Programmation mobile et réalité augmentée
Compte Rendu TP2

REST API et SQLite

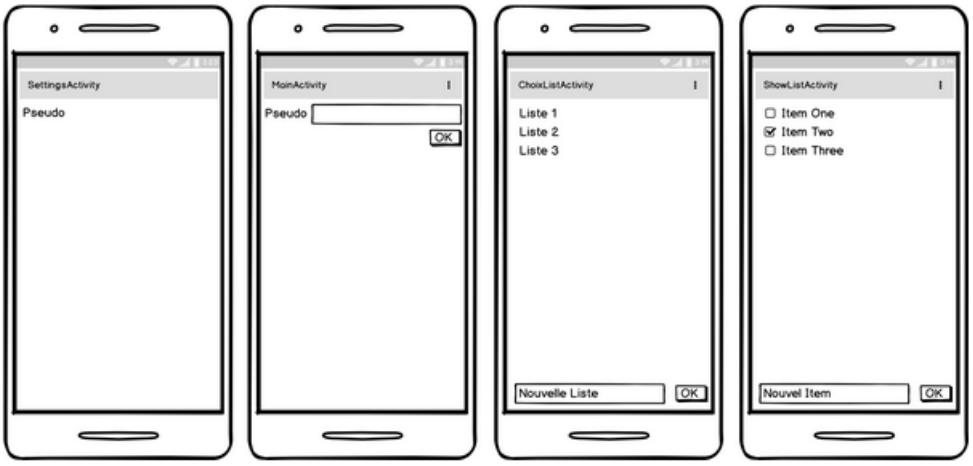
Andreis Gustavo Malta Purim
Lucas Oliveira Saintrain

Le 27 Juin 2022

1. Introduction

L'objectif de ce TP est d'explorer comment fonctionne une connexion API et comment avoir les données hors ligne avec une base de données SQLite, le tout utilisant Android.

Bien que pas complètement finis, nous avons développé ce TO du mieux que nous pouvions.



Le lien vers notre solution est :

https://github.com/PMR2022/TP2_SAINTRAIN_PURIM.git

2. Analyse / Développement

Pour la partie API, nous avons décidé d'utiliser Retrofit car il avait une configuration JSON automatisée, ce qui nous a semblé beaucoup plus rapide à utiliser. Fondamentalement, nous avons créé une classe appelée connecteur qui contiendra un objet de mise à niveau à l'intérieur avec la baseurl. Ensuite, cet objet implémentera une interface avec une série de requêtes (GET, POST, etc...) Nous avons dû créer des classes de données supplémentaires car l'API n'était pas exactement égale aux classes que nous avions créées précédemment.

```
data class Auth(val hash : String)
@Entity
data class apiListTD(@PrimaryKey val id:Int, val label:String, val userHash:String? = null)
@Entity
data class apiItemTD(@PrimaryKey val id:Int, val label:String, var checked:Boolean, val idList: Int? = -1)
data class apiLists(var lists : List<apiListTD>)
data class apiItems(var items : List<apiItemTD>)

class Connector(application: Application, baseurl: String = "http://tomnab.fr/todo-api/") {
    // API part
    private val retrofit: Retrofit = Retrofit.Builder()
        .baseUrl(baseurl)
        .addConverterFactory(GsonConverterFactory.create())
        .build()

    // API Calls
```

Ensuite, lorsque l'utilisateur se connecte à MainActivity, nous obtenons ses informations, effectuons l'accès et transmettons le hash à ChoixListActivity en tant que *intent bundle extra*

```
var hash: String? = null
try{
    hash = apiConnector.login(pseudo,password).hash
    bundle = bundle.apply {  this: Bundle
        putString("pseudo", pseudo)
        putString("hash", hash)
    }
} catch(exception:Exception){
    Toast.makeText(applicationContext,  text: "Erreur connexion", Toast.LENGTH_LONG).show()
    bundle = bundle.apply {  this: Bundle
        putString("pseudo", pseudo)
    }
```

Si la connexion ne fonctionne pas, le programme essaiera d'utiliser la base de données dont il dispose à la place, et si cela ne fonctionne pas, il obtiendra le JSON des préférences partagées de TP1.

Ensuite, sur ChoixListActivity, le programme va récupérer le hash donné et recréer la classe Profile et ajouter la List récupérée depuis l'API. Pour chaque liste, il obtiendra également les éléments. Cette méthode est plus facile à faire plutôt que d'obtenir chaque liste d'éléments pour chaque liste cliquée, car elle fait du profil une structure cohérente, plutôt que des appels d'API déconnectés.

```
var apiLists: apiLists?
try {
    apiLists = apiConnector.getLists(hash)
}catch(exc: Exception){
    Toast.makeText(applicationContext, text: "Using local DB",Toast.LENGTH_SHORT).show()
    apiLists = apiConnector.getLists(hash)
    apiLists = apiConnector.dbGetLists(hash)
}
if (apiLists != null) {
    for (apiList in apiLists.lists){
        val newItems = mutableListOf<ItemTD>()
        for (apiItem in apiConnector.getItems(hash, apiList.id).items){
            newItems.add(ItemTD(apiItem.label,apiItem.checked,apiItem.id,apiItem.id))
        }
        val newList = ListTD(apiList.label,apiList.id)
        newList.setItems(newItems)
        profile!!.addList(newList)
    }
} else {
    Toast.makeText(applicationContext, text: "API and DB failed to connect",Toast.LENGTH_SHORT).show
}
// Saves new user profile
```

Pour la partie DB, nous avons utilisé une structure Dao, et chaque fois qu'il fait une modification sur la liste, il modifie également la DB.

```
@androidx.room.Database(entities = [apiListTD::class, apiItemTD::class],version = 8)
abstract class DatabasePMR : RoomDatabase() { abstract fun daoDB() : DaoInterface }

@Dao
interface DaoInterface{
    // Get
    @androidx.room.Query("SELECT * FROM apiListTD WHERE userHash LIKE :hash")
    suspend fun getLists(hash:String) : List<apiListTD>
    @androidx.room.Query("SELECT * FROM apiItemTD WHERE id LIKE :id")
    suspend fun getItems(id:Int) : List<apiItemTD>
    // Insert / Update
    @androidx.room.Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertList(lists : apiListTD)
    @androidx.room.Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertItem(items : apiItemTD)
}
```

3. Conclusion

Nous avons eu quelques problèmes personnels pour ce TP (nous étions tous les deux en train de déménager juste après les examens), nous n'avons donc pas pu terminer toutes les tâches correctement. La partie API ne fonctionne que sur les Gets et Puts de base alors que la DB n'a pas été correctement testée lorsqu'elle n'a pas d'Internet. En général, il est censé fonctionner, mais il peut contenir des erreurs.

C'était un défi intéressant et nous avons beaucoup appris en le faisant.