Smart contrat Android

Prérequis

- Android Studio➔Installation : https://developer.android.com/studio/install
- Installez JAVA 11 ➔ Installation : JAVA 11
- Installez l'API Android 31 sur Android Studio

Pour utiliser l'API Android Kalima, il est recommandé d'avoir préalablement lu la documentation Android Example et d'avoir installé le JDK java dans sa version 11 au minimum.

Un exemple complet de smart contrat Android est disponible sur notre GitHub publique : KalimaAndroidExample.

Afin de créer des smarts Contrats Android, il vous suffit juste l'API **kalima-android-lib-release.aar,** donc vous devriez implémenter cette API dans votre projet afin de pouvoir créer des smarts contrats Java.

1- Smarts contrats ?

Un Smart Contrat, ou contrat intelligent, est un programme informatique qui automatise l'exécution d'un ensemble d'instructions prédéfinies lorsque des conditions préalables sont réunies.

Durant son exécution, toutes les étapes de validation sont enregistrées au niveau de la blockchain. Ces enregistrements assurent la sécurisation de l'ensemble des données en empêchant leur modification ou leur suppression.

Smarts contrats Android == « KalimaCacheCallback » que nous verrons par la suite

Les smarts contrats Android sont sous la forme d'une instance de la classe KalimaCacheCallback. Ce callback est implémenté pour réagir à certains évènements comme la création d'un nouveau cache ou encore l'arrivée de nouvelles transactions.

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 à chaque fois qu'une donnée sera ajoutée / modifiée en mémoire cache. Le callback a deux paramètres : L'adresse qui vous permet de savoir à quelle adresse la donnée est arrivée, et 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 à chaque fois qu'un message sera supprimé en mémoire cache. Il possède deux paramètres : L'adresse sur laquelle 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 à chaque fois qu'une déconnexion ou une reconnexion a lieu avec l'un des Master 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), à 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);
```