Léo Rongières

Rapport TEA 3

Introduction

L'objectif est de reprendre l'application des TEA précédents et d'intégrer un mode Offline, en stockant les listes et items dans une base de données sur l'appareil. En mode offline on peut seulement afficher, cocher et décocher les items. L'analyse s'articule donc en deux parties non égales, lire l'état réseau, la création et manipulation de la base de données

Analyse

Lire l'état du réseau

J'ai utilisé la bibliothèque ConnectivityManager pour lire l'état du réseau, dans la méthode isNetworkAvailable() :

```
private fun isNetworkAvailable(): Boolean {
    val connectivityManager : ConnectivityManager! = getSystemService(ConnectivityManager::class.java)
    if (connectivityManager != null) {
        val capabilities : NetworkCapabilities? =
            connectivityManager.getNetworkCapabilities(connectivityManager.activeNetwork)
        if (capabilities != null) {
            if (capabilities.hasTransport(NetworkCapabilities.TRANSPORT_CELLULAR)) {
                return true
            } else if (capabilities.hasTransport(NetworkCapabilities.TRANSPORT_WIFI)) {
                return true
            } else if (capabilities.hasTransport(NetworkCapabilities.TRANSPORT_ETHERNET)) {
                return true
            }
        }
    }
    return false
}
```

On vérifie que chaque réseau est indisponible

La base de données

J'ai utilisé l'ORM Room pour implémenter et utiliser une base de données dans l'application pour la persistance de données.

Léo Rongières

Tout d'abord on a défini nos tables de la forme :

```kotlin
@Entity(tableName = "lists")
data class ListEntity(
    @PrimaryKey val id: String,
    @ColumnInfo(name = "label") val label: String,
    @ColumnInfo(name = "user_id") val userId: String
)
```

Ensuite il faut définir les fonctions associées pour manipuler la table :

```kotlin
@Dao
interface UserDao {
    new *
    @Insert(onConflict = OnConflictStrategy.REPLACE)
    suspend fun insertUser(user: UserEntity)

    new *
    @Query("SELECT * FROM users WHERE username = :username")
    suspend fun getUserByUsername(username: String): UserEntity?
}
```

Une fois qu'on définit tout ça pour les tables « users », « lists » et « items », on définit la base de données :

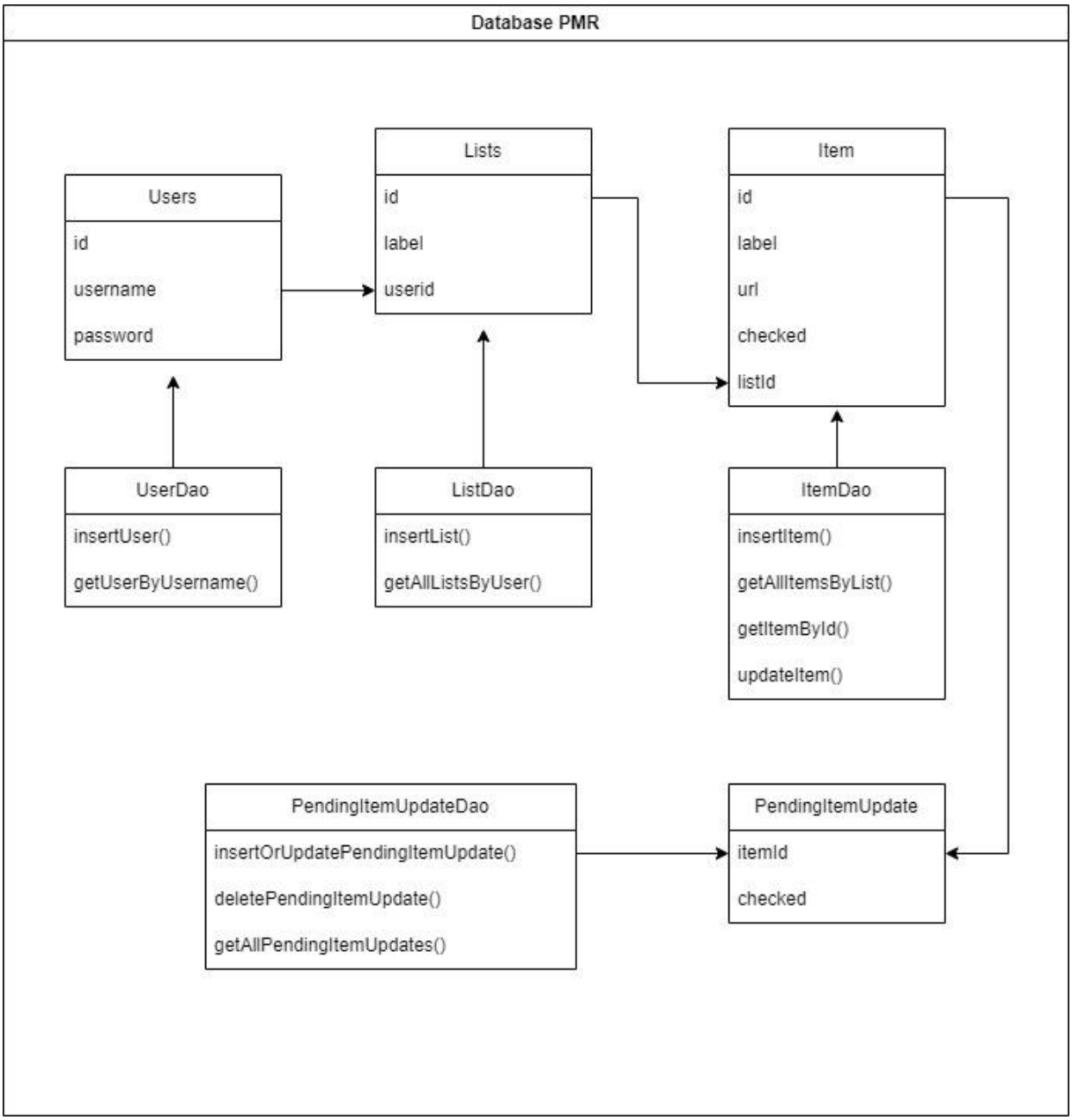
```kotlin
@Database(entities = [UserEntity::class, ListEntity::class, ItemEntity::class, PendingItemUpdate::class], version = 5, exportSchema = false)
abstract class AppDatabase : RoomDatabase() {
    new *
    abstract fun userDao(): UserDao
    new *
    abstract fun listDao(): ListDao
    new *
    abstract fun itemDao(): ItemDao
    new *
    abstract fun pendingItemUpdateDao(): PendingItemUpdateDao
    new *
    companion object {
        @Volatile
        private var Instance: AppDatabase? = null

        new *
        fun getDatabase(context: Context): AppDatabase {
            // if the Instance is not null, return it, otherwise create a new database instance.
            return Instance ?: synchronized( lock: this) {
                Room.databaseBuilder(context, AppDatabase::class.java, name: "PMR") Builder<AppDatabase>
                    .fallbackToDestructiveMigration()
                    .build() AppDatabase
                    .also { Instance = it }
            }
        }
    }
}
```

On a mis toutes nos tables en paramètres, nos DAO dans la fonction et ajouté la fonction getDatabase() pour initialiser la DB dans les classes.

Léo Rongières

Voici un résumé de la base de données :

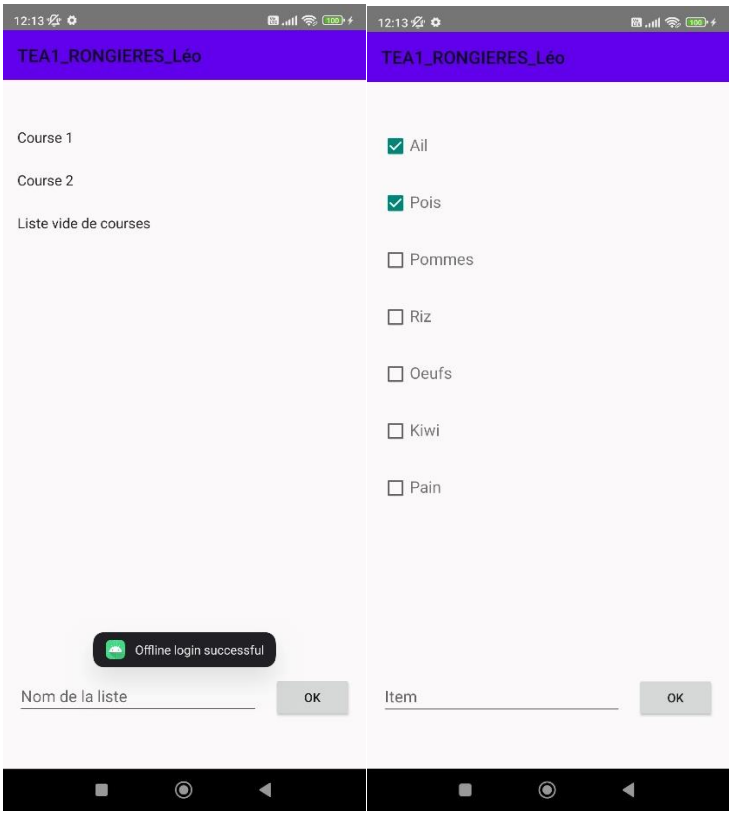


J'ai ajouté une classe PendingItem update qui stocke les modifications faites sur les items durant le mode Offline. Cela permet qu'au retour de la connexion internet, on update seulement les valeurs de checked qui ont été modifiées.

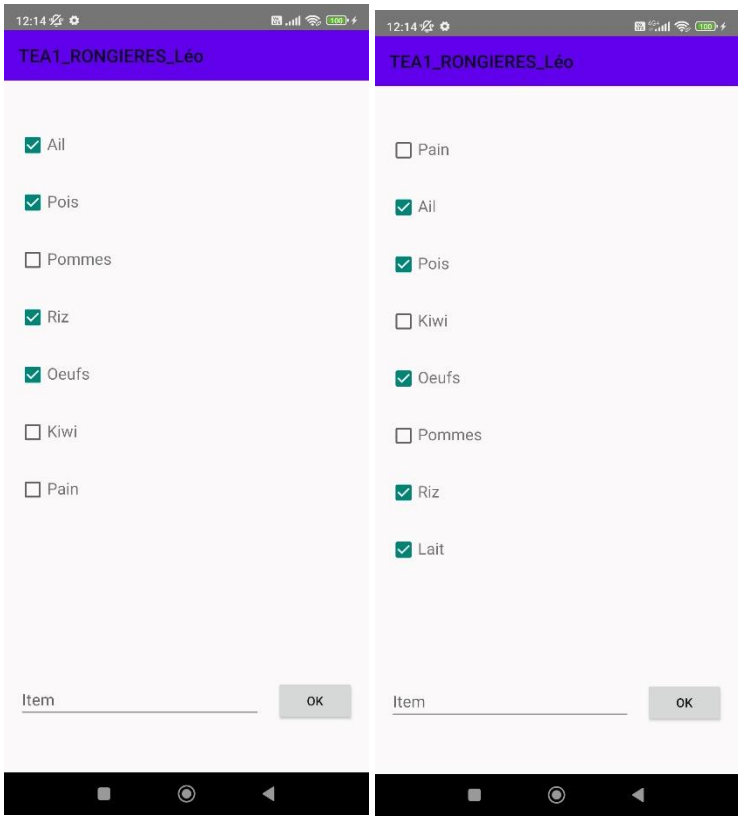
Pour utiliser le mode offline, j'ai repris les fonctions du TEA 2 en ajutant une disjonction de cas en fonction de la valeur de isNetworkAvailable() :

- En mode connecté, on charge les données et les stocke dans la base de données, on utilise les fonctions Dao pour les changements et les ajouts de listes
- En mode offline, on lit les données à partir de la DB, enregistre les modifications. On met dans la boucle OnResume de la classe ShowListActivity une fonction syncPendingItemUpdatesWithAPI() qui va update les modifications offline sur le serveur.

Léo Rongières



Connexion à l'application en mode hors ligne, lecture de la DB



Modifications des items, vérification de l'update dans l'API

Conclusion

L'application fonctionne, les données sont correctement stockées dans le cache avec l'ORM Room. Cependant on remarque que l'ordre des items et listes affichées n'est pas le même. C'est un point à améliorer.

Bibliographie

https://developer.android.com/codelabs/basic-android-kotlin-compose-persisting-data-room?hl=fr#0

https://developer.android.com/training/data-storage/room?hl=fr

https://developer.android.com/training/basics/network-ops/reading-network-state?hl=fr