ESIEE PARIS

Année 2019-2020
Semestre 2

Rapport Intermédiaire d'avancement de Projet Androïd

Avancement du 02/04/2020

Projet réalisé par :

MONTANI Maÿlis

Lien du Repos Github :

https://github.com/mayisu7798/ChuckNorrisJokesApp

Projet encadré par :

DUPONCHEL Nicolas

Consignes du mail

A la suite de votre mail nous indiquant les modalités de rendus pour le TP, j'ai ajouté un fichier README.md à mon git. J'ai également ajouté un dossier dans lequel j'ajouterai mes rapports d'avancement au format PDF.

Temps : ~ 10 minutes

Commit : Ajout d'un README de présentation du projet au git
Commit : Ajout des rapports d'avancement au Git

Partie II - 1 : Create the data model matching the API

Importer Kotlinx.serialization dans notre projet :

Nous devions importer Kotlinx.serialization dans notre projet. J'ai donc dû ajouter quelques lignes dans les fichiers `build.gradle (ChuckNorrisJokesApp)` et `build.gradle (:app)`.

Toutes les lignes "classpath" ont été ajoutées au fichier `build.gradle (ChuckNorrisJokesApp)`. J'ai également changé la version de kotlin en 1.3.61.

Toutes les lignes "implementation" ont été ajoutées au fichier `build.gradle (:app)`. Nous devions également ne pas oublier d'ajouter les lignes relatives au kotlin serialization runtime que j'ai mis en version 0.14.0 pour sa compatibilité avec la version 1.3.61 de kotlin.

Note : J'ai eu beaucoup de mal à comprendre qu'il fallait aussi changer des choses dans le fichier `build.gradle (ChuckNorrisJokesApp)` puisqu'on y avait pas encore touché pour le moment.
Temps : ~ 1 heure

Création de la classe de données Json `Joke` :

Nous devions ensuite créer une classe de données `Joke`. On fait précéder le mot clé "classe" par "data" pour indiquer qu'il ne s'agit que d'un objet contenant des données. On ajoute également le tag `@Serializable` avant la classe afin de pouvoir y traiter les données issues d'un fichier JSON.

Il nous était également demandé de changer le nom de certaines propriétés de l'objet pour les adapter au format de nommage "camel case". Pour cela, on utilise le tag `@SerialName` avant le nom de la variable.

En lançant le test demandé, celui-ci fonctionnait pour tout.

Note : J'ai mis beaucoup de temps à trouver une définition de Serialisable qui me parlait. J'avais besoin de comprendre ce que c'était. Lorsque j'ai lancé le test la première fois, je me suis aperçue que je n'avais pas mis les noms des variables dans le bon ordre. J'ai donc dû les changer.
Temps : ~ 1 heure

Commit : Ajout de la classe Joke et du test fonctionnel

Convertir notre liste statique précédente en liste de `Joke`

Nous souhaitons désormais utiliser notre nouvelle classe `Joke` pour stocker nos blagues sur Chuck Norris. Dans un premier temps, nous allons modifier le type de ce qu'attend la classe `JokeAdapter` en changeant de `List<String>` à `List<Joke>` (dans mon cas, on passait une liste de type `StaticList`, mais du coup j'ai adapté la suite du code pour utiliser le type `List<Joke>` directement). Cela implique évidemment d'adapter toutes les fonctions recevant un type `List<String>` en `List<Joke>` dans le code du fichier `JokeAdapter.kt`.

Dans un second temps, il nous faut transformer notre `List<String>` en `List<Joke>` dans le fichier `MainActivity.kt`. Pour cela, nous créons une fonction permettant de convertir les types. Voici le contenu de cette fonction, réalisée à l'aide de la fonction "map" :

```
private fun List<String>.toJokeList() : List<Joke> = map { stringJoke ->
    Joke(   categories = listOf("Rien", "Rien bis"),
            createdAt = "Paris",
            iconUrl = "Pas d'Url",
            id = "Pas d'ID",
            updatedAt = "Paris",
            url = "Pas d'Url",
            value = stringJoke )
}
```

Ainsi, il nous suffit ensuite d'appeler cette fonction à l'endroit où nous créons notre adapteur grâce à la ligne suivante : `JokeAdapter(StaticList.list.toJokeList())`

Note : J'ai eu énormément de mal à comprendre comment je devais m'attaquer à cette question et je ne comprenais pas vraiment l'énoncé. Après une heure de recherche, je vous ai envoyé un mail pour que vous m'aidiez. J'ai ensuite réussi à comprendre comment faire.
Temps : ~ 2 heures

Commit : Convertir notre liste statique précédente en liste de Joke

Partie II - 2 : Import Retrofit & Rx Java

Pour cette partie, nous devions simplement importer pleins d'outils qui nous seront utiles pour la suite. Ainsi, dans le fichier `build.gradle(:app)` nous avons dû ajouter ces lignes :

```
// Retrofit & Rx Java
implementation "com.squareup.retrofit2:retrofit:2.8.1"
implementation "io.reactivex.rxjava2:rxkotlin:2.3.0"
implementation "io.reactivex.rxjava2:rxandroid:2.1.0"
implementation "com.squareup.retrofit2:adapter-rxjava2:2.8.1"
// Retrofit answers serialization
implementation "com.jakewharton.retrofit:retrofit2-kotlinx-serialization-converter:0.5.0"
```

Et dans le fichier `AndroidManifest.xml`, nous avons dû ajouter cette ligne avant le tag `<application/>` :

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

Note : J'ai tenté de relancer ensuite l'application sur la simulation de téléphone proposé par l'application et les deux tests que nous avions fait précédemment échouent. Je suppose que c'est parce qu'on a ajouté des imports concernant la sérialisation. Comme la consigne ne demande pas particulièrement de lancer l'application, je ne vais pas tenir compte de cette erreur pour le moment.
Temps : ~ 15 minutes

Commit : Ajout des imports Retrofix et Rx Java

Partie II - 3 : Retrofit usage

API Interface

Il nous est demandé de créer une interface appelée "`JokeApiService`" qui contient une fonction `giveMeAJoke()` qui ira chercher les Joke sur internet au lien donné précédemment dans le sujet, à savoir `https://api.chucknorris.io/jokes/random`.

Temps : ~ 5 minutes
Commit : Ajout de l'interface JokeApiService et de sa fonction giveMeAJoke()

Retrofit Factory

Nous devons créer un objet `JokeApiServiceFactory` contenant une fonction de création d'instance de `JokeApiService`. Pour créer cette instance, nous allons alors utiliser `Retrofit.Builder` en lui ajoutant des informations comme son API, un converter factory et un adapter factory. Nous allons ensuite demander au builder ainsi créé de retourner une instance de `JokeApiService`.

```
object JokeApiServiceFactory {
    fun createJokeApiService () : JokeApiService {
        val builder : Retrofit = Retrofit.Builder()
            .baseUrl( baseUrl: "https://api.chucknorris.io/")
            .addConverterFactory(Json.asConverterFactory(MediaType.get("application/json")))
            .addCallAdapterFactory(RxJava2CallAdapterFactory.create())
            .build()
        return builder.create(JokeApiService::class.java)
    }
}
```

Note : Ca m'a semblé assez clair. Mais j'ai encore des erreurs lorsque je lance mon applications avec les tests liés à la sérialisation. J'attends encore d'avancer dans le sujet pour savoir si c'est normal ou non puisque le sujet ne nous demande toujours pas de lancer l'application.
Temps : ~ 30 minutes

Commit : Ajout d'un objet JokeApiServiceFactory et de sa fonction de création …

Partie II - 4 : Call Api to get one Joke

Il nous est demandé de créer une instance de `JokeApiService` dans le `MainActivity.kt` à l'aide de notre `JokeApiServiceFactory`. Nous devions alors récupérer une Joke à l'aide de la fonction `giveMeAJoke()` créée précédemment. Le résultat retourné par cette fonction étant un `Simple<Joke>` nous devions traiter l'erreur potentielle par l'utilisation de `subscribeBy()`, agrémentée par `subscribeOn(Scheduler.io())` pour faire tourner sur un autre Thread. Voici le code que nous obtenons :

```
val jokeService : JokeApiService = JokeApiServiceFactory.createJokeApiService()
val joke : Single<Joke> = jokeService.giveMeAJoke()
val resultSubscribe = joke.subscribeOn(Schedulers.io()).subscribeBy(
    onError = {println("Le Single<Joke> renvoie une erreur")},
    onSuccess = { joke -> println("Yaaay on peut utiliser la Jooooke : ${joke.value}")}
)
```

Notes :

- Je me suis aperçue que les erreurs dont je parlais précédemment étaient liées au fait que je n'éxécutais pas le MainActivity mais encore le test sur la sérialisation… Donc j'ai juste exécuté le MainActivity à la place.

- J'avais des erreurs incompréhensibles, j'ai cherché à quoi elles pouvaient être liées et je me suis dis que j'allais changer les versions de Kotlin et du runtime pour vérifier que l'erreur ne venait pas de là. Finalement, c'était de là qu'elle venait puisque la seule erreur qu'il me restait était celle indiquée dans le sujet.
    - Modification dans `build.gradle(ChuckNorrisJokesApp)` :
      ```
      ext.kotlin_version = '1.3.70'
      ```
    - Modification dans `build.gradle(:app)`
      ```
      implementation "org.jetbrains.kotlinx:kotlinx-seri
      alization-runtime:0.20.0"
      ```

- J'ai donc ajouté les quelques lignes que vous recommandiez dans le fichier `build.gradle(:app)` dans le bloc `android{...}` et mon application a fonctionné.
  ```
  compileOptions {
      sourceCompatibility JavaVersion.VERSION_1_8
      targetCompatibility JavaVersion.VERSION_1_8
  }
  ```

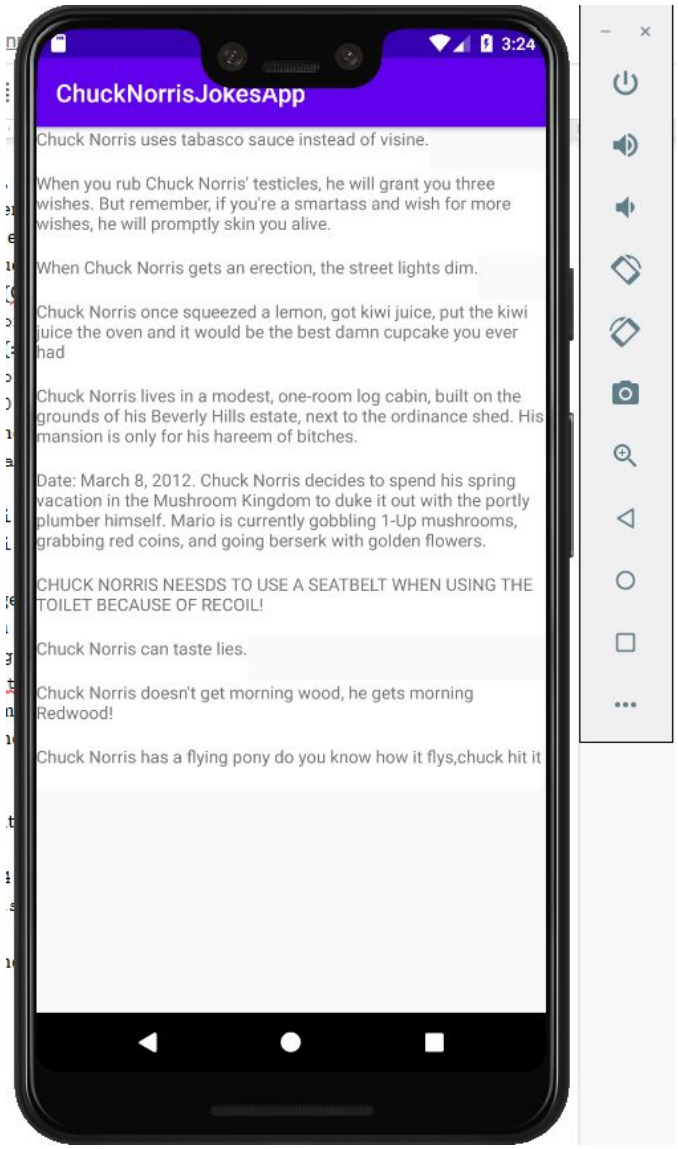
- J'avais cependant un affichage étrange lié au fait que j'avais mal modifié mon fichier JokeAdapter en laissant la ligne `holder.textView.text = listJokes[position].toString()` au lieu de la changer en `holder.textView.text = listJokes[position].value` dans la fonction `onBindViewHolder()`. Je l'ai donc modifié.

- Enfin, j'avais laissé une couleur de fond de texte un peu violente pour les yeux (rouge vif, avec écriture en noir), donc je l'ai changée parce que c'était plus joli (fond blanc).

Finalement, je peux bien lire dans mon LogCat la ligne que je voulais afficher en cas de réussite de lecture de la Joke :

```
I/System.out: Yaaay on peut utiliser la Jooooke : Chuck Norris is the reason Usain Bolt can run so fast.
```

Et mon application affiche bien les Jokes comme je les voulais (bien qu'on utilise pas encore pour le moment les Jokes récupérées sur internet, mais encore seulement les Jokes de ma liste statique convertie en `List<Joke>`):



Temps : ~ 2 heures
Commit : Appel de l'API pour obtenir une Joke & quelques rectifications de code

Partie II - 5 : Leaks killer

Il nous est demandé de pouvoir supprimer notre subscription à l'observable de Single. Pour cela, il faut dans un premier temps créer un attribut de la classe de type CompositeDisposable qui contiendra tous les observateurs créés :

```
class MainActivity : AppCompatActivity()
{
    private val disposable = CompositeDisposable()
```

Ensuite, il nous faut ajouter à cet attribut la valeur de retour de notre fonction subscribeBy() appliquée à notre Joke :

```
val resultSubscribe = joke.subscribeOn(Schedulers.io()).subscribeBy(
    onError = {println("Le Single<Joke> renvoie une erreur")},
    onSuccess = { joke -> println("Yaaay on peut utiliser la Jooooke : ${joke.value}")}
)
disposable.add(resultSubscribe)
```

Enfin, il faut override la fonction onDestroy() de notre MainActivity pour y ajouter le clear de notre attribut.

```
override fun onDestroy() {
    super.onDestroy()
    disposable.clear()
}
```

Temps : ~ 20 minutes
Commit : Suppression de l'observable du Simple par ajout d'un CompositeDisposable

Partie II - 6 : Conclusion

Note : Je m'arrête là pour aujourd'hui.
Bilan Temps : 7h20