Location Services

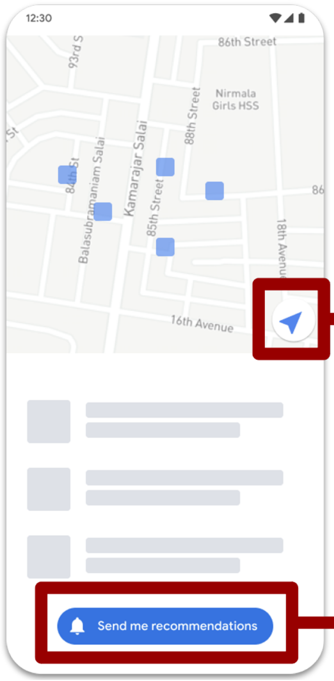
Jigisha Patel

Agenda

- Adding Location Services to Android apps
- Requesting Location Services Permissions
- Receiving Device Location
- Geocoding
- Displaying Location on Map

Location Services in Android

- Android's location permissions deal with the following categories of location access:
    - Foreground location
    - Background location



"Show current location" feature

Requires foreground location access

"Recommend nearby places" feature

Requires background location access

Image Source: https://developer.android.com/

Foreground Location

- If your app contains a feature that shares or receives location information only once, or for a defined amount of time, then that feature requires foreground location access.

- Some examples include the following:
  - Within a navigation app, a feature allows users to get turn-by-turn directions.
  - Within a messaging app, a feature allows users to share their current location with another user.

Foreground Location

- The system considers your app to be using foreground location if a feature of your app accesses the device's current location in one of the following situations:
    - An activity that belongs to your app is visible.
    - Your app is running a foreground service. When a foreground service is running, the system raises user awareness by showing a persistent notification. Your app retains access when it's placed in the background, such as when the user presses the Home button on their device or turns their device's display off.

Foreground Location Permission

- You declare a need for foreground location when your app requests either the ACCESS_COARSE_LOCATION permission or the ACCESS_FINE_LOCATION permission, as shown in the following snippet:

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

Foreground Location Permission cont...

- The level of precision depends on which permission you request:
  - **ACCESS_COARSE_LOCATION**
    - Provides location accuracy to within a city block.
  - **ACCESS_FINE_LOCATION**
    - Provides a more accurate location than one provided when you request ACCESS_COARSE_LOCATION.
    - This permission is necessary for some connectivity tasks, such as connecting to nearby devices over Bluetooth Low Energy (BLE)

Requesting Permissions

- Before fetching the location of the user, you should always request for the permission if not already granted.

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

} else {
    ActivityCompat.requestPermissions((Activity) context,
            this.permissionArray,
            this.LOCATION_PERMISSION_REQUEST_CODE);
}
```

Get the last known location

- Using the Google Play services location APIs, your app can request the last known location of the user's device.
- In most cases, the app would need the user's current location, which is usually equivalent to the last known location of the device.
- Specifically, use the **fused location provider** to retrieve the device's last known location.
- The fused location provider is one of the location APIs in Google Play services which manages the underlying location technology and provides a simple API so that you can specify requirements at a high level, like high accuracy or low power.
- It also optimizes the device's use of battery power.

Set up Google Play Services

- To access the fused location provider, your app's development project must include **Google Play services**.
- Download and install the Google Play services component via the SDK Manager and add the library to your project.
- Once the services are added, you should add the following dependency for Google Play Services in the app-level gradle.

```
dependencies {
   implementation 'com.google.android.gms:play-services-location:$mapsServiceVersion'
}
```

- The $mapsServiceVersion should be the version of Google Play Services that you want to use in the project such as “18.0.0”.
- For details, see the guide to Setting Up Google Play Services.

Create location services client

- In your activity's onCreate() method, create an **instance** of the Fused Location Provider Client as the following code snippet shows.

Get the last known location

- Once you have created the Location Services client you can get the last known location of a user's device.
- When your app is connected to these you can use the fused location provider's **getLastLocation()** method to retrieve the device location.
- The precision of the location returned by this call is determined by the permission setting you put in your app manifest.

Get the last known location cont…

- The getLastLocation() method returns a **Task** that you can use to get a Location object with the **latitude and longitude coordinates** of a geographic location.

- The location object may be **null** in the following situations:
  - Location is turned off in the device settings. The result could be null even if the last location was previously retrieved because disabling location also clears the cache.
  - The device never recorded its location, which could be the case of a new device or a device that has been restored to factory settings.
  - Google Play services on the device has restarted, and there is no active Fused Location Provider client that has requested location after the services restarted.

Location Settings

- If your app needs to request location or receive permission updates, the device needs to enable the appropriate system settings, such as GPS or Wi-Fi scanning.

- Rather than directly enabling services such as the device's GPS, your app specifies the required level of accuracy/power consumption and desired update interval, and the device automatically makes the appropriate changes to system settings.

- These settings are defined by the **LocationRequest** data object.

Location Request

- **setInterval()** - This method sets the rate in **milliseconds** at which your app prefers to receive location updates. Note that the location updates may be somewhat faster or slower than this rate to optimize for battery usage, or there may be no updates at all (if the device has no connectivity, for example).

- **setFastestInterval()** - This method sets the fastest rate in milliseconds at which your app can handle location updates. Unless your app benefits from receiving updates more quickly than the rate specified in setInterval(), you don't need to call this method.

- **setPriority()** - This method sets the priority of the request, which gives the Google Play services location services a strong hint about which location sources to use.
- The priority of PRIORITY_HIGH_ACCURACY, combined with the ACCESS_FINE_LOCATION permission setting that you've defined in the app manifest, and a fast update interval of 5000 milliseconds (5 seconds), causes the fused location provider to return location updates that are accurate to within a few feet. This approach is appropriate for mapping apps that display the location in real time.

- The following values are supported for the priority:
- **PRIORITY_BALANCED_POWER_ACCURACY** - Use this setting to request location precision to within a city **block**, which is an accuracy of approximately 100 meters. This is considered a **coarse** level of accuracy, and is likely to consume less power.
- **PRIORITY_HIGH_ACCURACY** - Use this setting to request the most **precise** location possible.
- **PRIORITY_LOW_POWER** - Use this setting to request **city-level** precision, which is an accuracy of approximately 10 kilometers. This is considered a coarse level of accuracy, and is likely to consume less power.
- **PRIORITY_NO_POWER** - Use this setting if you need negligible impact on power consumption, but want to receive location updates when available. With this setting, your app does not trigger any location updates, but receives locations triggered by other apps.

Request location updates

- Before requesting location updates, your app must connect to location services and make a location request.
- Once a location request is in place you can start the regular updates by calling **requestLocationUpdates()**.

- Depending on the form of the request, the fused location provider either invokes the **LocationCallback.onLocationResult()** callback method and passes it a list of Location objects.

- The accuracy and frequency of the updates are affected by the location permissions you've requested and the options you set in the location request object.

Define the location update callback

- The fused location provider invokes the **LocationCallback.onLocationResult()** callback method.

- The incoming argument contains a **list of Location object** containing the location's latitude and longitude.

Stop location updates

- Consider whether you want to stop the location updates when the activity is no longer in focus, such as when the user switches to another app or to a different activity in the same app.
- This can be handy to reduce power consumption, provided the app doesn't need to collect information even when it's running in the background.

Geocoding

- Geocoding is used when converting between coordinates and user-friendly place names.
- User place names are represented by Address object, which contains properties for specifying the street name, city name, country name, postal code, latitude, longitude and many more.

Display Location on Google Map

Adding Map

- The **Google Maps Android API** allows you to include maps and customized mapping information in your app.

- Google Maps Android API is part of the Google Play services platform.
- To use Google Maps, you should set up the Google Play services SDK in your app development project.

Steps to add Maps to App

1. Add the Google Maps Activity/Fragment in the project
2. Set up Google Maps API key in GCP console
3. Add the API key to Android project
4. Add the Maps dependency

Create Google Maps Activity/Fragment

- In the existing project, **right click** on source package, then click **New** then choose **Google -> Google Maps Activity** or **Fragment -> Google Maps Fragment**
- After providing name for the Activity/Fragment, click **Finish.**
- When the build is finished, Android Studio opens the **google_maps_api.xml** file in the editor that contains instructions on getting a Google Maps API key before you try to run the application.

Set up Google Maps API key in GCP console

- To **create an API key**:
  - Go to Google Cloud Platform Console
  - Create New Project and Select the same
  - Go to the APIs & Services -> Click on Enable APIs and Services
  - Search/select Maps SDK for Android and then click Enable on the SDK page
  - Go to the Credentials page, click Create credentials > API key. The API key created dialog displays your newly created API key.
  - Click Close.

- The new API key is listed on the Credentials page under API keys.

Adding the API key to the App

- You should not check your API key into your version control system, so it is recommended that it should be stored in the **local.properties** file, which is located in the root directory of your project so that it can be securely referenced by your app.

1. Open the **local.properties** in your project level directory, and then add the following code to the file. Replace YOUR_API_KEY with your API key.

```
MAPS_API_KEY=YOUR_API_KEY
```

Adding the API key to the App cont…

2. In your **app-level build.gradle** file, add this code in the defaultConfig element to allow Android Studio to read the Maps API key from the local.properties file at build time and then inject the mapsApiKey build variable into your Android manifest.

```
android {
   defaultConfig {
      // ...
      // Set the properties within `local.properties` into a `Properties` class so that values
      // within `local.properties` (e.g. Maps API key) are accessible in this file.
      Properties properties = new Properties()
      if (rootProject.file("local.properties").exists()) {
         properties.load(rootProject.file("local.properties").newDataInputStream())
      }

      // Inject the Maps API key into the manifest
      manifestPlaceholders = [ mapsApiKey : properties.getProperty("MAPS_API_KEY", "") ]
   }
}
```

Adding the API key to the App cont...

3. In your **AndroidManifest.xml** file, add the following code as a child of the application element.

```
<meta-data
   android:name="com.google.android.geo.API_KEY"
   android:value="${mapsApiKey}" />
```

4. Save the files and sync your project with Gradle.

Restrict API key

- Restricting API Keys adds security to your application by ensuring only authorized requests are made with your API Key.

1. Go to the **APIs & Services > Credentials** page.
2. Select the API key that you want to set a restriction on. The API key property page appears.
3. Under **Key restrictions**, set the following restrictions:
   - Application restrictions:
     a. Select **Android apps**.
     b. Click **+ Add package name and fingerprint**.
     c. Enter your package name and SHA-1 certificate fingerprint.
   - API restrictions:
     a. Click Restrict key.
     b. Select Maps SDK for Android from Select APIs dropdown. (If the Maps SDK for Android is not listed, you need to enable it.)
4. To finalize your changes, click **Save**.

Add the Maps dependency

- Add the following Maps dependency in the **app-level gradle** file.

```
dependencies {
   implementation 'com.google.android.gms:play-services-maps:$mapsServiceVersion'
   // ...
}
```

where mapsServiceVersion should be the version of the maps Play services that you want to use with your app such as “18.0.0”

Display map in Activity

- Add a Fragment object to the Activity that will handle the map. The easiest way to do this is to add a **<fragment>** element to the layout file for the Activity.
- Implement the **OnMapReadyCallback** interface and use the **onMapReady(GoogleMap)** callback method to get a handle to the GoogleMap object.
- The GoogleMap object is the internal representation of the map itself. To set the view options for a map, you modify its GoogleMap object.
- Call **getMapAsync()** on the fragment to register the callback.

References

- https://developer.android.com/training/location/permissions
- https://developer.android.com/training/permissions/requesting
- https://developer.android.com/training/location/request-updates
- https://developer.android.com/training/maps
- https://developers.google.com/maps/documentation/android-sdk/start
- https://developers.google.com/maps/documentation/android-sdk/get-api-key#restrict_key
- https://developers.google.com/maps/documentation/android-sdk/config
- https://developers.google.com/maps/documentation/android-sdk/map
- https://developers.google.com/maps/documentation/android-sdk/polygon-tutorial