Understanding and Using Services in Android: Background & Foreground Services

Codeible · Follow
5 min read · Oct 2, 2021

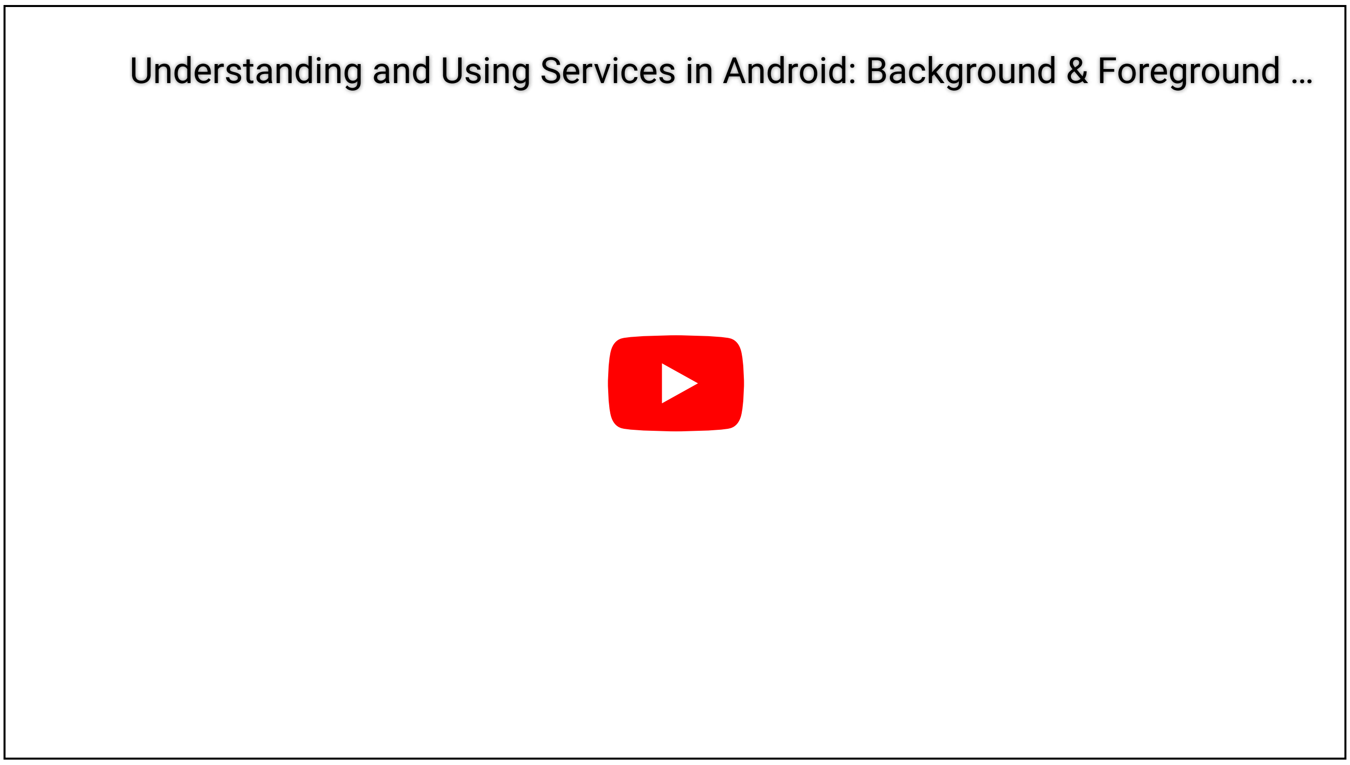
Listen Share More

Hello, in this article, I will show you how to utilize Background and Foreground Services in Android.

Follow on Twitter: Codeible (@codeible) / Twitter.

View article on Codeible.com.

Understanding and Using Services in Android: Background & Foreground Services
Hello, in this tutorial, I will show you how to utilize Background and Foreground Services in Android. There are 3...
codeible.com


Understanding and Using Services in Android: Background & Foreground ...

There are 3 types of services in Android:

(1) Background

(2) Foreground

(3) Bound

Each of these terms are misleading because it is not describing the behavior of how each service are used, but it is describing how they are terminated.

```
For Android Developers, a Service is a component that runs on the
background to perform long-running tasks.
```

> *A **Background Service** is a service that runs only when the app is running so it'll get terminated when the app is terminated.*
>
> *A **Foreground Service** is a service that stays alive even when the app is terminated.*
>
> *And a **Bound Service** is a service that runs only if the component it is bound to is still active.*

Let's see how we can create a Background Service.

Creating a Background Service

To create a Background Service, (1) create a new Class and have it extend the Service class. (2) Inside the class, override the onBind() and onStartCommand() methods.

```java
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;

import androidx.annotation.Nullable;

public class MyBackgroundService extends Service {
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {

        return super.onStartCommand(intent, flags, startId);
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

MyBackgroundService.java hosted with ❤ by GitHub view raw

The onStartCommand() method is called every time we start the service either by calling startService() or startForegroundService(). This is where we want to define what the service will do.

For now, we'll create a thread that'll print a message to the terminal every 2 seconds.

```java
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    new Thread(
            new Runnable() {
                @Override
                public void run() {
                    while (true) {
                        Log.e("Service", "Service is running...");
                        try {
                            Thread.sleep(2000);
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                }
            }
    ).start();
    return super.onStartCommand(intent, flags, startId);
}
```

MyBackgroundService.java hosted with ❤ by GitHub view raw

Adding the service to the app

Now that we have our Background Service, we need to let the app know about this service.

Go to the manifests file. Inside the application element, add a service element and use the **android:name** attribute to add the service to the app.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.codeible.servicestutorial">
    <application
        ...>
            ...
        </activity>

        <service android:name=".MyBackgroundService"></service>

    </application>

</manifest>
```

background-service.xml hosted with ❤ by GitHub view raw

Starting the Service

```
@Override

protected void onCreate(Bundle savedInstanceState) {

   super.onCreate(savedInstanceState);

   setContentView(R.layout.activity_main);

   Intent serviceIntent = new Intent(this,
       MyBackgroundService.class);

}
```

Call startService() and pass in the intent.

```
startService(serviceIntent);
```

If we run the app and look at the terminal, a message will be displayed every 2 seconds.

If we minimize the app, the service will continue to run.

If we terminate the app, it will stop.

Creating Foreground Services

Now let's see how we create a Foreground Service. Create a new class and have it extend the Service class.

Override the onBind() and onStartCommand() methods.

```java
import android.app.Service;
import android.content.Intent;
import android.os.IBinder;
import androidx.annotation.Nullable;

public class MyForegroundService extends Service {
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        return super.onStartCommand(intent, flags, startId);
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

MyForegroundService.java hosted with ❤ by GitHub view raw

Inside the onStartCommand() method, copy the code from the BackgroundService and paste it inside.

```java
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    new Thread(
            new Runnable() {
                @Override
                public void run() {
                    while (true) {
                        Log.e("Service", "Service is running...");
                        try {
                            Thread.sleep(2000);
                        } catch (InterruptedException e) {
                            e.printStackTrace();
                        }
                    }
                }
            }
    ).start();
    return super.onStartCommand(intent, flags, startId);
}
```

MyBackgroundService.java hosted with ❤ by GitHub view raw

Go to the manifests file and add the service to the app.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.codeible.servicestutorial">
    <application
        ...>
            ...
        </activity>

        <service android:name=".MyForegroundService"></service>

    </application>

</manifest>
```

ForegroundService.xml hosted with ❤ by GitHub view raw

In order for us to use Foreground Services, we need to add the FOREGROUND_SERVICE permission in the manifests file to enable it.

```
<uses-permission
android:name=”android.permission.FOREGROUND_SERVICE”></uses-
permission>
```

Starting the Foreground Service

Now we need to start the service.

Go to the MainActivity file. In the onCreate() method, replace the BackgroundService class with the ForegroundService class for the intent.

Instead of calling startService(), call startForegroundService().

```
Intent serviceIntent = new Intent(this, MyForegroundService.class);
startForegroundService(serviceIntent);
```

If we run the app, the service should start. However, if we terminate the app, it will stop.

This is because we have not put the service in the Foreground yet. Calling startForegroundService() only starts the service, we still need to put it in the Foreground state.

Go back to the Foreground Service class. In the onStartCommand() method, call startForeground().

```
public int onStartCommand(Intent intent, int flags, int startId) {
   ...
   startForeground(ID, NOTIFICATION);
   return super.onStartCommand(intent, flags, startId);
}
```

It takes 2 arguments. An id for the notification and the notification itself.

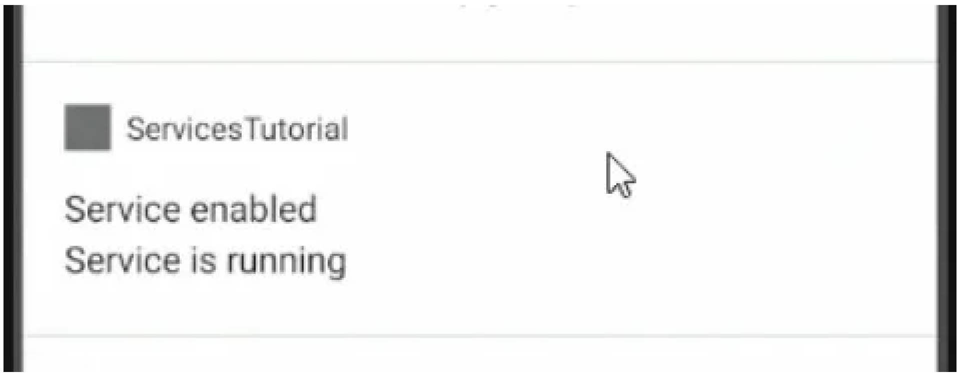
The reason we need to provide a notification for a Foreground Service is because we need to let the user know that there is a service from our app that is running even when the app is terminated. The notification cannot be removed until the service is terminated.

Pass in an ID and a notification object.

```java
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    new Thread(
            new Runnable() {
                @Override
                public void run() {
                    while (true) {
                        Log.e("Service", "Service is running...");
                        try {
                            Thread.sleep(2000);
                        } catch (InterruptedException e) {
                             e.printStackTrace();
                        }
                    }
                }
            }
    ).start();

    final String CHANNELID = "Foreground Service ID";
    NotificationChannel channel = new NotificationChannel(
            CHANNELID,
            CHANNELID,
            NotificationManager.IMPORTANCE_LOW
    );

    getSystemService(NotificationManager.class).createNotificationChannel(channel);
    Notification.Builder notification = new Notification.Builder(this, CHANNELID)
            .setContentText("Service is running")
            .setContentTitle("Service enabled")
            .setSmallIcon(R.drawable.ic_launcher_background);

    startForeground(1001, notification.build());
    return super.onStartCommand(intent, flags, startId);
}
```

MyForegroundService.java hosted with ❤ by GitHub view raw

If we run the app and terminate the app, the service should still be running and we'll see the notification in the notification tray.


ServicesTutorial

Service enabled
Service is running

Although it is working as intended, we still need to do one more thing to properly handle Foreground Services. Because they stay alive even after the app is terminated, whenever the app gets relaunched, another service will be created, and we'll end up with multiple of the same service running at the same time.

To stop this from happening, every time we want to start the service, we need to check if it is already running first.

Go back to the MainActivity file. Create a Boolean method call foregroundServiceRunning() and have it return false by default.

```java
public boolean foregroundServiceRunning(){
    return false;
}
```

MainActivity.java hosted with ❤ by GitHub view raw

Inside the method get a reference to the ActivityManager using getSystemService().

Then use a For Loop and loop through all the active services that are running for the app.

During each iteration, check if there is a service that matches our Foreground Service. If there is, return true.

```java
public boolean foregroundServiceRunning(){
    ActivityManager activityManager = (ActivityManager) getSystemService(Context.ACTIVITY_SERVIC
    for(ActivityManager.RunningServiceInfo service: activityManager.getRunningServices(Integer.M
        if(MyForegroundService.class.getName().equals(service.service.getClassName())) {
            return true;
        }
    }
    return false;
}
```

MainActivity.java hosted with ❤ by GitHub view raw

In the onCreate() method, check if the service is not running first. If it is not running, we want to start the service.

```java
if(!foregroundServiceRunning()) {
   Intent serviceIntent = new Intent(this,
     MyForegroundService.class);
   startForegroundService(serviceIntent);
}
```

If we run the app, and then terminate it, and then relaunch the app, it'll not start the service again.

Restarting the Foreground Service on Reboot

Sometimes you may want to restart a Foreground Service when the user reboots the system. We can achieve this by using a BroadcastReceiver.

The purpose of the Broadcast Receiver is to send or receive messages from the Android System.

When the user reboots their device, Android will send out a message telling everyone that the system was rebooted. We need to create a BroadcastReceiver to receive that message so we can restart Foreground Service.

To create a BroadcastReceiver, create a new class and have it extend the BroadcastReceiver class.

Open in app ↗

```java
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;

public class MyBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {

    }
}
```

MyBroadcastReceiver.java hosted with ❤ by GitHub view raw

Now that we have the BroadcastReceiver class, we need to add the BroadcastReceiver in the manifests file to let the app know about it.

Go to the Manifests file.

(1) Add a receiver element and use the android:name attribute to add the receiver.

(2) Inside the receiver, add an intent filter element to let the app will know what the receiver will be listening for.

(3) Add the BOOT_COIMPLETED action so it'll listen for when the app is rebooted.

(4) When that is done, enable the RECEIVE_BOOT_COMPLETE permission.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.codeible.servicestutorial">
    <uses-permission android:name="android.permission.FOREGROUND_SERVICE"></uses-permission>
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"></uses-permission
    <application
        ...>
        <activity android:name=".MainActivity">
            ...
        </activity>
        ...
        <receiver android:name=".MyBroadcastReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"></action>
            </intent-filter>
        </receiver>

    </application>

</manifest>
```

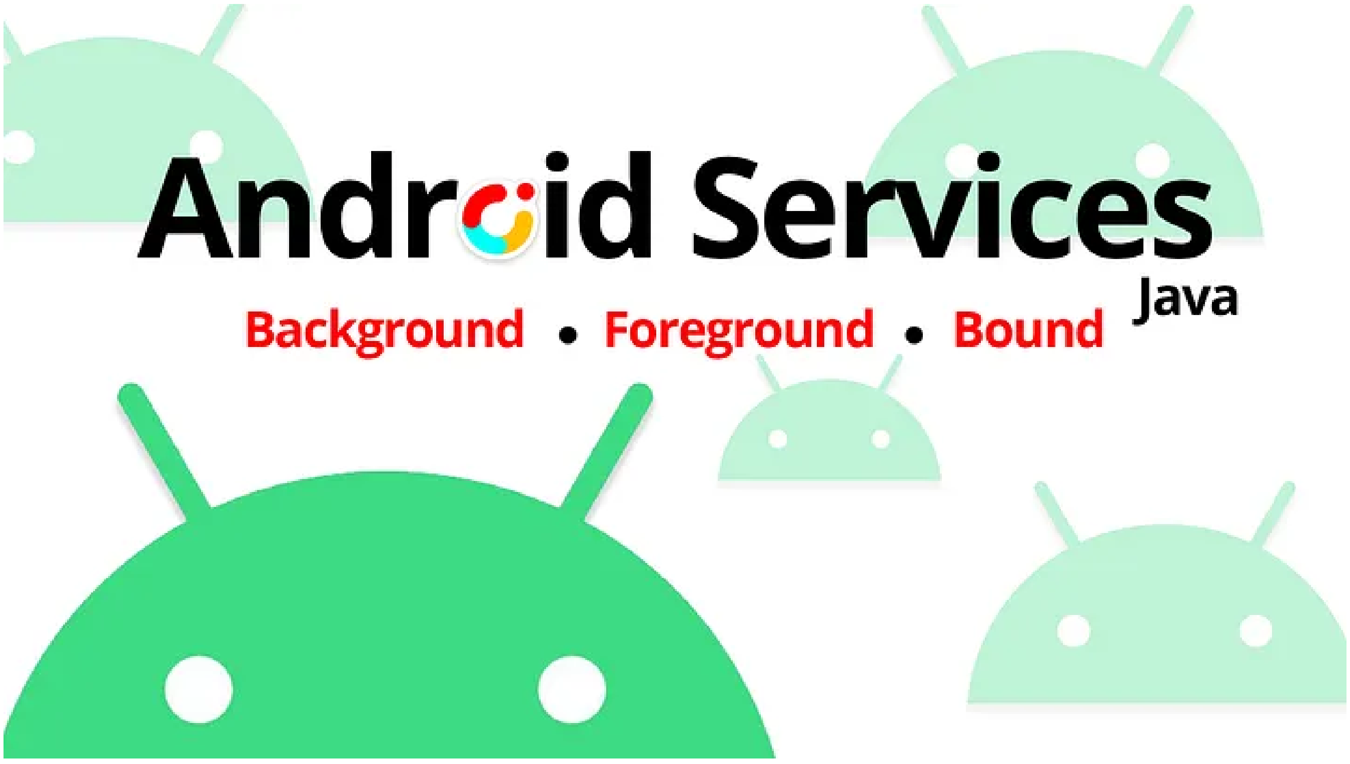
broadcast-receiver.xml hosted with ❤ by GitHub view raw

Now go back to the BroadcastReceiver class. In the onReceive() method, check if the action we received from the intent is equal to ACTION_BOOT_COMPLETED. If it is, we want to start the Foreground Service.

```java
@Override
public void onReceive(Context context, Intent intent) {
   if(intent.getAction().equals(Intent.ACTION_BOOT_COMPLETED)) {
    Intent serviceIntent = new Intent(context,
      MyForegroundService.class);
    context.startForegroundService(serviceIntent);
   }
}
```

If we run the app and then restart the device the service should start up automatically on its own.

Click here to download the project.



Codeible.com Image

Understanding and Using Services in Android: Background & Foreground Services

Hello, in this tutorial, I will show you how to utilize Background and Foreground Services in Android. There are 3...

codeible.com

Android App Development | AndroidDev | Java | Codeible | Android

Follow

Written by Codeible

121 Followers

Spreading Coding Knowledge to Viewers

More from Codeible

Codeible

Adding, Loading, and Using JavaScript in Angular

Learn how to load and use JavaScript code inside your Angular project

3 min read · Sep 26, 2022

51 2

Android
android
Notifications
with Firebase Cloud Messaging
Android Notifications
Like, Share, and Subscribe
Credit Card Transactions in Angular with PayPal
Free
The Complete Angular Course

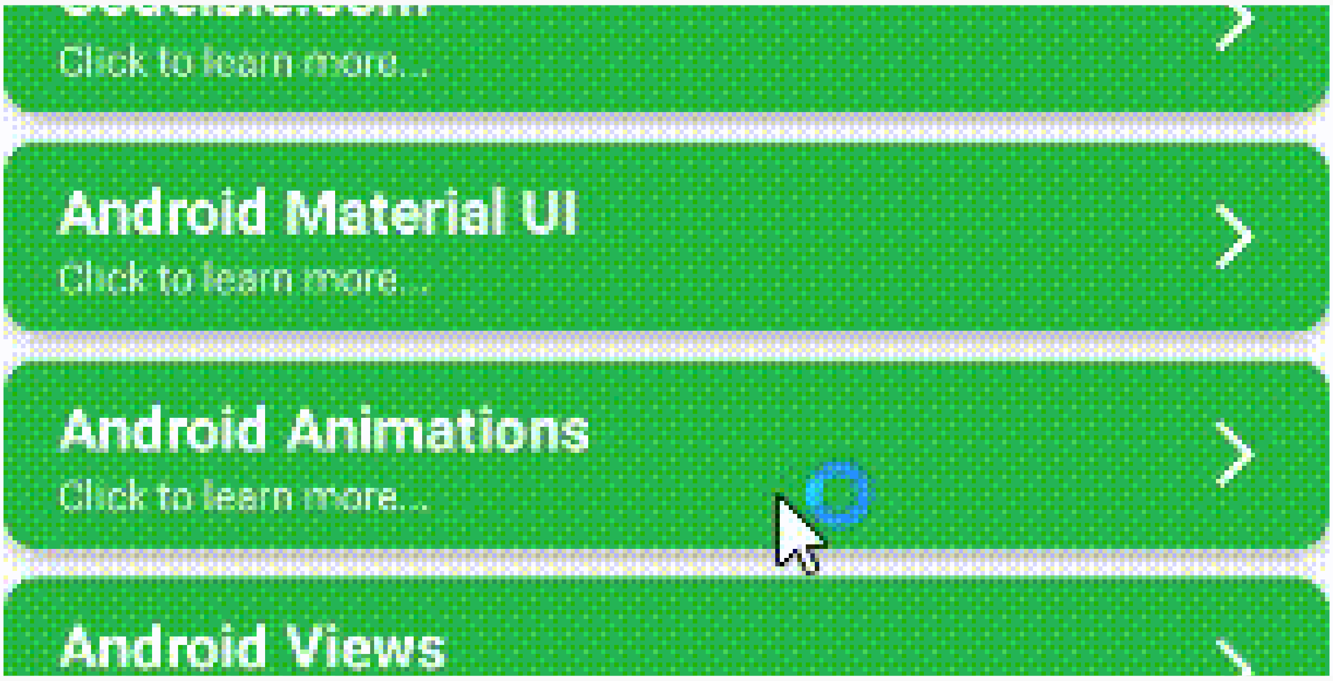
Codeible

Android Notifications with Firebase Cloud Messaging

By King (Codeible.com)

6 min read · Aug 9, 2021

124 1



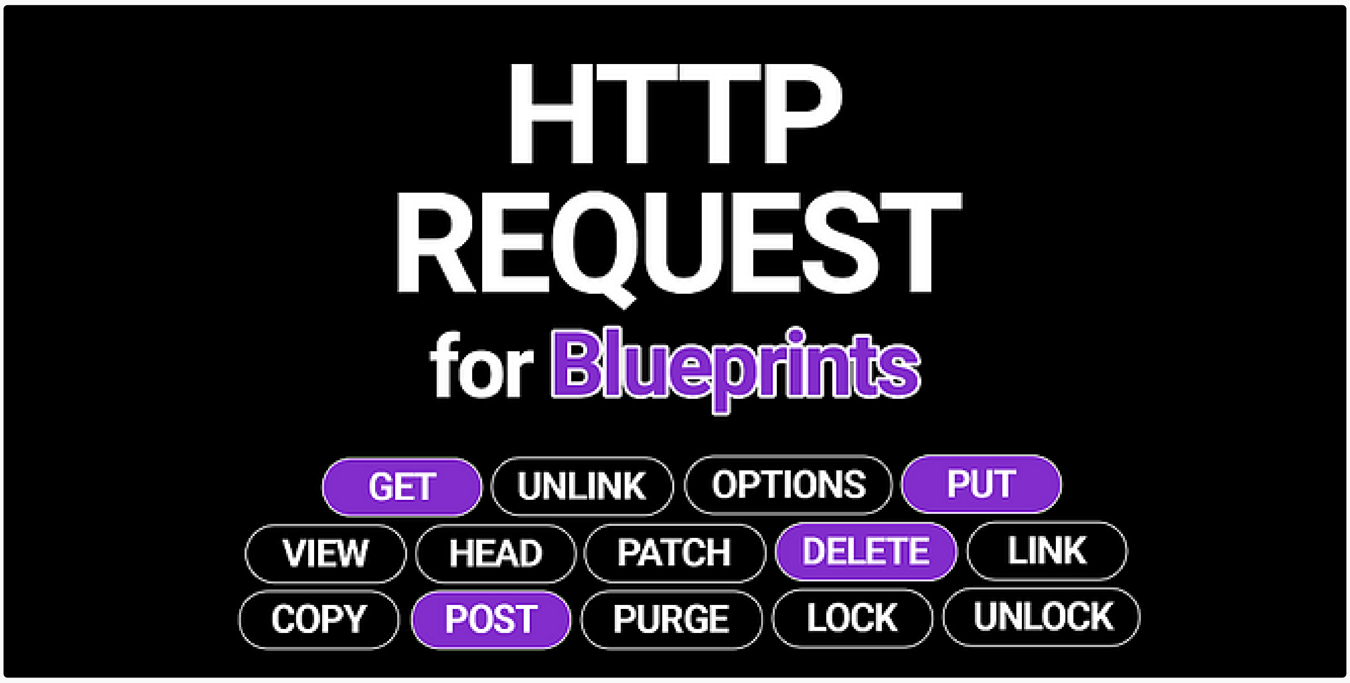
Codeible

Swipe or Slide and Drag and Drop items in RecyclerView

Hello, in this tutorial, I'll be showing you how to add swipe and drag gesture events to a RecyclerView.

5 min read · Aug 25, 2022

37 1



Codeible

Making a HTTP Request in Unreal Engine Blueprints using One Node

Hello, in this tutorial, I will show you how to make http requests in Unreal Engine using the HTTP Requests for Blueprints plugin with one...

4 min read · Mar 12

41

See all from Codeible

Recommended from Medium

Abdellah Ibn El Azraq

How to use WorkManager with Hilt in Android

In the fast-paced world of mobile app development, managing background tasks effectively is a crucial aspect of creating robust and…

3 min read · Oct 8

13

Kirill Rozov in ProAndroidDev

What’s new in Android 14 for developers

Review of most important changes in API and new features in Android 14 that developers need to adopt

21 min read · Oct 5

1.97K 11

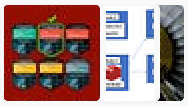
Lists

General Coding Knowledge

20 stories · 645 saves

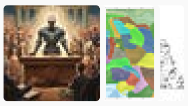
Medium Publications Accepting Story Submissions

155 stories · 1175 saves



New_Reading_List

174 stories · 215 saves



Staff Picks

529 stories · 494 saves

Manu Aravind

How to secure Api key in Android

Securing API keys in Android is important to prevent unauthorized access to your APIs and sensitive information. Here are some approaches...

1 min read · Aug 24

3 1

Benoit Ruiz in Better Programming

Advice From a Software Engineer With 8 Years of Experience

Practical tips for those who want to advance in their careers

22 min read · Mar 21

12.5K 241

Mukul Jangir in Stackademic

Best Practises In ViewModel

Why do we need ViewModel?

5 min read · Oct 30

452 10

Abhishek Pathak

Secured Shared Preferences in Android

In Android app development, data storage is a crucial aspect to consider. One of the most commonly used methods to store small amounts of...

4 min read · Jul 28

50

See more recommendations