:layout_marginTop="131dp"
        android:theme="@style/Base.ThemeOverlay.AppCompat.Dark"
        android:textColorLink="?attr/colorBackgroundFloating"
        style="@style/Base.ThemeOverlay.AppCompat.Dark"
        android:layout_alignParentTop="true"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />

</RelativeLayout>
```

El botón de activación

colors.xml

```
<resources>
    <color name="colorPrimary">#8A0829</color>
    <color name="colorPrimaryDark">#0000</color>
    <color name="colorAccent">#FFFFFF</color>
</resources>
```

Los colores que tendrán algunas partes de la aplicación

AndroidManifest.xml

```
<uses-permission android:name="android.permission.NFC"/>
<uses-feature
    android:name="android.hardware.nfc"
    android:required="false" />
```

El permiso que se ocupa para el uso del NFC en la aplicación

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="PruebaNFC-FINAL"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN" />

            <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
    </activity>
</application>
```

MainActivity.java

```
public class MainActivity extends AppCompatActivity {

    NfcAdapter nfcAdapter;
    ToggleButton tglReadWrite;
    EditText txtTagContent;
```

Algunos métodos que ocupamos para NFC
Como lo es el ayudante para obtener el adaptador NFC predeterminado
El botón de activación
Y donde tendrá el texto del contenido de la etiqueta

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    nfcAdapter = NfcAdapter.getDefaultAdapter(this);
    tglReadWrite = (ToggleButton)findViewById(R.id.tglReadWrite);
    txtTagContent = (EditText) findViewById(R.id.txtTagContent);

    if(nfcAdapter!=null && nfcAdapter.isEnabled()) {
        //Toast.makeText(this, "NFC available!", Toast.LENGTH_LONG).show();

    }else{
        //Toast.makeText(this, "NFC not available :(", Toast.LENGTH_LONG).show();
        finish();
    }
}
```

Se inicia la aplicación, se Obtendrá la `Bundle` nulo cuando la actividad get empieza el horario de firt y se pondrá en uso cuando la orientación actividad se ve modificado.
Nfcadapter recibirá el defaultadapter

Si el nfc adapter es diferente de null y está habilitado  entonces se podrá usar y estará disponible de lo contrario no se podrá usar la aplicación

```
@Override
protected void onNewIntent(Intent intent) {
    super.onNewIntent(intent);

    if(intent.hasExtra(NfcAdapter.EXTRA_TAG))
    {
        Toast.makeText(this,"Intento de NFC recibido!", Toast.LENGTH_SHORT).show();

        if(tglReadWrite.isChecked()){

            Parcelable[] parcelables = intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES);
            if(parcelables  != null && parcelables.length>0){

                readTextFromMessage((NdefMessage)parcelables[0]);
```

Los intentos son operaciones que se realizaran
El intent.hasExtra para comprobar si un extra con un nombre se aprobó en el intento
Si es verdad desplegara un mensaje en la parte inferior del celular con un Toast.makeText diciendo intento de NFC Recibido
Lo comprobara y lo guardara en un arreglo de Parcelable

```
            }else{
                Toast.makeText(this,"No se encontraron mensajes NDEF", Toast.LENGTH_SHORT).show();

            }


        }else {
            Tag tag = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG);
            NdefMessage ndefMessage = createNdefMessage(txtTagContent.getText()+"");

            writeNdfMessage(tag, ndefMessage);
        }


    }

}
```

Si es falso desplegara un Toast.makedText diciendo que no se encontraron mensajes NDEF que es un formato de intercambio de datos NFC
Si no se recibió el mensaje NFC entonces se hará un método para poder escribir el mensaje NFC

```
private void readTextFromMessage(NdefMessage ndefMessage) {

    NdefRecord[] ndefRecords = ndefMessage.getRecords();
    if(ndefRecords != null && ndefRecords.length>0){

        NdefRecord ndefRecord = ndefRecords[0];
        String tagContent = getTextFromNdefRecord(ndefRecord);
        txtTagContent.setText(tagContent);
        String url;
        if(tagContent.startsWith("http://"))
            url=tagContent;
        else
            url="http://"+tagContent;
        Intent i=new Intent(Intent.ACTION_VIEW);
        i.setData(Uri.parse(url));
        startActivity(i);


    }else{
        Toast.makeText(this,"No hay records NDEF",Toast.LENGTH_SHORT).show();
    }

}
```

Se hará un método para leer el mensaje
NDefRecord obtiene  los registros NDEF dentro del mensaje NDEF,
Que se almacenara en el tagcontent
Lo que está en azul es para poder abrir el navegador se hace un string
Prueba si el contenido que tiene tagcontent comienza con http://
Si comienza así entonces tagcontent  se almacena en el url
Si no comienza así se le agrega el http://
Se hace un Intent
Y se verifica noto el url con el parse
E inicializa con el startActivity
Si no hay ningún registro NDEF entonces despliega el mesaje diciendo que no hay records NDEF

```
@Override
 protected void onResume() {

    super.onResume();

    enableForegroundDispatchSystem();
 }
```

Es para continuar con la aplicación que se detuvo

Es similar que onCreate

```
private void enableForegroundDispatchSystem(){

    Intent intent = new Intent(this, MainActivity.class).addFlags(Intent.FLAG_RECEIVER_REPLACE_PENDING);
    PendingIntent pendingIntent = PendingIntent.getActivity(this,0,intent,0);
    IntentFilter[] intentFilters = new IntentFilter[]{};
    nfcAdapter.enableForegroundDispatch(this, pendingIntent, intentFilters, null);
}
```

Este método es el que utiliza onResume

```
private void formatTag(Tag tag, NdefMessage ndefMessage){
      try {
          NdefFormatable ndefFormatable = NdefFormatable.get(tag);
          if(ndefFormatable==null){
              Toast.makeText(this,"Tag no es ndef formateable", Toast.LENGTH_SHORT).show();
              return;
          }
          ndefFormatable.connect();
          ndefFormatable.format(ndefMessage);
          ndefFormatable.close();
          Toast.makeText(this,"Etiqueta ha sido escrita", Toast.LENGTH_SHORT).show();


      }catch (Exception e)
      {
          Log.e("formatTag", e.getMessage());
      }

  }
```

Este método es para tener acceso a las operaciones de formato NDEF en un tag
Primero verifica si es formatable , si lo que tiene la etiqueta es null, es que no se puede formatear a NDEF.
Nota: las etiquetas no pueden ser cualquiera tienen que ser compatible con el celular que uses.
Si no es null es que si se pudo escribir el mensaje en la etiqueta

```
private void writeNdfMessage(Tag tag, NdefMessage ndefMessage)
{
    try {
        if(tag == null){
            Toast.makeText(this,"Objeto Etiqueta no puede ser null", Toast.LENGTH_SHORT).show();
            return;
        }

        Ndef ndef = Ndef.get(tag);

        if(ndef == null){
            //Formatea etiqueta conel formato ndef y escribe el mensaje
            formatTag(tag, ndefMessage);
```

Es un método de verificación

```
            }else{
                ndef.connect();

                if(!ndef.isWritable())
                {
                    Toast.makeText(this,"Etiqueta no se puede escribir", Toast.LENGTH_SHORT).show();

                    ndef.close();
                    return;
                }

                ndef.writeNdefMessage(ndefMessage);
                ndef.close();
                Toast.makeText(this,"Etiqueta ha sido escrita1", Toast.LENGTH_SHORT).show();

            }

        }catch (Exception e){
            Log.e("writeNdefMessage", e.getMessage());
        }
    }
```

Si la etiqueta no es  null  entonces verifica si la etiqueta se puede escribir si no se pude despliega un mensaje diciendo que la etiqueta no se pude escribir
Y lo cierra
Si no es diferente entonces quiere decir que si se pudo escribir y desplegara un mensaje donde dirá la etiqueta se ha escrito

```
private NdefRecord createTextRecord(String content){
    try{
        byte[] language;
        language = Locale.getDefault().getLanguage().getBytes("UTF-8");

        final byte[] text = content.getBytes("UTF-8");
        final int languageSize = language.length;
        final int textLength = text.length;
        final ByteArrayOutputStream payload = new ByteArrayOutputStream(1+ languageSize + textLength);

        payload.write((byte) (languageSize & 0x1F));
        payload.write(language,0,languageSize);
        payload.write(text,0,textLength);

        return new NdefRecord(NdefRecord.TNF_WELL_KNOWN,NdefRecord.RTD_TEXT,new byte[0],payload.toByteArray());
    }catch (UnsupportedEncodingException e){
        Log.e("createTextRecord",e.getMessage());
    }
    return null;
}
```

En este método se crea un Nuevo registro NDEF que contenga datos de texto UTF-8
Para codificar el lenguaje que se está utilizando y no reciba otros símbolos extraños

```
private NdefMessage createNdefMessage(String content){

    NdefRecord ndefRecord = createTextRecord(content);

    NdefMessage ndefMessage = new NdefMessage(new NdefRecord[]{ndefRecord});

    return ndefMessage;
}
```

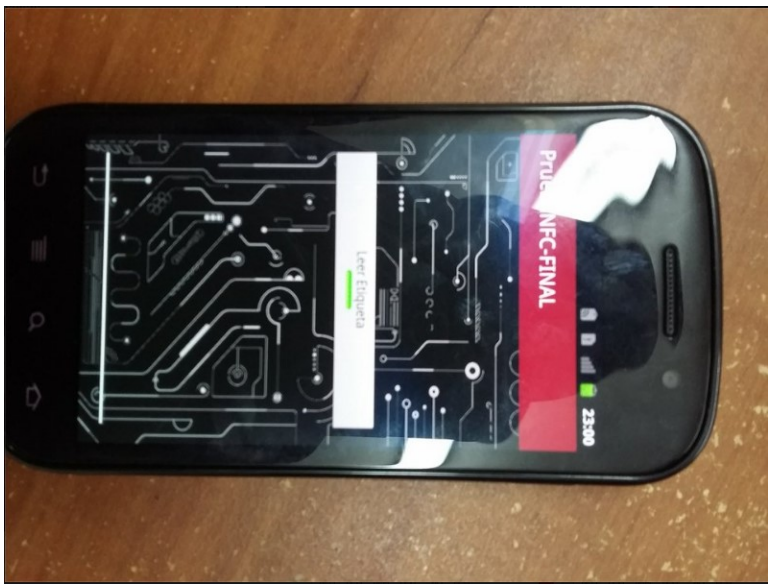
Se crea el NDEFMessage

```
public void tglReadWriteOnClick(View view) { txtTagContent.setText(""); }
```

Este es el método para cuando se le de click al botón de escribir, puedas escribir en la etiqueta

```
public String getTextFromNdefRecord(NdefRecord ndefRecord){
    String tagContent = null;
    try{
        byte[] payload = ndefRecord.getPayload();
        String textEncoding = ((payload[0] & 128) == 0) ? "UTF-8" : "UTF-16";
        int languageSize = payload[0] & 0063;
        tagContent = new String(payload,languageSize+1,payload.length-languageSize-1,textEncoding);

    }catch (UnsupportedEncodingException e){
        Log.e("getTextFromNdefRecord",e.getMessage(),e);
    }
    return tagContent;
}
/*////////////////////////////////////////////////////*/


}
```

Este es el método para poder obtener texto de los registros ya con la codificación del lenguaje y se almacena en la cadena de bytes
Y se devolverá el contenido de la etiqueta
Si ndefRecord no tiene contenido de carga útil  entonces se devolverá una matriz de bytes vacía



**Todo lo que puedes hacer con el NFC, empezando con aplicaciones**

## Automatizar perfiles

# Hogar

Compartir la clave del wifi

Al llegar a casa

Para dormir

# Pago

BBVA Wallet

# Identificación

Además, la conectividad NFC de nuestros terminales puede utilizarse para identificarse en determinados casos.

Algunas compañías aereas estaban dando la **posibilidad de hacer check-in** con nuestros teléfonos gracias a esta conectividad, algo que sin duda facilita y agiliza el proceso.

# Facilitar tareas a personas mayores

# Todo un mundo de posibilidades

# Transmisión de Datos vía Wi-Fi

# Transmisión de Datos en Android (WiFi)

Android permite a sus aplicaciones acceder y ver el estado de conexiones inalámbricas a muy bajo nivel. Las aplicaciones casi pueden ver toda la información relacionada con una conexión WiFi.
La información que puede ser observada por una aplicación incluye la velocidad de conexión, la dirección IP, el estado de negociación, entre otros. Las aplicaciones también pueden escanear, agregar, guardar, terminar e iniciar conexiones WiFi.
Android provee una API llamada **WifiManager** para manejar todos los aspectos de una conectividad WiFi. Se puede instanciar esta clase llamando al método **getSystemService**. Su sintaxis es la siguiente:

```
WifiManager mainWifiObj;
mainWifiObj = (WifiManager) getSystemService(Context.WIFI_SERVICE);
```

Para poder escanear una lista de redes WiFi, también se debe registrar un BroadcastReceiver. Se registra utilizando el método registerReceiver con un objeto de tipo Receiver como argumento. Su sintaxis es la siguiente:

```
class WifiScanReceiver extends BroadcastReceiver {
   public void onReceive(Context c, Intent intent) {
   }
}

WifiScanReceiver wifiReciever = new WifiScanReceiver();
```

El scan puede iniciarse llamando al método **startScan** de la clase WifiManager. Este método regresa una lista de objetos **ScanResult**. Se puede acceder a cualquier objeto utilizando el método **get** de lista. Su sintaxis es la siguiente:

```
List<ScanResult> wifiScanList = mainWifiObj.getScanResults();
String data = wifiScanList.get(0).toString();
```

Aparte de escanear, se tiene más control en una conexión inalámbrica utilizando los métodos definidos en la clase WifiManager, unos ejemplos incluyen:

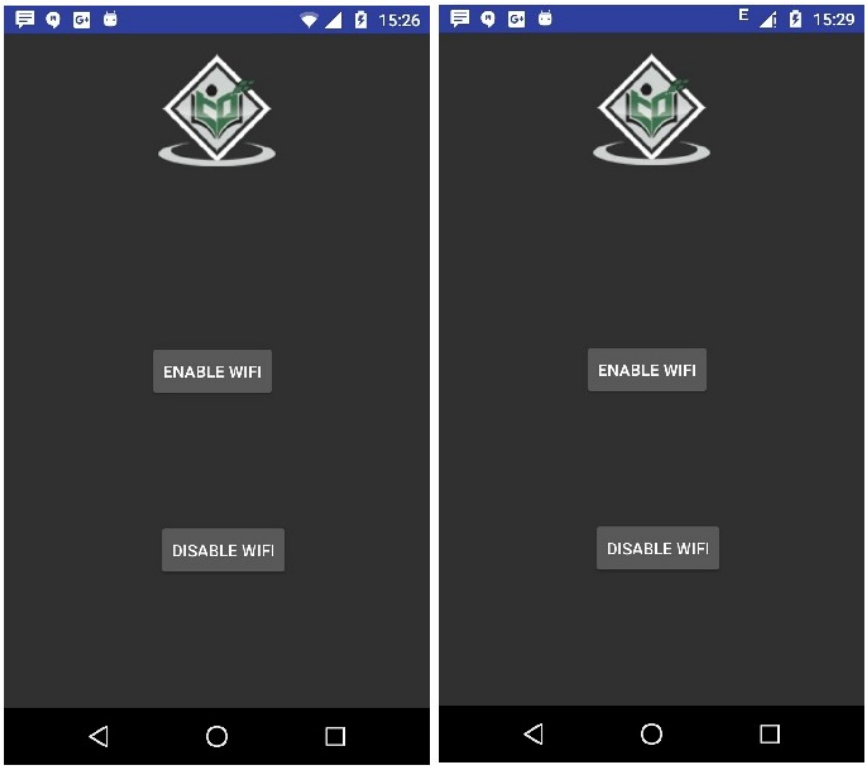
| No. | Método y descripción |
|---|---|
| 1 | **addNetwork (WifiConfiguration config):**<br>Este método agrega una nueva descripción de una red a un grupo de redes ya configuradas. |
| 2 | **createWifiLock (String tag):**<br>Este método crea un WifiLock para bloquear la pantalla de bloqueo de algún dispositivo. |
| 3 | **disconnect():**<br>Este método se desasocia del punto de acceso actualmente activo. |
| 4 | **enableNetwork (int netId, boolean disableOthers):**<br>Este metodo permite asociarse a una red previamente configurada. |
| 5 | **getWifiState():**<br>Este método regresa el estado de una conexión WiFi. |
| 6 | **isWifiEnabled():**<br>Este método regresa si está o no habilitado el WiFi. |
| 7 | **setWifiEnabled():**<br>Este método habilita o deshabilita una conexión WiFi. |
| 8 | **updateNetwork (WifiConfiguration config):**<br>Este método actualiza la descripción de una red ya existente. |

# Ejemplo

Aquí se presenta un ejemplo sencillo de una aplicación que controla el WiFi de un dispositivo.

```
package com.example.sairamkrishna.myapplication;

import android.net.wifi.WifiManager;
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class MainActivity extends Activity {
   Button enableButton,disableButton;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      enableButton=(Button)findViewById(R.id.button1);
      disableButton=(Button)findViewById(R.id.button2);

      enableButton.setOnClickListener(new OnClickListener(){
         public void onClick(View v){
            WifiManager wifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);
            wifi.setWifiEnabled(true);
         }
      });

      disableButton.setOnClickListener(new OnClickListener(){
         public void onClick(View v){
            WifiManager wifi = (WifiManager) getSystemService(Context.WIFI_SERVICE);
            wifi.setWifiEnabled(false);
         }
      });
   }
}
```

Al correr la aplicación se muestra la siguiente:


15:26
ENABLE WIFI
DISABLE WIFI
E
15:29
ENABLE WIFI
DISABLE WIFI