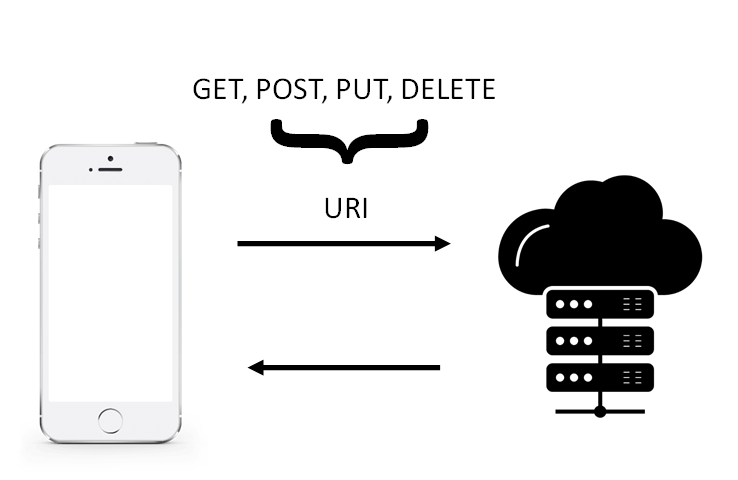
Communiquer avec une API REST

1. Objectif

•Communiquer avec une API REST

•Exploiter les données d’une API REST

1. Présentation



1. Retrofit
   1. Présentation

La plupart des applications mobiles consomment des API REST JSON pour récupérer toutes les données nécessaires à leur fonctionnement.

Retrofit est aujourd’hui une des façons les plus simples d’implémenter des appels à des webservices REST.

Il s’agit d’un client REST. Cette librairie permet d’implémenter plus facilement et rapidement des requêtes réseau sur Android (Java ou Kotlin).

Retrofit nous évite ainsi d’installer manuellement toutes les parties nécessaires à l’exécution d’une requête, comme par exemple la gestion des réponses JSON ou la création d’une [AsyncTask](https://blog.axopen.com/2016/03/les-asynctask/).

Cela permet un gain de temps conséquent et un code plus clair pour des performances équivalentes.

Depuis, Retrofit 2.6 supporte l’utilisation des coroutines. Cela permet de ne plus utiliser de callback ou la fonction enqueue.

* 1. Mise en place

1.Ajouter les dépendances

2.Ajouter les permissions

3.Créer le model

4.Créer le service client

5.Définir le style pour une ligne de liste

6.Définir un adapter​​​​​​​

7.Initialiser et lier les éléments

* 1. Ajouter les dépendances

Le code suivant montre les dépendances nécessaires pour utiliser Retrofit.

//Retrofit

dependencies {

implementation "com.squareup.moshi:moshi:$version\_moshi"

implementation "com.squareup.moshi:moshi-kotlin:$version\_moshi"

implementation "com.squareup.retrofit2:converter-moshi:$version\_retrofit"

}

* 1. Ajouter les permissions

Le code suivant montre les permissions nécessaires au bon fonctionnement de Retrofit. En effet, il est nécessaire que l'application ait la permission d'accéder à internet afin d'accéder au service REST.

<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="fr.eni.randomchuck">

<uses-permission android:name="android.permission.INTERNET" />

​

[…]

* 1. Créer le model

Le code suivant montre une entité permettant de stocker un enregistrement de données récupéré par Retrofit. Les propriétés de l'entité doivent avoir le même nom que le nom des propriétés renvoyées par le service REST sinon l'annotation @Json doit être utilisée. Par exemple, la propriété image permet de stocker la propriété icon\_url de l'enregistrement renvoyé par le service REST.

data class Joke(

val id: String,

@Json(name = "icon\_url") val image: String,

@Json(name = "value") val joke: String

)

​

* 1. Créer le service client

Le code suivant permet de représenter le service REST définit à l'adresse définit dans la variable BASE\_URL ("https://api.chucknorris.io/jokes/"). Ce service offre une méthode qui permet d'interroger l'adresse suivante : https://api.chucknorris.io/jokes/random"

interface ChuckService {

companion object {

val BASE\_URL = "https://api.chucknorris.io/jokes/"

val moshi = Moshi.Builder().add(KotlinJsonAdapterFactory()).build()

val retrofit = Retrofit.Builder()

.addConverterFactory(MoshiConverterFactory.create(moshi))

.baseUrl(BASE\_URL)

.build()

}

@GET("random")

suspend fun randomFact(): Joke

}

​

object ChuckApi {

val retrofitService: ChuckService by lazy { retrofit.create(ChuckService::class.java) }

}

​

* 1. Utiliser le service client

Le code suivant permet de montrer comment utiliser le service REST dans un ViewModel. On notera l'utilisation d'une coroutine car il est recommandé de consulter des sources de données extérieur dans un thread parallèle.

class RandomjokeViewModel(application: Application): AndroidViewModel(application)

{

val joke = MutableLiveData<Joke>()

val status = MutableLiveData<String>()

​

fun randomPersonne()

{

viewModelScope.launch {

try {

var result = ChuckApi.retrofitService.randomFact()

status.value = "OK"

joke.value = result

}

catch (e: Exception)

{

status.value = "KO : ${e.message}"

}

}

}

}

1. Conclusion

Grâce a sa simplicité, Retrofit est largement utilisé pour interroger des services REST.