

RAPPORT TP2/3 PMR

Marie LENGLET

Pablo SKEWES

I) Création d'un user sur Postman	2
II) Séquence 2	3
II-A) Retrofit 2	3
II-B) Les activités	4
II-B-1) SettingsActivity	4
II-B-2) MainActivity	5
II-B-3) ChoixListActivity	6
II-B-C) ShowListActivity	6
II-C) Exemples avec d'autres users	8
III) Séquence 3	10
III-A) Principe	10
III-B) Bases de données	11
III-C) Résultats	12
Conclusion	12

I) Création d'un user sur Postman

Sur postman, on crée l'utilisateur "marie" (login="marie" et password="marie") :



Cet utilisateur a 4 listes : "list_test_marie", "todo list marie 1", "todo list marie 2" et "todo list pablo 3".

Ces listes ont les items suivants :

- la liste "list_test_marie" d'identifiant 1189



- la liste "todo list marie 1" d'identifiant 1190



- la liste "todo list marie 2" d'identifiant 1191



- la liste "todo list pablo 3" d'identifiant 1192



II) Séquence 2

II-A) Retrofit 2

Nous avons utilisé **Retrofit 2** pour gérer les requêtes auprès de l'API.

Voici la documentation sur Retrofit : <https://square.github.io/retrofit/>

Voici un exemple de requête POST avec Retrofit 2 :

```
@POST("lists/{id_list}/items")
fun postItem(
    @Path(value = "id_list", encoded = true) idList : Int,
    @Query(value = "label", encoded = true) label : String,
    @Query(value = "url", encoded = true) url : String,
    @Header("hash") hash: String,
) : Call<Item>
```

Cette requête POST crée un Item dont la description est "label" dans la liste dont l'identifiant est "id_list".

Pour faire cette requête, on a besoin du hash, qui est stocké dans les sharedpreferences (fichier appelé sharedPrefTokens) dès l'authentification du début.

Dans les activités, lorsqu'on fait un appel à l'API, on le fait dans un thread différent du main grâce aux coroutines.

Voici un exemple avec l'authentification dans la MainActivity :

```

129     private fun authenticate(user: String, pass: String) {
130         CoroutineScope(context = SupervisorJob() + Dispatchers.Main).launch{ this: CoroutineScope
131             try {
132                 val response: Response<AuthenticationResponse> = api.authenticate(user, pass)
133                 Log.e(TAG, response.toString())
134                 if (response.isSuccessful) {
135                     val data: AuthenticationResponse = response.body()!!
136                     //Log.d(TAG, data.toString())
137                     Log.e(TAG, msg: "HAAAASH : " + data.hash)
138                     sharedPrefTokens.edit().putString("token", data.hash).apply()
139                     sharedPrefTokens.edit().putString(user, data.hash).apply()
140                     canLogin = true
141                     return@launch
142                 } else {
143                     Log.e(TAG, msg: "INVALID USER")
144                     return@launch
145                 }
146             } catch (e : Exception){
147                 Log.e(TAG, msg: "Exception found :\n $e")
148                 return@launch
149             }
150         }
151     }

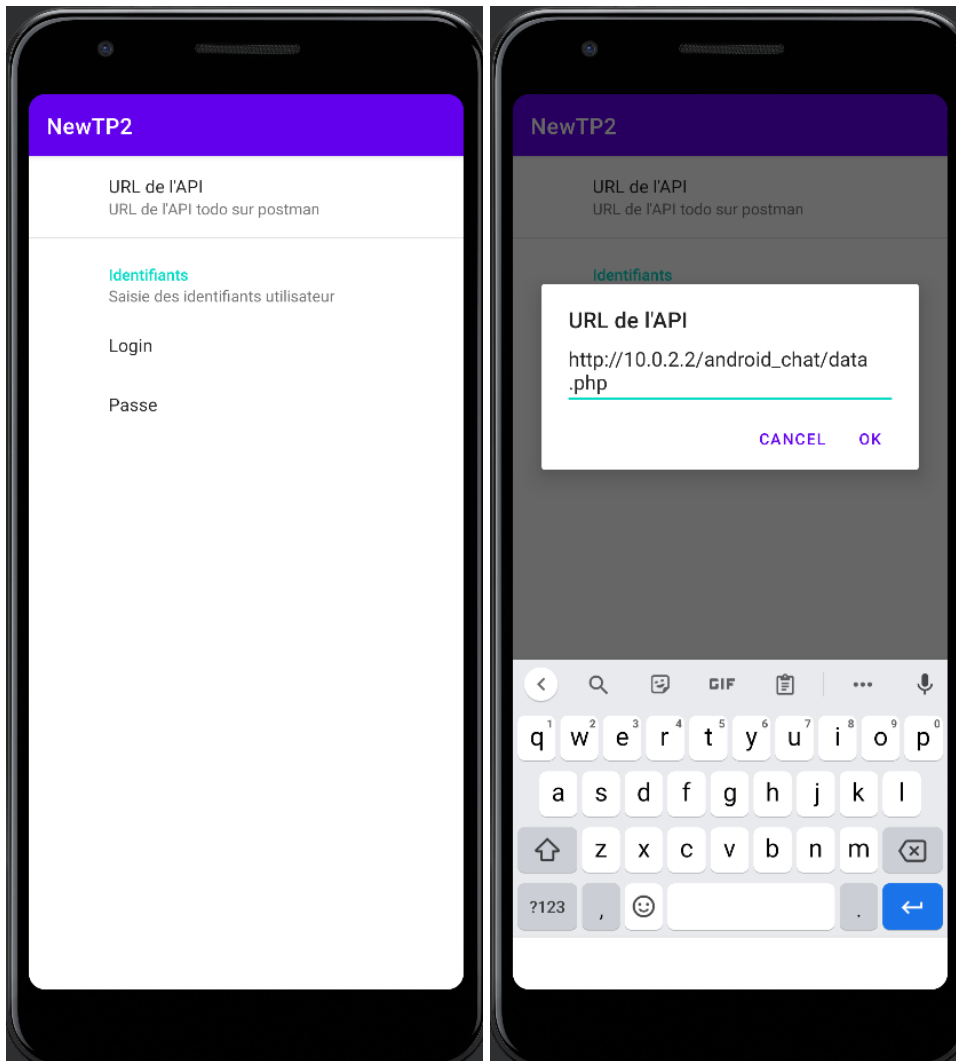
```

II-B) Les activités

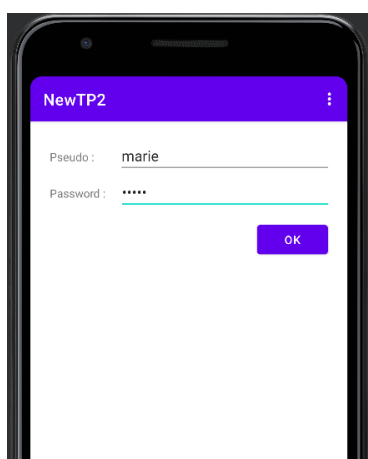
Nous avons réussi à faire la majorité de ce qu'il nous était demandé.

II-B-1) SettingsActivity

Voici la SettingsActivity, contenant l'URL de base :



II-B-2) MainActivity



Voici la MainActivity :

- Le dernier pseudo entré est automatiquement renseigné.

- Le bouton “OK” fonctionne que si on a renseigné un mot de passe, et si le couple pseudo-password correspond à un user de l’API.
- Lorsqu’on appuie sur le bouton OK, le clavier disparaît et le texte est cleared.
- Attention, le bouton OK doit parfois être cliqué 2 fois pour bien fonctionner. Nous n’avons pas trouvé d’où vient ce léger bug. Il faut donc parfois un petit peu insister sur le bouton OK.

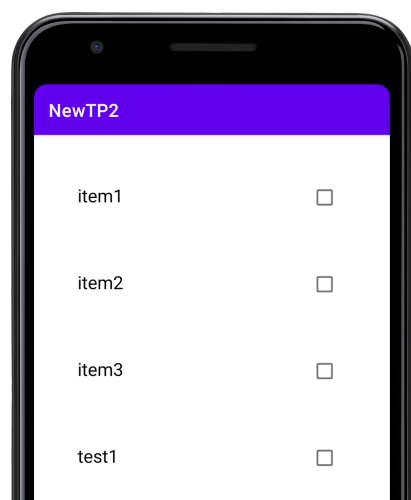
II-B-3) ChoixListActivity



On voit bien que les 4 listes de l’user “marie” s’affichent correctement.

Lorsque l’on clique sur une liste, on est redirigé vers ShowListActivity, où l’on peut voir les items appartenant à la liste cliquée.

II-B-C) ShowListActivity



La ShowListActivity fonctionne bien.

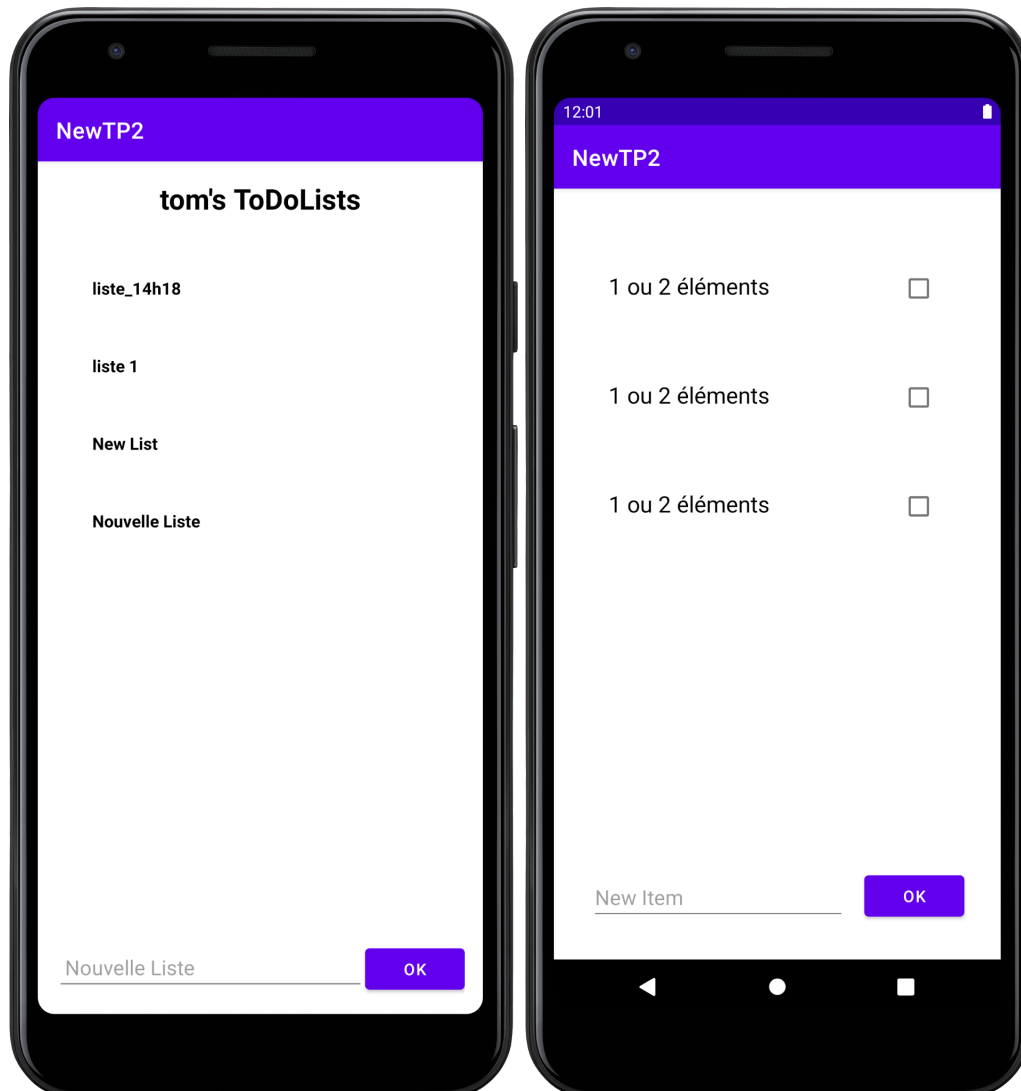
Il est possible de créer un nouvel item en bas, grâce à une requête POST.

Cependant, parfois le label du nouvel item créé s’affiche mal. Il suffit alors de retourner à l’activité d’avant, puis de

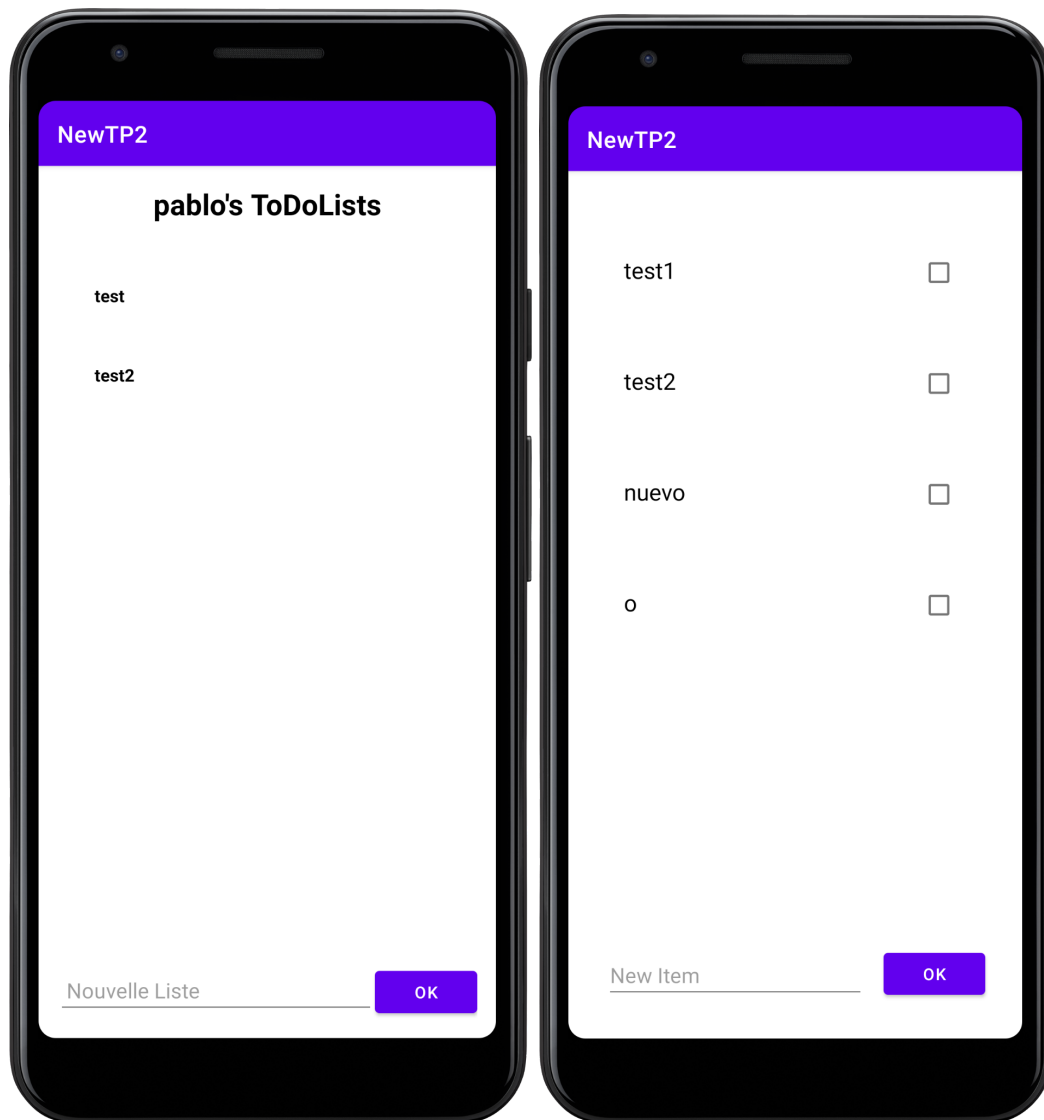
recliquer sur la liste concernée. On voit alors l'item apparaître normalement.

II-C) Exemples avec d'autres users

Voici un exemple avec l'utilisateur **tom** (password = web), puis la liste "liste 1" :



Voici un exemple avec l'utilisateur **pablo** (password = pablo), puis la liste "test2" :



III) Séquence 3

III-A) Principe

On utilise des threads différents du main.

On définit d'abord la base de données :

```
1 package com.example.newtp2.bdd
2
3 import ...
4
5
6
7
8
9
10 @Database(entities = [TodoList::class, Item::class, User::class], version = 1)
11 abstract class DataBase : RoomDatabase() {
12
13     abstract fun listDao() : ListDao
14
15 }
```

On modifie également les classes du modèle, et on crée un Dao, qui sert pour la gestion de l'accès de la bdd.

On adapte nos fonctions pour que si l'appel à l'API ne fonctionne pas (à cause de l'absence de réseau), alors on fait appel à la base de données (try-catch).

Voici un exemple avec la fonctions getItems :

```
135 private fun getItems(){
136     val hash = sharedPrefTokens.getString( key: "token", defValue: "")
137     //val id_list = sharedPrefTokens.getInt("id_list",1991)
138     CoroutineScope( context: SupervisorJob() + Dispatchers.Main).launch{ this: CoroutineScope
139         try {
140             val response: Response<ItemResponse> = RetrofitInstance.api.getItems(idList, hash!!).awaitResponse()
141             Log.d(TAG, msg: "response found : \n " + response.toString() + " hash " + hash)
142             if (response.isSuccessful) {
143                 val data: ItemResponse = response.body()!!
144                 Log.d(TAG, data.items.toString())
145                 val itemsFound = data.items
146                 for (item in itemsFound) item.idList = idList
147                 saveItems(itemsFound)
148                 Log.e(TAG, msg: "todolists found : \n " + itemsFound.toString())
149                 withContext(Dispatchers.Main) { this: CoroutineScope
150                     for (newItem in itemsFound) {
151                         myitems.add(newItem)
152                         rvItems.adapter!!.notifyItemInserted( position: myitems.size - 1)
153                     }
154                 }
155                 adapter.display(itemsFound)
156             }
157         }
158     }
159 }
```

```

157     } catch(e : Exception){
158         val itemsFound = listDao.getItems(idList)
159         for (item in itemsFound) item.idList = idList
160         //saveItems(itemsFound)
161         Log.e(TAG, msg: "todolists found : \n" + itemsFound.toString())
162         withContext(Dispatchers.Main) { this: CoroutineScope
163             for (newItem in itemsFound) {
164                 myItems.add(newItem)
165                 rvItems.adapter!!.notifyItemInserted( position: myItems.size - 1)
166             }
167             adapter.display(itemsFound)
168         }
169         Log.e(TAG, msg: "Exception found : \n $e")
170         withContext(Dispatchers.Main){ this: CoroutineScope
171             Toast.makeText(applicationContext, text: "ça marche pas", Toast.LENGTH_LONG).show()
172         }
173     }
174 }
175 }

```

III-B) Bases de données

On peut voir que les bases de données se remplissent bien :

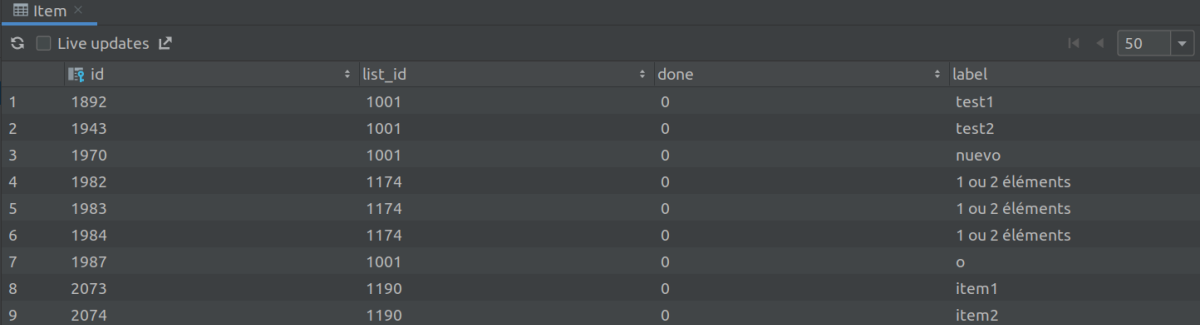
TABLE USERS :

User ×		
<input type="checkbox"/> Live updates ↗		
	id	pseudo
1	1	tom
2	2	isa
3	380	srp_ali
4	381	toto
5	383	xuran
6	384	flo
7	385	luiz_math
8	386	colyan
9	387	pablo
10	388	maxime

TABLE TODOLISTS :

ToDoList ×			
<input type="checkbox"/> Live updates ↗			
	id	idUser	label
1	1189	403	list_test_marie
2	1190	403	todo list marie 1
3	1191	403	todo list marie 2
4	1192	403	todo list pablo 3

TABLE ITEMS :



	id	list_id	done	label
1	1892	1001	0	test1
2	1943	1001	0	test2
3	1970	1001	0	nuevo
4	1982	1174	0	1 ou 2 éléments
5	1983	1174	0	1 ou 2 éléments
6	1984	1174	0	1 ou 2 éléments
7	1987	1001	0	o
8	2073	1190	0	item1
9	2074	1190	0	item2

III-C) Résultats

Notre application fonctionne bien off-line, à l'exception d'un léger bug : MainActivity envoie l'idUser à ChoixListActivity grâce à un intent. Ce idUser sert dans ChoixListActivity à sélectionner remplir correctement la base de donnée TODOLIST (chaque todolist a un attribut idUser).

Cependant, L'idUser est récupéré dans une coroutine de la fonction authenticate, et parfois cette coroutine met pas mal de temps à finir. Dans ce cas, la coroutine n'a pas le temps de setter la valeur idUser avant qu'elle ne soit utilisée dans ChoixListActivity. Dans ce cas, l'idUser envoyé n'est pas le bon.

Nous avons essayé de régler ce bug mais nous n'avons pas réussi.

Cependant, pour tester notre mode offline, nous avons testé avec l'user "marie", en passant à ChoixListActivity le bon identifiant de marie (idUser=403). Dans ce cas là, **lorsque le bon idUser est envoyé, le mode offline fonctionne parfaitement, et l'on a accès aux bonnes listes et bons items.**

Conclusion

Pour conclure, nous avons beaucoup appris lors de cet électif et avec ce TP.

Notre application fonctionne globalement bien, online comme offline.

Cependant, il reste des pistes d'amélioration. Il faut notamment régler le bug de idUser évoqué en partie III. On pourrait également faire en sorte de pouvoir ajouter des

