# Android example

## Table of Contents

# 1. Prerequisite

- Android Studio ➔ Installation: https://developer.android.com/studio/install
- Install JAVA 11 ➔ Installation: JAVA 11
- Install Android 31 API on Android Studio

# 2. API Android

Kalima provides an API for Android in the form of an aar library, which allows you to launch a service that connects to the blockchain, and that provides the necessary elements to communicate with this service.

We can also according to our needs, follow the Java Tutorial and simply use the Kalima jar, but the advantage of a service is that it can run in the background, and therefore allow the reception of data even when the Android application is closed, to be able for example to receive notifications.

We will see during this document the different steps to follow to start its service to connect to a Kalima blockchain.

Noted that your project must be in Android X to work with this service.

## a. Adding the library into the project

To get started, you need to import the Kalima android service library into your project.

Place kalima-Android-lib-Release.aar in a folder " libs " in your module folder. For example, if your module is called app, place the library in app/libs/. Then in the Gradle of your module add:

```
repositories {
    flatDir {
        dirs 'libs'
    }
}
```

As well as:

```
dependencies {
    implementation (name: 'kalima-android-lib-release', ext: 'aar')
}
```

## b. Kalima Service Preferences

KalimaServicePreferences is an object that will allow us to configure a number of elements via the shared preferences of Android, it is the equivalent of the configuration file that we use with the Kalima jar.

Below is an example of a configuration:

```
// Create android preferences to configure KalimaService for this app
KalimaServicePreferences kalimaServicePreferences =
KalimaServicePreferences.getInstance(getApplicationContext(), APP_NAME);
kalimaServicePreferences.setNodeName("ExampleNode");
// You need to choose an unused port
kalimaServicePreferences.setServerPort(9100);
// Notaries address and ports (ipAddress:port), separated with ","
kalimaServicePreferences.setNotariesList("62.171.131.154:9090,62.171.130.233:9090
,62.171.131.157:9090,144.91.108.243:9090");
// Set list of addresses we want for notifications
kalimaServicePreferences.setNotificationsCachePaths(new
ArrayList<String>(Arrays.asList("/StAubin/Underfloor_1/alarms")));
// Set class path of notification receiver
// You will receive broadcast in this receiver when new data arrives in Addresses
you choose with setNotificationsCachePaths
// Then you can build a notification, even if the app is closed
kalimaServicePreferences.setNotificationsReceiverClassPath("org.kalima.kalimaandr
oidexample.NotificationReceiver");
```

Let's see the usefulness of the different configurations:

- setNodeName➔ Use name that allows you to recognize your node
- setServerPort➔ This port is not normally used on client nodes. You can choose any port
- setNotariesList➔ Allows you to configure the list of addresses and ports of the Master Nodes on which you want to connect, in the form ip:port, separated by commas
- setNotificationsCachePath➔ Allows you to choose the addresses on which you want to be notified (this principle will be explained later)
- setNotificationsReceiverClassPath➔ Allows you to tell the service the path of the class that will handle notifications (this principle will be explained later)

### c. Kalima Cache Callback

The object KalimaCacheCallback will allow us to react in an event-driven way to interactions with the blockchain. Below is an example of instantiation:

```
kalimaCacheCallback = new KalimaCacheCallback.Stub() {
  @Override
  // Callback for incoming data
  public void onEntryUpdated(String cachePath, KMsgParcelable kMsgParcelable)
  throws RemoteException {
    Log.d("onEntryUpdated", "cachePath="  + cachePath + ", key=" +
    kMsgParcelable.getKey());
  }

  // Callback for deleted data
  @Override
  public void onEntryDeleted(String cachePath, String key) throws RemoteException
  {
    Log.d("onEntryDeleted", "cachePath="  + cachePath + ", key=" + key);
  }

  @Override
  public void onConnectionChanged(int i) throws RemoteException {
  }
};
```

As we can see, 3 callbacks are available:

- onEntryUpdated➔ This callback will be called whenever data is added /modified in cache. The callback has two parameters: The address that allows you to know where the address the data arrived, and the message in the form of KMsgParcelable (which allows you to retrieve the information of the message). An easy example of using this Callback would be updating a message list. A message has been added or changed, so you update your list.
- onEntryDeleted➔ This callback will be called every time a message is deleted in cache. It has two settings: The address on which the message was deleted, and the key of the deleted message. A simple example of using this callback would be updating a message list. A message has been deleted, so you'llkeep it off your display list.
- onConnectionChanged➔ This callback will be called each time a disconnection or reconnection occurs with one of the Master Nodes. This Callback can for example be useful to warn the user that he is no longer connected to the Blockchain, or that one of the notary no longer responds.

To work, we must switch to the service the implementation of our Callback, when connecting our activity and the service (which will be detailed just after), with the help of the following line:

```
kalimaServiceAPI.addKalimaCacheCallback(kalimaCacheCallback);
```

Similarly, when our activity is no longer in the foreground, we will want to delete the implementation of the callback, thanks to the following line, which we can for example use in the function onPause of the activity life cycle:

```
kalimaServiceAPI.unregisterKlimaCacheCallback(kalimaCacheCallback);
```

### d. Connection between an activity and the service

Before you can communicate with the service from an activity, you have to "connect" them together. To do this, it is necessary to call bindService at the start of your activity, in the function onResume Like what:

```
Intent intent = new Intent(this, KalimaService.class);
intent.putExtra(KalimaService.APP_NAME, APP_NAME);
bindService(intent, serviceConnection, BIND_AUTO_CREATE);
```

This, while having previously instantiated serviceConnection :

```
serviceConnection = new ServiceConnection() {

  @Override
  public void onServiceConnected(ComponentName name, IBinder service) {
    kalimaServiceAPI = KalimaServiceAPI.Stub.asInterface(service);

    try {

      kalimaServiceAPI.addKalimaCacheCallback(kalimaCacheCallback);

    } catch (RemoteException e) {
      e.printStackTrace();
    }

  }

  @Override
    public void onServiceDisconnected(ComponentName name) {
  }
};
```

The onServiceConnected function will be called when your activity is well connected to the service. It is then in this function that we must initialize kalimaServiceAPI (which we will look into next), and add our possible callback(s). You can also add code at that time, to initialize a display list with the data cached at that time for example.

### e. Kalima Service API

The KalimaServiceAPI object will make it possible to communicate with the service, to interact with the blockchain. Let's see the main functions it contains:

- get(String address, String key) Returns the message with the key "key" ➔present at the address address, via an object of type KMsgParcelable
- getAll(String address, boolean reverseDirection, KMsgFilter) Returns an ArrayList of KMsgParcelable of ➔all messagespresent at the address address. You can change the order of the arrayList if necessary via "reverseDirection", and add a filter via KMSgFilter.
- set(String address, String key, byte[] body, String tll) Allows you to create a transaction on the blockchain. The TTL (Time To Live) allows you to choose the lifetime of the message, in seconds (-1 so that the message is always active)➔
- delete( String address, String key) Creates a transaction on the blockchain to delete the "key" message ➔at the "address" address.

### f. Starting the service

Once started, the service does not stop. The best is to start it:

- When starting the phone thanks to a BoradcastReceiver➔ https://www.dev2qa.com/how-to-start-android-service-automatically-at-boot-time/
- At the launch of your application, that is to say in your first activity

Example of starting the service:

```
@Override
protected void onResume() {
Great. onResume();
 yew(permissionsStorage) {
Intent intent = new Intent(this, KalimaService. class);
intent. putExtra(KalimaService. APP_NAME, APP_NAME);
if(! isMyServiceRunning()) {
if (Build. VERSION. SDK_INT >= Build. VERSION_CODES. (O) {
      startForegroundService(intent);
    } else {
      startService(intent);
    }
  }
  bindService(intent, serviceConnection, BIND_AUTO_CREATE);
 }
}
```

And the detail of the function isMyServiceRunning :

```
private boolean isMyServiceRunning() {
 ActivityManager manager = (ActivityManager)
getSystemService(Context. ACTIVITY_SERVICE);
 yew (manager != null) {
for (ActivityManager. RunningServiceInfo service:
manager. getRunningServices(Integer. MAX_VALUE)) {
yew (KalimaService. class. getName(). Equals(
service. service. getClassName())) {
      return true;
    }
  }
 }
return false;
}
```

### g. Notifications

The service is designed so that one can react to transactions even when the application is closed, to create notifications for example. We have seen above that we can choose the addresses for whichwe want to be notified, as well as the class that will manage the notifications.

In fact the service will send a broadcast when a message arrives to one of the addresses who you have chosen. To create notifications, simply create a class that inherits from Broadcast receiver (the class whose path you have passed via KalimaServicePreferences). This class must be declared in the Manifest as follows:

```
<receiver android:name=". NotificationReceiver"
 Android:exported="true">
< >
<!-- always set name to org. kalima. NOTIFICATION_ACTION -->
  <android action:name="org.kalima.NOTIFICATION_ACTION"/>
</intent-filter>
</receiver>
```

Finally, here is an example of a Broadcast implementation Receiver :

```
public class NotificationReceiver extends BroadcastReceiver {

   /*
   With this class, you can launch notifications even if your app is closed.
   To achieve that, you need to declare this receiver in your Manifest file.
   See example in AndroidManifest.xml.
    */

   private final String CHANNEL_ID = "org.kalima.kalimandroidexample.notifications";
   private static int NOTIFICATION_ID = 0;

   @Override
   public void onReceive(Context context, Intent intent) {
KMsgParcelable kMsgParcelable = intent. getParcelableExtra(KalimaService. EXTRA_MSG);
String cachePath = intent. getStringExtra(KalimaService. EXTRA_CACHE_PATH);
Log. d("notification"," "cachePath=" + cachePath + "key=" + kMsgParcelable. getKey());

      You can choose an activity to start, when user click on notification
Intent dialogIntent = New Intent(context, MainActivity. class);
PendingIntent PendingIntent = PendingIntent. getActivity(context, 0, dialogIntent, 0);
      createNotificationChannel(context);

NotificationCompat. Builder builder = New NotificationCompat. Builder(context, CHANNEL_ID)
. setSmallIcon(Android. A. drawable. stat_sys_warning)
            . setContentTitle(cachePath)
. setContentText(kMsgParcelable. getKey())
            . setContentIntent(pendingIntent)
            . setAutoCancel(true)
. setPriority(NotificationCompat. PRIORITY_HIGH);

NotificationManagerCompat notificationManager = NotificationManagerCompat. from(context);
notificationManager. Notify(NOTIFICATION_ID, Builder. Build());
      NOTIFICATION_ID++;
   }

   private void createNotificationChannel(Context context) {
      Create the NotificationChannel, but only on API 26+ because
      the NotificationChannel class is new and not in the support library
yew (Build. VERSION. SDK_INT >= Build. VERSION_CODES. (O) {
         String description = "Alarms channel";
Int importance = NotificationManager. IMPORTANCE_HIGH;
         NotificationChannel channel = New NotificationChannel(CHANNEL_ID, CHANNEL_ID, importance);
channel. setDescription(description);
         Register the channel with the system; you can't change the importance
         or other notification behaviors after this
NotificationManager notificationManager = context. getSystemService(NotificationManager. class);
notificationManager. createNotificationChannel(channel);
      }
   }
}
```

## 3. Running, installing

You can test your app directly on your PC via the Android emulator, test it on your own Android device, or generate an APK to install later on any Android device.

### a. Android Emulator

To create a virtual Android device on your PC, follow this tutorial:
https://developer.android.com/studio/run/managing-avds

You can then test your application on the virtual device:

https://developer.android.com/studio/run/emulator

### b. Installation on a physical device

To install the app directly on your device:

https://developer.android.com/studio/run/device

### c. Generate APK

You can also generate an APK to test your app. You then just need to install this APK on a device to install the app.

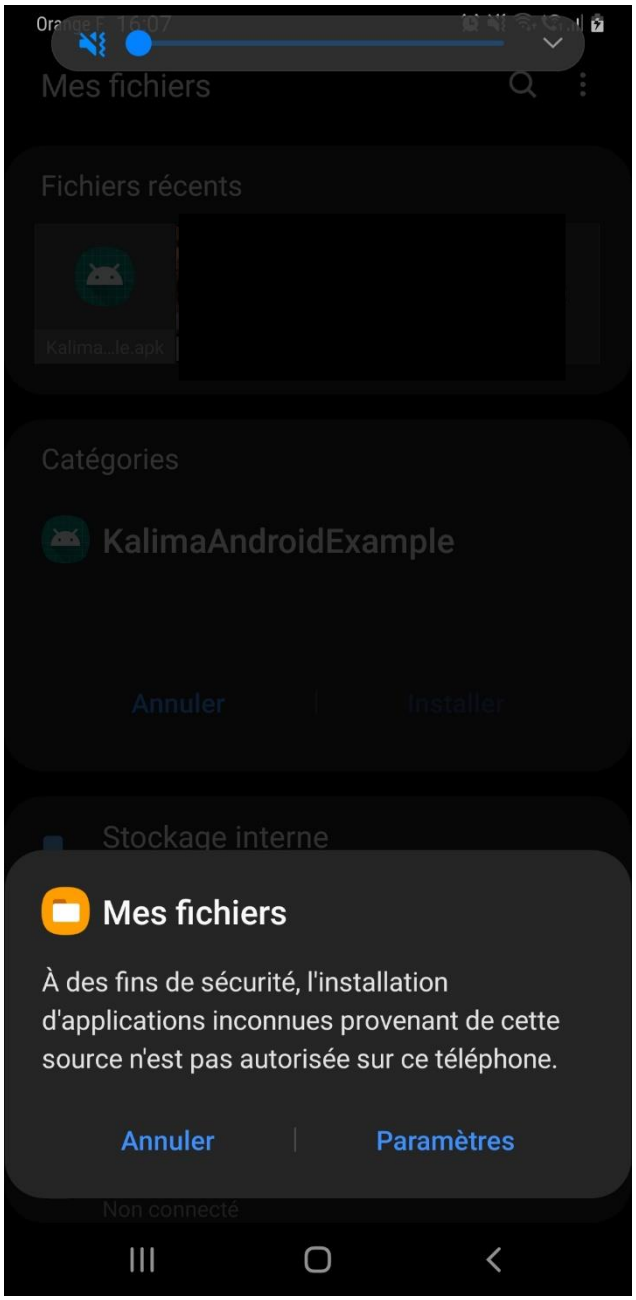
To generate the APK: https://medium.com/@ashutosh.vyas/create-unsigned-apk-in-android-325fef3b0d0a

Then copy the APK to your device, and head to its location via the "My Files" app. Click on the APK to install it. The installation of applications via "My Files" must be allowed. If a message appears regarding this point, click Settings and allow the installation (see screenshots below).

### d. Execution

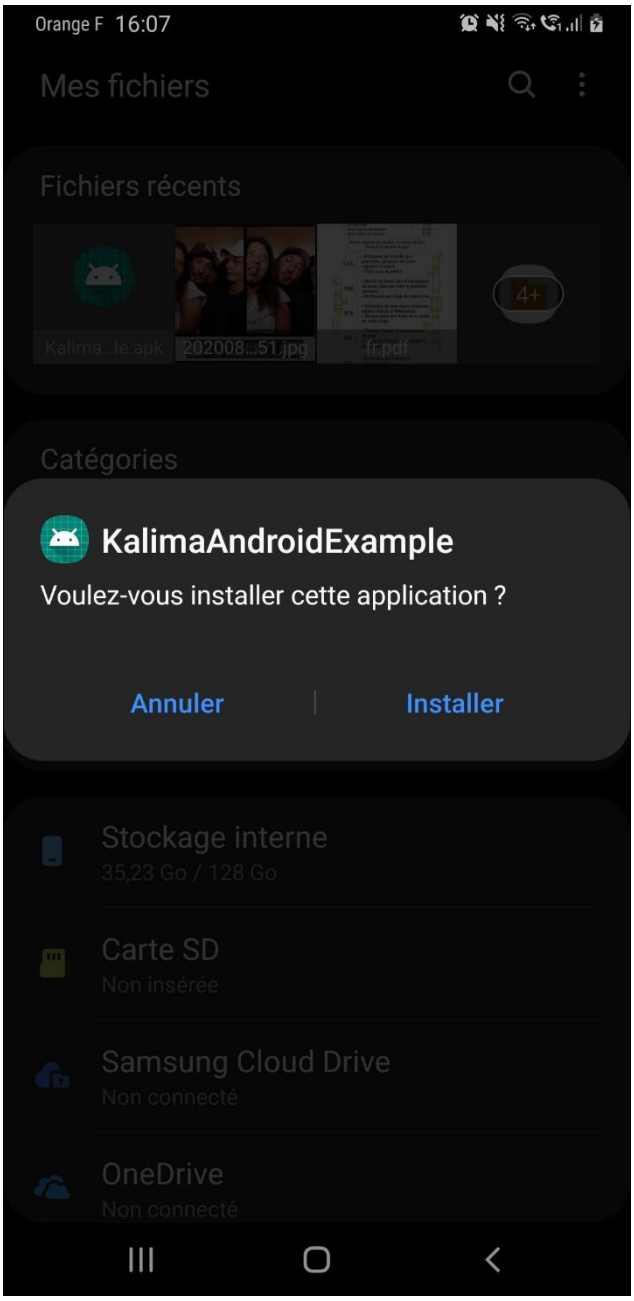
At startup, the application will launch a page to fill in the serialId. You must enter the serialId provided to you by Kalima.

Then we arrive on a simple screen that lists the addresses of the demonstration, namely: /alarms/fire and /sensors. If you click on an address, you can see the current values at that address. On the bottom of the screen, you can send a temperature to the /sensors address, via a text field and a button. With each new transaction on one of the two addresses, the application will create a notification.


Mes fichiers
Fichiers récents
Kalima...le.apk
Catégories
KalimaAndroidExample
Annuler
Installer
Stockage interne
Mes fichiers
À des fins de sécurité, l'installation d'applications inconnues provenant de cette source n'est pas autorisée sur ce téléphone.
Annuler
Paramètres
Non connecté

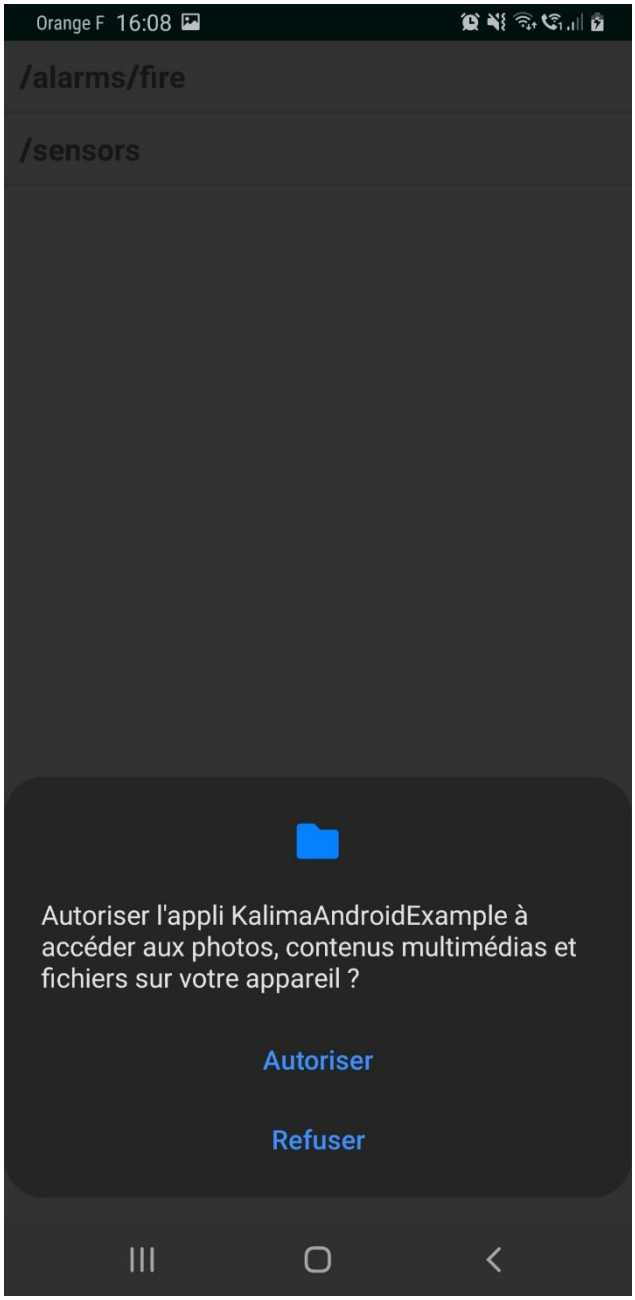

Orange F 16:17
Installation applis inconnues
Mes fichiers
11.5.01.341
Autorisation depuis cette source
L'installation d'applications depuis cette source peut représenter un risque pour votre téléphone et vos données.

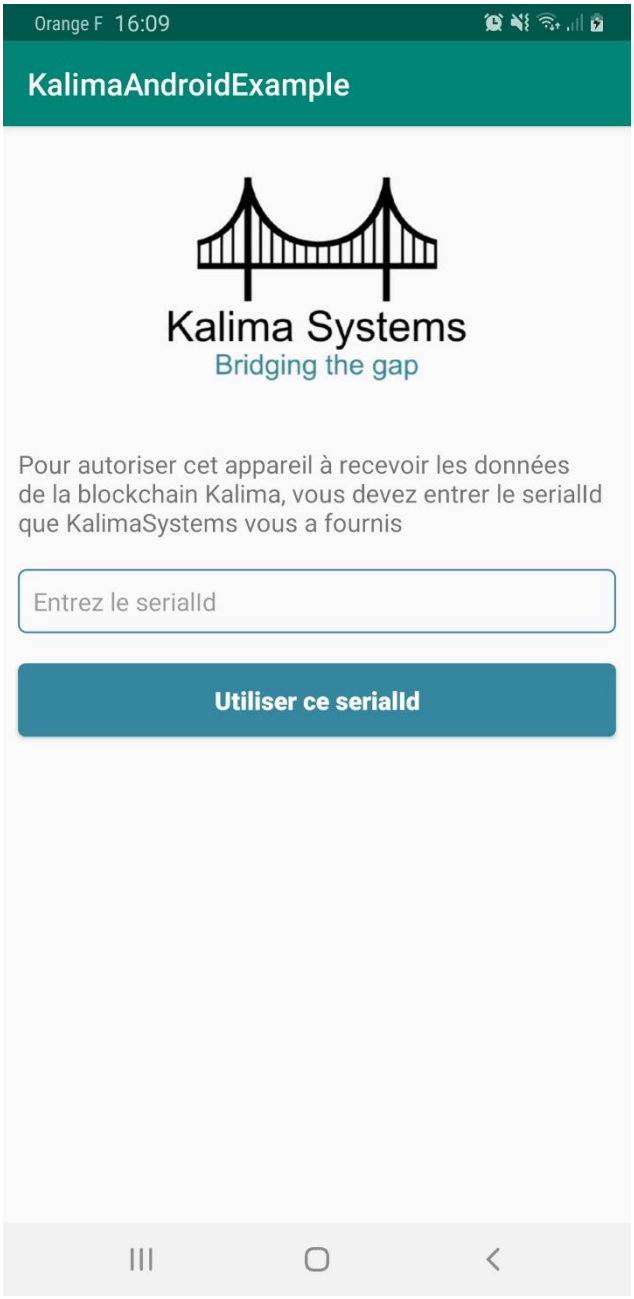
Finally, you can install the application:



### e. How the application works

On first launch, the application asks you for permission to access your files. This permission is necessary for the storage of files useful to Kalima. Once the authorization is accepted, a page will automatically open. Use this page to configure your serialId. The serialId must be provided to you by Kalima, it allows you to authenticate yourself on the Blockchain.



On the app's home page, you can see the list of existing tables (/alarms/fire and /sensors). At the bottom of the page, you can send a text. This text is sent to /alarms/fire with the unique key "key". If you are well authorized on the blockchain, sending and receiving must work. You must receive a notification each time you send. Also, if you click /alarms/fire, you should see the last message received.