Compte Rendu TEA 3

ED PMR 2024

LASCOUMES

L’objectif de ce TEA est de reprendre l’application du TEA précédent, et d’y ajouter un mode offline, afin de pouvoir cocher/décocher des items même lorsque le terminal n’est pas connecté à internet.

Pour mettre en place un mode offline, on commence par créer une base de données locale avec Room, permettant le stockage des listes/items et leur états (coché/décoché). On rajoute également une table items_sync, pour stocker, pour chaque item, la liste à laquelle il appartient, ainsi que si il est synchronisé avec l'API ou non. (On ne peut pas directement intégrer ces attributs à la table items sans causer des problèmes de désérialisation des réponses de l'API avec retrofit)

```kotlin
@Dao  new *
interface ItemToDoDao {

    @Query("SELECT * FROM lists WHERE id = :listId ")  new *
    suspend fun getList(listId: String): ListeToDo?

    @Query("SELECT * FROM lists")  new *
    suspend fun getLists(): List<ListeToDo>

    @Insert(onConflict = OnConflictStrategy.REPLACE)  new *
    suspend fun insertList(list: ListeToDo)

    @Update  new *
    suspend fun updateList(list: ListeToDo)

    @Query("SELECT items.* FROM items JOIN items_sync ON items.id = items_sync.itemId  WHERE items_sync.listId = :listId")
    suspend fun getListItems(listId: String): List<ItemToDo>

    @Query("SELECT * FROM items WHERE id = :itemId ")  new *
    suspend fun getItem(itemId: String): ItemToDo?

    @Insert(onConflict = OnConflictStrategy.REPLACE)  new *
    suspend fun insertItem(item: ItemToDo)

    @Update  new *
    suspend fun updateItem(item: ItemToDo)

    @Update  new *
    suspend fun updateSync(sync: ItemsSync)

    @Query("SELECT * FROM items_sync WHERE unsynced = 1")  new *
    suspend fun getUnsynced(): List<ItemsSync>

    @Query("SELECT * FROM items_sync WHERE itemId = :itemId ")  new *
    suspend fun getSync(itemId: String): ItemsSync?

    @Insert(onConflict = OnConflictStrategy.REPLACE)  new *
    suspend fun insertSync(sync: ItemsSync)
```

Ainsi, pour chaque appel à l'API (dans ShowListActivity ou ChoixListActivity), on vérifie si internet est accessible. Si ce n'est pas le cas, on affiche la base de données interne, et lors des changements d'état, on les sauvegarde et on marque l'item comme "unsynced". Si le réseau est accessible, on fait des appels API comme dans le TEA précédent, tout en sauvegardant les changements dans la base de données locale:

```
private fun updateItems(hash: String, listId: String) {  Minebox260 *
    lifecycleScope.launch {
        val db = AppDatabase.getDatabase( context: this@ShowListActivity)
        if (isNetworkAvailable()) {
            try {
                if (DataProvider.isInitialized()) {
                    val response = DataProvider.getItems(hash, listId)
                    if (response.success) {
                        response.items?.forEach { item ->
                            if (db.itemDao().getItem(item.id) != null) {
                                db.itemDao().updateItem(item)
                            } else {
                                db.itemDao().insertItem(item)
                            }
                            if (db.itemDao().getSync(item.id) == null) {
                                val sync = ItemsSync(itemId = item.id, listId = listId)
                                db.itemDao().insertSync(sync)
                            }
                        }
                        adapter.show( items: response.items ?: listOf())
                    } else {
                        showPopup("Failed to load items")
                    }
                }
            } catch (e: HttpException) {
                if (e.code() == 403) {
                    showPopup("Failed to authenticate. Please log out and try again")
                } else
                    showPopup("Failed to load items")
            } catch (e: Exception) {
                Log.e( tag: "PMR",  msg: e.message ?: "")
            }
        } else {
            val items = db.itemDao().getListItems(listId)
            adapter.show(items)
        }

    }
}
```

Enfin, on met en place un NetworkChangeReceiver, qui, lorsqu'il va détecter un changement d'état du réseau, et si internet est accessible, va lancer une synchronisation des données, en récupérant tous les items marqués comme unsynced dans la base de données locale, et en envoyant des requêtes correspondantes à l'API :

```kotlin
object sync {  new *

    suspend fun items(context: Context) {  new *
        val hash = context.getSharedPreferences( name: "user", Context.MODE_PRIVATE).getString( key: "hash",  defValue: "")
            ?: return
        val db = AppDatabase.getDatabase(context)
        val unsyncedItems = db.itemDao().getUnsynced()

        for (sync in unsyncedItems) {
            try {
                val item = db.itemDao().getItem(sync.itemId)
                if (item != null) {
                    val response = DataProvider.putItem(hash, sync.listId, item.id, item.checked)
                    if (response.success) {
                        sync.unsynced = false
                        db.itemDao().updateSync(sync)
                    }
                }

            } catch (e: Exception) {
                Log.e( tag: "PMR",  msg: e.message ?: "")
                return
            }
        }
    }
```

L’application ne permet ici que la modification de l’état des items en mode hors ligne (les boutons d’ajouts sont désactivés), mais on pourrais imaginer rendre l’ajout d’items possibles, ainsi que l’ajout de listes (il faudrait alors prendre en compte la synchronisation des listes, et l’attribution d’ID par l’API lors de la création d’un nouvel item/list)