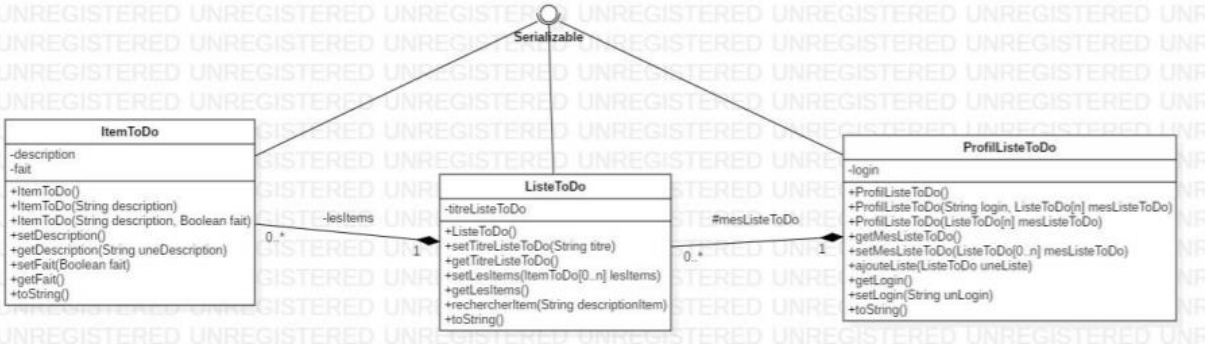
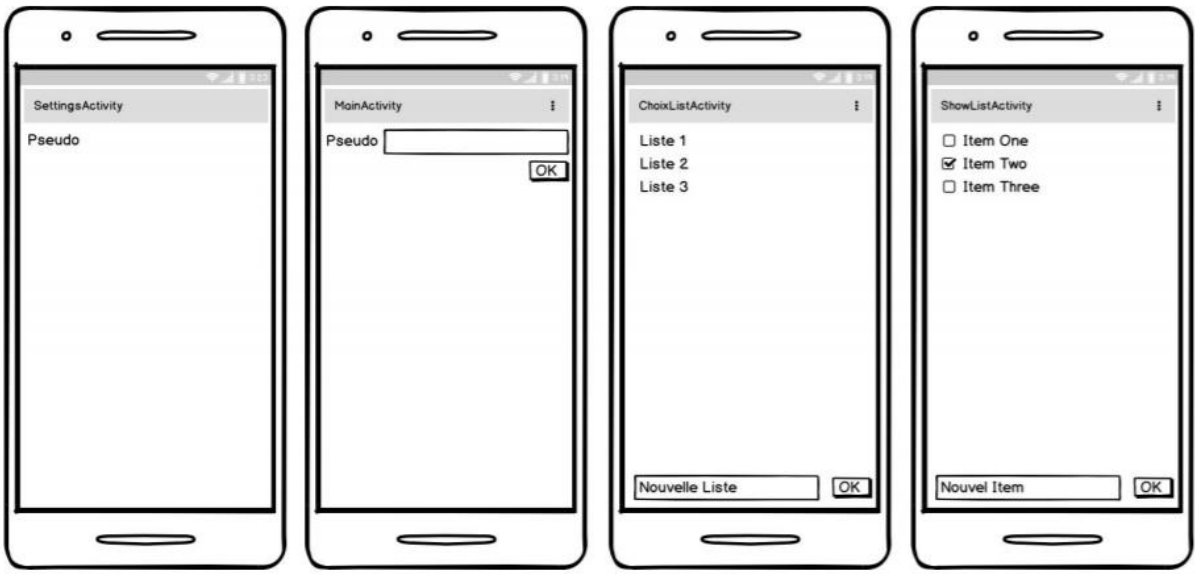
Compte rendu application ToDoList séquence 2

Introduction

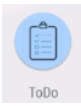
Ce projet a pour objectif l'élaboration d'une application ToDoList sur Android avec Java. **Les données sont stockées sur un serveur. On fera les download et upload via une API.**

Pour rappel, voici le diagramme de classe de l'application et les mockups des pages à afficher :





Par rapport aux consignes énoncées, voici les choix techniques :

- Les profiles des utilisateurs sont stockés dans des fichiers au format json.
- Un bouton Suppr. pour supprimer des items a été ajouté à la ShowListActivity
- L'icone de l'application est celle-ci : 

- Pour les deux recyclerView de choixListActivity et ShowListActivity, on a utlisé deux RecyclerViewAdapter différents (pas d'heritage).
- **Pour les appels à l'API, on utilise la librairie Retrofit.**

Analyse

Classes représentant les données

On dispose de trois classes pour représenter les données de l'application :

- La classe ItemToDo qui représente un item d'une liste
- La classe ListeToDo qui représente une liste
- La classe ProfileListeToDo qui représente un utilisateur

Le lien entre ses classes est détaillé dans le diagramme de classes présenté en introduction.

Modeles

Par rapport à la séquence précédente, on a implémenté 3 classes : une pour vérifier l'accès à internet de l'application InternetCheck, une contenant les fonctions d'appels à l'API TodoInterface, et une représentant les données que l'on peut récupérer avec les appels à l'API RetroMain.

InternetCheck : Cette classe créer un socket et tente de se connecter à google.com (ip 8.8.8.8). Si la connexion réussie, c'est que l'on est connecté à internet. Pour utiliser des sockets, on doit créer un nouveau thread. C'est pourquoi pour utiliser cette classe, l'objet qui instance InternetCheck doit contenir la méthode isConnectedToInternet(Boolean internet) qui sera appelé après la tentative de connexion à google.com. Dans ce projet, isConnectedToInternet est utilisé dans MainActivity qui définie isConnectedToInternet comme suit :

```
@Override
public void isConnectedToInternet(Boolean internet) {
    if (!internet){
        btnOk.setEnabled(false);
        Toast.makeText( context: this,  text: "Aucune connexion internet.", Toast.LENGTH_SHORT).show();
    } else {
        btnOk.setEnabled(true);
        Toast.makeText( context: this,  text: "Connexion internet établie.", Toast.LENGTH_SHORT).show();
    }
}
```

TodoInterface : On a beaucoup utilisé Postman pour écrire cette classe. Rien d'exceptionnel, on crée une fonction pour chaque appel @Get ou @POST etc. Les retours de ces fonctions sont des instances de la classe RetroMain. De plus si l'on a une liste contenant d'autres objets dans le json reçu par l'API, par exemple lorsque l'on reçoit la liste des utilisateurs, l'instance de RetroMain reçu crée une liste d'objets RetroMain également représentant la liste des utilisateurs. Cela permet de n'avoir qu'un modèle de donnée, mais demande à connaitre parfaitement le résultat de la requête http, d'où l'utilisation de Postman.

RetroMain : Définit les variables retournées par les requêtes http. Voir TodoInterface ci-dessus pour plus de détails. Les variables définies doivent être sérialisable d'où le @SerializedName(« nomvariable »).

Activities

- MainActivity

Est l'activité de démarrage de l'application. Elle permet de saisir un pseudo et un mot de passe pour se connecter à l'API. Si la connexion réussie, on passe à ChoixListActivity.

Au lancement de l'application, s'il n'y a pas de connexion internet, le bouton ok est désactivé, rendant les tentatives de connexion impossibles.

- SettingActivity

Préférences de l'application. Elle comporte dorénavant deux préférences : le dernier pseudo saisi par l'utilisateur et le hash permettant d'effectuer des appels à l'API en tant qu'utilisateur connecté. Ce hash est initialisé à la fin d'une connexion réussie dans MainActivity.

- ChoixListActivity

On affiche les listes du profil dans un RecyclerView. Un clic sur une liste permet d'accéder à ShowListActivity.

Par rapport à la séquence précédente, on a juste changé la façon de sauvegarder et d'accéder aux données du profil ProfilListToDo. Maintenant, il n'y a plus de fonction saveData(), mais juste une fonction de chargement des données depuis l'API. Avant, on sauvegarder tout le profil dans un fichier. Maintenant on sauvegarde chaque action, en envoyant la modification à l'API. Donc les sauvegardes ne se font plus avec une fonction saveData() mais à chaque addListeToDo. Un ProfilListeToDo est tout de même crée et permet de bien gérer le recyclerView.

- ShowListActivity

Cette activité charge les items de la liste préalablement sélectionné grâce au même fichier utilisé précédemment. On analyse ensuite chaque item pour savoir si oui ou non il a été coché précédemment et on le coche si oui.

Les modifications par rapport à la séquence précédente sont similaires à celles de ChoixListActivity.

- RecyclerViewAdapterList & RecyclerViewAdapterItem

Sont les deux adaptateurs des recyclerView de ChoixListActivity et SHowListActivity respectivement.

Pour initialiser les items cochés, la fonction se trouve dans le onBindViewHolder() de RecyclerViewAdapterItem.

Pas de modifications par rapport à la séquence précédente.

Conclusion

Changer la localisation des données ne devrait impliquer de modifier uniquement les fonctions de chargement et de sauvegarde des données. Entre les deux séquences, d'autres modifications ont du être faites ce qui montre les limites de flexibilité du code généré. Cependant l'approche du problème n'était pas la même et avec un peu plus de recul à la séquence 1, on aurait pu faire en sorte d'avoir cette flexibilité sans trop de mal.

On peut penser à implémenter une fonction save(action) qui permettrait de sauvegarder les modifications liées à l'action réalisée, ce qui éviterait de devoir gérer les sauvegardes à chaque opérations sur le profil utilisateur.

Bibliographie

- RecyclerView : https://www.youtube.com/watch?v=Vyqz_-sJGFk&t=499s
- Ecriture/Lecture de fichiers sur android : https://support.google.com/docs/answer/49114?co=GENIE.Platform%3DAndroid&hl=fr
- Debug : stackoverflow.com
- Utilisation de retrofit : https://medium.com/@prakash_pun/retrofit-a-simple-android-tutorial-48437e4e5a23