IES DR LLUÍS SIMARRO
ENSENYAMENT PÚBLIC
XATIVA

Programación multimedia y dispositivos móviles

UD11. Google Maps.

Desarrollo de Aplicaciones Multiplataforma

Anna Sanchis Perales
Enric Giménez Ribes

ÍNDICE

1. GOOGLE MAPS

Google Maps nos proporciona un servicio de cartografía online que podremos utilizar en nuestras aplicaciones Android. Estudiaremos la versión 2 del API que incorpora interesantes ventajas respecto a la versión anterior. Entre estas ventajas destaca el menor tráfico intercambiado con el servidor, la utilización de fragments y los gráficos en 3D. Como inconveniente resaltar que la nueva versión solo funciona en dispositivos con Google Play instalado.

Conviene destacar que a diferencia de Android, Google Maps no es un software libre, por lo que está limitado a una serie de condiciones de servicio. Podemos usarlo de forma gratuita siempre que nuestra aplicación no solicite más de X codificaciones geográficas al día. Podemos ofrecer información adicional sobre las ubicaciones del mapa y facilitar la interacción con el usuario como agregar marcadores, polígonos y superposiciones al mapa.

Información más completa de este API la encontramos en:

https://developers.google.com/maps/documentation/android-sdk/overview?hl=es-419

1.1. Obtención de una clave Google Maps

Para poder utilizar este servicio de Google, igual como ocurre cuando se utiliza desde una página web, va a ser necesario registrar la aplicación que lo utilizará. Tras registrar la aplicación se nos entregará una clave que tendremos que indicar en la aplicación.

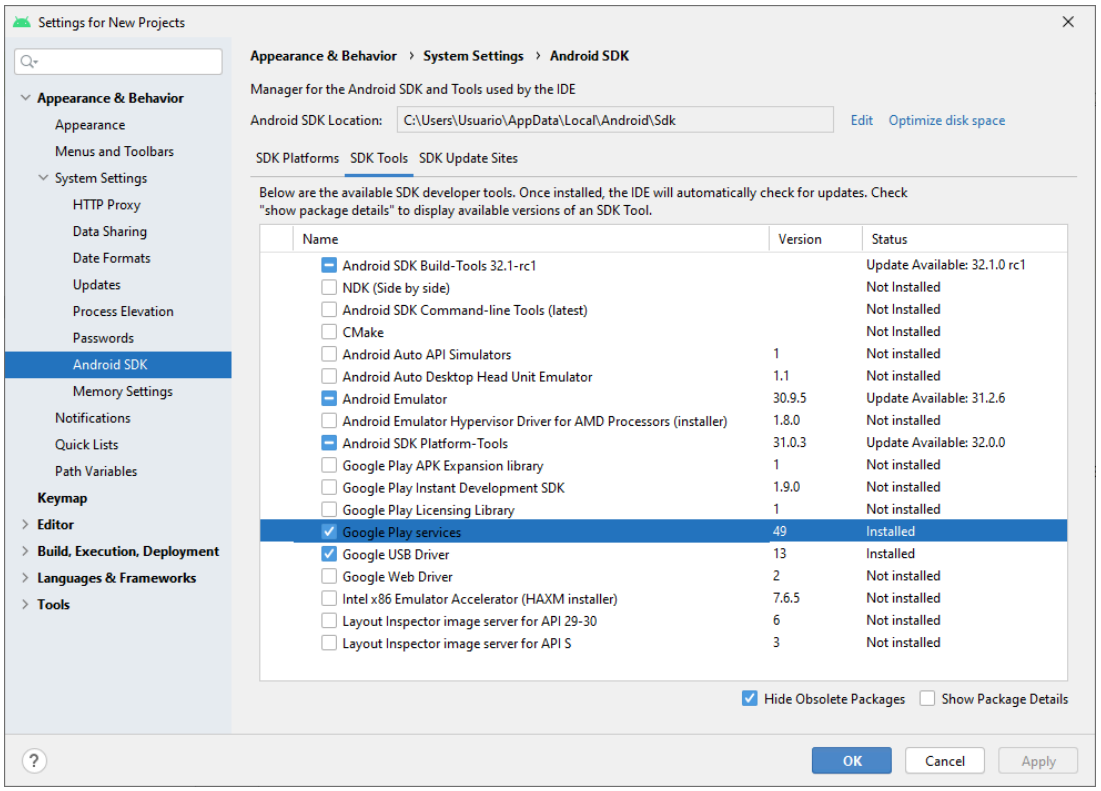
Realmente vamos a necesitar dos claves diferentes, una durante el proceso de desarrollo y otra para la aplicación final. La razón es que se genera una clave diferente en función del certificado digital con el que se firma la aplicación. En la fase de desarrollo las aplicaciones también han de ser firmadas digitalmente, pero en este caso el SDK utiliza un certificado especial (el certificado de depuración), utilizado sólo en la fase de desarrollo.

En caso de querer distribuir tu aplicación, una vez terminada tendrás que firmarla con un certificado digital propio. Recuerda que será necesario reemplazar la clave Google Maps, por otra, esta última asociada al certificado digital usado en la fase de distribución. En los siguientes puntos de ésta unidad lo vemos en detalle.

1.2. Instalando Google APIs

Este paso es rápido y sencillo. Los pasos para instalar el Google Play Service SDK, son los siguientes:

1. Abrimos el **Android SDK Manager**.
2. Seleccionamos **SDK Tools** y seleccionamos **Google Play services**.



3. Y luego hacemos click en el botón Apply para instalar.
4. Aceptamos la licencia y a continuación empezará a descargarse e instalarse las librerías necesarias del paquete elegido.
5. Una vez instalado cerramos el Android SDK Manager y volvemos a la ventana principal del IDE Android Studio.

1.3. Creando un proyecto de Google Maps

Ahora vamos a crear un sencillo ejemplo que nos permite visualizar un mapa centrado en las coordenadas geográficas detectadas por el sistema de posicionamiento.

1. Creamos un nuevo proyecto denominado **Tema11App1**.
2. En el layout activity_main.xml tendremos el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:padding="16dp"
   tools:context=".MainActivity">

   <Button
       android:id="@+id/btnMapGeneral"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_marginTop="16dp"
       android:text="MOSTRAR MAPA GENERAL"
       app:layout_constraintEnd_toEndOf="parent"
       app:layout_constraintHorizontal_bias="0.5"
       app:layout_constraintStart_toStartOf="parent"
       app:layout_constraintTop_toTopOf="parent" />


   <Button
       android:id="@+id/btnMapUbicacion"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_marginTop="16dp"
       android:text="MOSTRAR UBICACIÓN ACTUAL"
       app:layout_constraintEnd_toEndOf="parent"
       app:layout_constraintStart_toStartOf="parent"
       app:layout_constraintTop_toBottomOf="@+id/btnMapGeneral" />


   <Button
       android:id="@+id/btnMapMarkers"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content"
       android:layout_marginTop="16dp"
       android:text="MOSTRAR PUNTOS (MARKERS) ALMACENADOS"
       app:layout_constraintEnd_toEndOf="parent"
       app:layout_constraintStart_toStartOf="parent"
       app:layout_constraintTop_toBottomOf="@+id/btnMapUbicacion" />


   <Button
       android:id="@+id/btnMapPolilineas"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:text="MOSTRAR POLILÍNEAS ENTRE PUNTOS"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/btnMapMarkers" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

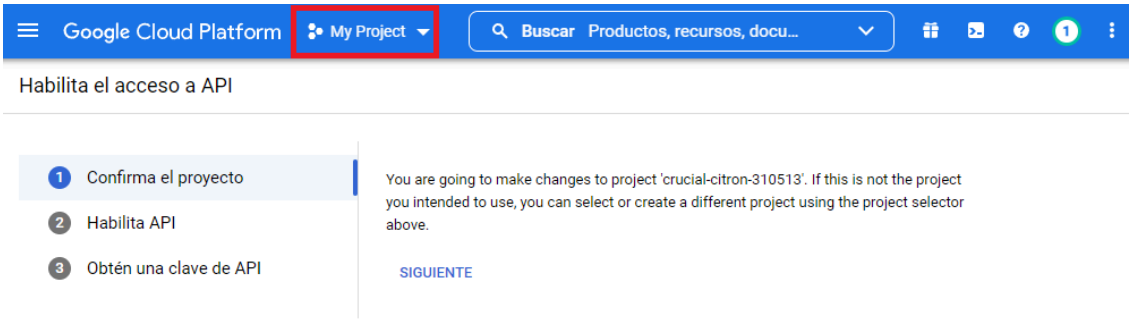
3. Comenzaremos trabajando para mostrar simplemente el mapa, el código en la clase MainActivity.kt, inicialmente, será el siguiente:

```
class MainActivity : AppCompatActivity() {
    private lateinit var binding: ActivityMainBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)

        binding = ActivityMainBinding.inflate(layoutInflater)
        setContentView(binding.root)

        binding.btnMapGeneral.setOnClickListener {
            val intent = Intent(this, MapsActivity::class.java)
            startActivity(intent)
        }
    }
}
```

4. Tras ello es necesario crear la actividad que mantiene el mapa. Si nos vamos a **File → New → Google → Google Maps Views Activity** nos permite crear una actividad basada en Google Maps automáticamente.
5. Con todo esto se nos presentará un asistente que nos permitirá crear una actividad basada en Google Maps. En nuestro caso la actividad recibirá el nombre de MapsActivity, ya que así lo hemos llamado en la intención usada en MainActivity.java. Tras asignar el nombre pulsaremos Finish y creará automáticamente la actividad basada en Google Maps.
6. El asistente, una vez que termina, crea dos archivos:
    a. La actividad: **MapsActivity.kt.**
    b. El layout: **activity_maps.xml.**

7. En el archivo AndroidManifest.xml nos encontramos con una dirección a la consola de Google, allí será donde generamos la clave para este proyecto (obligatorio para el funcionamiento de los mapas). Para ello accedemos a la consola de google, https://console.developers.google.com/

- Tienes que estar logueado en una cuenta de Google para realizar la solicitud.
- Nos aparecerá la consola de desarrolladores "Google Cloud Platform".



- Tendrás que seleccionar un proyecto existente o crear un nuevo proyecto. En nuestro caso, crearemos un nuevo proyecto, he utilizado el nombre de Tema11App1.

- Tras la creación debemos confirmar el proyecto, habilitar API y obtener una clave de API.
- También debemos habilitar el uso de Mapas en Android, para ello en el apartado Biblioteca, habilitar Maps SDK for Android.
- En tu archivo AndroidManifest.xml, ve a com.google.android.geo.API_KEY y actualiza

el android:value attribute de la siguiente manera:

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="MAPS_API_KEY" />
```

- Para solicitar el permiso ACCESS_FINE_LOCATION y ACCES_COARSE_LOCATION, agrega este código al elemento manifest:

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

- Si ejecutamos la aplicación, recuerda utilizar un emulador que tenga la play store instalada, obtenemos lo siguiente:

8. Vamos a visualizar el código de la actividad **MapsActivity.kt** para ver que está ocurriendo exactamente:

```
class MapsActivity : AppCompatActivity(), OnMapReadyCallback {

   private lateinit var mMap: GoogleMap
   private lateinit var binding: ActivityMapsBinding

   override fun onCreate(savedInstanceState: Bundle?) {
```

```
        super.onCreate(savedInstanceState)

        binding = ActivityMapsBinding.inflate(layoutInflater)
        setContentView(binding.root)

        // Obtain the SupportMapFragment and get notified when the map is ready to be
used.
        val mapFragment = supportFragmentManager
            .findFragmentById(R.id.map) as SupportMapFragment
        mapFragment.getMapAsync(this)
    }

    /**
     * Manipulates the map once available.
     * This callback is triggered when the map is ready to be used.
     * This is where we can add markers or lines, add listeners or move the camera. In
this case,
     * we just add a marker near Sydney, Australia.
     * If Google Play services is not installed on the device, the user will be
prompted to install
     * it inside the SupportMapFragment. This method will only be triggered once the
user has
     * installed Google Play services and returned to the app.
     */
    override fun onMapReady(googleMap: GoogleMap) {
        mMap = googleMap

        // Add a marker in Sydney and move the camera
        val sydney = LatLng(-34.0, 151.0)
        mMap.addMarker(MarkerOptions().position(sydney).title("Marker in Sydney"))
        mMap.moveCamera(CameraUpdateFactory.newLatLng(sydney))
    }
}
```

Y el código del layout **activity_maps.xml** es este:

```
<?xml version="1.0" encoding="utf-8"?>
<fragment
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:map="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MapsActivity" />
```

9. Como podemos observar es un **Fragment**, por lo que la forma de trabajar de los mapas será mediante Fragments.

   Por ello, haremos uso del Fragment Manager, que es el componente del sistema operativo encargado de gestionar los fragmentos de una aplicación. Este será invocado mediante el método supportFragmentManager(), y una vez que lo tenemos, solicitaremos una referencia al fragmento de mapa mediante el método findFragmentById(). El tipo de objeto que deseamos obtener es SupportMapFragment.

   El método getMapAsync() se usa para establecer el callback (la devolución de la llamada) en el fragmento. En este caso, es obligatorio ponerlo.

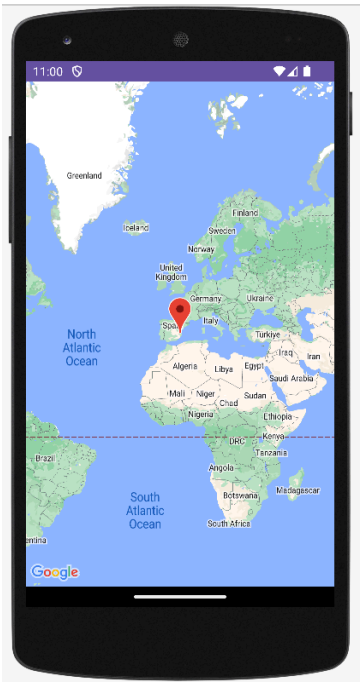
10. La clase MapsActivity implementa el interface OnMapReadyCallback, que nos permite usar el mapa cuando este se encuentra disponible. Dicho interfaz nos obliga a implementar, por defecto, el método onMapReady(), y en el mismo nos encontramos de ejemplo el marcador de la ciudad de Sydney. Dicho marcador se crea instanciando un objeto de tipo LatLng, que representa un par de coordenadas de latitud y longitud, almacenada como grados.

    Aunque lo vamos a cambiar por el siguiente código para clarificar el uso del mismo:

```
override fun onMapReady(googleMap: GoogleMap) {
    mMap = googleMap

    // Añadimos la posición del IES Simarro y movemos la cámara

    val iesSimarro = LatLng(38.986, -0.532944)
    mMap.addMarker(MarkerOptions().position(iesSimarro).title("IES Lluís simarro"))
    mMap.moveCamera(CameraUpdateFactory.newLatLng(iesSimarro))
}
```

    Tras crear el objeto LatLng y asignar sus coordenadas creamos un objeto MarkerOptions y le asignamos el objeto con las coordenadas (que será la posición), así como un título. Tras ello añadimos el objeto MarkerOptions al mapa y movemos la cámara mediante el método moveCamera(). Al mover la cámara comprobamos que dentro invocamos a la clase CameraUpdateFactory, donde dicha clase contiene métodos que cambian la cámara.

11. Si ejecutamos el programa el resultado de la ejecución sería el siguiente:



1.3.1. Posicionándonos en la ubicación actual

1. Continuaremos trabajando con el proyecto **Tema11App1**.
2. Ahora vamos a añadir la posibilidad de que nos posicione en la ubicación actual donde nos encontramos. Para ello vamos a modificar el código de **MainActivity.kt**:

```
class MainActivity : AppCompatActivity() {
   private lateinit var binding: ActivityMainBinding

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)

       binding = ActivityMainBinding.inflate(layoutInflater)
       setContentView(binding.root)

       binding.btnMapGeneral.setOnClickListener {
           val intent = Intent(this, MapsActivity::class.java)
           intent.putExtra("operacion", 0)
           startActivity(intent)
       }

       binding.btnMapGeneral.setOnClickListener {
           val intent = Intent(this, MapsActivity::class.java)
           intent.putExtra("operacion", 1)
           startActivity(intent)
       }
   }
```

```
}
```

De esta forma pasaremos un parámetro denominado “operacion” a la actividad MapsActivity.kt con la intención de que esta sepa que queremos hacer, si mostrar el mapa general, mostrar la ubicación actual…

3. En la clase **MapsActivity.kt** modificaremos el método onCreate() de la siguiente forma:

```
class MapsActivity : AppCompatActivity(), OnMapReadyCallback {

   private lateinit var mMap: GoogleMap
   private lateinit var binding: ActivityMapsBinding
   private  var operacion: Int? = 0

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)

       binding = ActivityMapsBinding.inflate(layoutInflater)
       setContentView(binding.root)

       // Obtain the SupportMapFragment and get notified when the map is ready to be
used.
       val mapFragment = supportFragmentManager
           .findFragmentById(R.id.map) as SupportMapFragment
       mapFragment.getMapAsync(this)

       operacion = intent.getIntExtra("operacion", 0)
   }
```

De esta forma recogeremos la operación que vamos a realizar y la almacenaremos en una variable que vamos a utilizar en el siguiente punto.

4. Modificaremos el método **onMapReady()** para que realice la operación que deseamos, según el botón que hemos pulsado en la pantalla de inicio, e implementamos el método privado **ubicacionActual()**, que nos situará en la ubicación actual donde nos encontramos. El código será el siguiente:

```
override fun onMapReady(googleMap: GoogleMap) {
   mMap = googleMap

   when (operacion) {
       0 -> {
```

```
            posicionSimarro()
        }
        1 -> {
            ubicacionActual()
        }
    }
}

private fun posicionSimarro() {
    // Añadimos la posición del IES Simarro y movemos la cámara

    val iesSimarro = LatLng(38.986, -0.532944)
    mMap.addMarker(MarkerOptions().position(iesSimarro).title("IES Lluís simarro"))
    mMap.moveCamera(CameraUpdateFactory.newLatLng(iesSimarro))
}


private fun ubicacionActual() {
    // Seteamos el tipo de mapa
    mMap.mapType = GoogleMap.MAP_TYPE_HYBRID

    if (ContextCompat.checkSelfPermission(
            this,
            Manifest.permission.ACCESS_FINE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED ||
        ContextCompat.checkSelfPermission(
            this,
            Manifest.permission.ACCESS_COARSE_LOCATION
        ) != PackageManager.PERMISSION_GRANTED
    ) {
        Toast.makeText(this, "MyLocation Enabled", Toast.LENGTH_SHORT).show()
        return
    }

    // Activamos la capa o layer MyLocation
    mMap.isMyLocationEnabled = true
}
```

En el método anterior estamos obteniendo una referencia del objeto mapa para manipular sus propiedades, en este caso, configuramos:

- El tipo de mapa, donde existen 5 tipos de mapas:
    - MAP_TYPE_HYBRID: Mapa satelital con una capa transparente de las principales calles.
    - MAP_TYPE_NONE: Mapa sin capas.

- MAP_TYPE_NORMAL: Mapa normal o básico.
- MAP_TYPE_SATELLITE: Mapa satelital sin etiquetas ni capas adicionales.
- MAP_TYPE_TERRAIN: Mapa de terreno.

- Activamos la capa MyLocation, la cual nos permite obtener la ubicación actual del usuario en un pequeño punto azul.

Otra posibilidad para coger la ubicación actual:

```
private fun ubicacionActual2() {
    val fusedLocationClient: FusedLocationProviderClient =
LocationServices.getFusedLocationProviderClient(this)
    if (ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED ||
        ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED
    ) {
        // TODO: Consider calling
        //    ActivityCompat#requestPermissions
        // here to request the missing permissions, and then overriding
        //   public void onRequestPermissionsResult(int requestCode, String[]
permissions,
        //                                          int[] grantResults)
        // to handle the case where the user grants the permission. See the
documentation
        // for ActivityCompat#requestPermissions for more details.
        return
    }

    fusedLocationClient.lastLocation
        .addOnSuccessListener(this) { location ->
            // Got last known location. In some rare situations this can be null.
            if (location != null) {
                // Logic to handle location object
                val posActual = LatLng(location.latitude, location.longitude)
                mMap.addMarker(MarkerOptions().position(posActual).title("Estás
aquí"))
                mMap.moveCamera(CameraUpdateFactory.newLatLng(posActual))
            }
        }
}
```

Para más información sobre la ubicación actual os recomiendo visitar este apartado del developer:

https://developers.google.com/maps/documentation/android-sdk/location

1.3.2. Usando los Markers de Google Maps

Aunque ya los hemos utilizado inicialmente, y los hemos explicado, vamos a definirlos con algo más de profundidad. Un **Marker** es una especie de icono indicador que permite señalar en el mapa la ubicación de un punto definido por una Latitud y una Longitud.

Para dibujar un Marker necesitamos las coordenadas de Latitud y Longitud. Por tanto, vamos a crear un método nuevo que reciba un objeto del tipo LatLng el cual corresponde con un objeto de posición en Google Maps, que recibirá también como parámetro un String que corresponde con el título del marcador y otro String que corresponde con información de detalle del marcador, por lo que este método tendrá el siguiente código:

```
// Método que nos permite crear marcadores
private fun setMarker(position: LatLng, titulo: String, info: String) {

   // Agregamos marcadores para indicar sitios de interés
   val myMarker = mMap.addMarker(MarkerOptions()
       .position(position)
       .title(titulo)  // Agrega un título al marcador
       .snippet(info)  // Agrega información detalle relacionada

.icon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_GREEN))) //
Color del marcador
}
```

Por lo que, comenzaremos de la siguiente forma:

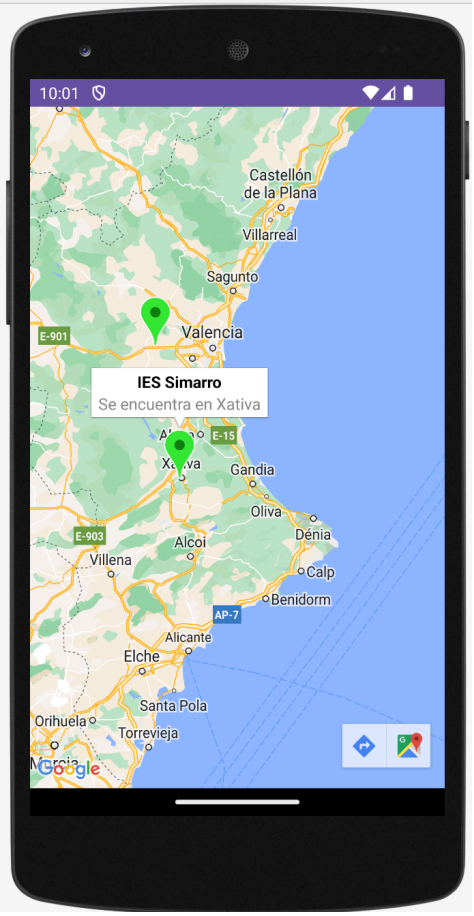
1. Continuaremos trabajando con el proyecto **Tema11App1**.
2. Nos vamos a la clase MainActivity.kt y modificamos el método onCreate() añadiendo y recogiendo en este caso el botón btnMapMarkers, y añadiendo una nueva operación. Omitimos el código por ser totalmente repetitivo a los puntos anteriores.
3. Nos vamos a la clase MapsActivity.kt y añadiremos el método setMarker() anterior.
4. Tras ello vamos a definir dos objetos de tipo LatLng que van a contener las coordenadas de dos centros educativos.

```
private val iesSimarro = LatLng(38.986, -0.532944)
private val fpCheste = LatLng(39.476488, -0.648207)
```

5. Modificamos el método onMapReady() para que recoja la nueva operación:

```
...
2 -> {
    setMarker(iesSimarro, "IES Simarro", "Se encuentra en Xativa")
    setMarker(fpCheste, "CIPFP Cheste", "Se encuentra en Cheste")

}
...
```

6. Si ejecutamos la aplicación y pulsamos dicho botón, ampliando con el zoom, veremos:



Para más información de los marcadores os recomiendo que visitéis esta página del developer:

- https://developers.google.com/maps/documentation/android-sdk/marker

1.3.3. Dibujando Polilíneas en Google Maps

Las **polilíneas** en Google Maps Android API, son segmentos de líneas que se trazan uniendo 2 o más puntos de latitud y longitud representados por objetos de la clase LatLng. Las polilíneas se representan empleando la clase Polyline cuyo constructor espera un objeto del tipo PolylineOptions.

Para no hacerlo demasiado complicado vamos a crear una nueva ruta a una cadena de comidas rápidas de hamburguesas

Comenzaremos de la siguiente forma:

1. Continuaremos trabajando con el proyecto **Tema11App1**.
2. Nos vamos a la clase MainActivity.kt y modificamos el método onCreate() añadiendo y recogiendo en este caso el botón btnMapsPolilineas, y añadiendo una nueva operación. Omitimos el código por ser totalmente repetitivo a los puntos anteriores.
3. A continuación, definiremos la polilínea con las posiciones intermedias para ir a la hamburguesería. El código de la declaración de la misma quedaría de la siguiente forma (todo estos puntos lo hemos calculado en la página de Google Maps, https://www.google.es/maps/) en la clase MapaActivity.kt (ojo, la polilínea comienza en el IES Dr. Lluís Simarro y acaba en el parking de la hamburguesería, no en el mismo restaurante, por eso la última coordenada no es la del restaurante):

```
...
private val hamburgueseria = LatLng(39.478992, -0.426408)

private val POLILINEA = PolylineOptions()
    .add(iesSimarro)
    .add(LatLng(39.482039, -0.422245))
    .add(LatLng(39.481244, -0.422760))
    .add(LatLng(39.478858, -0.424340))
    .add(LatLng(39.478352, -0.424717))
    .add(LatLng(39.478418, -0.425030))
    .add(LatLng(39.478762, -0.425824))
    .add(LatLng(39.479084, -0.426598))
    .add(LatLng(39.479337, -0.427238))
    .add(LatLng(39.479367, -0.427436))
```

```
    .add(LatLng(39.479287, -0.427600))
    .add(LatLng(39.479145, -0.427570))
    .add(LatLng(39.479061, -0.427397))
    .add(LatLng(39.479095, -0.427119))
    .add(LatLng(39.479080, -0.426925))
    .add(LatLng(39.479000, -0.426538))
    .add(LatLng(39.478808, -0.426067))
    .add(LatLng(39.478609, -0.425992))
    .add(LatLng(39.478544, -0.425843))
    .add(LatLng(39.478406, -0.425858))
...
```

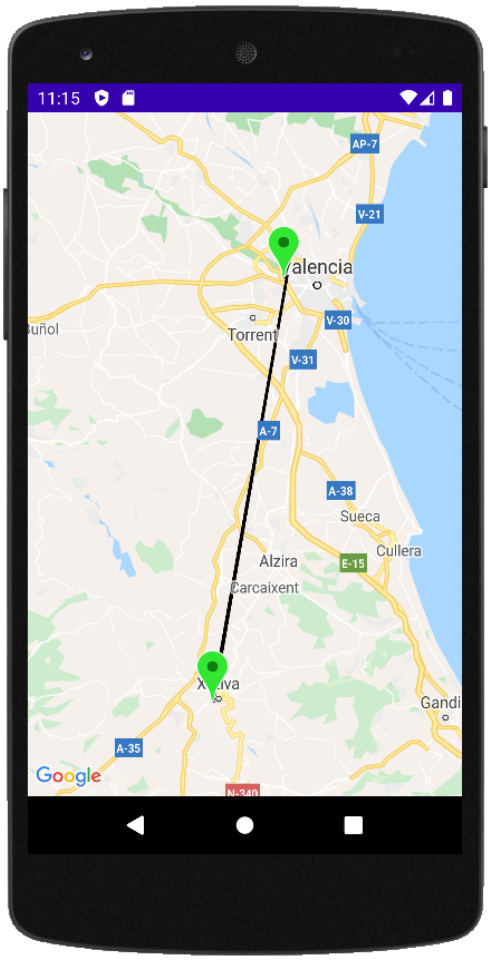
4. Crearemos un método nuevo que se llamará setPolyline(), que se encarga de añadir la polilínea al mapa. El código será el siguiente:

```
// Método que añade la polilínea al mapa
private fun setPolilyne(options: PolylineOptions) {
    val polyline: Polyline = mMap.addPolyline(options)
}
```

5. Ya solo nos queda modificar el método onReadyMap() para añadir la nueva operación:

```
...
3 -> {
    setMarker(iesSimarro, "IES Simarro", "Se encuentra en Xativa")
    setMarker(hamburgueseria, "McDonald's", "Se encuentra en Mislata")
    setPolilyne(POLILINEA)
}
...
```

6. Si ejecutamos veremos el resultado final.



Para más información del dibujo de formas en maps os recomiendo que visitéis esta página del developer:

- https://developers.google.com/maps/documentation/android-sdk/shapes

1.3.4. Más ejemplos

Para ampliar las funcionalidades de la API de Google Maps, podemos acceder a los recursos siguientes:

- https://cursokotlin.com/capitulo-28-google-maps-en-android-con-kotlin/
- https://cursokotlin.com/capitulo-30-google-maps-en-android-con-kotlin-parte-2-localizacion

-en-tiempo-real-kotlin/

- https://cursokotlin.com/polylines-en-google-maps-con-kotlin-parte-3-capitulo-37/