

# **INSTITUTO TECNOLÓGICO DE LA LAGUNA**

## **DESARROLLO DE APLICACIONES EN ANDROID**

Profesora: Lina Ernestina Arias Hernández

SEMESTRE: AGO – DIC 2024

### **Manual de aplicación**

**Brandon Eduardo Reyes Hernández - 21130931**

**Roberto Grijalva Sánchez - 20130054**

ING. SISTEMAS COMPUTACIONALES

Torreón, Coahuila

22/11/2024

## Tabla de contenidos

<b>Configuración de api de Google</b>	<b>2</b>
<b>Documentación de Atributos de clase</b>	<b>11</b>
<b>Mostrar el mapa en pantalla e inicializar sensores</b>	<b>13</b>
<b>Inicio de onSensorChanged</b>	<b>14</b>
<b>Botón timer</b>	<b>16</b>
<b>Botón reset</b>	<b>17</b>
<b>Final de onCreate</b>	<b>18</b>
<b>Inicialización y actualización de temporizador</b>	<b>19</b>
<b>Método requestLocationPermission</b>	<b>20</b>
<b>Método requestActivityRecognitionPermission</b>	<b>21</b>
<b>Método getLastKnownLocation</b>	<b>22</b>
<b>Método onRequestPermissionsResult</b>	<b>23</b>
<b>Método onDestroy</b>	<b>24</b>
<b>Método onMapReady</b>	<b>25</b>
<b>Referencias</b>	<b>26</b>

## Configuración de api de Google

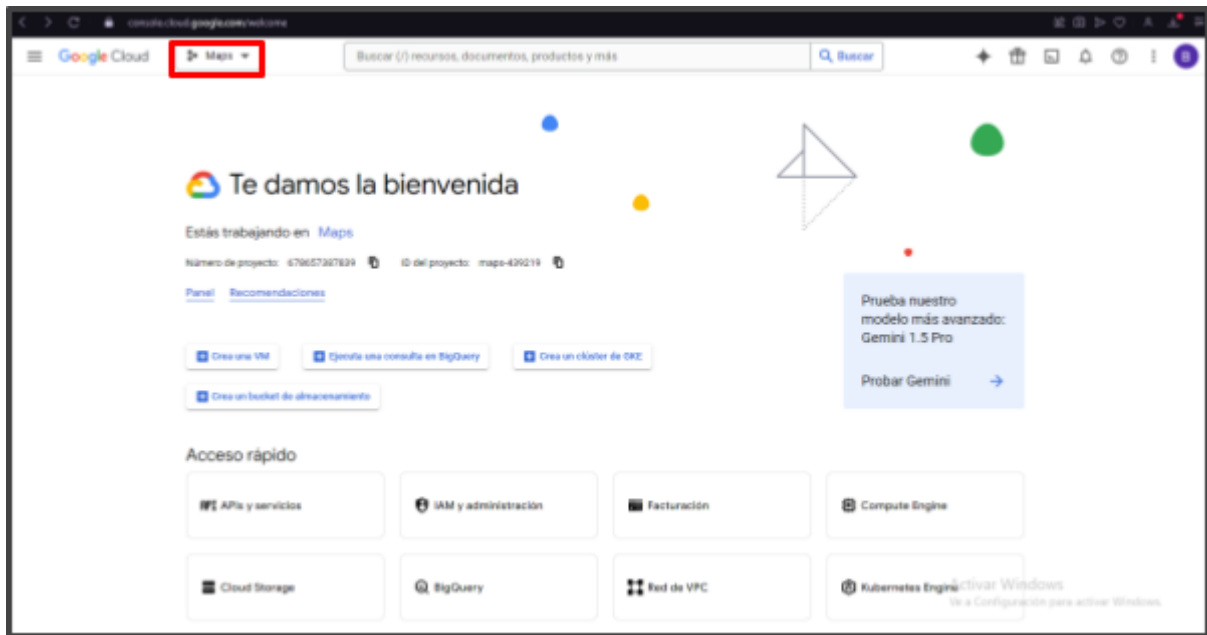
Primero, para poder hacer uso de las librerías necesarias, en el proyecto, dentro de “**build.gradle.kts (:app)**”, nos dirigimos a dependencias y agregamos “**implementation(libs.play.services.location)**” y “**implementation(libs.play.services.maps)**”. Nos quedaría algo así (Figura 1):

```
dependencies {  
  
    implementation(libs.appcompat)  
    implementation(libs.material)  
    implementation(libs.activity)  
    implementation(libs.constraintlayout)  
    implementation(libs.play.services.tasks)  
    implementation(libs.play.services.location)  
    implementation(libs.play.services.maps)  
    testImplementation(libs.junit)  
    androidTestImplementation(libs.ext.junit)  
    androidTestImplementation(libs.espresso.core)  
}
```

(Figura 1).

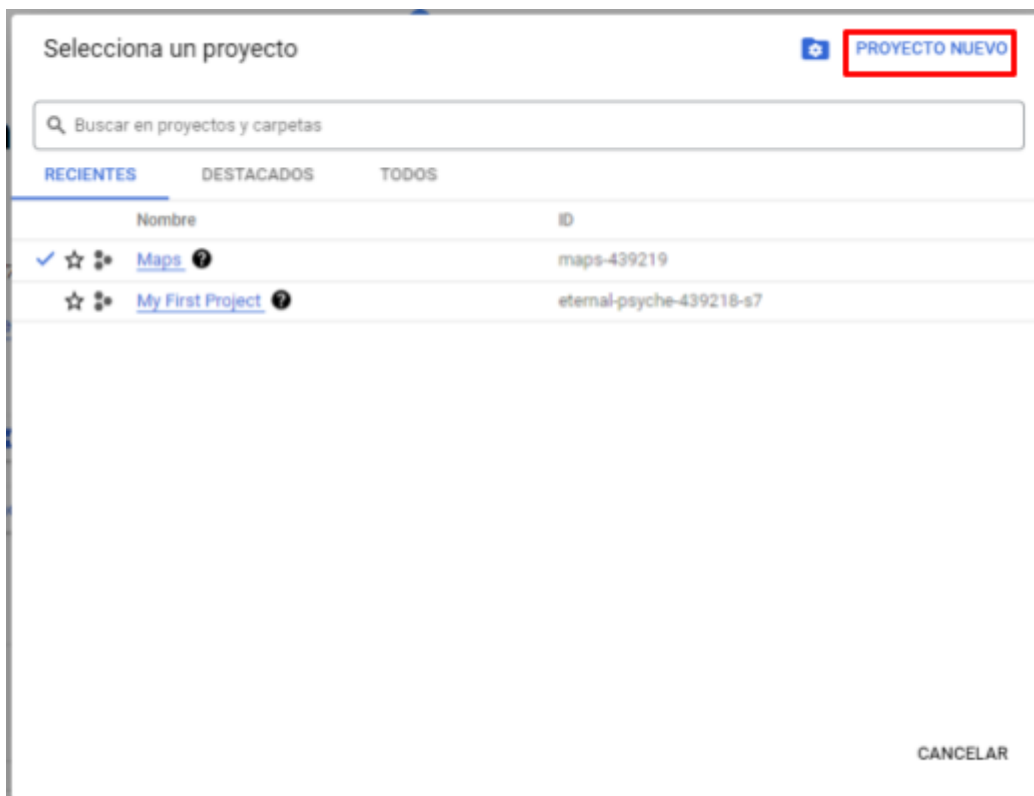
Se tendrá que sincronizar el proyecto.

Para el siguiente paso, antes tenemos que crear un proyecto en “<https://console.cloud.google.com>”, primero hay que registrarse y loguearse. Una vez dentro, nos mostrará la siguiente ventana (Figura 2), dentro de ella, damos clic en el apartado de “**Maps**” (el cual está marcado en rojo).



(Figura 2).

Se nos abrirá una ventana y damos click en “**NUEVO PROYECTO**” (Figura 3).



(Figura 3).

Nos redirige a la página donde crearemos el proyecto y solamente colocamos el nombre que le queramos asignar, en el apartado de ubicación, lo dejamos tal cual como está (“**Sin organización**”), una vez hecho esto, damos click en crear. (Cabe resaltar que solamente se tienen cierta cantidad de proyectos gratuitos como límite,

si excedemos ese límite, se tendrá que ya sea, eliminar proyectos, o aumentar la cuota con un formulario y posiblemente realizando pagos), (Figura 4).

### Proyecto nuevo



Tienes 23 projects restantes en tu cuota. Solicita un incremento o borra algunos proyectos. [Más información](#)

[MANAGE QUOTAS](#)

Nombre del proyecto \*  
MyMaps

ID del proyecto: mymaps-442602. No se puede cambiar más adelante. [EDITAR](#)

Ubicación \*  
 Sin organización

[EXPLORAR](#)

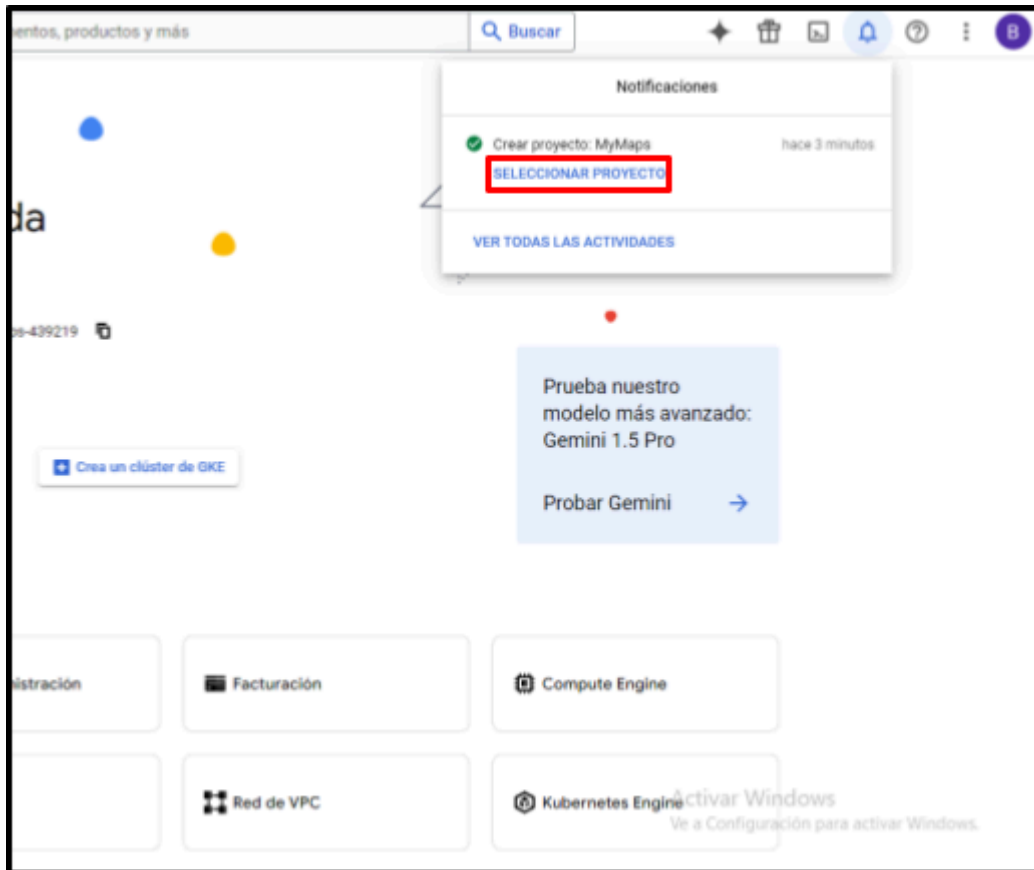
Organización o carpeta superior

CREAR

CANCELAR

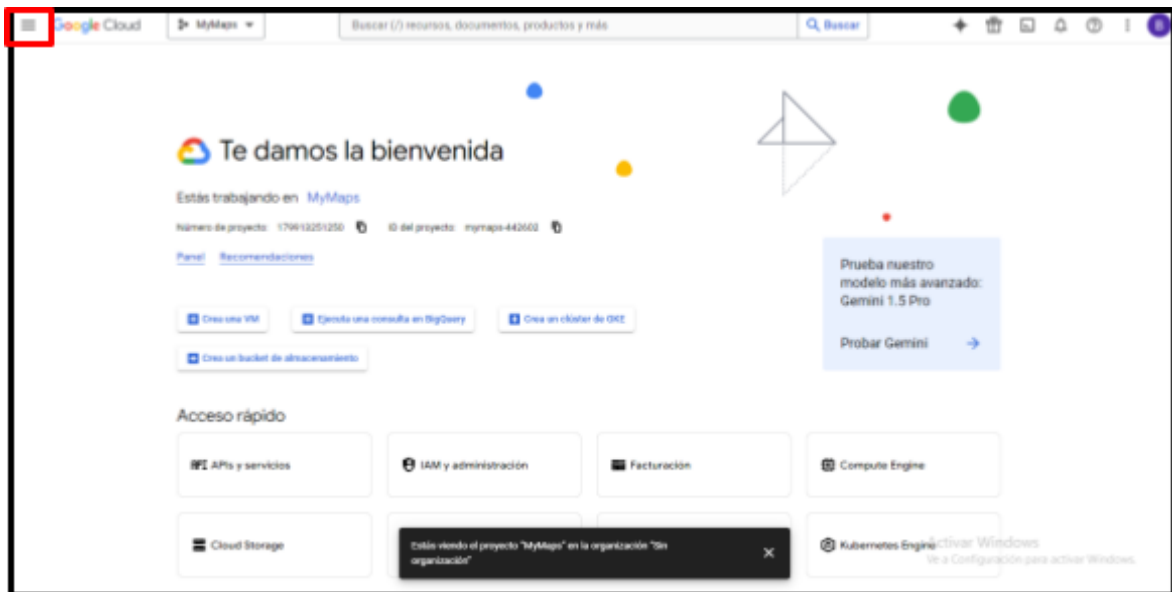
(Figura 4).

Ahora, esperamos un poco mientras se crea el proyecto, y una vez se termine de crear, nos aparecerá una notificación mencionando que ya está listo y damos click en **“SELECCIONAR PROYECTO”** (Figura 5).

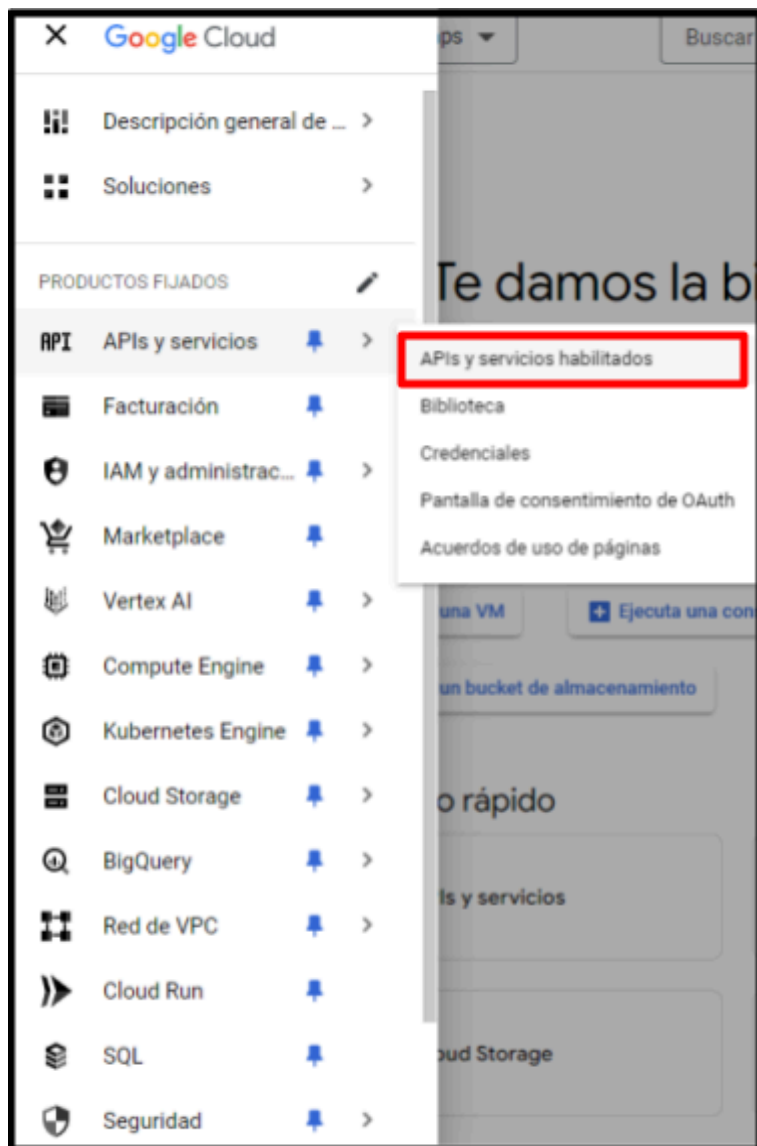


(Figura 5).

Abrirá el proyecto que creamos, ahora, abrimos el menú de navegación (Figura 6) y en la sección de **“APIs y servicios”**, damos click en **“APIs y servicios habilitados”** (Figura 7).



(Figura 6).



(Figura 7).

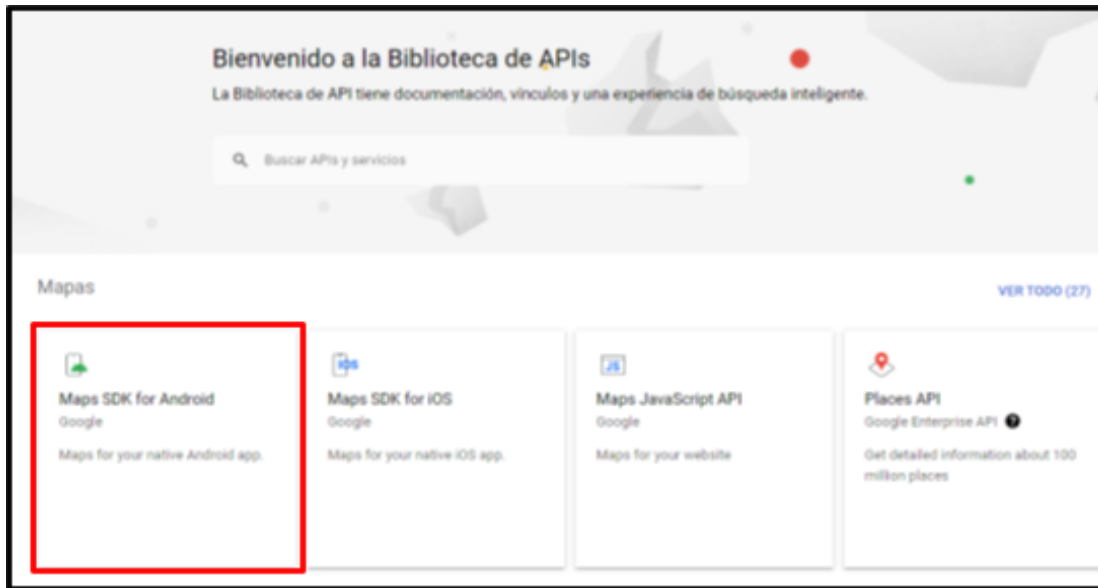
Ahora, en la ventana que se abrió, damos click en “**HABILITAR APIS Y SERVICIOS**”, (Figura 8).



(Figura 8).



Nos mostrará todos los servicios que se pueden utilizar, en nuestro caso, utilizamos “**Maps SDK for Android**”, así que, seleccionamos ese (Figura 9).



(Figura 9).

Ahora, simplemente damos click en “**Habilitar**” (Figura 10).



(Figura 10).

Luego de esperar unos segundos, nos dirá que se ha generado nuestra clave de API (se recomienda mantener habilitada la casilla para recibir notificaciones en caso de que estemos a punto de exceder el crédito mensual), en esa ventana, damos click en “**IR A GOOGLE MAPS PLATAFORM**” (Figura 11).



(Figura 11).

Una vez hecho eso, en el “**AndroidManifest.xml**”, se agregó la etiqueta para hacer uso del api de Google (esto nos permite utilizar el mapa en nuestra aplicación), en el identificador “**name**” se hace igual y en el de “**value**” se debe ingresar una clave de API (Figura 12):

```
<meta-data
    android:name="com.google.android.geo.API_KEY"
    android:value="AIzaSyBWax5np-I8YW5b0NpsK6zdBAwXvQyUAdE"
/>
```

(Figura 12).

Además, dependiendo de los permisos que estos tendrán que estar declarados en el mismo xml (la diferencia entre **FINE\_LOCATION** y **COARSE\_LOCATION** es la precisión con la que lo hace cada uno, el primero, con una precisión de 10 ms o superior y el segundo siendo de segundos, se utiliza el que más se adecúe de acuerdo al proyecto que se vaya a realizar) (Figura 13):

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

(Figura 13).

Para poder mostrar el mapa, se hace uso de un fragment en el “**activity\_main.xml**”, le configuramos los parámetros a gusto y le agregamos la línea “**android:name=“com.google.android.gms.maps.SupportMapFragment”** (esto para poder mostrar en la interfaz el mapa) (Figura 14):

```
<fragment
    android:id="@+id/map"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="300dp"
    map:cameraZoom="10"
    map:uiRotateGestures="true"
    map:uiZoomControls="true" />
```

(Figura

14).

## Documentación de Atributos de clase

Primero, tenemos los atributos de clase (Figura 15), (Figura 16):

**private SensorManager sensorManager;**

Se utiliza para lograr acceder a los sensores que se tienen disponibles, es un administrador de sensores, por así decirlo.

**private Sensor stepCounterSensor;**

Este es el sensor de pasos, el cual nos permite almacenar los pasos que haya detectado el sensor y así poder contabilizarlos

**private SensorEventListener stepListener;**

Este evento, permite reconocer cuándo ha cambiado algún dato nuevo en el sensor, y así poder manejar las actualizaciones necesarias, como la interfaz de usuario, las asignaciones en variables, inicio de programa, etc.

**private FusedLocationProviderClient fusedLocationClient;**

Da acceso a los servicios de “Fused Location Provider”, los cuales son de Google y sirven para obtener la posición actual del dispositivo.

**private TextView stepCountTextView, locationTextView;**

Son parte de la interfaz gráfica, en “stepCountTextView” se muestran los pasos que se han dado, mientras que en el “locationTextView”, muestran la localización con latitud y longitud.

**private GoogleMap nMap;**

Se puede usar gracias a “SupportMapFragment”, con “nMap” nos permite mostrar la ubicación actual del usuario en el mapa.

**private static final int LOCATION\_PERMISSION\_REQUEST\_CODE = 1;**

Nos permite dar a entender al programa que vamos a solicitar el permiso de ubicación

**private static final int ACTIVITY\_RECOGNITION\_REQUEST\_CODE = 2;**

En este caso, se usa también para dar a conocer el permiso, pero esta vez de actividad del usuario.

**private TextView distanceTextView, timeTextView, caloriesTextView, stepsByminutesTextView;**

Son parte de la interfaz de usuario.

- **distanceTextView:** Muestra la distancia recorrida en metros.
- **timeTextView:** Muestra el tiempo transcurrido.
- **caloriesTextView:** Muestra las calorías quemadas estimadas.

- **stepsByminutesTextView:** Muestra el promedio de pasos dados por minuto.

**boolean isRunning = false;**

Se utiliza para saber cuando el usuario dio click en el botón de inicio.

**private ImageView timer, reset;**

Son ImageViews que se utilizan para implementar los botones tanto de inicio/parar, como de reiniciar.

**private int secondsElapsed = 0;**

Se utiliza para almacenar el tiempo transcurrido desde el inicio del temporizador.

**private Handler handler = new Handler();**

Permite gestionar el uso de ejecuciones repetitivas, por ejemplo, en el temporizador, se utiliza para poder actualizar el temporizador sin problemas.

```
public class MainActivity extends AppCompatActivity implements OnMapReadyCallback {

    //Uso de sensor
    private SensorManager sensorManager; 5 usages
    //Contador de pasos
    private Sensor stepCounterSensor; 3 usages
    //Evento del sensor
    private SensorEventListener stepListener; 4 usages

    private FusedLocationProviderClient fusedLocationClient; 2 usages

    // textviews que vamos a usar para el contador de pasos y localización
    private TextView stepCountTextView, locationTextView; 3 usages

    //Mapa de google
    private GoogleMap mMap; 2 usages

    //Permisos
    private static final int LOCATION_PERMISSION_REQUEST_CODE = 1; 2 usages
    private static final int ACTIVITY_RECOGNITION_REQUEST_CODE = 2; 2 usages

    //Apartir de aquí, lo nuevo
    //Nuevos textfields
    private TextView distanceTextView, timeTextView, caloriesTextView, stepsByminutesTextView; 3 usages

    //Boolean para controlar el botón del timer
    boolean isRunning = false; 4 usages
}
```

(Figura 15).

```
//Botones timer y reset
private ImageView timer, reset; 4 usages

//Contador de segundos
private int secondsElapsed = 0; 5 usages
private Handler handler = new Handler(); 2 usages
```

(Figura 16).

## Mostrar el mapa en pantalla e inicializar sensores

**SupportMapFragment mapFragment = (SupportMapFragment)  
getSupportFragmentManager().findFragmentById(R.id.map);**

Esta clase es proporcionada por la API de Google Maps, el cual simplemente nos permite implementar un mapa de Google y asignarlo “map” de la interfaz y así mostrar el mapa.

**mapFragment.getMapAsync(this);**

Una vez mostrado el mapa, realiza un proceso asíncrono, gracias a esto es que se muestra el mapa en pantalla (cabe resaltar que, para hacer uso de un mapa, debemos implementar la API de Google y para esto se debe asignar en “AndroidManifest.xml”, esto se explica al final).

**sensorManager = (SensorManager) getSystemService(SENSOR\_SERVICE);**

Simplemente inicializa el “sensorManager” y le asigna el servicio “SENSOR\_SERVICE” y así poder hacer uso de sensores en general.

**stepCounterSensor =**

**sensorManager.getDefaultSensor(Sensor.TYPE\_STEP\_COUNTER);**

Asigna a “stepCounterSensor” un sensor de tipo contador de pasos, para lograr llevar a cabo el conteo de los pasos, esto devolverá un valor flotante (Figura 17).

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

    stepCountTextView = findViewById(R.id.stepCountTextView);
    locationTextView = findViewById(R.id.locationTextView);
    SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map);
    mapFragment.getMapAsync(callback: this);

    // Inicializar el sensor de pasos
    sensorManager = (SensorManager) getSystemService(SENSOR_SERVICE);
    stepCounterSensor = sensorManager.getDefaultSensor(Sensor.TYPE_STEP_COUNTER);
```

(Figura 17).

## Inicio de onSensorChanged

```
if (stepCounterSensor != null) {
```

Verifica si el sensor de contador de pasos se encuentra disponible para su uso.

```
stepListener = new SensorEventListener() {
```

```
    @Override
```

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

Esto nos permite verificar cuándo hay un cambio en el sensor, o sea, que se han detectado pasos.

```
final int[] stepCount = {(int) event.values[0]};
```

Simplemente almacena la cantidad de pasos totales detectados.

```
double stepLength = 0.75;
```

```
final double[] distance = {stepCount[0] * stepLength};
```

Se multiplica la cantidad de pasos detectados por una longitud promedio, para poder almacenar la distancia promedio recorrida.

```
double MET = 3.5;
```

```
int peso = 70;
```

```
double horas = (double) secondsElapsed / 3600;
```

```
final double[] cal = {MET * peso * horas};
```

Primero se declara la una variable double “MET” con 3.5, el cuál es una unidad de medida aproximada que se gastaría al realizar cierta actividad física. Luego, se asigna el peso en kg, posteriormente se calculan las horas gracias a los segundos que se tienen almacenados. y con esas variables, se calcula el aproximado de calorías quemadas y se almacena en “cal”.

```
double minutes = (double) secondsElapsed / 60;
```

```
final double[] pm = {stepCount[0] / minutes};
```

De los segundos, se calculan los minutos transcurridos y se almacenan en “minutes”, luego en “pm” se almacena la cantidad de pasos promedio dados por minuto (Figura 18)

```

if (stepCounterSensor != null) {
    stepListener = new SensorEventListener() {
        @Override
        public void onSensorChanged(SensorEvent event) {
            final int[] stepCount = {(int) event.values[0]};

            //Calcular distancia
            double stepLength = 0.75;
            final double[] distance = {stepCount[0] * stepLength};

            //Calcular calorías
            double MET = 3.5;
            int peso = 70;
            double horas = (double) secondsElapsed / 3600;
            final double[] cal = {MET * peso * horas};

            //Calcular pasos/minuto
            double minutes = (double) secondsElapsed / 60;
            final double[] pm = {stepCount[0] / minutes};
        }
    };
}

```

Figura 18).



## Botón timer

Cuando se da un click, si el booleano “isRunning” es falso, lo cambia a true, para decir que se está ejecutando y así manejar los cambios correctamente.

Después, a timer (el botón en sí) le asigna una nueva imagen (stop).

Luego, manda a llamar a startTimer (esto se explica después).

Al botón reset, lo vuelve inaccesible (que el usuario no lo pueda clickear), esto para evitar fallos.

Luego, muestra los pasos realizados por el usuario, la distancia recorrida, las calorías quemadas y el promedio de pasos por minuto.

Si cuando se clickea el botón, el booleano “isRunning” es verdadero, primero, lo cambia a false.

Al botón timer, le asigna una imagen para representar que está detenida la aplicación y que puede iniciarla.

### **handler.removeCallbacks(runnable);**

Esto simplemente detiene el temporizador por medio del objeto handler, de esta forma ya no se estará actualizando.

Y finalmente habilita el botón “reset” para su uso (Figura 19).

```
timer.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View v) {
        if (!isRunning) {
            isRunning = true;
            timer.setImageResource(R.drawable.stop);
            startTimer();
            reset.setEnabled(false);
            //int stepCount = (int) event.values[0];
            stepCountTextView.setText("Pasos: " + stepCount[0]);
            distanceTextView.setText("Distancia recorrida: " + distance[0] + " metros");
            caloriesTextView.setText("Calorias consumidas: " + cal[0]);
            stepsByminutesTextView.setText("Pasos por minuto: " + pm[0]);
        } else {
            isRunning = false;
            timer.setImageResource(R.drawable.start);
            handler.removeCallbacks(runnable);
            reset.setEnabled(true);
            event.values[0] = 0;
        }
    }
});
```

(Figura 19).

## Botón reset

Cambia los textos que se muestran en pantalla, los cuales son, los de distancia recorrida, el tiempo transcurrido, las calorías quemadas, los pasos por minuto y los pasos totales contados (en ese orden).

Asigna el valor de las variables a cero, para que cuando se vuelva a iniciar la ejecución, puedan mostrar los datos correspondientes de forma correcta (Figura 20).

```
//Boton reset
reset.setOnClickListener(new View.OnClickListener(){
    @Override
    public void onClick(View v){
        distanceTextView.setText("Distancia recorrida:-");
        timeTextView.setText("Tiempo transcurrido: 00:00");
        caloriesTextView.setText("Calorias consumidas:-");
        stepsByminutesTextView.setText("Pasos por minuto:-");
        stepCountTextView.setText("0");

        distance[0] = 0;
        stepCount[0] = 0;
        cal[0] = 0;
        pm[0] = 0;
    }
});
}
```

(Figura 20).

## Final de onCreate

Primero, por medio del “sensorManager” se pone a registrar tanto a “stepListener”, como a “stepCountSensor”, de los cuales “stepListener” es el objeto que manejará los pasos registrados y “stepCountSensor” nos dirá cuando haya un cambio en el sensor, para poder realizar tanto el conteo de pasos, como la actualización de la interfaz de usuario.

El else que podemos observar aquí, es del if que se hace mención anteriormente: **“if (stepCounterSensor != null) {”**.

Lo que quiere decir, que si no está disponible el sensor encargado de contar los pasos, manda un mensaje a la pantalla mencionando que no se encuentra disponible el sensor.

**fusedLocationClient = LocationServices.getFusedLocationProviderClient(this);**

Inicializa el cliente de ubicación proporcionado por Google, este nos permitirá obtener la ubicación actual del usuario.

Manda a llamar una serie de métodos (Figura 21):

- **requestLocationPermission();**
- **requestActivityRecognitionPermission();**
- **getLastKnownLocation();**

```
        sensorManager.registerListener(stepListener, stepCounterSensor, SensorManager.SENSOR_DELAY_NORMAL);
    } else {
        Toast.makeText(context: this, text: "Step counter sensor no disponible!", Toast.LENGTH_SHORT).show();
    }

    // Inicializar cliente de ubicación
    fusedLocationClient = LocationServices.getFusedLocationProviderClient(activity: this);

    // Solicitar permisos
    requestLocationPermission();
    requestActivityRecognitionPermission();

    // Obtener la ubicación una vez que los permisos sean concedidos
    getLastKnownLocation();
}
```

(Figura 21).

## Inicialización y actualización de temporizador

**private void startTimer(){ runnable.run(); }**

Es un método que nos permite arrancar el objeto Runnable (el cual permite hacer uso de instancias por medio de un hilo, y runnable o run significa que se ha iniciado un hilo, el cual estará siendo usado constantemente) por medio de run(). Lo cual, dicho de otra forma, nos permitirá iniciar el temporizador.

**if (isRunning) {**

Esta condición simplemente verifica si se ha iniciado la ejecución (si han dado click al botón "buttonTimer").

**int minutes = secondsElapsed / 60;**

**int seconds = secondsElapsed % 60;**

Calculan tanto los minutos como los segundos, ya que la variable "secondsElapsed" (encargada de almacenar el tiempo transcurrido), toma el tiempo en segundos.

**timeTextView.setText(String.format("Tiempo transcurrido: %02d:%02d", minutes, seconds));**

Al "timeTextView", le asigna un texto mostrando los minutos y segundos.

**secondsElapsed++;**

**handler.postDelayed(this, 1000);**

secondsElapsed se increments en uno para registrar un nuevo segundo y gracias a handler, se configuró para que este método se ejecute cada segundo, asegurando que secondsElapsed siga incrementando correctamente y que el tiempo se muestre al usuario cada segundo (Figura 22).

```
private void startTimer() { runnable.run(); }

private Runnable runnable = new Runnable() { 2 usages
    @Override
    public void run() {
        if (isRunning) {
            int minutes = secondsElapsed / 60;
            int seconds = secondsElapsed % 60;
            timeTextView.setText(String.format("Tiempo transcurrido: %02d:%02d", minutes, seconds));
            secondsElapsed++;
            handler.postDelayed(this, delayMillis: 1000);
        }
    }
};
```

(Figura 22).

## Método requestLocationPermission

Lo primero que podemos observar es la condición:

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

Lo que hace es sencillo, verifica si no se han otorgado permisos al acceso de la localización.

De ser ese el caso, entonces, por medio de

“**ActivityCompat.requestPermissions**” pide al usuario permisos de “**ACCESS\_FINE\_LOCATION**”, el cual es para obtener la ubicación aproximada del dispositivo. También pide permisos de “**ACCESS\_BACKGROUND\_LOCATION**” el cual permite que se pueda acceder a la ubicación incluso si se está ejecutando la aplicación en segundo plano, finalmente, con “**LOCATION\_PERMISSION\_REQUEST\_CODE**” se añadirá un indicador del permiso que estamos solicitando (Figura 23).

```
// Solicitar permiso de ubicación
private void requestLocationPermission() { 2 usages
    if (ContextCompat.checkSelfPermission(context: this, Manifest.permission.ACCESS_FINE_LOCATION)
        != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions(activity: this,
            new String[]{Manifest.permission.ACCESS_FINE_LOCATION, Manifest.permission.ACCESS_BACKGROUND_LOCATION},
            LOCATION_PERMISSION_REQUEST_CODE);
    }
}
```

(Figura 23).

## Método requestActivityRecognitionPermission

Tiene un if similar que el método anterior, solo que esta vez sirve para corroborar que no se tenga permiso de reconocimiento de actividad.

En caso de no tenerlo, por medio del “**ActivityCompat**”, pide al usuario el permiso “**ACTIVITY\_RECOGNITION**” para saber cuándo hay actividad física por parte del usuario, con “**ACTIVITY\_RECOGNITION\_REQUEST\_CODE**” se añadirá un indicador que dé a conocer qué permiso solicitamos (Figura 24).

```
// Solicitar permiso de reconocimiento de actividad
private void requestActivityRecognitionPermission() { 1 usage
    if (ContextCompat.checkSelfPermission( context: this, Manifest.permission.ACTIVITY_RECOGNITION)
        != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: this,
            new String[]{Manifest.permission.ACTIVITY_RECOGNITION},
            ACTIVITY_RECOGNITION_REQUEST_CODE);
    }
}
```

(Figura 24).

## Método getLastKnownLocation

Lo primero que se aprecia a ver es:

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

Es solamente para validar que se tenga permisos para acceder a la localización del dispositivo.

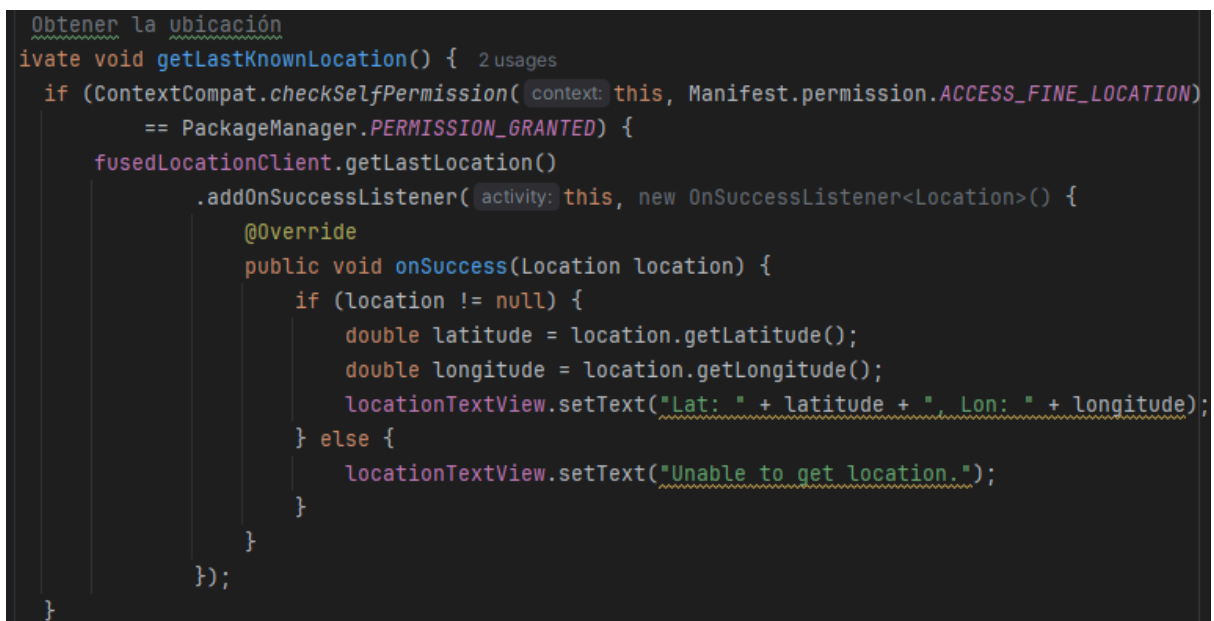
```
fusedLocationClient.getLastLocation().addOnSuccessListener(this, new
OnSuccessListener<Location>() {
```

Con esta línea, lo que quiere decir, es que, “**fusedLocationClient**” manda a llamar a “**getLastLocation**”, obteniendo un resultado exitoso, agrega un “**OnSuccessListener**” a la clase “**Location**” para poder manejarlo.

```
if (location != null) {
```

Una vez se obtuvo la ubicación, se verifica que esta no esté vacía, y por medio de la variable “**latitude**” se almacena la latitud de la ubicación que se obtuvo. Por medio de la variable “**longitude**” se almacena la longitud de la ubicación que se obtuvo. Posteriormente se imprimen los resultados en “**locationTextView**”.

Si resultase que la ubicación fue nula, se imprime el mensaje “Unable to get location.” por medio de “**locationTextView**” (Figura 25).

A screenshot of a code editor showing the implementation of the getLastKnownLocation method. The code is in Java and uses Android's Location API. It starts with a comment 'Obtener la ubicación' and a private void method signature. Inside, there's a permission check using ContextCompat.checkSelfPermission. If the permission is granted, it calls fusedLocationClient.getLastLocation().addOnSuccessListener. Inside the success listener, there's an @Override annotation and a public void onSuccess method. This method checks if the location is not null. If it is, it gets the latitude and longitude and sets the text of a TextView with a formatted string. If it's null, it sets the text to 'Unable to get location.'.

```
Obtener la ubicación
private void getLastKnownLocation() { 2 usages
    if (ContextCompat.checkSelfPermission(context: this, Manifest.permission.ACCESS_FINE_LOCATION)
        == PackageManager.PERMISSION_GRANTED) {
        fusedLocationClient.getLastLocation()
            .addOnSuccessListener(activity: this, new OnSuccessListener<Location>() {
                @Override
                public void onSuccess(Location location) {
                    if (location != null) {
                        double latitude = location.getLatitude();
                        double longitude = location.getLongitude();
                        locationTextView.setText("Lat: " + latitude + " Lon: " + longitude);
                    } else {
                        locationTextView.setText("Unable to get location.");
                    }
                }
            });
    }
}
```

(Figura 25).

## Método onRequestPermissionsResult

**super.onRequestPermissionsResult(requestCode, permissions, grantResults);**

Se llama a sí mismo automáticamente si el usuario interactúa con alguna solicitud de permisos. “**requestCode**” es lo que ayuda a identificar el permiso de sensor que se está solicitando usar, si resulta ser igual a

“**LOCATION\_PERMISSION\_REQUEST\_CODE**”, entonces se sabrá que usaremos la ubicación. Posteriormente, se verifica si “**grantResults**” (El cual contiene ayuda a comprobar si el permiso fue concedido). Esto se hace posible por lo siguiente:

```
if (grantResults.length > 0 && grantResults[0] ==  
PackageManager.PERMISSION_GRANTED) {  
    getLastKnownLocation();
```

Este if, verifica que “grantResults” no esté vacío, además de esto, el permiso que se alojó al inicio de “grantResults” haya sido concedido. De ser así, obtiene la última ubicación del dispositivo.

En caso de que el permiso resultara denegado, se imprime el mensaje “Location permission denied!” en pantalla, por medio de un Toast.

En el caso de que el permiso a verificar sea el de actividad, omite el código de ubicación y hace la misma verificación, pero con actividad, si resulta ser concedido, se imprime un mensaje que lo indica y en el caso de ser denegado, lo indica con un mensaje (Figura 26).

```
// Manejar la respuesta de los permisos  
@Override  
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {  
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);  
    if (requestCode == LOCATION_PERMISSION_REQUEST_CODE) {  
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {  
            getLastKnownLocation();  
        } else {  
            Toast.makeText(context, this, text: "Location permission denied!", Toast.LENGTH_SHORT).show();  
        }  
    } else if (requestCode == ACTIVITY_RECOGNITION_REQUEST_CODE) {  
        if (grantResults.length > 0 && grantResults[0] == PackageManager.PERMISSION_GRANTED) {  
            // Permiso de reconocimiento de actividad concedido  
            Toast.makeText(context, this, text: "Activity recognition permission granted!", Toast.LENGTH_SHORT).show();  
        } else {  
            Toast.makeText(context, this, text: "Activity recognition permission denied!", Toast.LENGTH_SHORT).show();  
        }  
    }  
}
```

(Figura 26).



## Método onDestroy

Cuando una actividad deja de usarse, se elimina memoria y libera recursos, para evitar que el equipo se alente, en este caso, el if verifica si el sensor de pasos no está vacío, del ser así, libera los recursos (Figura 27).

```
@Override
protected void onDestroy() {
    super.onDestroy();
    if (sensorManager != null && stepListener != null) {
        sensorManager.unregisterListener(stepListener);
    }
}
```

(Figura 27).

## Método onMapReady

Este método se invoca automáticamente cuando el mapa de Google está listo para ser usado.

**mMap = googleMap;**

Se le asigna el mapa que se obtuvo, a la variable global mMap

Después, por medio del if se verifica con “**checkSelfPermission**”(con él, se puede revisar si se tienen concedidos los permisos que se desea utilizar) si se tienen permisos para el acceso a la ubicación, de ser así, pasa lo siguiente:

**mMap.setMyLocationEnabled(true);**

Lo que hace es que le activa la capa de ubicación actual en el mapa, esto hace que el usuario pueda ver su ubicación en tiempo real.

En el caso de no tener permisos de acceso a la ubicación, llama a la función “**requestLocationPermission();**” para solicitar el permiso (Figura 28).

```
@Override no usages
public void onMapReady(@NonNull GoogleMap googleMap) {
    mMap = googleMap;

    // Verificar permisos de ubicación
    if (ContextCompat.checkSelfPermission(context: this,
        Manifest.permission.ACCESS_FINE_LOCATION) == PackageManager.PERMISSION_GRANTED) {
        mMap.setMyLocationEnabled(true); // Activa la capa de ubicación en el mapa
    } else {
        requestLocationPermission();
    }
}
```

(Figura 28).

## Referencias

El proyecto fue realizado gracias y en base a las siguientes fuentes de información y tutoriales:

[1]

“Información general de sensores | Sensors and location | Android Developers”. Android Developers. Accedido el 23 de noviembre de 2024. [En línea]. Disponible: [https://developer.android.com/develop/sensors-and-location/sensors/sensors\\_overview?hl=es-419](https://developer.android.com/develop/sensors-and-location/sensors/sensors_overview?hl=es-419)

[2]

dev.xcheko51x. *Obtener ubicación GPS con kotlin y android studio*. (26 de mayo de 2023). Accedido el 23 de noviembre de 2024. [Video en línea]. Disponible: <https://www.youtube.com/watch?v=ChTUvldz-tk>

[3]

freeCodeCamp.org. *Android Studio Tutorial - Build a GPS App*. (26 de enero de 2021). Accedido el 23 de noviembre de 2024. [Video en línea]. Disponible: [https://www.youtube.com/watch?v=\\_xUcYfbtfsI](https://www.youtube.com/watch?v=_xUcYfbtfsI)

[4]

Philipp Lackner. *How to Track Your Users Location in the Background in Android - Android Studio Tutorial*. (28 de septiembre de 2022). Accedido el 23 de noviembre de 2024. [Video en línea]. Disponible: <https://www.youtube.com/watch?v=Jj14sw4Yxk0>

[5]

Programación Android by AristiDevs. *Cómo conseguir la LOCALIZACIÓN en TIEMPO REAL con LOCATION SERVICE en KOTLIN*. (29 de junio de 2023). Accedido el 23 de noviembre de 2024. [Video en línea]. Disponible: [https://www.youtube.com/watch?v=k\\_XaQAVua4A](https://www.youtube.com/watch?v=k_XaQAVua4A)

[6]

Códigos de Programación - MR. *Cargar mapas y seleccionar ubicación - Android Studio | Java*. (03 de noviembre de 2022). Accedido el 03 de noviembre de 2024. [Video en línea]. Disponible: [https://www.youtube.com/watch?v=Df\\_CruIKxFc&t](https://www.youtube.com/watch?v=Df_CruIKxFc&t)