Le Guillou Eve
Thomas Iona

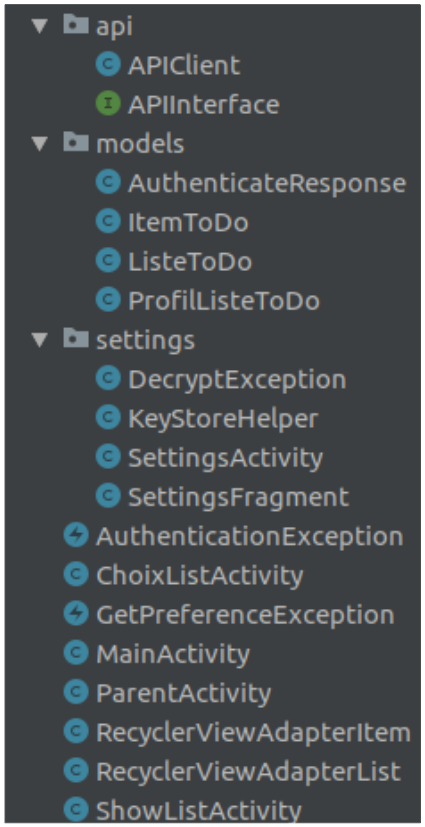
Compte-rendu Application TodoList
Séquence 2

Introduction

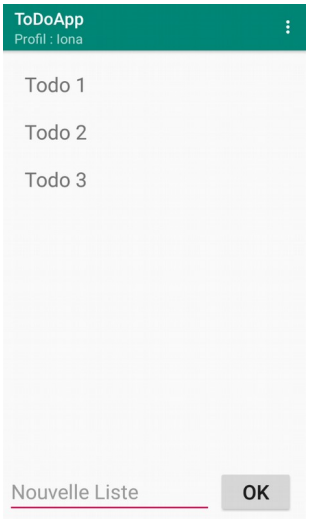
Le mini-projet de cette séquence était le développement d'une application android en Java permettant de gérer des todo listes associées à des profils d'utilisateur, en faisant appel à une API REST. Nous avons choisi d'utiliser la bibliothèque Retrofit pour implémenter les appels à l'API.

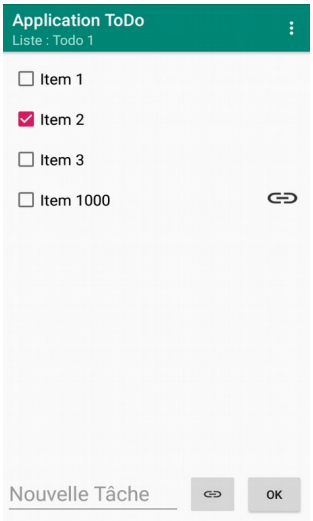
En résumé, on dispose d'une activité MainActivity sur lequel l'utilisateur rentre son pseudo et son mot de passe. L'application lance un appel à l'API pour vérifier si l'utilisateur existe et si le mot de passe correspond. Si oui, l'API renvoie un hash, et l'utilisateur passe à l'activité ChoixListActivity. Cette activité affiche les listes correspondant au profil. On peut en ajouter ou aller sur une liste déjà existante. Dans l'activité « ShowListActivity » on affiche les tâches à faire incluses dans la liste et leur état (faites ou non). Enfin une « SettingsActivity » est accessible à tout instant en utilisant le menu de l'actionBar.

Dans nos sources on dispose donc des classes suivantes qu'on détaillera dans l'analyse :



Les différentes activités ont pour apparence :





MainActivity ChoixListActivity ShowListActivty SettingsActivity

Afin de remplacer le logo par défaut et de rendre notre application plus identifiable nous avons mis en place ce logo :

Fonctionnalités mise en place dans la séquence 2 :

- Vérification de l'accès internet

- Les données proviennent désormais de l'API. Elles sont récupérées grâce à Retrofit.

- L'authentification s'effectue automatiquement dans la mesure du possible (préférences présentes et non erronées)

- Option de déconnexion dans le menu de l'actionBar.

- Chiffrement du mot de passe lorsqu'il est stocké dans la bbd à l'aide de l'AndroidKeyStore.

- Ajout de nouvelles listes et de nouveaux items, y compris avec url

- Intent implicite pour ouvrir les liens associés à certains items

1. Analyse

i. ParentActivity

Dans un effort de factorisation du code, on crée une nouvelle activité dont vont hériter toutes les autres activités sauf SettingsActivity. On y regroupe ainsi toutes les opérations et déclarations de méthodes effectuées à répétition, notamment :
- Les méthodes et attributs permettant de gérer le menu dans l'ActionBar et l'actionBar.
- La méthode alerter permettant de créer à la fois un log et un Toast pour un message.
- La méthode recupPreference qui permet de récupérer une préférence
- La méthode disconnect qui retire toutes les préférences liées à un utilisateur (pseudo/mot de passe / hash)

ii. MainActivity (activité de connexion)

MainAcitvity est l'activité servant à la connexion. La première fois on y entre son pseudo et mot de passe. Ceux-ci sont sauvegardés dans les préférences. On stockera également le hash à chaque nouvelle connexion.

Le mot de passe est stocké une fois hashé grâce à une paire de clé gérée par le AndroidKeyStore. La protection de ce mot de passe a été implémentée en se fondant sur un article disponible dans la bibliographie. Le code correspondant à l'interaction avec l'AndroidKeyStore se trouve dans *KeyStoreHelper* dans le packet settings. Il a été récupéré et adapté aux besoins de l'application notamment en ajoutant/modifiant la gestion des exceptions. On évite ainsi de stocker un mot de passe en clair dans les préférences. Dans une autre version, on pourrait essayer d'utiliser les EncryptedSharedPreferences disponibles en version alpha (cf bibliographie).

Si un utilisateur est enregistré dans les préférences, l'application essaiera de l'authentifier automatiquement et passera à l'activité suivante.

Cette activité est ensuite retirée de la pile des activités, puisqu'on considère que l'utilisateur préférera ne pas repasser par celle-ci. Pour se déconnecter et retrouver l'activité de connexion, une option déconnexion dans le menu de l'actionBar et un bouton « Log out » dans les préférences permettrons de vider les préférences utilisateurs.

Si internet n'est pas disponible, l'activité mainActivity apparaîtra avec un bouton de connexion désactivé.

iii. SettingsActivity

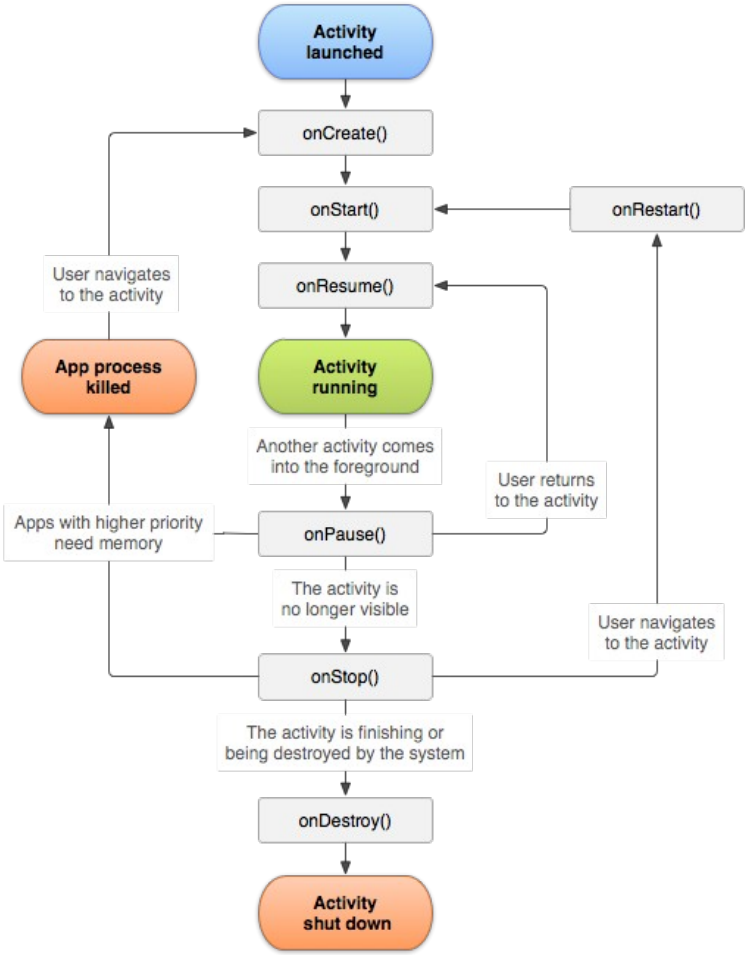
Cette activité est accessible grâce au menu de l'actionBar. On peut accéder au menu Préférences, on peut entrer son pseudo, son hash d'identification et l'URL de base de l'API. Entrer un pseudo ou un hash à la main risque de casser l'authentification.

Un bouton log out est également disponible pour changer proprement d'utilisateur.

iv. ChoixListActivity

Depuis cette activité, on peut accéder aux Listes Todo déjà créées et en ajouter de nouvelles.

Cette activité peut être chargée lorsqu'on arrive de MainActivity, auquel cas elle récupère les informations du profil depuis les préférences. Elle peut également être « chargée » lorsqu'on revient de « ShowListActivity ».



OnCreate n'est appelée qu'à la création de l'activité alors que onStart sera appelée à chaque fois que l'activité est chargée, donc on fait la majorité de nos actions dans onStart, ainsi quand on reviendra de ShowListActivities, elle s'exécutera ce qui permet notamment de charger le profil à chaque fois pour ne pas perdre d'informations. Dans onStart, on va d'abord créer et mettre en place l'adaptateur du RecyclerView, vide, puis on va lancer l'appel à l'API pour récupérer les listes et, en cas de succès de l'appel, on met à jour le RecyclerView. Cette manière de faire permet de ne pas avoir à recréer un RecyclerView Adaptateur à chaque appel à l'API.

Lorsqu'on crée une nouvelle liste, on fait un appel à l'API pour l'ajouter, puis on récupère les listes à nouveau, on vide le champ texte pour une meilleure expérience utilisateur, et on notifie l'adaptateur qu'il y a eu des changements ce qui lui permet de recalculer la nouvelle vue.

En cliquant sur une des listes, on ouvre une activité ShowListActivity en lui passant l'identifiant de la liste et son titre dans un Bundle.

v. ShowListActivity

Depuis cette activité, on peut accéder aux tâches déjà créées et en ajouter de nouvelles. Les tâches sont affichées grâce à un RecyclerView dont l'adaptateur est défini dans RecyclerViewAdapterItem. Lors de onCreate, comme expliqué précédemment dans ChoixListActivity, on initialise le recyclerView et, selon l'orientation, le layout manager, puis on fait un appel à l'API pour récupérer les tâches en transmettant à l'API l'identifiant de la liste, pour ainsi accéder aux tâches de la ToDo dans laquelle on est. Chaque tâche est associée à une checkbox, laquelle peut être cochée ou décochée en cliquant dessus.

vi. SettingsActivity

Cette activité permet d'afficher les préférences de l'utilisateur, ici limitées à son pseudonyme, à son hash et à l'URL de base de l'API. On peut accéder à cette activité depuis toutes les autres activité de l'application.

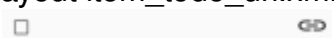
Elle utilise un fragment pour les préférences qui hérite de PreferenceFragment-Compat. Cette dernière classe est spécialisée pour la gestion des préférences à l'aide de fragments. Le fragment est ajouté statiquement dans le layout activity_settings.xml.

vii. RecyclerViewAdapterList

Cette classe n'a pas été modifiée par rapport à la séquence 1.

vii.bis RecyclerViewAdapterItem

On a dorénavant des items qui peuvent être avec ou sans url. On doit donc adapter l'affichage selon la présence ou non d'un url.
Pour ceci on a créer un nouveau layout item_todo_url.xml :



Il est constitué d'une checkbox avec son texte et d'une icône de lien sur lequel on peut cliquer.

Pour pouvoir gérer cette différence on utilise la méthode :

```
public int getItemViewType(int position) {
    if (null == itemToDos.get(position).getUrl()) {
        return ITEM_WITHOUT_URL;
    }
    return ITEM_WITH_URL;
}
```

… qui identifie le type de vue associée à un élément.

On l'utilise implicitement (le viewType est passé en paramètre) dans onCreateViewHolder pour inflate des layouts différents selon le viewType (qui varie selon si l'élément dispose ou non d'un url). On crée également une sous-classe UrlItemViewHolder de ItemViewHolder qui permet de gérer l'imageView rajouté pour les liens.

Enfin dans le onBindViewHolder, si l'item a une url, on ajoute un onClickListener sur l'imageView pour déclencher un intent implicite pour ouvrir une page web.

viii. Internationalisation

Ces éléments n'ont pas été modifiés par rapport à la séquence 1.

iv. models package

On rassemble dans ce package les models nous servant à représenter nos objets et servant à Retrofit à parser les réponses dans l'API. On y retrouve :

- AuthenticateResponse

Cette classe est utilisée lors de l'appel à l'API permettant l'authentification, elle possède comme attribut les éléments de la réponse de l'API, ce qui permet à Retrofit de parser la réponse sous forme d'un objet utilisable.

- ItemToDo

Classe servant à représenter les items. On notera l'ajout d'un attribut url et la conversion du booléen fait en un entier par rapport à la version précédente pour se conformer aux données renvoyées par l'API. IsFait a donc été repensé pour renvoyer un booléen à partir de l'entier récupérer par Retrofit.

- ListeToDo

Classe représentant les listes. On a fait attention à ce que getLesItems ne renvoie pas de null même si Retrofit n'a pas initialisé lesItems.

- ProfilToDo

Classe représentant un profil utilisateur avec une liste de ListToDo.

Classe servant également à parser les réponses renvoyant la liste de ListToDo d'un utilisateur.

x. API Client

Cette classe permet d'instancier le service qui va contacter l'API utilisant la bibliothèque Retrofit. Elle implémente la méthode permettant de créer le service.

xi. API Interface

Cette interface permet de déclarer des méthodes d'appel à l'API, selon une écriture propre à Retrofit. On y déclare les routes de l'API et leurs paramètres.

Conclusion

Conduit de bout en bout, cet exercice permet de manipuler et mettre en œuvre la bibliothèque Retrofit pour manipuler des API. Nous avons dû repenser les classes afin qu'elles correspondent aux réponses de l'API, notamment pour gérer les checkbox, dont l'état n'était pas traduit par un booléen, mais par un 0 et un 1. L'ajout d'URL aux checkbox nous a aussi permis d'apprendre à gérer un RecyclerView avec différents type de

ViewHolder (un pour les items sans lien et un pour les items avec liens). Nous avons aussi mis en place un système permettant de gérer plus proprement les erreurs au niveau des préférences, en effet, lorsque celles-ci étaient vides, les importations dans les activités renvoyaient des messages d'erreur, mais ceux-ci n'étaient pas reconnus comme tels, et seulement comme chaînes de caractères. L'implémentation de l'exception permet ainsi à l'activité de produire une erreur lorsqu'il y a un problème. Nous avons aussi implémenté deux autres types d'exception, une pour gérer les problèmes d'authentification, et une pour gérer les problèmes de déchiffrement.

Cet exercice nous a aussi permis de réfléchir à la structure d'une activité, notamment lors des interactions entre l'adaptateur du RecyclerView et les appels API. Nous avons aussi pu apprendre à mettre en place un Intent implicite et à stocker un mot de passe chiffré.

Perspectives

Il subsiste encore de nombreuses perspectives d'amélioration. Comme on l'avait mentionné dans le premier compte rendu, on peut ajouter une classe intermédiaire pour factoriser les méthodes onCreate des activités ShowListActivty et ChoixListActivity, ainsi réduire encore le code, sans surcharger MainActivity. On peut aussi penser à pouvoir masquer et supprimer les tâches déjà faites. L'historique des pseudo n'a pas été mis en place.

Dans les nouvelles améliorations qui s'offrent à nous on peut mentionner l'amélioration de la connexion automatique. Celle-ci s'effectue pour l'instant sur la MainActivity qui devient parfois visible lorsque que l'authentification implicite à l'aide des préférence prend trop de temps. On pourra également vérifier que l'appareil dispose d'une application permettant de répondre à l'intent implicit permettant d'ouvrir une page web à l'aide de resolveActivity(). On pourrait éventuellement améliorer l'interface utilisateur en utilisant un bouton d'action flottant (cf bibliographie) plutôt que l'interface actuelle. On pourrait aussi pouvoir mettre en place un système d'inscription, pour créer de nouveaux utilisateurs.

Bibliographie

Manipuler Retrofit :
https://guides.codepath.com/android/consuming-apis-with-retrofit
https://square.github.io/retrofit/
Cours de M. Boukadir :
https://github.com/pmr2019/sequence2/branches
Faire un recyclerView avec différents types de lignes :
https://stackoverflow.com/questions/25914003/recyclerview-and-handling-different-type-of-row-inflation
Chiffrer les mots de passe :
https://medium.com/@ali.muzaffar/securing-sharedpreferences-in-android-a21883a9cbf8
https://developer.android.com/topic/security/data
Intent Implicit :
https://developer.android.com/guide/components/intents-filters#ExampleSend