

Tema 6. Mapas y GPS

Contenido

1	Bibliografía	2
2	Introducción	2
3	Ubicación actual en la clase Aplicacion	2
4	CasosUsoLocalización. Permisos de localización.....	3
4.1	Disecionando el código.....	8
4.2	Introducimos CasosUsoLocalizacion en MainActivity	12
5	Usando mapas en la aplicación MisLugares.....	14
5.1	Introducción	14
5.2	Conectando mi Aplicación con GoogleMaps. La consola google	14
5.2.1	La API de Google maps.....	14
5.2.2	Las librerías.....	24
5.3	Funcionalidad de la API de GoogleMaps. La cámara.	25
5.3.1	Edificios en 3D en el mapa	25
5.3.2	La posición de la cámara	26
5.3.3	Ubicación del objetivo) target.....	27
5.3.4	Rumbo (orientación), bearing	27
5.3.5	Inclinación (ángulo de visión), tilt	28
5.3.6	Enfocar	30
5.3.7	Moviendo la cámara.....	32
5.3.8	Cambiar el nivel de zoom y establecer el zoom mínimo / máximo	32
5.4	El layout para el fragment del mapa	33
5.5	La Actividad Mapa MapActivity. La Api de GoogleMaps.....	34
5.5.1	Entendiendo el código.....	37
5.5.2	OnMapReady.....	37
5.5.3	onInforWindowClick.....	40
6	Añadiendo distancias al RecyclerView	40
6.1	Modificamos el Layout para elemento_lista.xml	41
6.1.1	Calculo de distancias en la tierra. Fórmula del semiverseno	42
6.2	Modificando el adaptador para añadir las distancias	44
7	Tarea.....	45

1 Bibliografía

El gran libro de Android de Jesús Tomas Girones.

Este tema comprende el **tema 7** del libro Gran Libro de Android.

<http://www.androidcurso.com/index.php/mooc/programa-mooc>

<https://developer.android.com/>

2 Introducción

El objetivo de este tema es **introducir el uso de mapas en mis lugares**, así como las localizaciones. Dependemos del proveedor **de servicios GPS**, así como del **proveedor de servicios de internet** para las ubicaciones geográficas. Para realizar **los cálculos de información de localización geográfica**, Android usa **o los satélites y el GPS**, o las **antenas del proveedor de servicios de telefonía**. No me voy a extender en cómo funciona pero si alguien está interesado lo puede encontrar en el siguiente enlace.

<http://www.androidcurso.com/index.php/491>

Igualmente introduciremos la **API de Google Maps** en nuestra aplicación, que además nos permitirá **asociar nuestros lugares con puntos en el mapa** y en su caso proporcionar al usuario información extra. Para ello necesitaremos **acceder a la consola de Google**, con nuestra propia **cuenta de Google** y asociar con **un sistema de clave pública y privada nuestra aplicación** con la **Api de Google maps**. Es un sistema de seguridad de Google para intentar evitar ataques cibernéticos a sus sistemas de mapas y navegación. Cada aplicación debe verificarse en la consola de Google, evitando accesos indeseados y teniendo controlados a los creadores de aplicaciones, al igual que las claves de subida y carga de Google Play como ya explicamos anteriormente.

Igualmente, y empezaremos por ahí, **necesitamos asignar permisos de localización a la aplicación**. Vamos a empezar por ahí. Al igual que con **los permisos de almacenamiento tenemos que gestionar los permisos de localización**. Pero además, tenemos que gestionar la **red del proveedor de servicios y la red GPS** de nuestro dispositivo. Vamos a crear un nuevo caso de uso llamado **CasosUsoLocalización** para realizar esta tarea.

3 Ubicación actual en la clase Aplicacion

Al introducir las ubicaciones geográficas en nuestra aplicación, vamos a aprovechar las ventajas de tener disponible nuestra ubicación actual. Entre otras cosas, vamos a guardar nuestra ubicación actual en el contexto de aplicación, en la clase Aplicacion.

Definimos en aplicacion la propiedad posición actual.

```
public class Aplicacion extends Application {  
    private LugaresBD lugares;  
    private AdaptadorLugaresBD adaptador;  
    public GeoPunto posicionActual = new GeoPunto(0.0, 0.0);  
}
```

En CasosUsoLocalizacion vamos a modificar nuestra posición actual. Esta es la razón por la que **inicializamos CasosUsoLocalizacion en MainActivity**, para tener la ubicación accesible desde el principio.

4 CasosUsoLocalización. Permisos de localización

La manera **de pedir permisos va a ser similar al caso de fotografías y almacenamiento**. Pero en este caso **es más complicado porque manejamos las variantes del proveedor** de GPS y de telefonía.

Primero vamos a añadir en el **manifest** el permiso en cuestión, en este caso: **ACCESS_FINE_LOCATION**. Tenemos dos permisos básicos para acceder a la localización del dispositivo. Uno de ellos es ACCESS_COARSE_LOCATION, que nos permite acceder a una localización aproximada del dispositivo. ACCESS_FINE_LOCATION si embargo nos permite acceder a una localización precisa del mismo.

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"  
tools:node="replace"/>
```

Vamos a crear **una nueva clase CasosUsoLocalizacion** como sigue, y como siempre vamos a explicar el código.

```
import android.Manifest;  
import android.app.Activity;  
import android.content.DialogInterface;  
import android.content.pm.PackageManager;  
import android.location.Location;  
import android.location.LocationListener;
```

```

import android.location.LocationManager;
import android.os.Bundle;
import android.util.Log;

import com.example.mislugares2019.Adaptadores.AdaptadorLugares;
import com.example.mislugares2019.Adaptadores.AdaptadorLugaresBD;
import com.example.mislugares2019.modelo.GeoPunto;
import com.example.mislugares2019.presentacion.Aplicacion;

import androidx.appcompat.app.AlertDialog;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;

import static android.content.Context.LOCATION_SERVICE;

public class CasosUsoLocalizacion implements LocationListener,
ActivityCompat.OnRequestPermissionsResultCallback {
    private static final String TAG = "MisLugares2019";
    private Activity actividad;
    private int codigoPermiso;
    private LocationManager manejadorLoc;
    private Location mejorLoc;
    private GeoPunto posicionActual;
    private AdaptadorLugaresBD adaptador;
    private static final long DOS_MINUTOS = 2 * 60 * 1000;

    public CasosUsoLocalizacion(Activity actividad, int codigoPermiso)
    {
        this.actividad = actividad;
        this.codigoPermiso = codigoPermiso;
        manejadorLoc = (LocationManager)
        actividad.getSystemService(LOCATION_SERVICE);
        posicionActual = ((Aplicacion) actividad.getApplication())
        .posicionActual;
        adaptador = ((Aplicacion)
        actividad.getApplication()).getAdaptador();
        if (!hayPermisoLocalizacion()) {

            solicitarPermiso(Manifest.permission.ACCESS_FINE_LOCATION,
                "Necesitas permisos de localización para que el
mapa funcione ",
                codigoPermiso, this.actividad);
        }
        ultimaLocalizacion();
    }

    public static void solicitarPermiso(final String permiso, String
        justificacion, final int requestCode, final Activity
        actividad) {
        if
        (ActivityCompat.shouldShowRequestPermissionRationale(actividad,
            permiso)) {
            new AlertDialog.Builder(actividad)
                .setTitle("Solicitud de permiso")
                .setMessage(justificacion)
                .setPositiveButton("Ok", new
        DialogInterface.OnClickListener() {
            public void onClick(DialogInterface dialog,
            int whichButton) {

```

```

ActivityCompat.requestPermissions(actividad,
                                new String[]{permiso},
requestCode);
                                })).show();
    } else {
        ActivityCompat.requestPermissions(actividad,
            new String[]{permiso}, requestCode);
    }
}

public boolean hayPermisoLocalizacion() {
    return (ActivityCompat.checkSelfPermission(
        actividad, Manifest.permission.ACCESS_FINE_LOCATION)
        == PackageManager.PERMISSION_GRANTED);
}

void ultimaLocalizacion() {
    try {
        if (hayPermisoLocalizacion()) {
            if
(manejadorLoc.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
actualizaMejorLocaliz (manejadorLoc.getLastKnownLocation(
                        LocationManager.GPS_PROVIDER));
            }
            if
(manejadorLoc.isProviderEnabled(LocationManager.NETWORK_PROVIDER)) {
actualizaMejorLocaliz (manejadorLoc.getLastKnownLocation(
                        LocationManager.NETWORK_PROVIDER));
            } else {
solicitarPermiso(Manifest.permission.ACCESS_FINE_LOCATION,
                "Sin el permiso localización no puedo
mostrar la distancia" +
                " a los lugares.", codigoPermiso,
actividad);
            }
        }
    } catch (SecurityException e) {
        System.out.println("Error en permisos");
    }
}

public void permisoConcedido() {
    ultimaLocalizacion();
    activarProveedores();
    adaptador.notifyDataSetChanged();
}

private void activarProveedores() {
    try {
        if (hayPermisoLocalizacion()) {
            if

```

```

(manejadorLoc.isProviderEnabled(LocationManager.GPS_PROVIDER)) {

    manejadorLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,
        20 * 1000, 5, this);

    }
    if
    (manejadorLoc.isProviderEnabled(LocationManager.NETWORK_PROVIDER)) {

    manejadorLoc.requestLocationUpdates(LocationManager
        .NETWORK_PROVIDER, 10 * 1000, 10, this);

    }
    } else {

    solicitarPermiso(Manifest.permission.ACCESS_FINE_LOCATION,
        "Sin el permiso localización no puedo mostrar
    la distancia" +
        " a los lugares.", codigoPermiso,
    actividad);

    }

    } catch (SecurityException s) {

        System.out.println("Error en Permisos");

    }

    }

    @Override public void onLocationChanged(Location location) {
        Log.d(TAG, "Nueva localización: "+location);
        actualizaMejorLocaliz(location);
        adaptador.notifyDataSetChanged();
    }
    @Override public void onProviderDisabled(String proveedor) {
        Log.d(TAG, "Se deshabilita: "+proveedor);
        activarProveedores();
    }
    @Override public void onProviderEnabled(String proveedor) {
        Log.d(TAG, "Se habilita: "+proveedor);
        activarProveedores();
    }
    @Override
    public void onStatusChanged(String proveedor, int estado, Bundle
    extras) {
        Log.d(TAG, "Cambia estado: "+proveedor);
        activarProveedores();
    }

    private void actualizaMejorLocaliz(Location localiz) {
        if (localiz != null && (mejorLoc == null
            || localiz.getAccuracy() < 2*mejorLoc.getAccuracy()
            || localiz.getTime() - mejorLoc.getTime() >
    DOS_MINUTOS)) {
            Log.d(TAG, "Nueva mejor localización");
            mejorLoc = localiz;
            ((Aplicacion)
    actividad.getApplication()).posicionActual.setLatitud(
        localiz.getLatitude());
        }
    }

```

```
((Aplicacion)
actividad.getApplication()).posicionActual.setLongitud(
    localiz.getLongitud());

    }
}

public void activar() {

    try {
        if (hayPermisoLocalizacion())
            activarProveedores();
    } catch (SecurityException e) {

    }

}

public void desactivar() {

    try {
        if (hayPermisoLocalizacion())

            manejadorLoc.removeUpdates(this);

    } catch (SecurityException e) {

    }

}

@Override public void onRequestPermissionsResult(int requestCode,
String[]
permissions, int[] grantResults) {
    if (requestCode == codigoPermiso
        && grantResults.length == 1
        && grantResults[0] ==
PackageManager.PERMISSION_GRANTED)
        permisoConcedido();

}

@Override public void onRequestPermissionsResult(int requestCode,
String[]
permissions, int[] grantResults) {
    if (requestCode == codigoPermiso
        && grantResults.length == 1
        && grantResults[0] ==
PackageManager.PERMISSION_GRANTED)
        permisoConcedido();

}

}
```

4.1 Diseccionando el código

La parte de petición de permisos es exactamente igual que para los permisos de almacenamiento que vimos anteriormente.

```
private LocationManager manejadorLoc;  
private Location mejorLoc;  
  
private static final long DOS_MINUTOS = 2 * 60 * 1000;
```

Nos fijamos ahora en el interface **que implementamos, LocationListener**. **LocationListener** Se utiliza para recibir notificaciones del **LocationManager** cuando **la ubicación ha cambiado**. **Debemos sobrescribir cuatro métodos** que enumeraré a continuación.

Las siguientes propiedades de la clase nos ayudaran a manejar proveedores y localizaciones.

LocationManager es el manejador de localizaciones de Android. Esta clase proporciona acceso a los servicios de ubicación del sistema. Estos servicios permiten que las aplicaciones obtengan actualizaciones periódicas de la ubicación geográfica del dispositivo, o que activen una aplicación especificada (con un intent) cuando el dispositivo entra en la proximidad de una ubicación geográfica determinada.

<https://developer.android.com/reference/kotlin/android/location/LocationManager>

Location Una clase de datos que representa una ubicación geográfica. Una ubicación puede consistir en una latitud, longitud, time-stamp y otra información como rumbo (bearing) dirección geográfica (norte, sur, etc.), altitud y velocidad.

<https://developer.android.com/reference/android/location/Location>

DOS_MINUTOS: es el tiempo en milisegundos.

onLocationChanged: se activa o ejecuta cuando cambia la localización.

onProviderDisabled: se ejecuta cuando el proveedor de servicios se deshabilita, por una acción del usuario (apagado, modo avión, etc.) o una pérdida de red.

onProviderEnabled: se ejecuta cuando el proveedor de servicios se habilita.

onStatusChanged: cambia el estado del proveedor.

Para cada una de estas situaciones debemos llevar a cabo acciones. **Cuando cambie la localización debemos recoger la nueva localización**. Cuando cambien el estado de los proveedores debemos activar los proveedores.

// Métodos de la interfaz LocationListener

```
@Override public void onLocationChanged(Location location) {
    Log.d(TAG, "Nueva localización: "+location);
    actualizaMejorLocaliz(location);
    adaptador.notifyDataSetChanged();
}

@Override public void onProviderDisabled(String proveedor) {
    Log.d(TAG, "Se deshabilita: "+proveedor);
    activarProveedores();
}

@Override public void onProviderEnabled(String proveedor) {
    Log.d(TAG, "Se habilita: "+proveedor);
    activarProveedores();
}

@Override
public void onStatusChanged(String proveedor, int estado, Bundle extras) {
    Log.d(TAG, "Cambia estado: "+proveedor);
    activarProveedores();
}
```

El método activar proveedores activa tanto el proveedor GPS como el de telefonía.

Si el proveedor está disponible, pide actualizaciones cada 20 segundos para el GPS, diez segundos para el proveedor de red telefónica. Sino hay permisos de localización, se solicitan.

```
private void activarProveedores() {

    try {
        if (hayPermisoLocalizacion()) {
            if (manejadorLoc.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
                manejadorLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,
                    20 * 1000, 5, this);
            }
            if (manejadorLoc.isProviderEnabled(LocationManager.NETWORK_PROVIDER)) {
                manejadorLoc.requestLocationUpdates(LocationManager
                    .NETWORK_PROVIDER, 10 * 1000, 10, this);
            }
        } else {
            solicitarPermiso(Manifest.permission.ACCESS_FINE_LOCATION,
                "Sin el permiso localización no puedo mostrar la distancia" +
                " a los lugares.", codigoPermiso, actividad);
        }
    }
```

```

    } catch (SecurityException s) {

        System.out.println("Error en Permisos");

    }
}

```

`manejadorLoc.isProviderEnabled` comprueba que el proveedor de servicios GPS o Red está activado, para poder peticiones de ubicación.

Al método `manejadorLoc.requestLocationUpdates` le proporcionamos **el proveedor de red o GPS**, un tiempo de espera de 10 segundos para Network y 20 para GPS, la distancia mínima en metros, y un `LocationListener` para esperar la respuesta, en este caso nuestra propia clase **CasoUsoLocalizacion es un LocationListener**. solicita peticiones de actualización de la ubicación geográfica.

```

manejadorLoc.requestLocationUpdates(LocationManager.GPS_PROVIDER,
    20 * 1000, 5, this);

```

```

manejadorLoc.requestLocationUpdates(LocationManager
    .NETWORK_PROVIDER, 10 * 1000, 10, this);

```

Activar y desactivar las actualizaciones de localización nos permite que nuestra aplicación reciba o no las **actualizaciones de información geográfica**. `manejadorLoc.removeUpdates(this);` para todas las actualizaciones de localización. Ya hemos visto cómo actúa `activarProveedores`.

```

public void activar() {

    try {
        if (hayPermisoLocalizacion())
            activarProveedores();
    } catch (SecurityException e) {

    }

}

public void desactivar() {

    try {

```

```

        if (hayPermisoLocalizacion())

            manejadorLoc.removeUpdates(this);

    } catch (SecurityException e) {

    }

}

```

Actualiza la **mejor localización**, sólo va a actualizar hacia una localización nueva si la precisión es menor que dos veces la precisión de la mejor localización que hemos obtenido anteriormente y el tiempo de obtención entre ambas localizaciones (timestamp) es mayor de dos minutos.

```

private void actualizaMejorLocaliz(Location localiz) {
    if (localiz != null && (mejorLoc == null
        || localiz.getAccuracy() < 2*mejorLoc.getAccuracy()
        || localiz.getTime() - mejorLoc.getTime() > DOS_MINUTOS))
    {
        Log.d(TAG, "Nueva mejor localización");
        mejorLoc = localiz;
        ((Aplicacion)
        actividad.getApplication()).posicionActual.setLatitud(
            localiz.getLatitude());
        ((Aplicacion)
        actividad.getApplication()).posicionActual.setLongitud(
            localiz.getLongitude());
    }
}

```

Con **ultimaLocalización** intentamos obtener la **mejorLocalización** posible de entre el GPS y la red de telefonía siguiendo el **criterio anterior** de **actualizaMejorLocalización**. Se guarda en el objeto Aplicación.

```

void ultimaLocalizacion(){

    try {
        if (hayPermisoLocalizacion()) {
            if
            (manejadorLoc.isProviderEnabled(LocationManager.GPS_PROVIDER)) {

                actualizaMejorLocaliz (manejadorLoc.getLastKnownLocation(
                    LocationManager.GPS_PROVIDER));
            }
            if
            (manejadorLoc.isProviderEnabled(LocationManager.NETWORK_PROVIDER)) {

                actualizaMejorLocaliz (manejadorLoc.getLastKnownLocation(
                    LocationManager.NETWORK_PROVIDER));
            }
        }
    }
}

```

```

        } else {

solicitarPermiso(Manifest.permission.ACCESS_FINE_LOCATION,
                  "Sin el permiso localización no puedo mostrar
la distancia" +
                  " a los lugares.", codigoPermiso,
actividad);
        }
    }

    } catch (SecurityException e) {

        System.out.println("Error en permisos");
    }
}

```

Por último si el permiso de localización es concedido cuando comprobamos en el **onRequestPermissionsResult**, llamamos a **permisoConcedido()**. Esto es exactamente igual que lo que hacíamos con los permisos de almacenamiento, del tema 5

```

@Override public void onRequestPermissionsResult(int requestCode,
String[]
permissions, int[] grantResults) {
    if (requestCode == codigoPermiso
        && grantResults.length == 1
        && grantResults[0] ==
PackageManager.PERMISSION_GRANTED)
        permisoConcedido();
}
}

```

Permiso concedido, fuerza la búsqueda de una nueva localización, activa los proveedores, y comunica al adaptador que ha habido cambios en la localización actual. Esto lo usaremos para que actualice el RecyclerView con las nuevas distancias, desde nuestra posición actual a los lugares almacenados.

```

public void permisoConcedido() {
    ultimaLocalizacion();
    activarProveedores();
    adaptador.notifyDataSetChanged();
}

```

adaptador.notifyDataSetChanged();, actualiza los datos del adaptador. En el siguiente tema meteremos las localizaciones para cada Lugar y completaremos esta información. Entre otras cosas hace que **se recargue o refresque la información en el RecyclerView**.

4.2 Introducimos CasosUsoLocalizacion en MainActivity

```

private CasosUsoLocalizacion usoLocalizacion;

```

```

---
-----

protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    lugares = ((Aplicacion) getApplication()).getLugares();
    adaptador = ((Aplicacion) getApplication()).getAdaptador();
    usoLugar = new CasosUsoLugar(this, lugares, adaptador);
    usoLocalizacion = new CasosUsoLocalizacion(this,
    SOLICITUD_PERMISO_LOCALIZACION);
}

```

Antes de ir al Mapa vamos a **comprobar que tenemos permisos de localización**. Lo haremos en el método asociado al Menu, onOptionsItemSelected.

```

@Override
public boolean onOptionsItemSelected(MenuItem item) {

---
---

if (id==R.id.menu_mapa) {

    if (usoLocalizacion.hayPermisoLocalizacion()) {
        Intent intent = new Intent(this, MapaActivity.class);
        startActivity(intent);
    } else {

        Toast.makeText(this, "Necesita permisos de localización para
abrir el mapa", Toast.LENGTH_LONG).show();
    }
}
}

```

Recordar que cuando no tenemos permisos de ubicación o no **tenemos la última ubicación geográfica podemos guardar los puntos sin posición**. Tenemos definido en GeoPunto, la opción **SIN_POSICION**, GeoPunto.*SIN_POSICION*.

5 Usando mapas en la aplicación MisLugares

5.1 Introducción

Para **añadir los mapas vamos a usar una actividad y un fragment**. Nos va a permitir reforzar el uso de fragmentes en aplicaciones como ya vimos para preferencias, y **además vamos a estudiar en detalle la clase GoogleMap** y todos los **listener** asociados para el control de la vista mapa que vamos a cargar.

Necesitaremos igualmente la **librería de GoogleMaps** donde están todo el **framework** de clases de **manejo de la vista mapas**.

5.2 Conectando mi Aplicación con GoogleMaps. La consola google

Seguir **los pasos indicados en el tutorial y en el libro, pero ojo están incompletos, os lo indico completos en el siguiente apartado 5.2.1**. Como veréis **es necesario un sistema de clave pública y clave privada** para que poder usar la Api de GoogleMaps, y es necesario **acceder a la consola de Google**. En el siguiente enlace está todo correctamente explicado. Seguirlo paso a paso.

<http://www.androidcurso.com/index.php/223>

En el libro **Gran Libro de Android** lo tenéis en el punto **7.3 Mapas**.

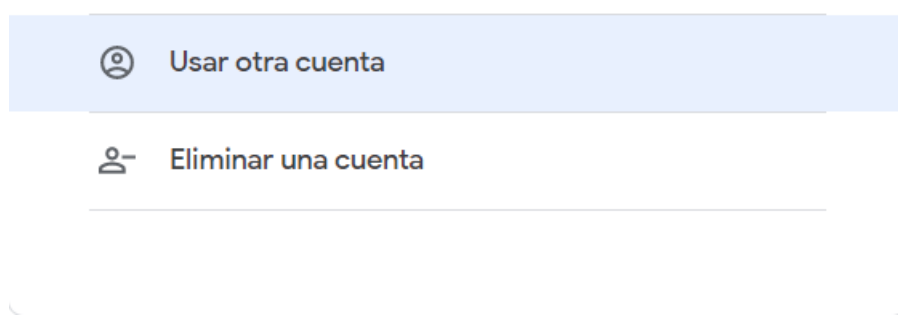
Dos nuevos detalles:

5.2.1 La API de Google maps

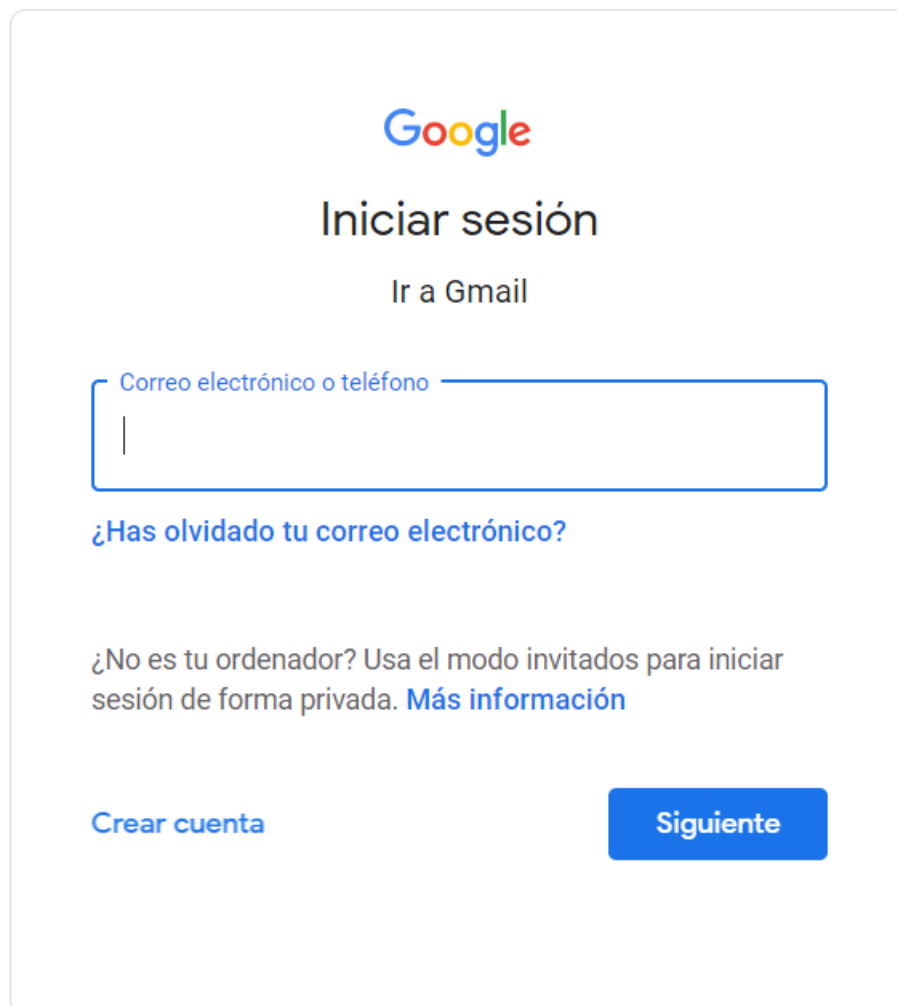
Pasos para habilitar la API de google

1. Crear el proyecto y las credenciales tal como indica el libro. Para ello accedemos a la consola Google en el siguiente enlace

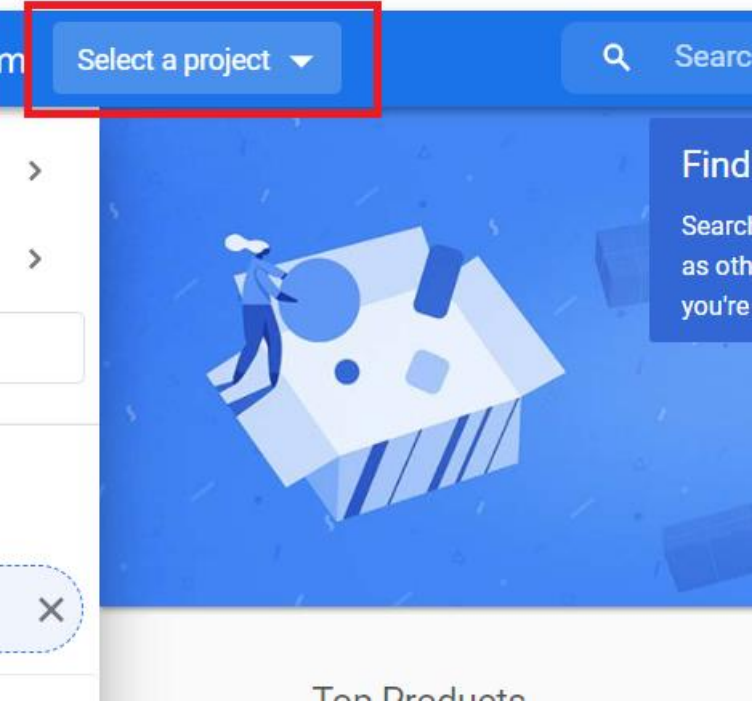
<https://console.cloud.google.com/?hl=es>



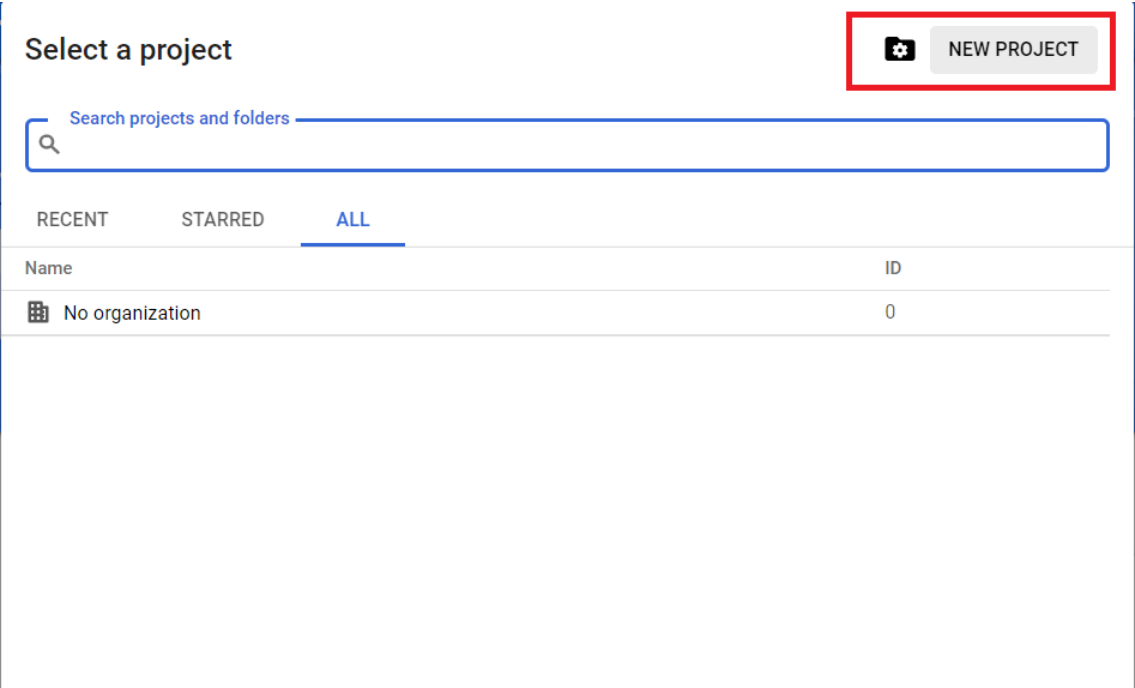
- a. Introducir vuestra cuenta Google. Mi consejo es que creéis una nueva: nombreakpellidosbrianda@gmail.com




- b. Crear un nuevo proyecto MisLugares22 pulsando en Select project, Nuevo proyecto




Nuevo proyecto




 You have 12 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)

[MANAGE QUOTAS](#)


Project name *
MisLugares22 

Project ID: mislugares22-339119. It cannot be changed later. [EDIT](#)

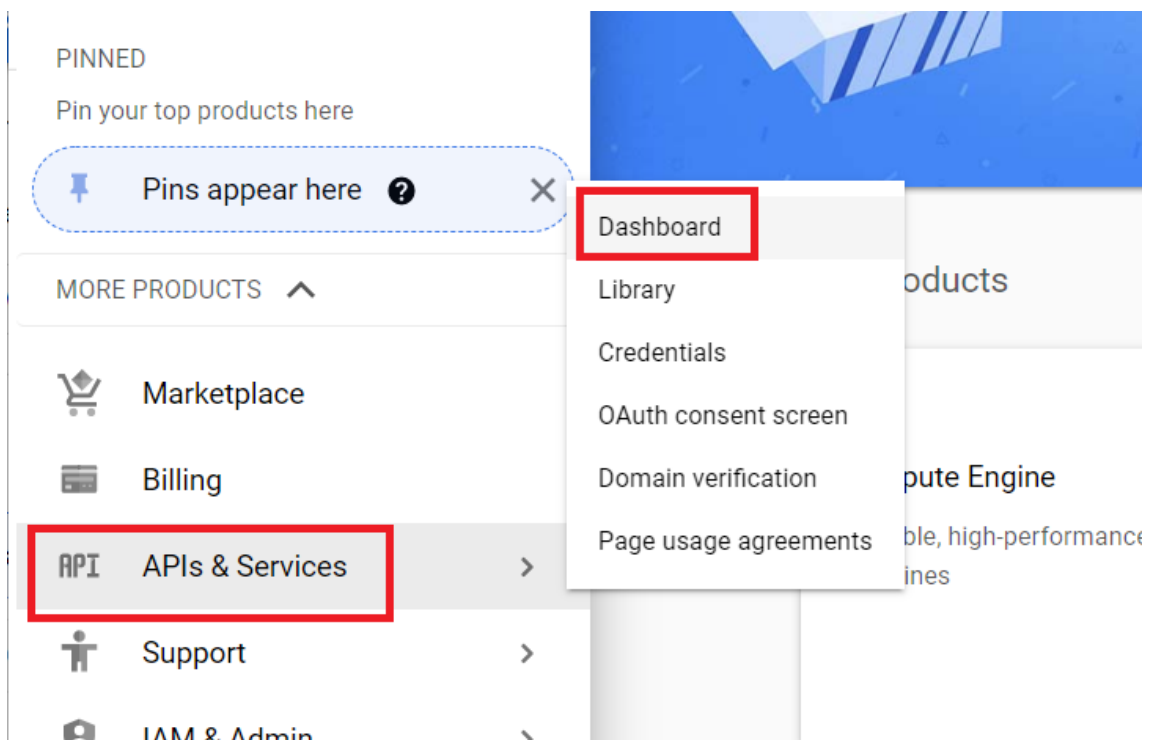
Location *
 No organization [BROWSE](#)

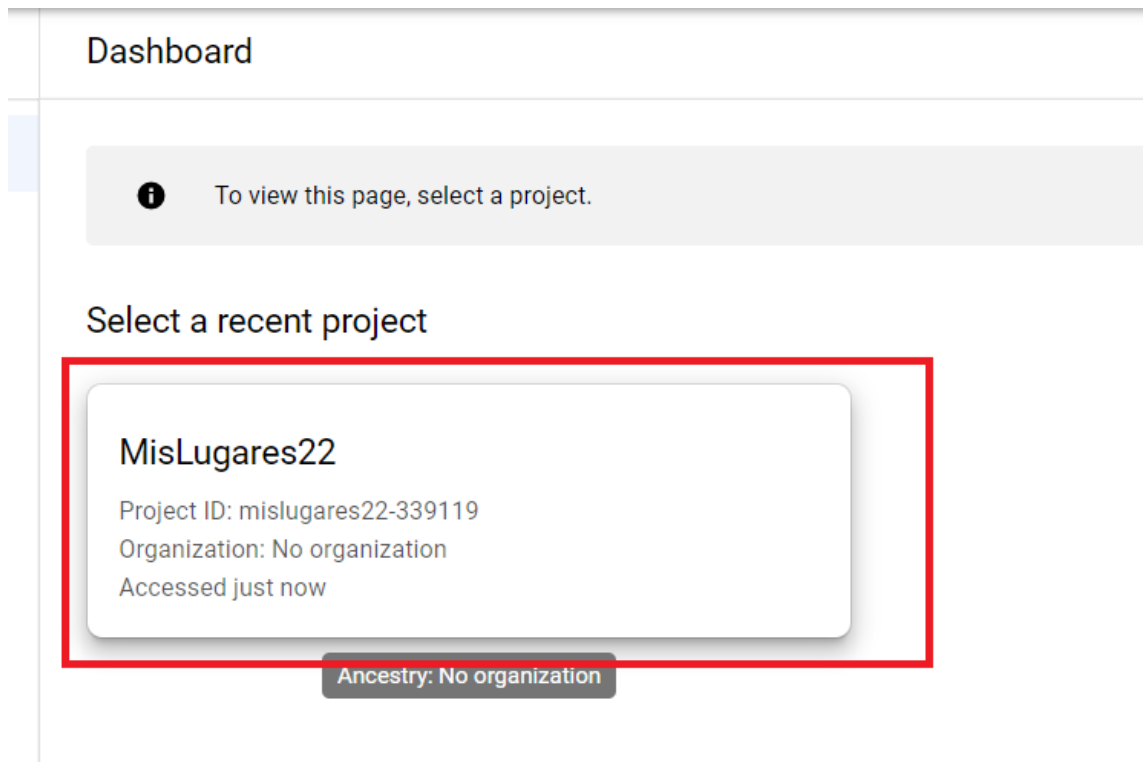
Parent organization or folder

CREATE CANCEL

Search now includes  as other categories to you're looking for

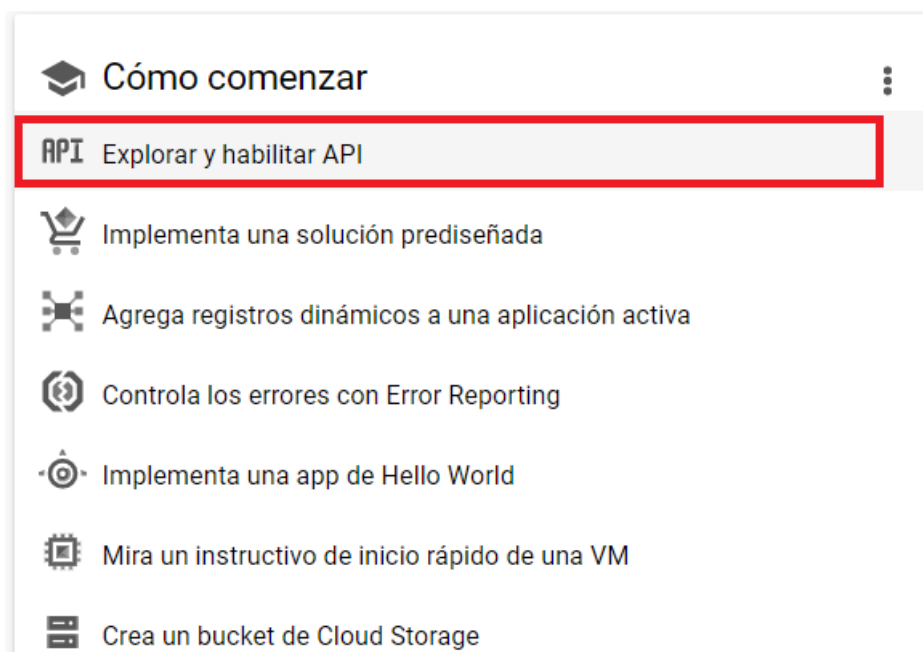
- c. Y finalmente pulsais en API and services DashBoard y seleccionar el proyecto pulsando en el.



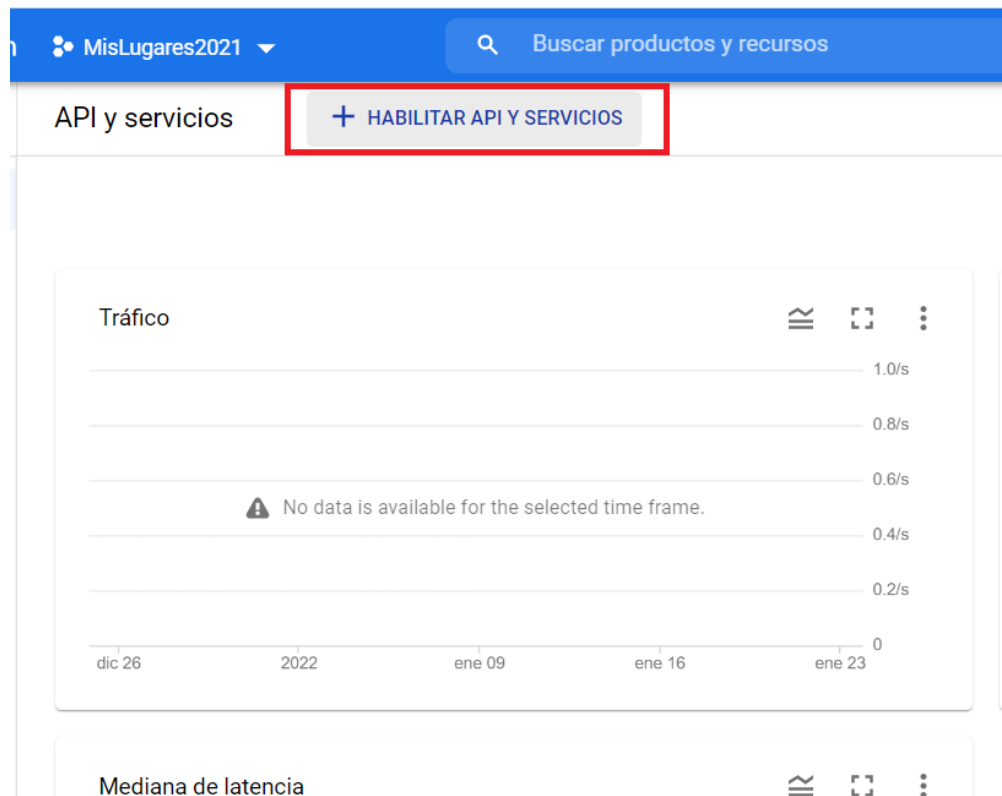


2. Ahora hay que habilitar la API de Google Maps antes de crear las credenciales. Para ello.

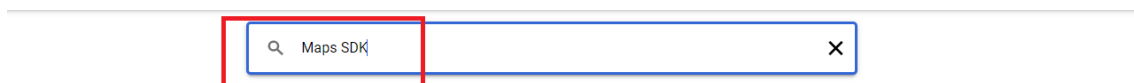
- a. Si lo hacéis después desde la consola de Google <https://console.cloud.google.com/?hl=es> pulsad en



- b. Si sigues el punto 1 directamente habilitar la API y servicios



c. Buscad Map SDK for Android y pulsad en el.



"Maps SDK"

3 resultados



Maps SDK for iOS

Google

Add maps based on Google Maps data to your iOS application with the Maps SDK for iOS. The SDK automatically handles access to the Google Maps servers, map display and response to user gestures such as clicks and drags.



Maps SDK for Android

Google

Add maps based on Google Maps data to your Android application with the Maps SDK for Android. The SDK automatically handles access to Google Maps servers, map display and response to user gestures such as clicks and drags.

d. Pulsa en habilitar.



Maps SDK for Android

Google

Maps for your native Android app.

ENABLE

Click to enable this API

- e. El nuevo estado de la API será el siguiente si volvéis a entrar.



Maps SDK for Android

Google

Maps for your native Android app.

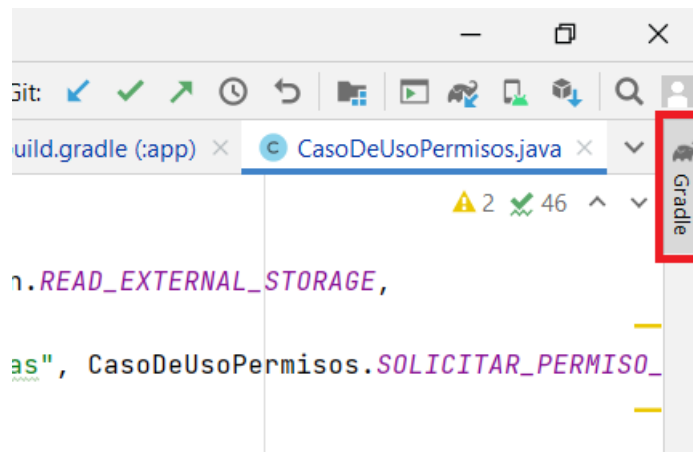
ADMINISTRAR



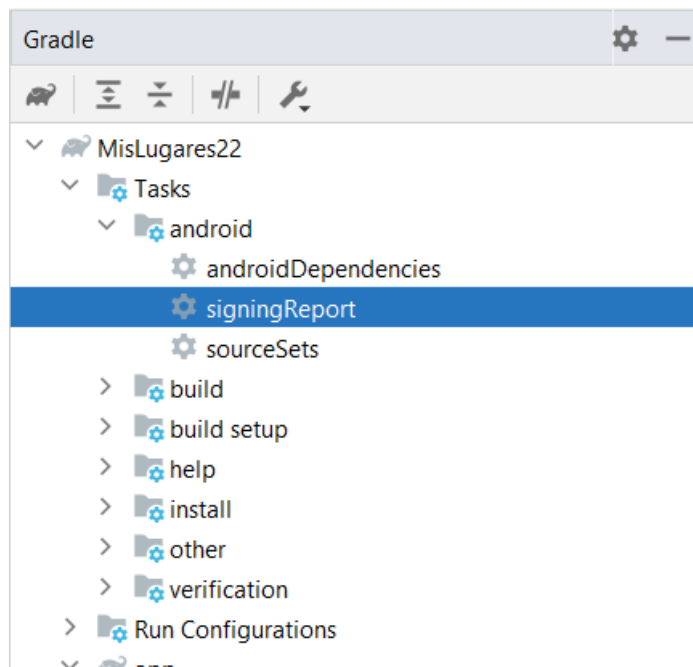
API habilitada

Haz clic para administrar esta API.

3. Es hora de crear las credenciales y asignar las claves de depuración al proyecto para que MisLugares pueda acceder a mapas.
 - a. En vuestro proyecto abrir la pestaña gradle de la parte superior izquierda del Android Studio pulsando en ella



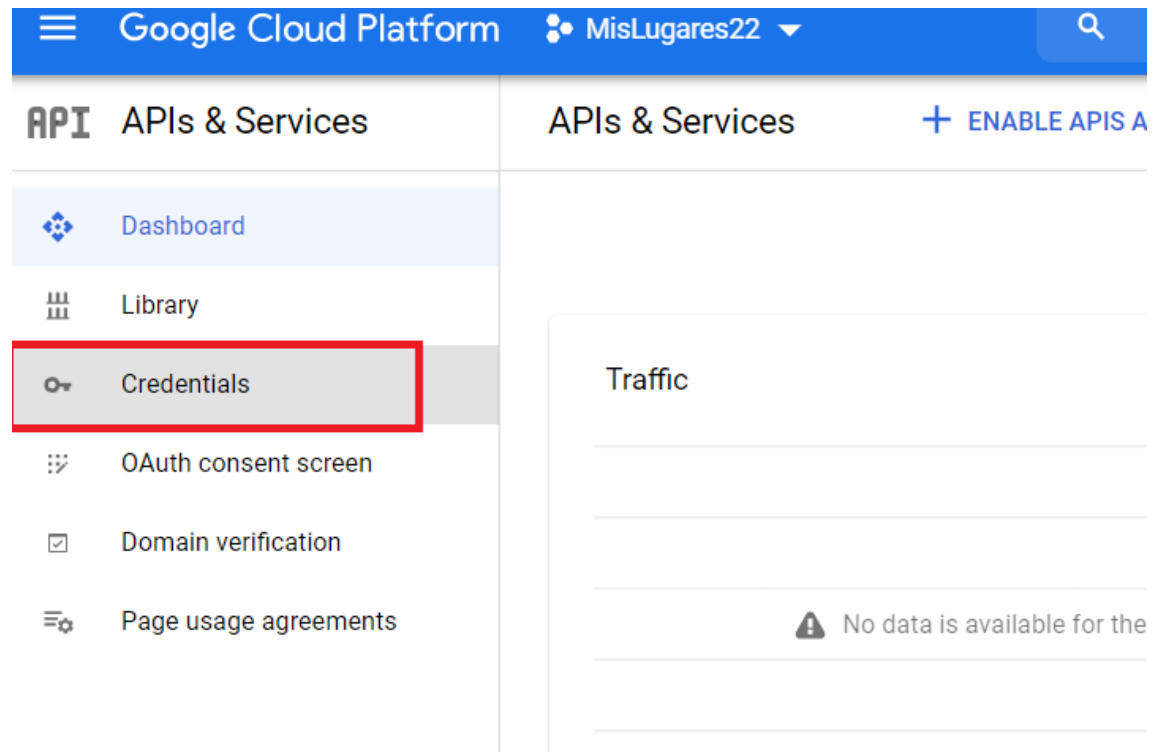
- b. Se despliega la zona de gradle y seleccionaremos la tarea signing report para ver las claves SH1 que asignaremos a nuestra credencial para dar acceso al proyecto de la Google console desde nuestra aplicación. Haced doble click sobre signingreport.



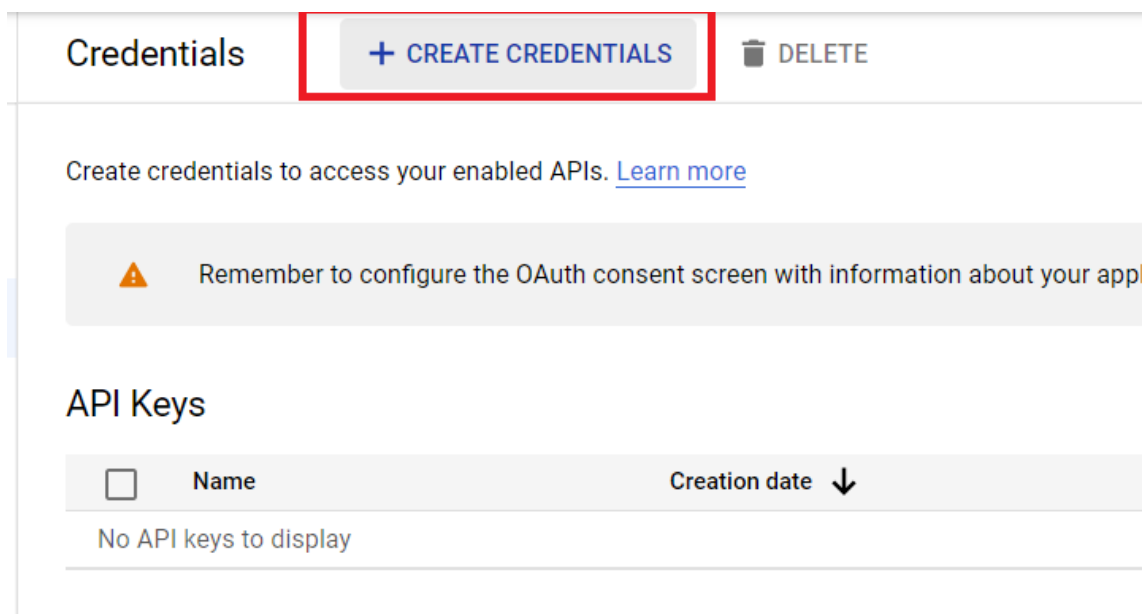
- c. Podéis copiar las claves mostradas en la consola de comandos. Esta borrada en mi caso a vosotros os aparecerá completa

```
> Task :app:signingReport
Variant: debug
Config: debug
Store: C:\Users\carlo\.android\debug.keystore
Alias: AndroidDebugKey
MD5: FA:
SHA1: F1
SHA-256:
Valid until: domingo, 6 de noviembre de 2050
-----
```

- d. Pulsad en el dashboard de la consola de Google para MisLugares22 en credentials




- e. Pulsad en Create Credentials y API Key y cerrar pulsando close



- f. Pulsad en la API ya creada porque vamos a restringirla.

API Keys

<input type="checkbox"/>	Name	Creation date ↓	Restrictions
<input type="checkbox"/>	 API key 1	Jan 23, 2022	None

g. Seleccionar Android Apps y pulsad en add an Item

Application restrictions

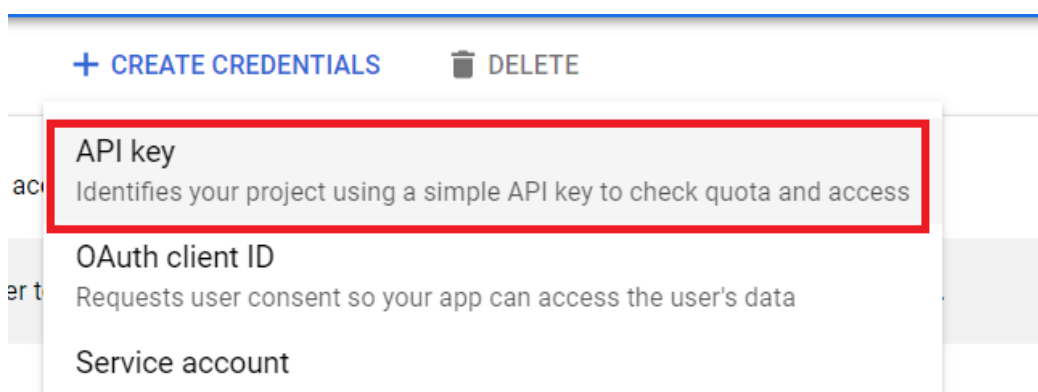
An application restriction controls which websites, IP addresses, or applications can use your API key. You can set one application restriction per key.

- ☐ None
- ☐ HTTP referrers (web sites)
- ☐ IP addresses (web servers, cron jobs, etc.)
- ☒ **Android apps**
- ☐ iOS apps

Restrict usage to your Android apps

Add your package name and SHA-1 signing-certificate fingerprint to restrict usage to your Android apps

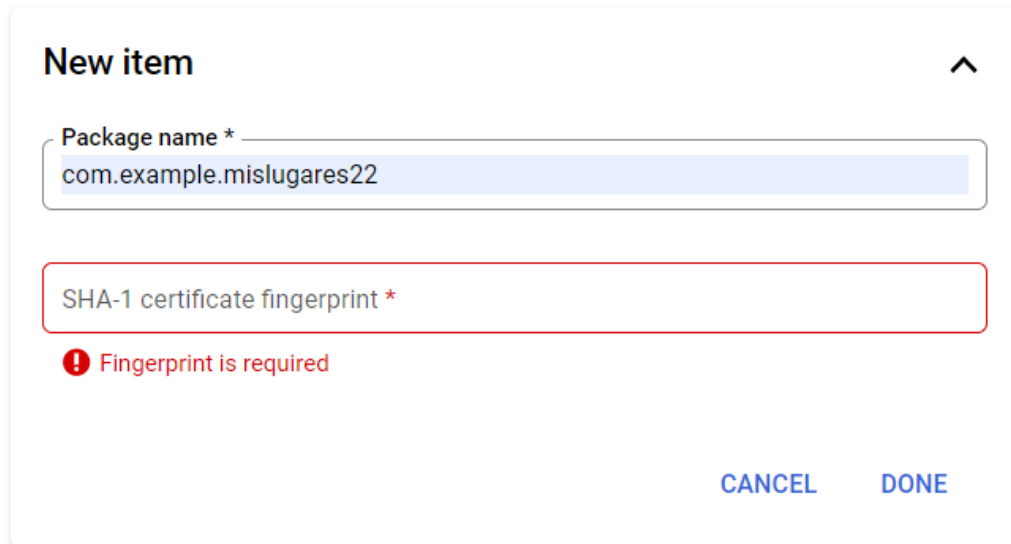
ADD AN ITEM



h. Añadir vuestro application id en el paquete y vuestra clave SHA-1 que habéis copiado de Android studio y aceptar.

Restrict usage to your Android apps

Add your package name and SHA-1 signing-certificate fingerprint to restrict usage to your Android apps



New item

Package name *

com.example.mislugares22

SHA-1 certificate fingerprint *

! Fingerprint is required

CANCEL DONE

5.2.2 Las librerías

Importante: usar las nuevas librerías en lugar de las del libro:

En el gradle de proyecto añadir en el classpath dentro del buildscript en dependencies:

El nuevo plugin gradle de mapas para manejo de claves de la API Google Maps.

```
classpath "com.google.android.libraries.mapsplatform.secrets-gradle-plugin:secrets-gradle-plugin:2.0.0"
```

La librería de servicios de Google que proporcionan entre otras cosas la Api de acceso a Google Maps.

```
classpath 'com.google.gms:google-services:4.3.10'
```

Os debe quedar mas o menos así

```
// Top-level build file where you can add configuration options common to all sub-projects/modules.  
buildscript {  
    repositories {  
        google()  
        mavenCentral()  
    }  
}
```



```
    }
    dependencies {
        classpath "com.android.tools.build:gradle:7.0.3"

        // NOTE: Do not place your application dependencies here; they
        belong // in the individual module build.gradle files

        // NOTE: Do not place your application dependencies here; they
        belong // in the individual module build.gradle files

        classpath "com.google.android.libraries.mapsplatform.secrets-
        gradle-plugin:secrets-gradle-plugin:2.0.0"
        classpath 'com.google.gms:google-services:4.3.10'
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}
```

Y en el gradle de aplicación o módulo, en dependencias añadir la librería de la API de acceso al servicio de mapas:

```
implementation 'com.google.android.gms:play-services-maps:18.0.2'
```

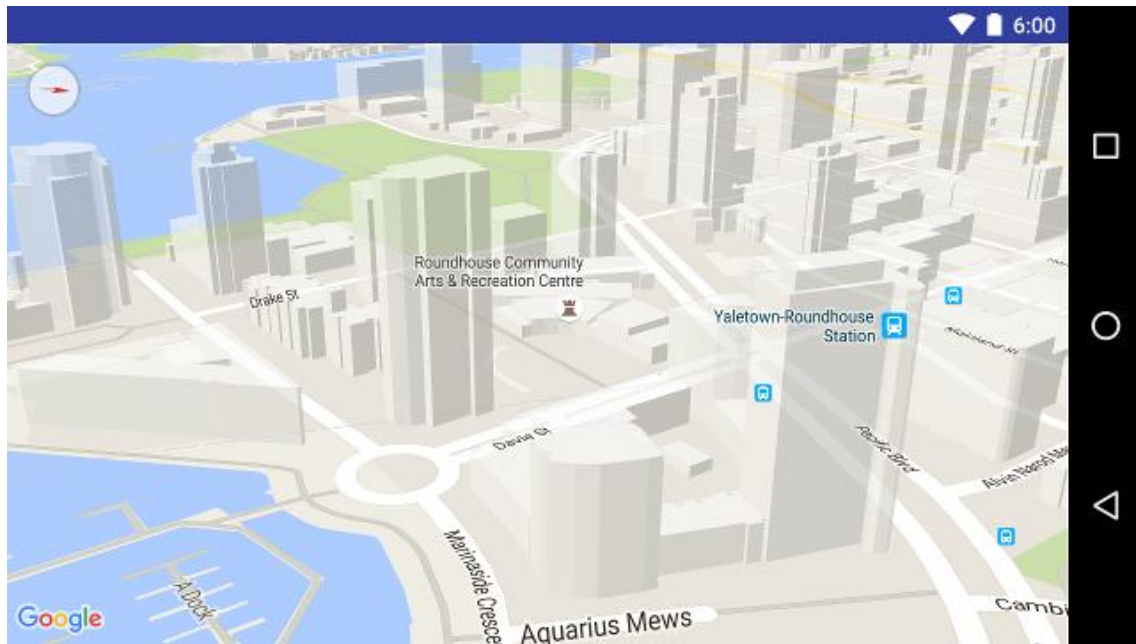
Ahora solo falta configurar el proyecto tal y como indica el libro en el apartado mapas y ya GoogleMaps funcionará en vuestra propia aplicación.

<http://www.androidcurso.com/index.php/223>

5.3 Funcionalidad de la API de GoogleMaps. La cámara.

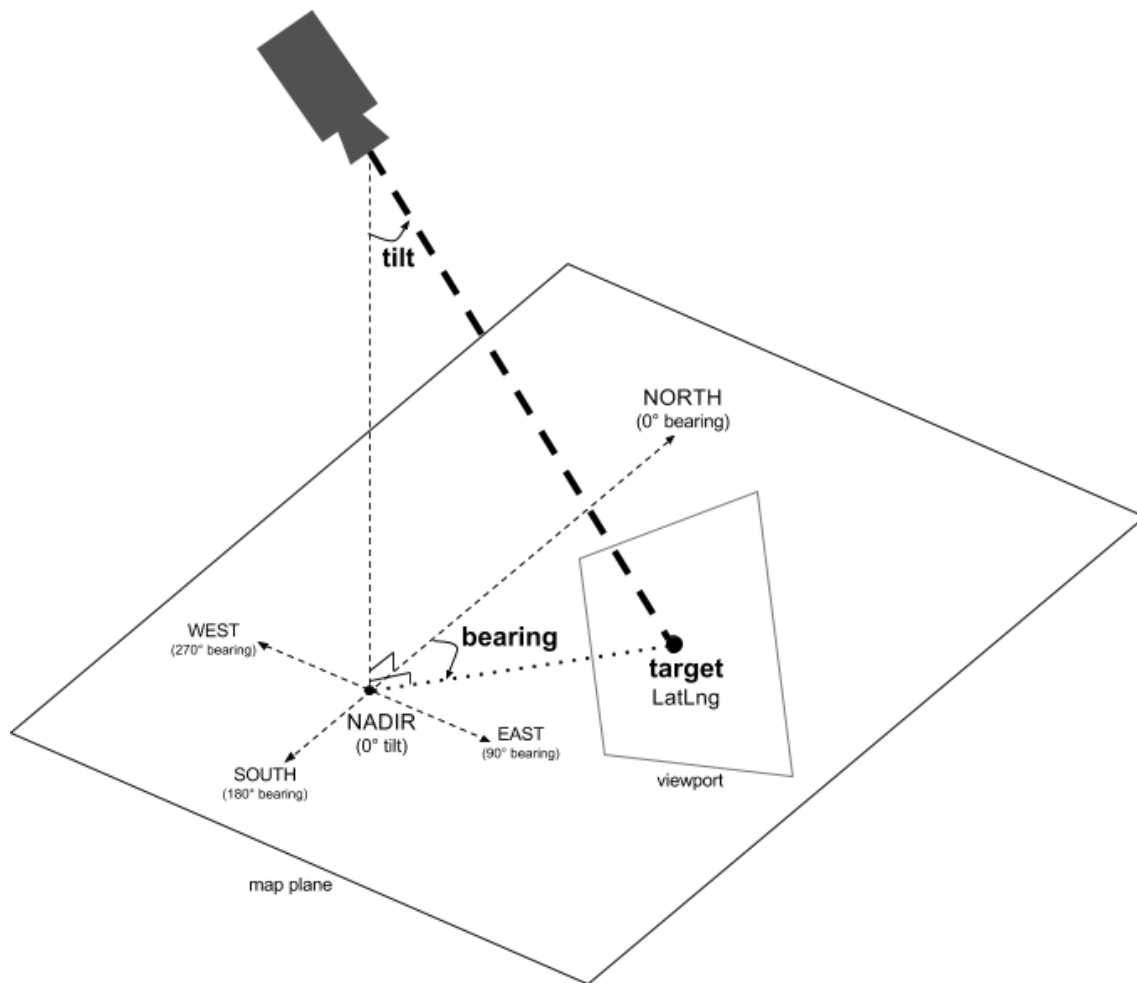
5.3.1 Edificios en 3D en el mapa

Muchas **ciudades**, cuando se ven **de cerca**, tendrán **edificios 3D visibles**, como se puede ver en la siguiente **imagen de Vancouver**, Canadá. Puede **deshabilitar** los edificios 3d llamando **GoogleMap.setBuildingsEnabled(false)**.



5.3.2 La posición de la cámara

La vista del mapa se modela como una **cámara que** mira hacia abajo en un plano plano. La **posición de la cámara** (y, por lo tanto, la representación del mapa) se especifica mediante las siguientes **propiedades: objetivo** (ubicación de **latitud** / **longitud**) , **rumbo** , **inclinación** y **zoom** .



5.3.3 Ubicación del objetivo) target

El objetivo de la cámara **es la ubicación del centro del mapa**, especificado como coordenadas de latitud y longitud.

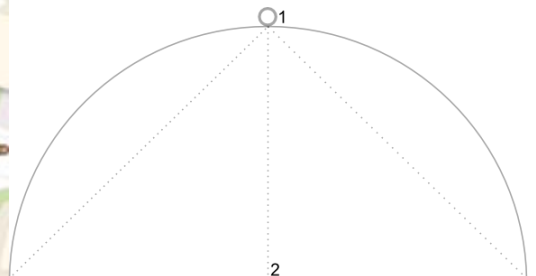
5.3.4 Rumbo (orientación), bearing

El rumbo **de la cámara es la dirección en la que apunta una línea vertical en el mapa**, medida en grados en sentido horario desde el norte. **Alguien que conduce** un automóvil a menudo gira un **mapa de ruta para alinearlo con su dirección de viaje**, mientras que los **excursionistas** que usan un mapa y una brújula generalmente **orientan el mapa de modo que una línea vertical apunte hacia el norte**. La API de Maps te **permite cambiar la alineación o rumbo de un mapa**. Por ejemplo, un **rumbo de 90 grados** da como resultado un mapa donde la **dirección hacia arriba apunta hacia el este**.

5.3.5 Inclinación (ángulo de visión), tilt

La inclinación define la posición de la cámara en un arco entre directamente sobre la posición central del mapa y la superficie de la Tierra, medida en grados desde el nadir (la dirección que apunta directamente debajo de la cámara). Cuando **cambia el ángulo de visión, el mapa aparece en perspectiva, con características lejanas que parecen más pequeñas y características cercanas que parecen más grandes**. Las siguientes ilustraciones demuestran esto.

En las imágenes a continuación, el ángulo de visión es de 0 grados. La primera imagen muestra un esquema de esto; la posición **1** es la posición de la cámara y la posición **2** es la posición actual del mapa. El mapa resultante se muestra debajo de él.



El mapa que se muestra con el ángulo de visión predeterminado de la cámara.

En las imágenes a continuación, el ángulo de visión es de 45 grados. Tenga en cuenta que la cámara no está inclinada a 45 grados; en cambio, se mueve a la mitad a lo largo de un arco entre la cabeza recta (0 grados) y el suelo (90 grados), hasta la posición **3**. La cámara todavía está apuntando al punto central del mapa, pero ahora el área representada por la línea en la posición **4** ahora es visible.



El mapa se muestra con un ángulo de visión de 45 grados. Un ángulo de visión de cámara de 45 grados

El mapa en esta captura de pantalla todavía está centrado en el mismo punto que en el mapa original, pero han aparecido **más características en la parte superior del mapa**. A medida que

aumenta el ángulo más allá de 45 grados, las características entre la cámara y la posición del mapa aparecen proporcionalmente más grandes, mientras que las características más allá de la posición del mapa aparecen proporcionalmente más pequeñas, produciendo un efecto tridimensional.

5.3.6 Enfocar

El nivel de zoom de la cámara determina la escala del mapa. A niveles de zoom más grandes se pueden ver más detalles en la pantalla, mientras que a niveles de zoom más pequeños se puede ver más del mundo en la pantalla. En el nivel de zoom 0, la escala del mapa es tal que el mundo entero tiene un ancho de aproximadamente 256dp ([píxeles independientes de la densidad](#)).

Aumentar el nivel de zoom en 1 duplica el ancho del mundo en la pantalla. Por lo tanto, en el nivel de zoom N, el ancho del mundo es de aproximadamente $256 * 2^N$ dp, es decir, en el nivel de zoom 2, el mundo entero tiene aproximadamente 1024dp de ancho. Tenga en cuenta que el nivel de zoom no necesita ser un número entero. El rango de niveles de zoom permitidos por el mapa depende de una serie de factores que incluyen la ubicación, el tipo de mapa y el tamaño de la pantalla. Cualquier número fuera del rango se convertirá al siguiente valor válido más cercano, que puede ser el nivel de zoom mínimo o el nivel de zoom máximo. La siguiente lista muestra el nivel de detalle aproximado que puede esperar ver en cada nivel de zoom:

- 1: mundo
- 5: Masa continental / continente
- 10: ciudad
- 15: calles
- 20: edificios

Las siguientes imágenes muestran la apariencia visual de diferentes niveles de zoom:



Un mapa en el nivel de zoom 5.



Un mapa en el nivel de zoom 15.



mapa en el nivel de zoom 20.

5.3.7 Moviendo la cámara

La API de Maps te permite cambiar qué **parte del mundo es visible en el mapa**. Esto se logra cambiando la **posición de la cámara** (en lugar de mover el mapa).

Cuando **cambia la cámara**, tiene la opción de **animar el movimiento resultante de la cámara**. La **animación se interpola entre los atributos actuales de la cámara y los nuevos atributos de la cámara**. También puedes controlar la duración de la animación.

5.3.8 Cambiar el nivel de zoom y establecer el zoom mínimo / máximo

`CameraUpdateFactory.zoomIn()` y `CameraUpdateFactory.zoomOut()` da un valor `CameraUpdate` que **cambia el nivel de zoom en una unidad**, mientras mantiene todas las demás propiedades iguales.

CameraUpdateFactory.zoomTo(float) le proporciona un valor CameraUpdate que **cambia el nivel de zoom al valor dado**, mientras mantiene todas las demás propiedades iguales.

CameraUpdateFactory.zoomBy(float) y le CameraUpdateFactory.zoomBy(float, Point) da un **valor CameraUpdate que aumenta (o disminuye, si el valor es negativo) el nivel de zoom por el valor dado**. Este último **fija el punto dado en la pantalla de modo que permanezca en la misma ubicación (latitud / longitud)** y, por lo tanto, puede cambiar la ubicación de la cámara para lograr esto.

5.4 El layout para el fragment del mapa

Empezamos por el Layout, el xml , donde vamos a inflar el fragment SupportMapFragment. Es una clase fragment predefinida para cargar mapas de Google. Para eso definimos un Layout y su Fragment que es donde se cargará el mapa.

```
<?xml                                version="1.0"                                encoding="utf-8" ?>
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <fragment                                android:id="@+id/mapa"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        class="com.google.android.gms.maps.SupportMapFragment"/>
</RelativeLayout>
```

Si os fijáis es una clase predefinida que hereda o extiende Fragment que se llama **com.google.android.gms.maps.SupportMapFragment**. Debemos igualmente añadir las librerías de GoogleMap en nuestro Graddle. Vamos a usar la versión 17 de la librería que concuerda con el resto de librerías añadidas de AndroidX y no da problemas de dependencias. Ya lo hicimos anteriormente.

Creamos el layout **mapa.xml**.

La clase SupportMapFragment Este uso de fragment **es la forma más sencilla de colocar un mapa en una aplicación android**. La clase proporciona una abstracción alrededor de un mapa para **manejar automáticamente todas las necesidades de ciclo de vida** de una actividad **mapa**. Al ser un fragment, **este componente se puede agregar inflando el XML anterior**.

5.5 La Actividad Mapa MapActivity. La Api de GoogleMaps

Encontrareis información extra en:

<http://www.androidcurso.com/index.php/284>

Creemos ahora el FragmentActivity sobre el que vamos a cargar nuestro mapa. Le llamamos MapActivity.

La clase **GoogleMap** es la base de funcionamiento **de la vista de mapa** debe ser adquirido usando **getMapAsync** del interface **OnMapReadyCallback**. Esta clase inicializa automáticamente el sistema de mapas y la vista de mapa. Cualquier objeto obtenido de la GoogleMap está asociado con la vista.

GoogleMap es el punto de entrada para todos los métodos relacionados con **el mapa**. **No puede crear una instancia** de un **GoogleMap** directamente, más bien, debe obtenerse uno del **getMapAsync()**, método de MapFragmento y MapView que haya agregado a su aplicación.

Es similar a **un objeto View**, pero **GoogleMap** solo se **puede leer y modificar desde el hilo de la interfaz de usuario de Android**. Llamar a métodos de **GoogleMap** desde otro hilo dará como **resultado una excepción**. Resumiendo, usarlo dentro **sólo de Actividades o fragmentos** asociados al mapa.

Se puede ajustar el punto de vista de un mapa cambiando la posición de la cámara (en lugar de mover el mapa). Se puede usar la cámara del mapa para establecer parámetros como ubicación, nivel de zoom, ángulo de inclinación y rumbo.

MapActivity.java

```
public class MapActivity extends FragmentActivity
    implements GoogleMap.OnInfoWindowClickListener,
    OnMapReadyCallback, ActivityCompat.OnRequestPermissionsResultCallback
{
    private GoogleMap mapa;
    private static final int MY_LOCATION_REQUEST_CODE = 1;

    private LugaresBD lugares;
    private CasosUsoLugar usoLugar;
    private AdaptadorLugaresBD adaptador;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.mapa);
    }
}
```

```

    lugares = ((Aplicacion) getApplication()).getLugares();
    adaptador = ((Aplicacion) getApplication()).getAdaptador();
    usoLugar = new CasosUsoLugar(this, lugares, adaptador);

    SupportMapFragment mapFragment = (SupportMapFragment)
getSupportFragmentManager().findFragmentById(R.id.mapa);
    mapFragment.getMapAsync(this);

}

@Override
public void onMapReady(GoogleMap googleMap) {
    mapa = googleMap;
    mapa.setMapType(GoogleMap.MAP_TYPE_NORMAL);

    if (ContextCompat.checkSelfPermission(this,
        android.Manifest.permission.ACCESS_FINE_LOCATION) ==
        PackageManager.PERMISSION_GRANTED) {

        mapa.setMyLocationEnabled(true);
        mapa.getUiSettings().setZoomControlsEnabled(true);
        mapa.getUiSettings().setCompassEnabled(true);
        // }
        if (lugares.tamanyo() > 0) {

            int _id = adaptador.idPosicion(0);
            GeoPunto p = lugares.elemento(_id).getPosicion();
            mapa.moveCamera(CameraUpdateFactory.newLatLngZoom(
                new LatLng(p.getLatitud(), p.getLongitud()),
12));

        }
        for (int n = 0; n < lugares.tamanyo(); n++) {
            int _id = adaptador.idPosicion(n);
            Lugar lugar = lugares.elemento(_id);
            GeoPunto p = lugar.getPosicion();
            if (p != null && p.getLatitud() != 0) {
                BitmapDrawable iconoDrawable = (BitmapDrawable)
                    ContextCompat.getDrawable(this,
lugar.getTipo().getRecurso());
                Bitmap iGrande = iconoDrawable.getBitmap();
                Bitmap icono = Bitmap.createScaledBitmap(iGrande,
                    iGrande.getWidth() / 7,
iGrande.getHeight() / 7, false);
                mapa.addMarker(new MarkerOptions()
                    .position(new LatLng(p.getLatitud(),
p.getLongitud())))

                .title(lugar.getNombre()).snippet(lugar.getDireccion())

                .icon(BitmapDescriptorFactory.fromBitmap(icono));
            }
        }

        mapa.setOnInfoWindowClickListener(this);
    } else {

```

```

        Toast.makeText(this, "No tiene permisos de
localizacion", Toast.LENGTH_LONG).show();

        new AlertDialog.Builder(this)
            .setTitle("Solicitud de permiso")
            .setMessage("Mensaje")
            .setPositiveButton("Ok", new
DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog,
int whichButton) {

                    ActivityCompat.requestPermissions(MapaActivity.this,
                        new
String[]{Manifest.permission.ACCESS_FINE_LOCATION},
MY_LOCATION_REQUEST_CODE);
                })
            .show();

    }

}

@Override
public void onInfoWindowClick(Marker marker) {
    for (int id = 0; id < lugares.tamanyo(); id++) {
        if (lugares.elemento(id).getNombre()
            .equals(marker.getTitle())) {
            Intent intent = new Intent(this,
VistaLugarActivity.class);
            intent.putExtra("id", (long) id);
            startActivity(intent);
            break;
        }
    }
}

public void onRequestPermissionsResult(int requestCode, String[]
permissions, int[] grantResults) {
    if (requestCode == MY_LOCATION_REQUEST_CODE) {
        if (permissions.length == 1 &&
            permissions[0] ==
Manifest.permission.ACCESS_FINE_LOCATION &&
            grantResults[0] ==
PackageManager.PERMISSION_GRANTED) {
            mapa.setMyLocationEnabled(true);
        } else {
            // Permission was denied. Display an error message.
        }
    }
}
}
}

```

5.5.1 Entendiendo el código

Vamos a empezar esta vez con los interfaces., **MapaActivity** además de extender de **FragmentActivity** implementa tres interfaces. `GoogleMap.OnInfoWindowClickListener`, `OnMapReadyCallback`, `ActivityCompat.OnRequestPermissionsResultCallback` {

OnMapReadyCallback obliga a implementa un método en la clase llamado **OnMapReady**. Este método se va a llamar cuando el mapa este listo para usarse y Obtengamos un objeto `GoogleMap` no nulo.

OnInfoWindowClickListener es un interface para controlar los eventos de click y tap sobre nuestro mapa. Nos va a permitir, entre otras cosas, **mostrar información extra** cuando hagamos click sobre los iconos que representan nuestros lugares en la aplicación. El método obligatorio a sobrescribir que controla esos clics es **onInfoWindowClick**.

OnRequestPermissionsResultCallback, nos sirve para la petición de permisos, pero ya hemos realizado este trabajo en `CasoUsoLocalización`. Al final del tema, será vuestra tarea modificar el código, quitar este interfaz y usar nuestro `CasoUsoLocalización` para la comprobación de permisos.

En las **propiedades de la clase** podemos distinguir una variable nueva **private GoogleMap** mapa. Como ya hemos dicho antes esta clase nos va a ayudar a controlar toda la vista Mapa.

Dentro del onCreate

OnCreate ()

```
SupportMapFragment mapFragment = (SupportMapFragment)
    getSupportFragmentManager().findFragmentById(R.id.mapa);
mapFragment.getMapAsync(this);
```

Inflamos nuestro mapFragment desde el **XML mapa.xml**. Y realizamos la llamada a **getMapAsync**. Cuando el mapa este listo, pues como ves se carga de manera asíncrona, se ejecutará el método de evento **onMapReady**.

5.5.2 OnMapReady

El método más importante es **onMapReady**, que va a darnos el control del mapa en la vista, donde vamos a marcar nuestros lugares.

```
@Override
public void onMapReady(GoogleMap googleMap) {
    mapa = googleMap;
    mapa.setMapType(GoogleMap.MAP_TYPE_NORMAL);

    if (ContextCompat.checkSelfPermission(this,
        android.Manifest.permission.ACCESS_FINE_LOCATION) ==
        PackageManager.PERMISSION_GRANTED) {

        mapa.setMyLocationEnabled(true);
        mapa.getUiSettings().setZoomControlsEnabled(true);
        mapa.getUiSettings().setCompassEnabled(true);
    }
}
```

Lo primero que quiero que veáis es que el método nos va a devolver un objeto **googleMap** para manejar la vista. Después de comprobar si tenemos **permisos de localización**, realizamos **tres tareas de configuración**. **getUiSettings** nos devuelve un objeto **UiSettings** que nos permitirá configurar nuestra mapa. Como veis habilitamos que nuestra configuración aparezca en el mapa con **setMyLocationEnabled(true)**. Con **setZoomControlsEnabled** activamos la posibilidad de usar **zoom** y la **brújula** con **setCompassEnabled(true)**.

```
if (lugares.tamanyo() > 0) {

    int _id = adaptador.idPosicion(0);
    GeoPunto p = lugares.elemento(_id).getPosicion();
    mapa.moveCamera(CameraUpdateFactory.newLatLngZoom(
        new LatLng(p.getLatitud(), p.getLongitud()), 12));
}
```

Con **mapa.moveCamera** movemos la cámara hacia el primer lugar de la lista. **newLatLngZoom(LatLng latLng, float zoom)** nos devuelve la cámara centrada en el punto geográfico que pasamos como parámetro.

LatLng es una clase inmutable que representa un par de coordenadas de latitud y longitud, almacenadas como grados. Un objeto inmutable es aquel cuyo estado no se puede cambiar una vez construido. Todos sus atributos han sido definidos como **final** y/o utiliza copia defensiva para protegerse frente a cambios desde el código cliente.

CameraUpdateFactory es una clase abstracta que nos devuelve objetos **CameraUpdate**, que cambian posición, zoom, ángulo de la cámara.

```
for (int n = 0; n < lugares.tamanyo(); n++) {
    int _id = adaptador.idPosicion(n);
    Lugar lugar = lugares.elemento(_id);
    GeoPunto p = lugar.getPosicion();
    if (p != null && p.getLatitud() != 0) {
        BitmapDrawable iconoDrawable = (BitmapDrawable)
            ContextCompat.getDrawable(this,
                lugar.getTipo().getRecurso());
    }
}
```

```

        Bitmap iGrande = iconoDrawable.getBitmap();
        Bitmap icono = Bitmap.createScaledBitmap(iGrande,
            iGrande.getWidth() / 7, iGrande.getHeight() / 7,
false);
        mapa.addMarker(new MarkerOptions()
            .position(new LatLng(p.getLatitude(), p.getLongitude()))

            .title(lugar.getNombre()).snippet(lugar.getDireccion())
            .icon(BitmapDescriptorFactory.fromBitmap(icono)));
    }
}

```

ContextCompat es una clase que nos ayuda a **acceder a recursos de la aplicación**.

[Drawable](#) `getDrawable(Context context, int id)` nos devuelve el recurso drawable asociado al id de la clase R.

`Bitmap iGrande = iconoDrawable.getBitmap();` nos devuelve el mapa de bits asociado al recurso drawable.

Un **Drawable** es una abstracción general de "algo que se puede dibujar". La mayoría de las veces tratará con **Drawable** como el tipo de recurso recuperado para dibujar cosas en la pantalla; la clase **Drawable** proporciona una API genérica para tratar con un recurso visual subyacente que puede adoptar una variedad de formas. A diferencia de a View, un **Drawable** no pueden recibir eventos o interactuar con el usuario.

Un **Drawable** que contiene un mapa de bits y puede ser en mosaico, estirarse o alinearse. Se puede crear un **BitmapDrawable** desde una ruta de archivo, una secuencia de entrada, inflando un XML o desde un objeto **Bitmap**.

Bitmap es el objeto de arquitectura Android que contiene directamente el gráfico de mapa de bits que puede ser dibujado. Permite manejar y transformar de gráficos de mapa de bits sin procesar, y debe dibujarse sobre un objeto Canvas.

Para dibujar en Android se necesitan cuando cosas. El objeto Canvas es el lienzo, donde dibujar. El objeto bitmap es el mapa de bits, se necesitan igualmente un Paint, que contiene los colores a dibujar, y una forma primitiva de dibujo (ya sea vectores, mapa de bits, formas como rectángulos, etc.) El objeto Bitmap contiene estas tres últimas, sólo necesita un lienzo para ser dibujado.

```

public static Bitmap createScaledBitmap (Bitmap src,

        int dstWidth,

        int dstHeight,

        boolean filter)

```

Nos crea un nuevo **Bitmap** a escala. `dstWidth` y `dstHeight` son las nuevas medidas. Con el **filter** a **true** se aplica sobre el **Bitmap** un filtro bilinear, que nos da una representación mejor del bitmap escalado, pero disminuye el rendimiento de la operación.

Para **quien quiera aprender las clases gráficas** en Android:

<http://www.androidcurso.com/index.php/recursos/35-unidad-4-graficos-en-android/132-clases-para-graficos-en-android>

Para finalizar `onMapReady`, `mapa.addMarker` nos permite añadir **objetos markers** a nuestro mapa. **Los marcadores identifican ubicaciones en el mapa**. El marcador predeterminado **usa un ícono estándar, común a la apariencia de Google Maps**. Es posible **cambiar el color, la imagen, el título o el punto de anclaje** del icono a través de la API, con el objeto `MarkerOptions`.

5.5.3 onInfoWindowClick

Sobreescribimos este método debido a que implementamos el **interfaz `OnInfoWindowClickListener`**. Nos va a permitir controlar los eventos `onClick` y `onTap` de pantalla. En particular va a capturar el evento cuando hagamos un click en uno de nuestros marcadores.

El **funcionamiento** es sencillo, cuando **hacemos click sobre nuestro marcador, `Marker`**, introducido en la **`onMapReady`**, nos va a llevar a `VistaLugarActivity` para mostrarnos la información del lugar. En el bucle `for` comprobamos a qué lugar **corresponde nuestro marcador** en el adapter.

```
@Override
public void onInfoWindowClick(Marker marker) {
    for (int pos = 0; pos < lugares.tamanyo(); pos++) {
        if (lugares.elemento(pos).getNombre()
            .equals(marker.getTitle())) {
            Intent intent = new Intent(this,
VistaLugarActivity.class);
            intent.putExtra("pos", (long) pos);
            startActivity(intent);
            break;
        }
    }
}
```

6 Añadiendo distancias al RecyclerView

Vamos a **modificar nuestro `RecivlerView`** para que **admita distancias entre nuestra ubicación actual y la ubicación de nuestros lugares**.

6.1 Modificamos el Layout para elemento_lista.xml

El primer paso será modificar la vista elemento_lista.xml y añadir el campo distancia.

```
<TextView android:id="@+id/distancia"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"

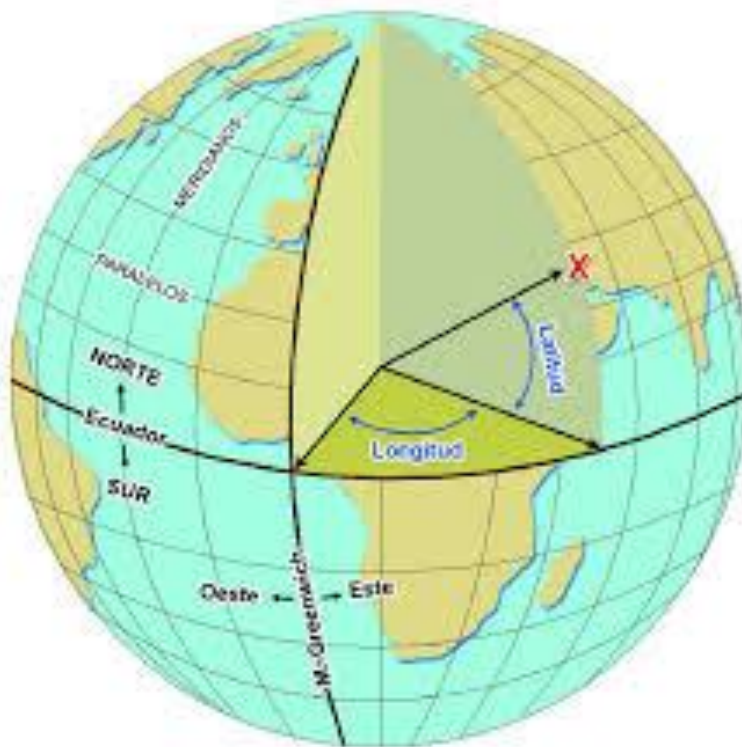
    app:layout_constraintTop_toBottomOf="@id/direccion"
    app:layout_constraintEnd_toEndOf="parent"
    android:gravity="right"
    android:text="... Km" />
```

Ahora para cada uno de nuestros lugares debemos calcular la distancia respecto a nuestra ubicación actual. En la clase GeoPunto tenemos el método implementado distancia. Entre el punto actual y el que pasamos como parámetro.

```
public double distancia(GeoPunto punto) {
    final double RADIO_TIERRA = 6371000; // en metros
    double dLat = Math.toRadians(latitud - punto.latitud);
    double dLon = Math.toRadians(longitud - punto.longitud);
    double lat1 = Math.toRadians(punto.latitud);
    double lat2 = Math.toRadians(latitud);
    double a = Math.sin(dLat/2) * Math.sin(dLat/2) +
        Math.sin(dLon/2) * Math.sin(dLon/2) *
            Math.cos(lat1) * Math.cos(lat2);
    double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    return c * RADIO_TIERRA;
}
```

La latitud y la longitud son ángulos con respecto al ecuador. La longitud es el ángulo horizontal, la latitud es el ángulo vertical.

6.1.1 Cálculo de distancias en la tierra. Fórmula del semiverseno



Aplicamos la fórmula para **obtener distancias a partir de latitud y longitud**, la **distancia en realidad va a ser el arco entre los dos puntos en una esfera**, la teoría matemática que sustenta este **cálculo es la del semiverseno**.

$$\text{Distancia} = 2 * R * \text{asin} \sqrt{\sin^2 \left(\frac{\Delta \text{lat}}{2} \right) + \cos(\text{lat1}) * \cos(\text{lat2}) * \sin^2 \left(\frac{\Delta \text{lon}}{2} \right)}$$

Lat → latitud

Lon → Longitud

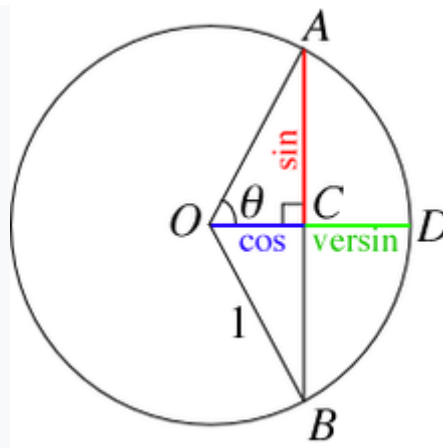
(lat1, lon1) → Latitud y longitud en el punto 1

(lat2, lon2) → Latitud y longitud en el punto 2

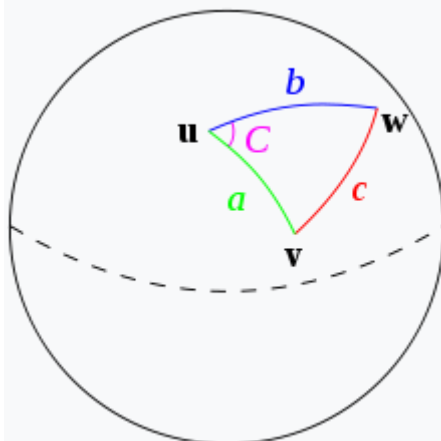
Δlat = lat2 - lat1

Δlon = lon2 - lon1

R = 6372.795477598 Km (Radio de la Tierra)



Usamos **esta fórmula** para calcular la distancia entre dos puntos de la tierra, suponiendo que es una esfera.



6.2 Modificando el adaptador para añadir las distancias

Modificamos el ViewHolder en el Adapter añadiendo el campo distancia:

```
public static class ViewHolder extends RecyclerView.ViewHolder {
    public TextView nombre, direccion;
    public ImageView foto;
    public RatingBar valoracion;
    public TextView distancia;
    public ViewHolder(View itemView) {
        super(itemView);
        nombre = itemView.findViewById(R.id.nombre);
        direccion = itemView.findViewById(R.id.direccion);
        foto = itemView.findViewById(R.id.foto);
        valoracion = itemView.findViewById(R.id.valoracion);
        distancia = itemView.findViewById(R.id.distancia);
    }
}
```

En el método personaliza añadimos el valor de la distancia desde nuestro punto a la ubicación actual.

```
// Personalizamos un ViewHolder a partir de un lugar
public void personaliza(Lugar lugar) {
    nombre.setText(lugar.getNombre());
    direccion.setText(lugar.getDireccion());
    int id = R.drawable.otros;
    switch (lugar.getTipo()) {
        case RESTAURANTE: id = R.drawable.restaurante; break;
        case BAR: id = R.drawable.bar; break;
        case COPAS: id = R.drawable.copas; break;
        case ESPECTACULO: id = R.drawable.espectaculos; break;
        case HOTEL: id = R.drawable.hotel; break;
        case COMPRAS: id = R.drawable.compras; break;
        case EDUCACION: id = R.drawable.educacion; break;
        case DEPORTE: id = R.drawable.deporte; break;
        case NATURALEZA: id = R.drawable.naturaleza; break;
        case GASOLINERA: id = R.drawable.gasolinera; break;
    }
    foto.setImageResource(id);
    foto.setScaleType(ImageView.ScaleType.FIT_END);
    valoracion.setRating(lugar.getValoracion());

    GeoPunto pos = ((Aplicacion)
itemView.getContext().getApplicationContext())
        .posicionActual;
    if (pos.equals(GeoPunto.SIN_POSICION) ||
        lugar.getPosicion().equals(GeoPunto.SIN_POSICION)) {
        distancia.setText("... Km");
    } else {
        int d = (int) pos.distancia(lugar.getPosicion());
        if (d < 2000) distancia.setText(d + " m");
        else distancia.setText(d / 1000 + " Km");
    }
}
```

```
}  
}
```

Cogemos la **posición actual del contexto de aplicación**.

```
GeoPunto pos=((Aplicacion)=  
itemView.getContext().getApplicationContext())  
                .posicionActual;
```

Si **no tenemos la posición de la ubicación actual**, es un punto indefinido, **no escribimos las distancias** en el RecyclerView

```
if (pos.equals(GeoPunto.SIN_POSICION) ||  
    lugar.getPosicion().equals(GeoPunto.SIN_POSICION)) {  
    distancia.setText("... Km");  
}
```

Si la **distancia es menor de 2000 metros**, la escribimos en **metros**, si es **mayor** en **kilómetros**.

```
if (d < 2000) distancia.setText(d + " m");  
else          distancia.setText(d / 1000 + " Km");  
}
```

7 Tarea

MapaActivity está incompleto. Par mejorarlo debemos **añadir las funcionalidades de permisos** de **CasosUsoLocalizacion** a la actividad. Vuestra tarea **será eliminar toda la petición directa y comprobación de permisos** y **usar nuestro caso de uso** en su lugar de manera apropiada.