Compte rendu travail application TODO Liste Séquence 1 : Justin Sabaté

Introduction :

Bonjour, voici le compte rendu du TP que j'ai réalisé seul, en réutilisant le code que M. Boukadir a développé lors de son cours pour les parties RecyclerView. Deux classes similaires à celle développée lors de ce cours permettent l'affichage soit d'une liste de todolistes ou d'une liste d'items sous formes de checkbox. Je me suis également inspiré de vos screencasts disponibles sur moodle pour la partie menus, settings, json (librairie gson) et utilisation des préférences de l'application.
Vous trouverez ci-dessous l'explication des choix que j'ai fait pour développer cette application, ma conclusion et des perspectives. Bien entendu le code commenté est disponible dans le dépôt git associé à ce document.

Analyse :

Gestion des classes

Le projet a été fait sous kotlin, on n'a donc pas besoin de développer les fonctions getters, setters, toString, etc décrits dans le diagramme donné dans le sujet.
Ainsi mes classes sont plus simples et plus compactes, comme les constructeurs et l'ensemble de ces fonctions sont générés à partir de ce code :

```
class ItemToDo(var description : String = "description", var fait : Boolean = false
)
```

```
class ListeToDo(var titreListeToDo : String,var active:Boolean=false) {
    var listItemsToDo : MutableList<ItemToDo> = mutableListOf()
}
```

```
class ProfilListeToDo( var login : String, var mesListToDo: MutableList<ListeToDo>
= mutableListOf(),var active:Boolean=false)
```

Tags :

Vous trouverez un tag associé à chaque liste sous forme de companion object (singleton) pour permettre le débuggage et d'avoir une vision de l'exécution du code via la console.

Utilisation des préférences de l'application pour stocker les données :

On remarque tout de suite les booléens « active » que j'ai rajouté aux classes ListeToDo et ProfilListeToDo. En effet pour passer les paramètres entre les vues je n'ai pas utilisé de bundle mais j'ai tout stocké à chaque fois dans les préférences de l'application. Ainsi vous retrouverez les fonctions « *loadPrefs* » « *getProfils* » et « *editPrefs* » dans mes quatre activités.
La première charge les préférences, la deuxième extrait l'objet contenant toutes les informations de ces préférences, stocké en json pour le convertir en objet et la troisième réalise l'action inverse à savoir la conversion de l'objet une fois modifié en json puis le stockage dans les péférences.

Elles permettent d'accéder à l'objet nommé *profils* stocké sous en json dans les préférences de l'application. Cet objet est une liste mutable d'objets ProfilListeToDo, qui comportent eux même une liste mutable de ListeToDo qui elles-mêmes contiennent une liste d'ItemToDo. En accédant à l'objet profils on a donc accès aux données de toute l'app.
Le paramètre *active* permet de savoir quel utilisateur est loggé, (active=true) et si on est dans l'activité ShowListActivity, quelle liste a été sélectionnée par cet utilisateur.

On retrouve dans les activités ChoixListActivity et ShowListActivity une fonction *RefreshRecyclerView*.
Elle permet de rafraichir l'affichage en appelant la fonction *getProfils* (c'est à dire en allant chercher l'objet profils dans les paramètres, qui contient toutes les informations sur les profils, les listes et les items qui les composent)

Le fait de ne pas utiliser de bundles a créé un problème principal. En effet, je dois appeler ma fonction de rafraichissement pour afficher les changements. Je dois donc recharger les données sauvegardées pour les afficher. En plus de cela je dois les recharger avant de les comparer ou les modifier. Je pense que ce chargement a pour conséquence de rendre les objets chargés différents en terme d'adresse de stockage. En plus de sûrement dupliquer le stockage, je ne peux donc pas comparer directement les objets (notamment quand je clique, l'objet cliqué (liste ou item) n'est pas exactement le même que l'objet dans la structure chargée). Ne trouvant pas de solution directe à ce problème, j'ai dû comparer les titres des listes et les descriptions des items. Pour éviter que deux objets différents puissent être confondus, j'ai donc interdit la saisie de deux listes avec le même titre ou deux items avec la même description.
Vous trouverez ces spécificités dans les fonctions *addItemToActiveList* de l'activité ShowListActivity et *addListToLoggedUser* de ChoixListActivity.

Gestion des menus :
On trouve un layout pour le menu dans le dossier res/menu.
On vient donc dans chaque activité appeler ce layout en overridant *onOptionsItemSelected* et *onCreateOptionsMenu* qui permettent d'afficher le menu dans chaque activité (sauf settings) et d'appeler l'intent qui va nous permettre de changer d'activité pour aller vers *SettingsActivity* si on clique sur le menu settings.

Gestion des recyclerviews :
J'ai développé deux recyclerviews qui prennent las layouts item et item_list pour les afficher sous forme de recyclerview. Les adapters ItemAdapter et ListAdapter permettent de réaliser les fonctions de base de ces recyclerviews, notamment l'inflate des textes et de l'état de la checkbox pour ItemAdapter.

Conclusion :
Globalement je dirais que le stockage dans les préférences de l'application à chaque étape au lieu de passer l'utilisateur et la liste actifs en bundles m'a compliqué la tâche. À l'avenir je ne stockerai dans ces données que pour pouvoir fermer et rouvrir l'application.
J'ai cependant pu créer l'application demandée avec un ajout personnel qui est que l'on ne peut pas créer deux listes identiques ou deux items identiques (au sein d'une même liste) pour un même utilisateur. Cela peut s'expliquer en terme d'expérience utilisateur, mais

j’aurais du utiliser les bundles pour que l’utilisateur puisse saisir deux fois les mêmes données.

Perspectives :

Une amélioration serait d’effectivement développer la version utilisant des bundles, je pense qu’elle simplifierait les choses à plusieurs endroits dans le code. J’ai cependant montré que c’était faisable avec cette structure de données unique (n’utilisant pas à la fois les préférences et des bundles).