Bilan du TEA2 de PMR

Introduction

L'objectif de ce troisième TEA est de rajouter en plus de la connexion entre notre application et une API en ligne, une base de données stockées en mémoire cache afin d'accéder aux données même quand le téléphone n'est pas connecté à internet.

L'interface de l'application ne change pas.

Présentation de quelques éléments du code

Le code est déjà commenté, mais nous revenons ici sur les éléments clés de la création d'une base de données :

A la différence des TEA précédents, cette fois-ci, on utilise un booléen afin de déterminer si le téléphone a accès ou non à internet. Pour connaître l'état de l'accès au réseau dans les autres activités, on ajoute un booléen dans les sharedPreferences qui vaut "true" si on a accès au réseau, sinon "false".

```
fun isNetworkAvailable(context: Context): Boolean {
    val connectivityManager =
        context.getSystemService(Context.CONNECTIVITY_SERVICE) as ConnectivityManager

    val network = connectivityManager.activeNetwork ?: return false
    val networkCapabilities =
        connectivityManager.getNetworkCapabilities(network) ?: return false

    return networkCapabilities.hasCapability(NetworkCapabilities.NET_CAPABILITY_INTERNET)
}
```

Si oui, alors l'application fonctionne exactement comme lors du TEA 2, elle fait une requête de type POST à l'API afin d'obtenir un token (le hash) qui lui sert de clé d'identification pour accéder aux informations stockées en ligne. On utilise des requêtes GET accompagnées du hash pour accéder aux listes, et des requêtes GET accompagnée du hash et de l'identifiant de la liste pour accéder aux items qu'elle contient. A chaque fois, pour ses requêtes, l'application utilise les sharedpreferences afin d'obtenir l'url de base de l'API avec laquelle elle doit communiquer, cet url de base est modifiable depuis les settings de l'application (menu 3 petits points en haut à droite de la MainActivity).

Sinon, l'application accède aux données via une base SQLite qu'elle a renseignée pour la dernière fois lorsqu'elle s'est connectée à l'API.

Ci-dessous on retrouve les définitions des classes des objets qui composent la base de données, les listes (un label et un id), les items (un id, un label, le booléen de la case à cocher et enfin l'id de la liste qui le contient)

```kotlin
//Création de l'objet ListOfItems pour la base de données (stockage des listes)
@Entity(tableName = "lists")
data class ListOfItems(
    @PrimaryKey val id: String,
    val label: String,
    val items: MutableList<Item> = mutableListOf()
)
```

```kotlin
//Création de l'objet ItemEntity pour la base de données (stockage des items)
@Entity(tableName = "items")
data class ItemEntity(
    @PrimaryKey val id: String,
    val label: String,
    val checked: String,
    val listId: String
)
```

On crée la nouvelle interface ListDao, qui va permettre d'utiliser des méthodes pour modifier les éléments de notre base de données. On remarque qu'on utilise la syntaxe SQL afin de faire les requêtes.

- getAllLists() et getItemsForList() permettent juste d'afficher les listes ou les items d'une liste (afin son id).

-insertList() et insertItem() permettent de rajouter des listes et des items à la base de données.

```kotlin
package com.example.tea

import androidx.room.Dao
import androidx.room.Insert
import androidx.room.OnConflictStrategy
import androidx.room.Query


//Méthodes de la base de données :
@Dao
interface ListDao {
    @Query("SELECT * FROM lists")
    fun getAllLists(): List<ListOfItems>

    @Query("SELECT * FROM items WHERE listId = :listId")
    fun getItemsForList(listId: String): List<ItemEntity>

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertList(list: ListOfItems)

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    fun insertItem(item: ItemEntity)

}
```

On implémente ces éléments dans ChoixListActivity et ShowListActivity, en définissant les variables de notre base de données.

```kotlin
123         private lateinit var database: AppDatabase
124         private lateinit var listDao: ListDao
```

On l'initialise :

```kotlin
140         // Initialisation Room et ListDao
141         database = Room.databaseBuilder(applicationContext, AppDatabase::class.java, name: "my-database")
142             .build()
```

Et on modifie les fonctions onResponse de ces deux activités afin que lors de la réponse de l'API, on stock les données obtenues également dans la base de données.

```
if (responseBody != null) {
    val lists = responseBody.lists

    GlobalScope.launch(Dispatchers.IO) { this: CoroutineScope
        for (listItem in lists) {
            val listOfItems = ListOfItems(listItem.id, listItem.label)
            listDao.insertList(listOfItems)

        }
    }

    // Récupérer les listes depuis la base de données
    val retrievedLists = listDao.getAllLists()
    adapter.items.addAll(retrievedLists)
    adapter.notifyDataSetChanged()
```

Difficultés

Dans un premier temps l'application crashait après la connexion. Il a donc fallu identifier la cause de ce problème, avec l'utilisation de débugger notamment.

On n'a malheureusement pas réussi à faire exécuter le programme car une erreur de compatibilité java intervient dès la compilation de l'application. Le code ne présente pas de problèmes apparents mais les bibliothèques utilisées entrent sûrement en conflit car ne vise pas la même version de java pour s'exécuter.

On n'a donc pas pu tester le code, cependant la structure de la base de données devrait être fonctionnelle et la partie affichage également.

Perspectives

On pourrait imaginer implémenter les boutons de suppression des listes et des items, et pourquoi pas des éléments accessoires comme des outils de filtre par ordre alphabétique, nombre d'items, …

Optimiser la lisibilité et aussi l'ordre du code.

Le côté esthétique peut aussi être grandement amélioré, et de nouveaux messages d'erreurs afin de comprendre la source du problème pourraient être affichés à l'écran.

Pourquoi pas une liste déroulante qui retient les anciens url d'api afin de ne pas avoir à les re-renseigner.

Sources

Surtout des forums tels qu'open classroom, stackoverflow, ainsi que le site

https://developer.android.com/kotlin?hl=fr qui donne des indications sur chaque élément de kotlin.

ChatGPT pour tout ce qui est débogage, reformulation.