MODUL 7
Android Permissions and Google Maps

THEME DESCRIPTION

In this module, students will understand how to request and obtain app permissions in Android. Students will gain a solid understanding of how to include local and global interactive maps in their app by using the Google Maps API and how to request permissions to use device features that provide richer functionality.

WEEKLY LEARNING OUTCOME (SUB-LEARNING OUTCOME)

Students will be able to create permission requests for their app and handle missing permissions.

TOOLS/SOFTWARE USED

- Android Studio

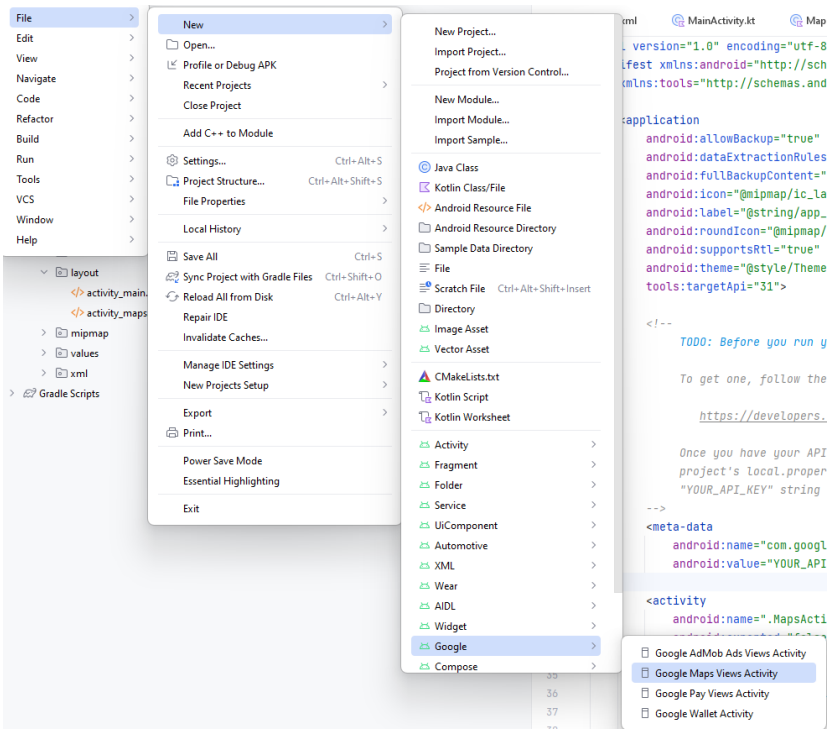
PRACTICAL STEPS

Part 1 - Requesting the location permission

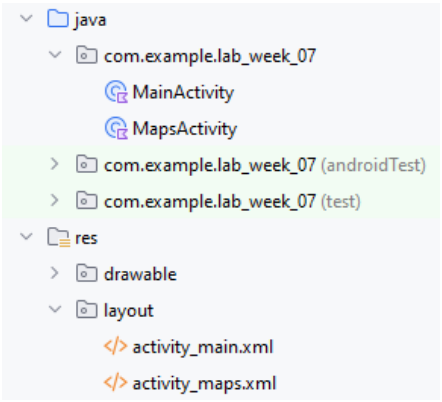
1. Open Android Studio and click **New Project.**
2. Choose the **Empty Views Activity** to start with.
3. Name your project **"LAB_WEEK_07"**.
4. Set the minimum SDK to **"API 24: Android 7.0 (Nougat)"**.
5. Click **Finish**, and let your android application build itself.
6. Create new **GitHub** repository for this project and commit.
7. In this part, we will be focusing on how we can **request the location permission from the user** in Android. First, import the necessary **Dependencies** to your **build.gradle.kts (Module :app)** and don't forget to **Gradle Sync**.

```
implementation(libs.androidx.activity.ktx)
implementation(libs.androidx.fragment.ktx)
```

8. For this tutorial, we will be using the google map activity to start off our app. Create a new **Google Activity** by clicking **File > New > Google > Google Maps Views Activity.**



9. 2 new files should appear, which are **MapsActivity.kt** and **activity_maps.xml**.



10. Now let's add the location permission in our manifest file as usual. Go to **AndroidManifest.xml** and add the code below before your application tag.

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

11. Don't forget to also change your main activity to the newly created **MapsActivity**. Update your **activity tags** in your **AndroidManifest.xml** to the code below.

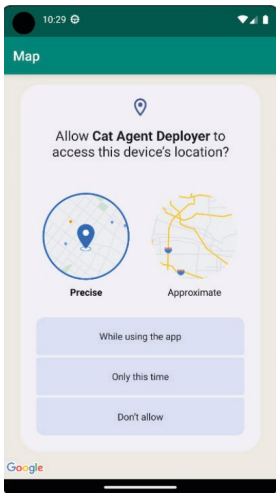
```
<activity
```

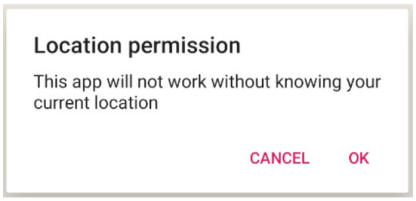
```
    android:name=".MapsActivity"
    android:exported="true"
    android:label="@string/title_activity_maps" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity
    android:name=".MainActivity"
    android:exported="false">
</activity>
```

12. Let's define the function that will be called when the user has granted the location permission. Add the code below at the end of your class below the **onMapReady function** in **MapsActivity.kt**.

```
private fun getLastLocation() {
    Log.d("MapsActivity", "getLastLocation() called.")
}
```

13. Now let's start asking the user for the required permission. Here's how this is gonna go, first, the user will be presented with the **system permission dialog**. Then if the user **denies** it, a **rationale dialog** will be shown next. A **rationale dialog** is a warning dialog to the user that the app will now work without the required permission. If the user presses **OK**, then the **system permission dialog** will be brought up again. But if the user presses **CANCEL**, then the app is stuck without the required permission. This sequence will be repeated **until the user grants the required permission** needed for the app.





14. To bring up the system permission request dialog, first create a variable to store an **ActivityResultLauncher** for the permission request. Add the code below at the very top of your class in **MapsActivity.kt**.

```
//This is the variable through which we will launch the permission request and
track user responses
private lateinit var requestPermissionLauncher: ActivityResultLauncher<String>
```

15. After creating the **ActivityResultLauncher** variable, let's create the function to register the **ActivityResult**. Add the code below at the very end of your **onCreate** callback in **MapsActivity.kt**.

```
//This is used to register for activity result
//The activity result will be used to handle the permission request to the user
//It accepts an ActivityResultContract as a parameter
//which in this case we're using the RequestPermission() ActivityResultContract
requestPermissionLauncher =
registerForActivityResult(ActivityResultContracts.RequestPermission()) {
       isGranted -> if (isGranted) {
           //If granted by the user, execute the necessary function
           getLastLocation()
       } else {
           //If not granted, show a rationale dialog
           //A rationale dialog is used for a warning to the user that the app will
```

```
now work without the required permission
            showPermissionRationale {
                requestPermissionLauncher.launch(ACCESS_FINE_LOCATION)
        }
    }
}
```

16. Now, let's declare the **showPermissionRationale** function to show the **rationale dialog**. Add the code below before your **getLastLocation function** in **MapsActivity.kt**.

```
//This is used to bring up a rationale dialog which will be used to ask the user
for permission again
//A rationale dialog is used for a warning to the user that the app will now work
without the required permission
//Usually it's brought up when the user denies the needed permission in the
previous permission request
private fun showPermissionRationale(positiveAction: () -> Unit) {
    //Create a pop up alert dialog that's used to ask for the required permission
again to the user
    AlertDialog.Builder(this)
        .setTitle("Location permission")
        .setMessage("This app will not work without knowing your current location")
        .setPositiveButton(android.R.string.ok) { _, _ -> positiveAction() }
        .setNegativeButton(android.R.string.cancel) { dialog, _ -> dialog.dismiss()
}
        .create().show()
}
```

17. All our declarations are done, now let's call our functions. But before that, we need to check if the user already has the permission granted for the app. Add the code below before your **showPermissionRationale** function in **MapsActivity.kt**.

```
//This is used to check if the user already has the permission granted
private fun hasLocationPermission() =
    ContextCompat.checkSelfPermission(this, ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED
```

18. Lastly, update your **onMapReady callback** to the code below in **MapsActivity.kt**. This will **launch** the **activity result** if the permission is not granted yet.

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

    //OnMapReady is called when the map is ready to be used
    //The code below is used to check for the location permission for the map
functionality to work
    //If it's not granted yet, then the rationale dialog will be brought up
    when {
        hasLocationPermission() -> getLastLocation()
        //shouldShowRequestPermissionRationale automatically checks if the user has
denied the permission before
        //If it has, then the rationale dialog will be brought up
        shouldShowRequestPermissionRationale(ACCESS_FINE_LOCATION) -> {
            showPermissionRationale {
                requestPermissionLauncher
                    .launch(ACCESS_FINE_LOCATION)
            }
        }
        else -> requestPermissionLauncher
            .launch(ACCESS_FINE_LOCATION)
    }
}
```

19. **Run** your app now and a **System Permission Request** should appear first. Then if you deny the request, a **Rationale Dialog** should appear next. If you press OK, then the **System Permission Request** will be brought up again. And if you deny it again, then try to press OK for the next **Rationale Dialog**, the next **System Permission Request** will not appear again. This is due to the change of default behavior from SDK 31 and onwards, which is that the **System Permission Request** will not be shown again after the **second time denying** it.

COMMIT to **GITHUB** at this point. **Commit Message: "Commit No. 1 – Create MapsActivity and location permission request"**.

Part 2 - Showing a map of the user's location

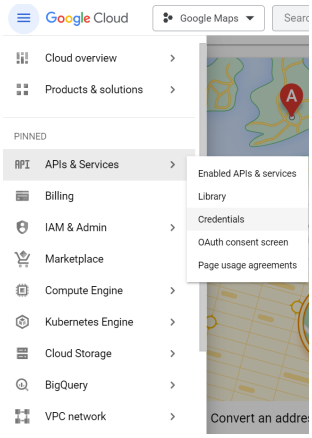
1. Continue your "**LAB_WEEK_07"** project.
2. Now that we have successfully got the user's consent to use the location permission, in this part, we will be focusing on how you can **set google maps to show your current location** in Android. First, **make sure** that you have the **dependency** below in your **Gradle File**. If it's not yet imported, add it to your **build.gradle.kts (Module:app)** and do a **Gradle Sync**.

```
implementation(libs.play.services.location)
```

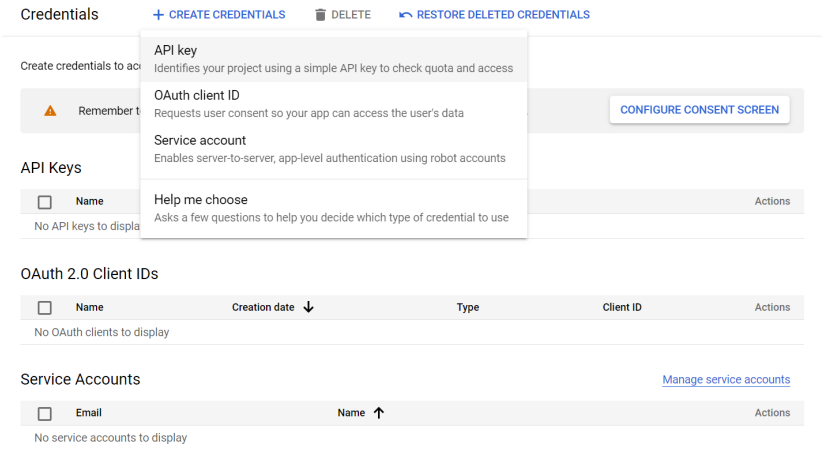
3. Next, in order to use the **Google Maps API service**, we first need the required **API Key** to be able to fetch data from Google Maps. Click the link below to start creating your **API Key**.

   https://console.cloud.google.com/welcome

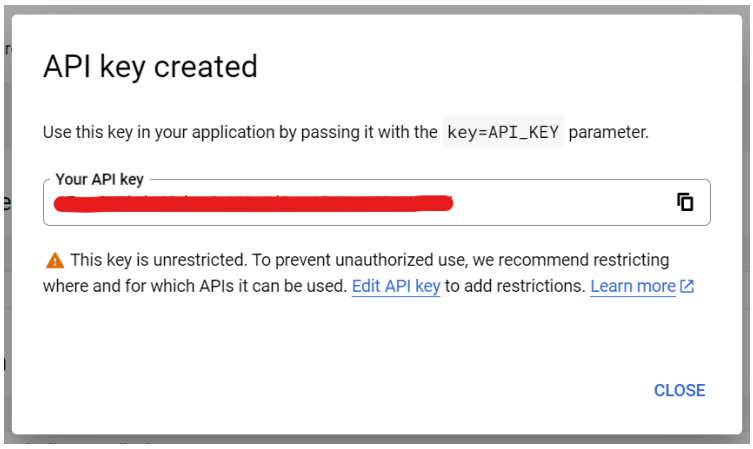
4. On your side navigation, navigate to **APIs & Services > Credentials**.



5. In the **Credentials Page**, click the **CREATE CREDENTIALS** button and choose the **API key** option.



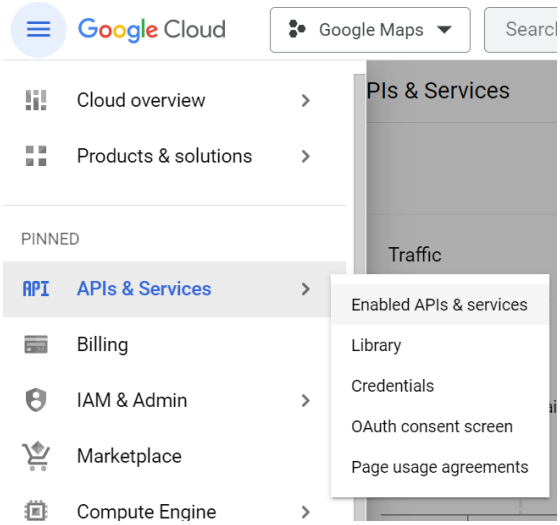
6. A dialog should pop up showing your **API Key**. Save it and never show it to anyone.



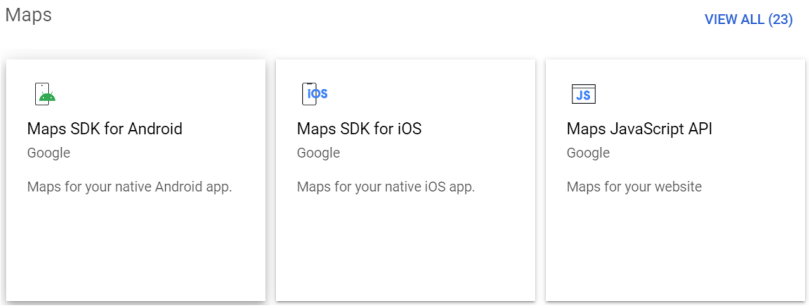
7. You've got your **API Key**, now let's add that to the application. Update your **meta-data** tag in **AndroidManifest.xml** and replace the **"YOUR_API_KEY"** with your **API Key**.

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

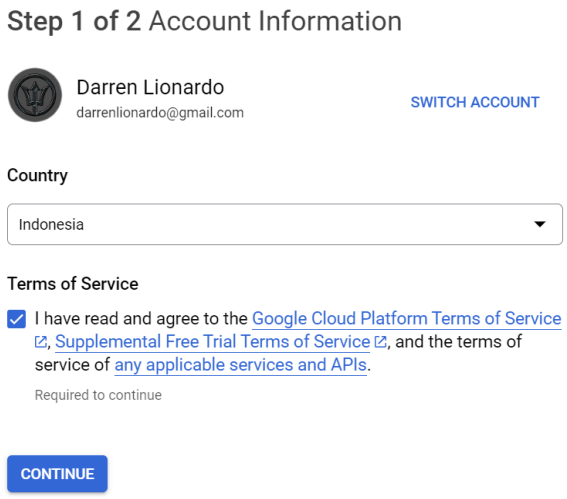
8. Still in your **Google Cloud Console**, we need to do 1 more thing and that is to enable the **Google Maps API for Android**. Navigate to **APIs & Services > Enabled APIs & services**.



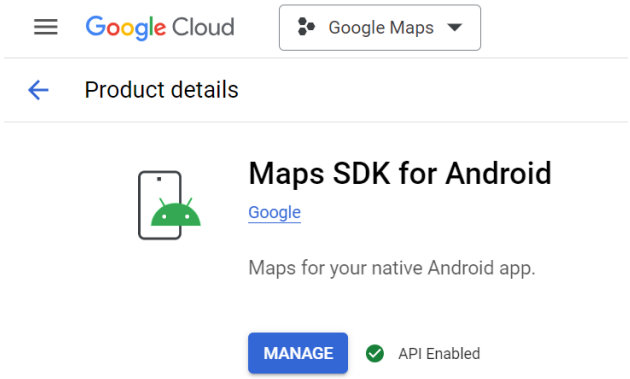
9. Click the **ENABLE APIS AND SERVICES** button and click the **Maps SDK for Android** option.



10. Once you're in, click the **ENABLE** button. You will then be presented with the following page.



11. **IGNORE** this page and you can straight up go back to the previous page to check if your **Maps SDK for Android** is enabled or not. If you continue the above page, then **Google Cloud** will ask you for your **Billing Info,** which is not needed for this tutorial. You may need to enter your own **Billing Info** to use other **Google Maps** features and enter the **Free Trial** provided by the **Google Cloud** itself.



12. Once **Enabled**, you're good to go and ready to set the location functionality in your project.
13. Next, let's create the functionality to get the user's current location. First, add the code below before your **onCreate callback** in **MapsActivity.kt**.

```
//A google play location service which helps us interact with Google's Fused
Location Provider API
//The API intelligently provides us with the device location information
private val fusedLocationProviderClient by lazy {
   LocationServices.getFusedLocationProviderClient(this)
}
```

14. After we've got the **API** that provides us with the user's location, we need to define a couple of other things. First is to **move the camera** to the provided location, and second is to **add a marker** at that position. Add the code below at the end of your **MapsActivity.kt** below the **getLastLocation** function.

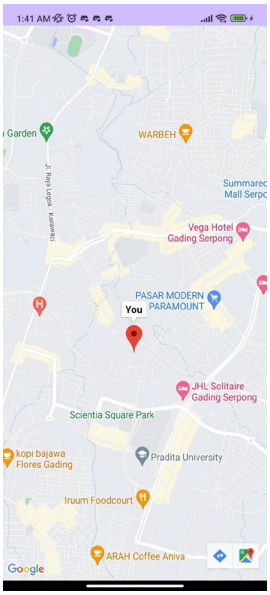
```
private fun updateMapLocation(location: LatLng) {
   mMap.moveCamera(CameraUpdateFactory.newLatLngZoom(
       location, 7f))
}

private fun addMarkerAtLocation(location: LatLng, title: String) {
   mMap.addMarker(MarkerOptions().title(title)
       .position(location))
}
```

15. Lastly, update your **getlastLocation** method to the code below in **MapsActivity.kt**.

```
//Executed when the location permission has been granted by the user
private fun getLastLocation() {
    if (hasLocationPermission()) {
        try {
            fusedLocationProviderClient.lastLocation
                .addOnSuccessListener { location: Location? ->
                    location?.let {
                        val userLocation = LatLng(location.latitude,
location.longitude)
                        updateMapLocation(userLocation)
                        addMarkerAtLocation(userLocation, "You")
                    }
                }
        } catch (e: SecurityException) {
```

```
            Log.e("MapsActivity", "SecurityException: ${e.message}")
        }
    } else {
        // If permission was rejected
        requestPermissionLauncher.launch(ACCESS_FINE_LOCATION)
    }
}
```

16. **Run** your app, and everything should be working as intended.



COMMIT to **GITHUB** at this point. **Commit Message: "Commit No. 2 – Add Google Maps API Key and pinpoint current user location"**.

ASSIGNMENT

Continue your **LAB_WEEK_07** project. Currently, you're just putting in your **API Key** straight into your **AndroidManifest.xml**. This is not a good practice as it's definitely not secured in any way. Provide a way so you can securely pass the required **API Key** for your **Google Maps API Service**. Use the guide below to help you achieve this.

https://developers.google.com/maps/documentation/android-sdk/start#add-key

COMMIT to **GITHUB** at this point. **Commit Message: "Commit No. 3 – Move Google Maps API Key to secure location"**.