Exemple Android

1. Prérequis

- Android Studio ➔ Installation : https://developer.android.com/studio/install

2. API Android

Kalima fournis une API pour Android sous forme d'une librairie aar, qui permet de lancer un service qui se connecte à la blockchain, et qui fournit les éléments nécessaires pour communiquer avec ce service.

On peut également suivant nos besoins, suivre le TutoClient et utiliser simplement le jar Kalima, mais l'avantage d'un service et qu'il peut tourner en tâche de fond, et donc permettre la réception de données même lorsque l'application Android est fermée, pour pouvoir par exemple recevoir des notifications.

Nous allons voir au cours de ce document les différentes étapes à suivre pour mettre en route se service afin de se connecter à une blockchain Kalima.

A noté que votre projet doit être en Android X pour fonctionner avec ce service.

a. Ajout de la librairie dans le projet

Pour commencer, vous devez importer la librairie du service android Kalima dans votre projet.

Placer kalima-android-lib-release.aar dans un dossier « libs » dans le dossier de votre module. Par exemple, si votre module s'appelle app, placer la librairie dans app/libs/. Puis dans le Gradle de votre module ajoutez :

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

Ainsi que :

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

b. Kalima Service Preferences

KalimaServicePreferences est un objet qui va nous permettre de configurer un certain nombre d'éléments via les préférences partagées de Android, c'est l'équivalent du fichier de configuration que l'on utilise avec le jar Kalima.

Ci-dessous un exemple de 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");
// Directory for Kalima files storage
kalimaServicePreferences.setFilesPath(Environment.getExternalStorageDirectory().t
oString() + "/Kalima/jit/KalimaAndroidExample/");
// Path for log directory
kalimaServicePreferences.setLogPath(Environment.getExternalStorageDirectory().toS
tring() + "/Kalima/jit/KalimaAndroidExample/");
kalimaServicePreferences.setLedgerName("KalimaLedger");
// Set list of CachePaths 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
CachesPaths you choose with setNotificationsCachePaths
// Then you can build a notification, even if the app is closed
kalimaServicePreferences.setNotificationsReceiverClassPath("org.kalima.kalimaandr
oidexample.NotificationReceiver");
```

Voyons l'utilité des différentes configurations :

- setNodeName ➔ Utilisez en nom qui vous permet de reconnaître votre nœud
- setServerPort ➔ Ce port n'est en principe pas utilisé sur les nœuds clients. Vous pouvez choisir n'importe quel port
- setNotariesList ➔ Permet de configurer la liste des adresses et ports des Notary Nodes sur lesquels on souhaite se connecter, sous la forme ip:port, séparés par des virgules
- setFilesPath ➔ Permet de choisir où seront stockés sur l'appareil les fichiers nécessaires à Kalima, ainsi que les logs éventuels
- setNotificationsCachePath ➔ Permet de choisir les caches paths sur lesquels on souhaite être notifiés (ce principe sera expliqué plus tard)
- setNotificationsReceiverClassPath ➔ Permet d'indiquer au service le chemin de la classe qui va gérer les notifications (ce principe sera expliqué plus tard)

c. Kalima Cache Callback

L'objet KalimaCacheCallback va nous permettre de réagir de manière évènementielle aux interactions avec la blockchain. Ci-dessous, un exemple d'instanciation :

```
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 {
  }
};
```

Comme on peut le voir, 3 callbacks sont disponibles :

- onEntryUpdated ➔ Ce callback sera appelé a chaque fois qu'une donnée sera ajoutée / modifiée en mémoire cache. Le callback a deux paramètres : Le cachePath qui vous permet de savoir sur quel cache la donnée est arrivée, est le message sous forme de KMsgParcelable (qui vous permet de récupérer les infos du message). Un exemple simple d'utilisation de ce Callback serait la mise à jour d'une liste de message. Un message a été ajouté ou modifié, donc vous mettez à jour votre liste.
- onEntryDeleted ➔ Ce callback sera appelé a chaque fois qu'un message sera supprimé en mémoire cache. Il possède deux paramètres : Le cachePath sur lequel le message a été supprimé, ainsi que la clé du message supprimé. Un Exemple simple d'utilisation de ce callback serait la mise à jour d'une liste de message. Un message a été supprimé, donc vous le retiré de votre liste d'affichage.
- onConnectionChanged ➔ Ce callback sera appelé a chaque fois qu'une déconnexion ou une reconnexion a lieu avec l'un des Notary Nodes. Ce Callback peut par exemple être utile pour avertir l'utilisateur qu'il n'est plus connecté à la Blockchain, ou que l'un des notary ne répond plus.

Pour fonctionner, nous devons passer au service l'implémentation de notre Callback, lors de la connexion entre notre activité et le service (qui sera détaillé juste après), a l'aide de la ligne suivante :

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

De même, lorsque notre activité n'est plus en premier plan, on va vouloir supprimer l'implémentation du callback, grâce à la ligne suivante, que l'on peut par exemple utiliser dans la fonction onPause du cycle de vie de l'activité :

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

d. Connexion entre une activité et le service

Avant de pouvoir communiquer avec le service depuis une activité, il faut les « connecter » ensemble. Pour ce faire, il faut appeler bindService au démarrage de votre activité, dans la fonction onResume par exemple :

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

Ceci, tout en ayant au préalable instancié 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) {
  }
};
```

La fonction onServiceConnected sera appelée lorsque votre activité sera bien connectée au service. C'est alors dans cette fonction qu'il faut initialiser kalimaServiceAPI (dont on va parler juste après), et ajouter notre ou nos callbacks éventuel(s). Vous pouvez également ajouter du code à ce moment-là, pour initialiser une liste d'affichage avec les données présentes en mémoire cache à cet instant par exemple.

e. Kalima Service API

L'objet KalimaServiceAPI va permettre de communiquer avec le service, pour interagir avec la blockchain. Voyons les fonctions principales qu'il contient :

- get(String cachePath, String key) ➔ Retourne le message avec la clé « key » présent dans le cache « cachePath », via un objet de type KMsgParcelable
- getAll(String cachePath, boolean reverseDirection, KMsgFilter) ➔ Retourne un ArrayList de KMsgParcelable de tout les messages présent dans le cache « CachePath ». On peut changer l'ordre de l'arrayList si besoin via « reverseDirection », et ajouter un filtre via KMSgFilter.
- set(String cachePath, String key, byte[] body, String tll) ➔ Permet de créer une transaction sur la blockchain. Le TTL (Time To Live) permet de choisir la durée de vie du message, en secondes (-1 pour que le message soit toujours actif)
- delete(String cachePath, String key) ➔ Crée une transaction sur la blockchain pour supprimer le message « key » dans le Cache « cachePath ».

f. Démarrage du service

Une fois démarré, le service ne s'arrête pas. Le mieux est donc de le démarrer :

- Au démarrage du téléphone grâce à un BoradcastReceiver ➔ https://www.dev2qa.com/how-to-start-android-service-automatically-at-boot-time/#:~:text=Start%20Android%20Service%20When%20Boot,android%20monitor%20log%20cat%20console.
- Au lancement de votre application, c'est-à-dire dans votre première activité

Example de démarrage du service :

```
@Override
protected void onResume() {
  super.onResume();
  if(permissionStorage) {
    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);
  }
}
```

Et le détail de la fonction isMyServiceRunning :

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

g. Les autorisations

Pour fonctionner, le service a besoin d'autorisation pour le stockage, il faut donc que votre activité principale demande cette autorisation à l'utilisateur :

```
private void checkPermissions(){
  if(ContextCompat.checkSelfPermission(getApplicationContext(),
  Manifest.permission.WRITE_EXTERNAL_STORAGE)!=
  PackageManager.PERMISSION_GRANTED){
    ActivityCompat.requestPermissions(this, new
    String[]{Manifest.permission.WRITE_EXTERNAL_STORAGE}, 0);
  } else {
    permissionStorage = true ;
  }
}
```

De plus il faut ajouter quelques permissions dans le manifest :

```
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.FOREGROUND_SERVICE" />
```

h. Les notifications

Le service est conçu pour que l'on puisse réagir a des transactions même lorsque l'application est fermée, pour créer des notifications par exemple. On a vu plus haut que l'on peut choisir les cache paths pour lesquels on veut être notifiés, ainsi que la classe qui va gérer les notifications.

En fait le service va envoyer un broadcast lorsqu'un message arrive sur l'un des cache paths qui vous avez choisis. Pour créer des notifications, il suffit donc de créer une classe qui hérite de Brodcast receiver (la classe dont vous avez passez le chemin via KalimaServicePreferences).

Cette classe doit être déclarée dans le Manifest de la manière suivante :

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

Enfin, voici un exemple d'implémentation du Broadcast 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.R.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
        if (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. Exécution, installation

Vous pouvez tester votre application directement sur votre PC via l'émulateur Android, la tester sur votre propre appareil Android, ou encore générer un APK à installer par la suite sur n'importe quel appareil Android.

a. Emulateur Android

Pour créer un appareil Android virtuel sur votre PC, suivre ce tuto : https://developer.android.com/studio/run/managing-avds

Vous pouvez ensuite tester votre application sur l'appareil virtuel :

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

b. Installation sur un appareil physique

Pour installer l'application directement sur votre appareil :

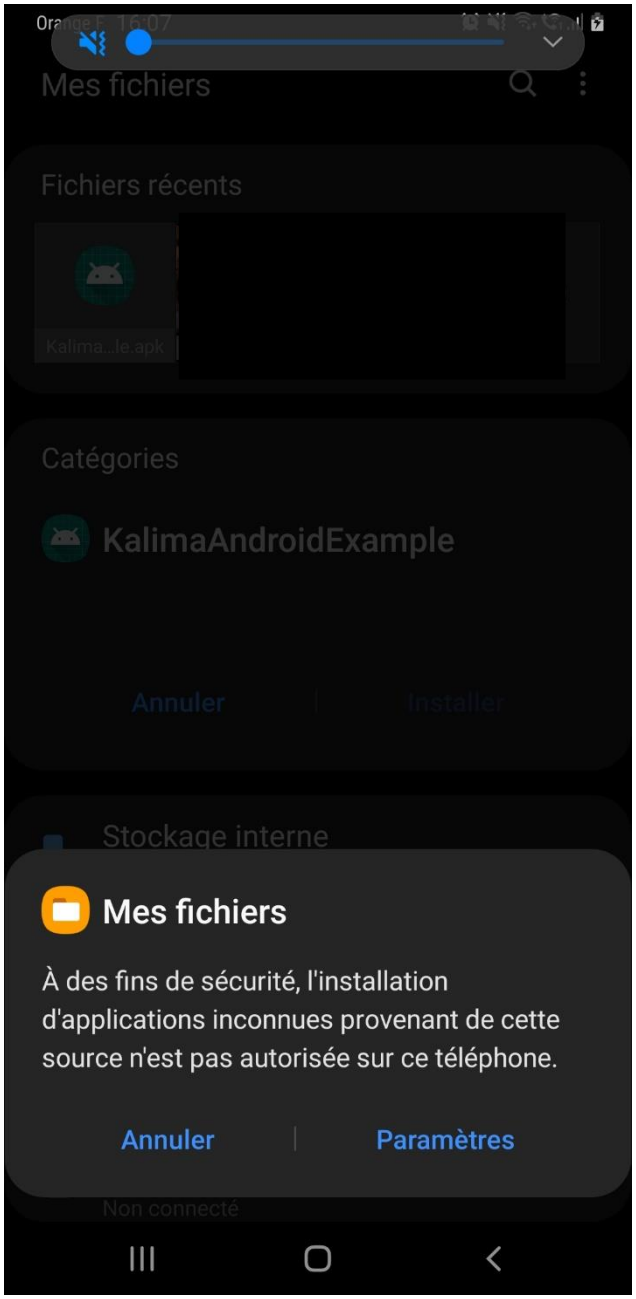
https://developer.android.com/studio/run/device

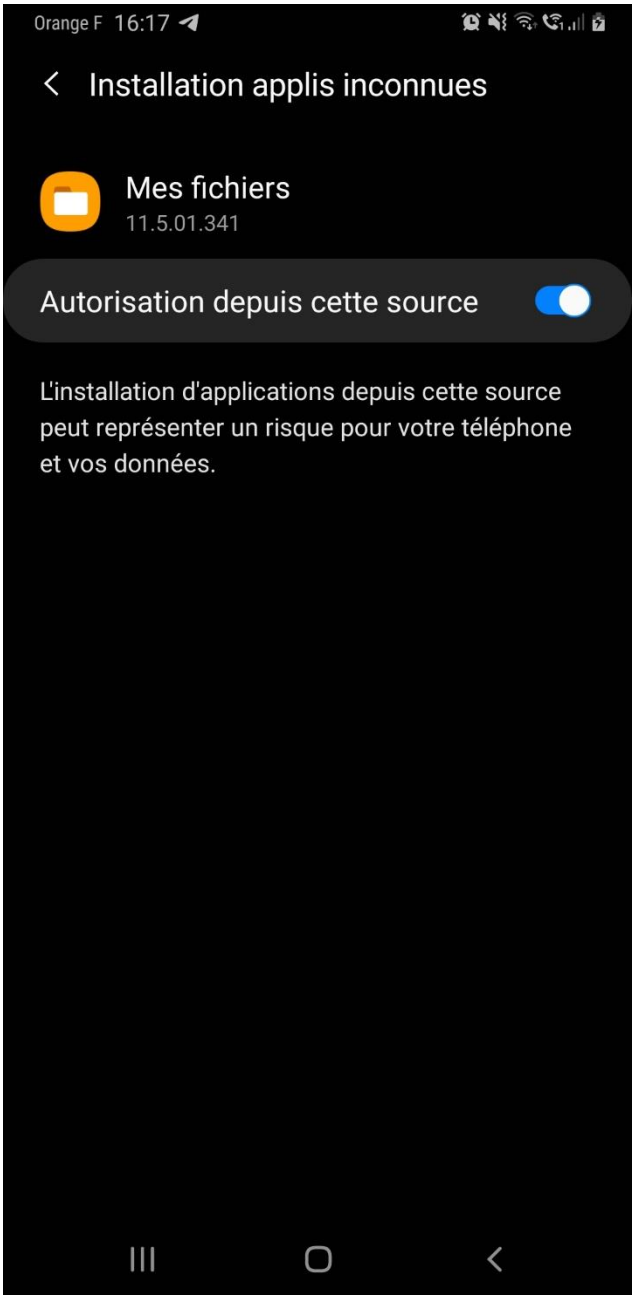
c. Générer l'APK

Vous pouvez également générer un APK pour tester votre application. Il vous suffit alors d'installer cet APK sur un appareil pour installer l'application.

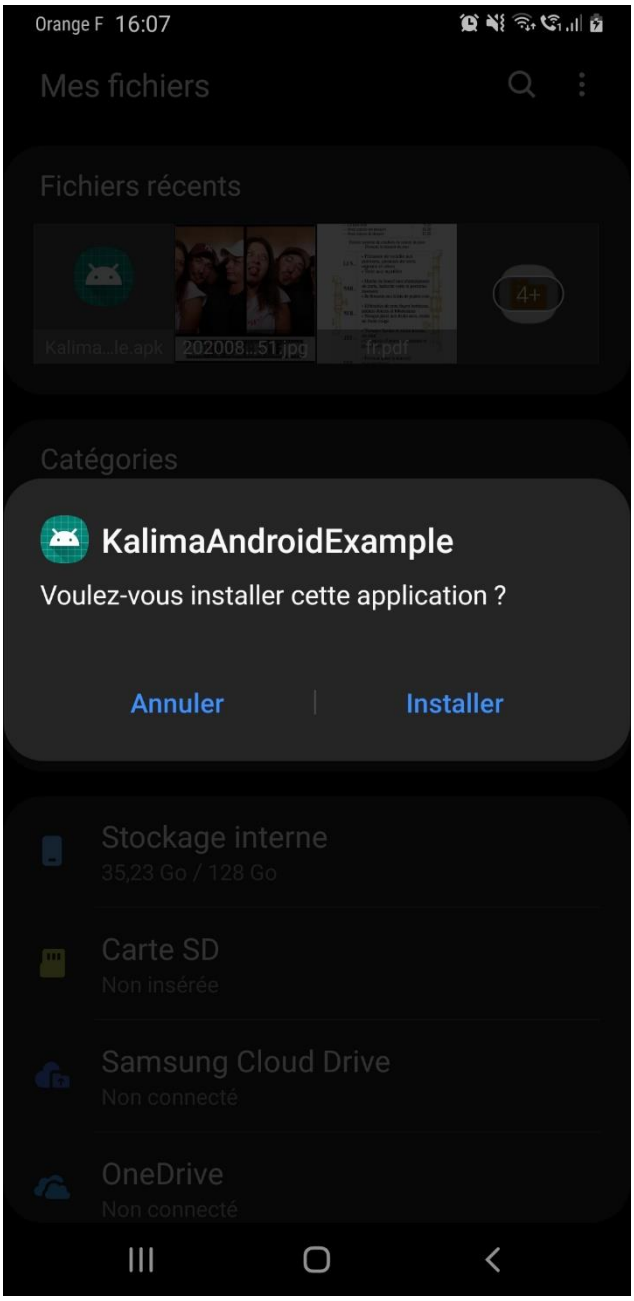
Pour générer l'APK : https://medium.com/@ashutosh.vyas/create-unsigned-apk-in-android-325fef3b0d0a

Copier ensuite l'APK sur votre appareil, et dirigez-vous à son emplacement via l'application « Mes Fichiers ». Cliquez sur l'APK pour l'installer. L'installation d'applications via « Mes Fichiers » doit être autorisée. Si un message apparaît concernant ce point, cliquez sur paramètres et autorisez l'installation (voir captures ci-après).


Mes fichiers
Fichiers récents
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

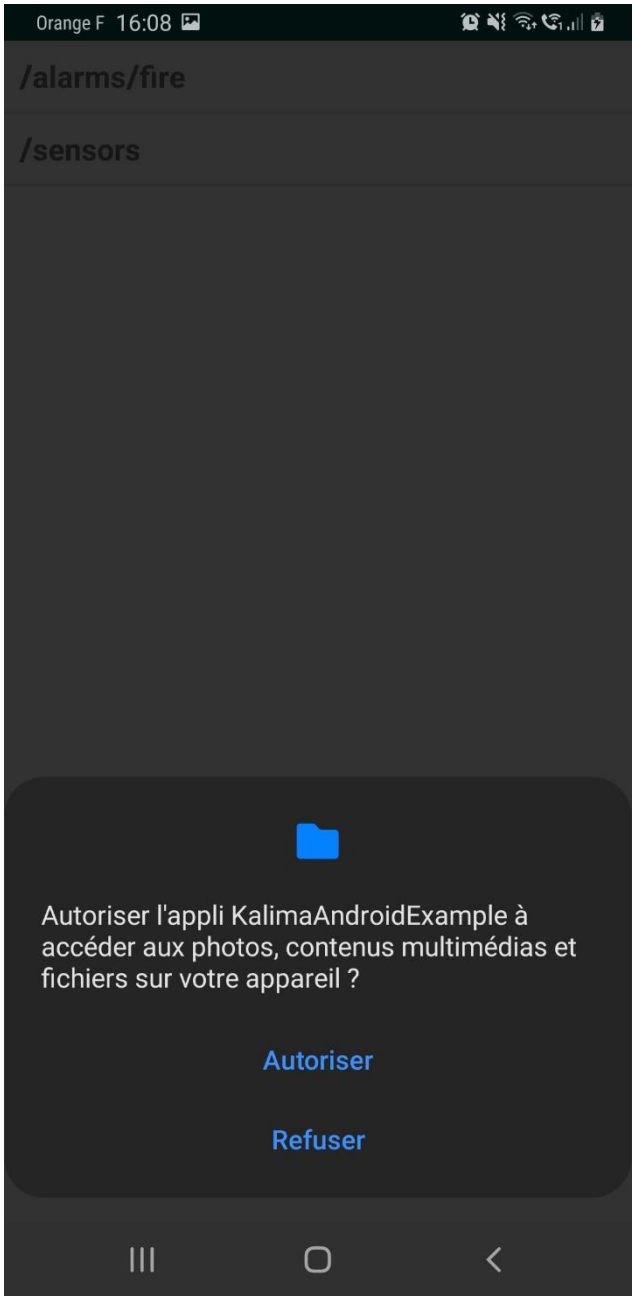

Orange F 16:17
Installation applis inconnues
Mes fichiers
11.5.01.341
Autorisation depuis cette source
L'installation d'applications depuis cette source peut représenter un risque pour votre téléphone et vos données.

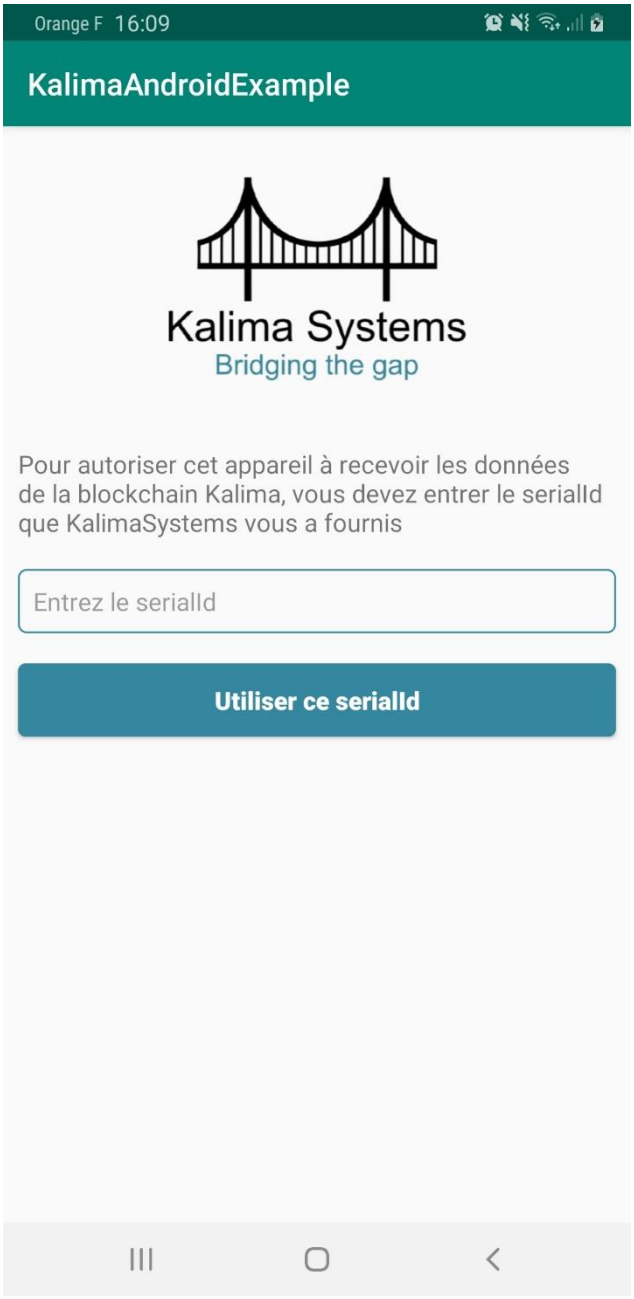
Enfin, vous pouvez installer l’application :



d. Fonctionnement de l'application

Au premier lancement, l'application vous demande l'autorisation d'accès à vos fichiers. Cette autorisation est nécessaire pour le stockage des fichiers utiles à Kalima. Une fois l'autorisation acceptée, une page va automatiquement s'ouvrir. Cette page permet de configurer votre serialId. Le serialId doit vous être fournis par Kalima, il permet de vous authentifier sur la Blockchain.





Sur la page d'acceuil de l'application, vous pouvez voir la liste des tables existantes (/alarms/fire et /sensors). Sur le bas de la page, vous pouvez envoyer un texte. Ce texte est envoyé dans /alarms/fire avec pour clé unique « key ». Si vous êtes bien autorisé sur la blockchain, l'envoi et la réception doivent fonctionner. Vous devez recevoir une notification a chaque envoi. De plus, si vous cliquez sur /alarms/fire, vous devez voir le dernier message reçu.