Compte rendu:
Séquence 2-3:

1. Introduction:

De nos jours, tout le monde se trouve pressé, débordé par de multiples tâches et activités à accomplir, ce qui nécessite une bonne organisation pour une meilleure gestion de temps et d'énergie.
Notre application répond parfaitement à ce besoin. Elle permet à l'utilisateur de s'inscrire pour enregistrer ses ToDo listes avec ses différents items.
En plus, elle permet à l'utilisateur de récupérer ses notes depuis un cloud distant, depuis n'importe quel terminal, sans pour autant dépendre de la connexion réseau pour fonctionner.

2. Analyse:

❖ **Activités:**

- *MainActivity:*

Il s'agit de l'activité de démarrage de notre application.
Un champ d'entrée de texte, ainsi qu'un champs de saisie de mot de passe sont proposés à l'utilisateur pour pouvoir saisir ses identifiants.
D'ailleurs, lors de la saisie du pseudo, une liste d'auto complétion s'affiche pour permettre à l'utilisateur de choisir parmi les pseudos qui sont déjà entrés avant.
A l'aide de la fonction `setOnClickListener`, appelée sur le bouton, on exécute la fonction `loginAndSaveData()` pour effectuer l'enregistrement des données:

- GenericActivity (abstract)

Il s'agit de l'activité de base de notre application.

Cette activité contient des constantes représentant les clés d'accès à nos préférences pour ne pas avoir à saisir ces chaînes de caractères sur toutes les activités.
Elle contient également la méthode `alerter()` permettant d'afficher des messages à l'utilisateur.
De plus, cette activité contient la déclaration abstraite de la fonction `updateData()`, qui est reprise par les autres activités et qui permet de sauvegarder les modifications apportées si la connexion réseau est disponible. Ce dernier test est effectué à l'aide de la fonction `isNetworkAvailable()`, également présente dans cette classe.

- *SettingsActivity:*

Cette activité est une activité de préférence (qui hérite de PreferenceActivity), qui permet de stocker l'historique des pseudos (logins) des utilisateurs.
Elle permet également de changer l'url de base de l'API.

- *ChoixListActivity:*

Cette activité permet à l'utilisateur de gérer ses ToDo listes.
Pour cela, en inscrivant son pseudo sur l'interface de MainActivity et en cliquant sur le bouton OK, l'utilisateur est dirigé vers cette deuxième activité.
La liste des ToDo Listes de l'utilisateur est présentée sous forme d'une liste de boutons ayant comme titre les noms de ces ToDo Listes. Ainsi, l'utilisateur peut accéder au contenu de chaque ToDo liste en cliquant sur le bouton correspondant.
Ce clic bouton lui permet de passer à l'activité suivante.

Tout comme MainActivity, cette activité est équipée de notre ToolbarMenu introduit dans la suite de ce rapport

La récupération de la liste des listes est effectuée via la méthode `getLists()`:
On utilise pour cela les coroutines, étant faciles à rédiger et maintenir, tout en effectuant un travail robuste en background, transparent au développeur. On utilise le Dispatcher par Défaut pour tous les appels web.
Cette fonction fait ensuite appel à la fonction `getLists()` du data provider (voir plus bas).

- *ShowListActivity:*

Cette activité permet de lister une liste d'items. Elle possède un fonctionnement similaire à l'activité précédente. En plus, elle permet de mettre à jour l'état "checked"

des items en les décochant/cochant, et ce, en faisant appel à la méthode `updateData()` définie dans cette activité.

❖ DataProvider:

- Récupération du token de l'utilisateur

Lorsque l'utilisateur valide sa saisie du pseudo/mot de passe, la fonction `getUserHash()`, définie dans le provider, permet d'effectuer la requête POST nécessaire pour récupérer le hash. Il est intéressant de noter que l'API accepte les données sous format form-data. Envoyer des données JSON par exemple dans le body de la requête enverra une réponse 400.
Heureusement, la librairie RetroFit est assez flexible et permet facilement ce genre de requêtes:

```
val requestBody: RequestBody = MultipartBody.Builder()
    .setType(MultipartBody.FORM)
    .addFormDataPart( name: "user", user)
    .addFormDataPart( name: "password", password)
    .build()
val hash = userService.getUserHash(requestBody).hash
hash  ^withContext
```

- Sauvegarde/Récupération des données

On a défini une classe responsable de récupérer et sauvegarder les données depuis l'API web, pour chaque entité, en utilisant son service respectif.
Ceci se déroule comme suit:

- On essaie de récupérer les données depuis l'API, puis de les mettre en cache dans la bdd SQLite, créée à l'aide de Room. Si la récupération échoue (en cas de non disponibilité du réseau), on récupère les données stockées au préalable dans le cache Room.

```
suspend fun getLists(hash: String): List<ListeToDo>  {
    return try {
        val lists = listsService.getLists(hash).lists.toList()
        saveLists(lists)
        lists
    } catch (exception: Exception) {
        listDao.getLists()
    }
}
```

- Tables base de données:

Pour définir les tables de notre base de données, nous avons repris les entités modèles définies dans le TP précédent. Il suffisait d'ajouter le champ id, et les annotations nécessaires pour qu'ils soient interprétés par Room. Nous avons ensuite défini des fonctions permettant de faire la liaison entre nos objets réponse et la définition des tables/modèles.

```
private fun List<ItemResponse>.toItem(): List<ItemToDo> {
    return this.map { itemResponse ->
        ItemToDo(itemResponse.id, itemResponse.label, itemResponse.checked)
    }
}
```

❖ Affichage des listes:

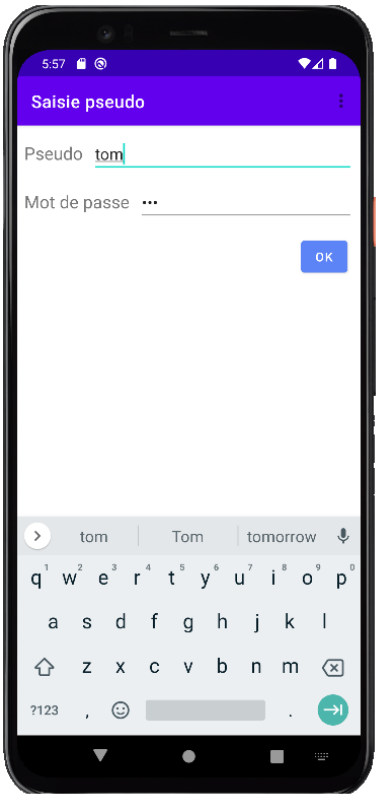
Pour les deux activités précédentes, et pour des raisons de performances, on utilise un RecyclerView pour l'affichage. On définit un "Adapter" et un "ViewHolder" pour les deux classes modèles concernées (ListeToDo et ItemToDo).
Lors de l'ajout d'un nouvel élément, on fait appel à la méthode `notifyItemInserted()` de l'"Adapter" afin de mettre à jour la vue.
L'activité "*ShowListActivity"* a pour particularité de permettre des interactions avec la liste affichée, à savoir, cocher et décocher un item affiché. Pour cela, on ajoute au "checkbox" un listener (`setOnCheckedChangeListener()`), lors du bind() des données de son élément afin de déclencher la mise à jour du status "fait" de l'item et sauvegarder les données.

❖ Toolbar Menu:

On a créé un module intitulé "ToolbarMenu" qui nous permet de mettre en place un "action bar" accompagné d'un Menu ayant comme élément unique le champ "Préférences".
Un simple clic sur ce champ permet de diriger l'utilisateur de n'importe quelle activité vers l'interface de SettingsActivity pour vérifier l'historique de saisie du pseudo.
L'idée était de créer un Toolbar commun à toutes les activités de notre application pour l'ajouter par la suite directement et éviter la redondance des parties de codes.
C'était une initiative d'optimisation du code qui nous a permis de gagner en temps et en performance de l'application.

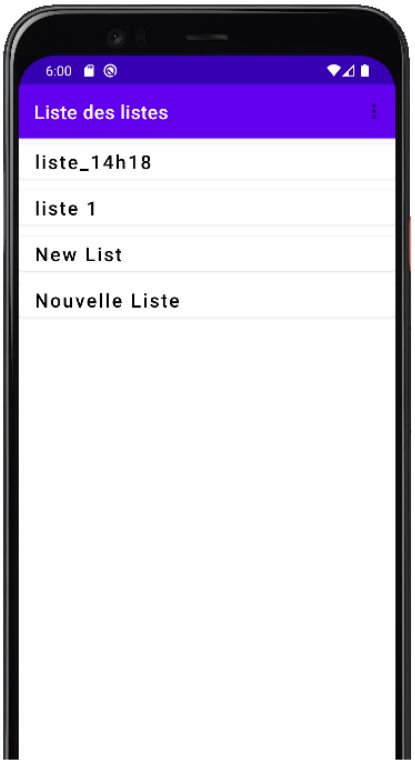
❖ Layouts et Exemple d'exécution:

- *MainActivity:*

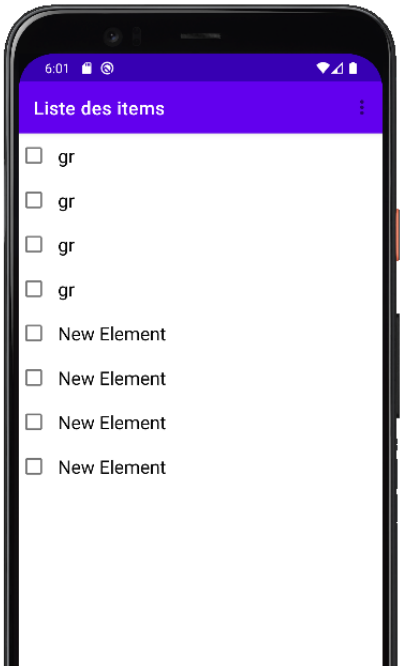


- *SettingsAcitivty:*

- *ChoixListActivity:*



- *ShowListActivity:*



3. Conclusion:

Vers la fin de ce TP, on a réussi à mettre en place une application qui répond aux objectifs fixés au démarrage. On a aboutit à une application fonctionnelle qui permet de gérer les données de multiples utilisateurs avec de nombreuses ToDo listes. On a en plus pu ajouter quelques améliorations par rapport à ce qui est juste attendu de ce travail; on a pu utilisé une “activité générique” qui permet de gérer les diverses méthodes communes et dont la plupart de nos activités héritent.
On a également mis en place un système de récupération de données et de mise en cache fonctionnel à l’aide des puissantes librairies RetroFit et Room.

4. Bibliographie:

- https://www.youtube.com/watch?v=fJEFZ6EOM9o&ab_channel=CodinginFlow
- https://www.youtube.com/watch?v=VE8iMXBLHQc&ab_channel=NgamCode
- https://www.youtube.com/watch?v=oh4YOj9VkVE&ab_channel=CodinginFlow
- https://www.youtube.com/watch?v=DMkzIOLppf4&ab_channel=CodinginFlow
- Converting Kotlin Data Class from JSON using GSON | Baeldung on Kotlin
- Why Are Kotlin Synthetics Deprecated and What Are the Alternatives? | by Sahil Sharma | Better Programming
- Kotlin Android- How implement CheckBox.OnCheckedChangeListener? - Stack Overflow
- What is the Kotlin double-bang (!!) operator? - Stack Overflow-syntax - Kotlin secondary constructor - Stack Overflow
- android - Retrofit 2 with only form-data - Stack Overflow
- sqlite - Android Room SQLite_ERROR no such table - Stack Overflow
- android - Add Header Parameter in Retrofit - Stack Overflow
- android - Retrofit and GET using parameters - Stack Overflow
- Using Retrofit with Kotlin Coroutines in Android (mindorks.com)
- android - Please provide a Migration in the builder or call fallbackToDestructiveMigration in the builder in which case Room will re-create all of the tables - Stack Overflow