How to Get the User’s Current Location in an Android App

In the modern age of mobile applications, location-based services have become increasingly important. Whether you're building a navigation app, a social media platform, or a service that provides personalized experiences based on user location, understanding how to retrieve the user's current location is a fundamental skill for any Android developer.

In this article, we'll walk through how to get the user’s current location in an Android app using Kotlin. We'll be referencing a simple project, which we’ll build step by step, that demonstrates how to:

1. Fetch the user's current coordinates (latitude and longitude).

2. Display the coordinates on the screen using a `TextView`.

3. Show the user’s current location on a Google Map.

4. Update the location every 10 minutes.

5. Stop location updates when the app is not in focus.

## Prerequisites

Before we dive in, you should have:

- A basic understanding of Kotlin and Android development.

- Android Studio installed.

- A Google Maps API key, which you can obtain by enabling the Google Maps API in the Google Cloud Console.

## Step 1: Setting Up the Project

Start by creating a new Android project in Android Studio. Select Kotlin as the language and choose a basic activity template.

Here's a quick rundown of the project setup:

- \*\*Project Name\*\*: LocationApp

- \*\*Package Name\*\*: `com.example.locationapp`

- \*\*Minimum SDK\*\*: API 21 (Lollipop)

Once your project is set up, we’ll move on to the core of our app.

## Step 2: Adding Necessary Permissions

To access the device's location, you need to request the appropriate permissions in your `AndroidManifest.xml` file. Add the following permissions:

```xml

<uses-permission android:name="android.permission.ACCESS\_FINE\_LOCATION" />

<uses-permission android:name="android.permission.ACCESS\_COARSE\_LOCATION" />

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

These permissions allow your app to access the user's location and connect to the internet to display maps.

**Step 3: Setting Up the ViewModel**

To handle location updates efficiently, we’ll use an AndroidViewModel. This allows us to manage UI-related data in a lifecycle-conscious way, ensuring that data survives configuration changes such as screen rotations.

Here's how the LocationViewModel is set up:

kotlin

Copy code

class LocationViewModel(application: Application) : AndroidViewModel(application) {

private val \_locationData = MutableLiveData<Location>()

val locationData: LiveData<Location> get() = \_locationData

private val \_latitude = MutableLiveData<Double>()

val latitude: LiveData<Double> get() = \_latitude

private val \_longitude = MutableLiveData<Double>()

val longitude: LiveData<Double> get() = \_longitude

private val locationManager = application.getSystemService(Context.LOCATION\_SERVICE) as LocationManager

private lateinit var locationListener: LocationListener

private val viewModelJob = Job()

private val viewModelScope = CoroutineScope(Dispatchers.Main + viewModelJob)

init {

startLocationUpdates()

}

fun startLocationUpdates() {

locationListener = object : LocationListener {

override fun onLocationChanged(location: Location) {

\_locationData.value = location

\_latitude.value = location.latitude

\_longitude.value = location.longitude

}

override fun onProviderEnabled(provider: String) {}

override fun onProviderDisabled(provider: String) {}

override fun onStatusChanged(provider: String?, status: Int, extras: Bundle?) {}

}

// Check for location permission

val hasFineLocationPermission = ContextCompat.checkSelfPermission(

getApplication(),

Manifest.permission.ACCESS\_FINE\_LOCATION

) == PackageManager.PERMISSION\_GRANTED

val hasCoarseLocationPermission = ContextCompat.checkSelfPermission(

getApplication(),

Manifest.permission.ACCESS\_COARSE\_LOCATION

) == PackageManager.PERMISSION\_GRANTED

if (hasFineLocationPermission || hasCoarseLocationPermission) {

if (locationManager.isProviderEnabled(LocationManager.GPS\_PROVIDER)) {

locationManager.requestLocationUpdates(

LocationManager.GPS\_PROVIDER,

0L,

0f,

locationListener

)

}

// Coroutine to update location every 10 minutes

viewModelScope.launch {

while (true) {

delay(10 \* 60 \* 1000L) // 10 minutes

if (locationManager.isProviderEnabled(LocationManager.GPS\_PROVIDER)) {

locationManager.requestSingleUpdate(LocationManager.GPS\_PROVIDER, locationListener, null)

}

}

}

} else {

// Handle the case where location permission is not granted

// You can prompt the user to grant permission here

}

}

fun stopLocationUpdates() {

locationManager.removeUpdates(locationListener)

}

override fun onCleared() {

super.onCleared()

stopLocationUpdates()

viewModelJob.cancel()

}

}

**Step 4: Creating the Main Activity**

The MainActivity is responsible for displaying the user’s location on the screen and showing it on a map. Here’s how you can set up the MainActivity:

kotlin

Copy code

class MainActivity : AppCompatActivity(), OnMapReadyCallback {

private lateinit var binding: ActivityMainBinding

private val locationViewModel: LocationViewModel by viewModels()

private lateinit var googleMap: GoogleMap

override fun onCreate(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

binding = ActivityMainBinding.inflate(layoutInflater)

setContentView(binding.root)

// Observe latitude and longitude

locationViewModel.latitude.observe(this, Observer { latitude ->

binding.latitudeTextView.text = getString(R.string.latitude\_text, latitude.toString())

})

locationViewModel.longitude.observe(this, Observer { longitude ->

binding.longitudeTextView.text = getString(R.string.longitude\_text, longitude.toString())

})

// Observe the combined location data to update the map

locationViewModel.locationData.observe(this, Observer { location ->

location?.let {

updateMap(it)

}

})

// Initialize fetching location

locationViewModel.startLocationUpdates()

}

private fun updateMap(location: Location) {

val latLng = LatLng(location.latitude, location.longitude)

googleMap.addMarker(MarkerOptions().position(latLng).title("Current Location"))

googleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(latLng, 15f))

}

override fun onMapReady(map: GoogleMap) {

googleMap = map

}

override fun onResume() {

super.onResume()

locationViewModel.startLocationUpdates()

}

override fun onPause() {

super.onPause()

locationViewModel.stopLocationUpdates()

}

override fun onDestroy() {

super.onDestroy()

locationViewModel.stopLocationUpdates()

}

}

**Step 5: Displaying the Coordinates**

Next, let’s display the user’s latitude and longitude on the screen. Open your activity\_main.xml layout file and add two TextView elements:

xml

Copy code

<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"

tools:context=".MainActivity">

<TextView

android:id="@+id/latitudeTextView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Latitude:"

app:layout\_constraintTop\_toTopOf="parent"

app:layout\_constraintStart\_toStartOf="parent"

android:layout\_marginTop="16dp"

android:layout\_marginStart="16dp"/>

<TextView

android:id="@+id/longitudeTextView"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Longitude:"

app:layout\_constraintTop\_toBottomOf="@id/latitudeTextView"

app:layout\_constraintStart\_toStartOf="parent"

android:layout\_marginTop="16dp"

android:layout\_marginStart="16dp"/>

</androidx.constraintlayout.widget.ConstraintLayout>

This layout will show the user's latitude and longitude as text, updated in real-time.

**Step 6: Integrating Google Maps**

To show the user's location on a map, you'll need to integrate Google Maps. Add a SupportMapFragment to your activity\_main.xml:

xml

Copy code

<fragment

android:id="@+id/map"

android:name="com.google.android.gms.maps.SupportMapFragment"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"/>

Update your MainActivity to implement the OnMapReadyCallback interface. You will also need to make sure that you’ve configured your Google Maps API key properly in the AndroidManifest.xml file.

**Step 7: Handling Permissions**

Since location services involve sensitive user data, you need to handle permissions carefully. Use Android’s permission model to request and manage location permissions, ensuring that your app behaves responsibly.

**Conclusion**

And there you have it—a simple Android application that fetches the user's current location, displays it on the screen, and shows it on a map. By using ViewModel and Kotlin coroutines, the app efficiently handles location updates and ensures that the user’s location is always current, even as they move around.

This project serves as a foundation for building more complex location-based features in your applications. You can extend this further by adding geofencing, location history, or even integrating with other APIs to provide more context-aware experiences for users.

Feel free to try out the code and customize it to fit your needs. Happy coding!