Compte rendu séquence 2 & 3

Séquence 2

Pour cette séquence, les premières modifications portaient sur la page d'accueil. Un nouveau champ *password* est apparu et le bouton *OK* n'est disponible que si le téléphone est connecté à Internet. Grâce à ce nouveau champ, il est possible de se connecter avec un compte sur *l'API RESTful* http://tomnab.fr/todo-api/.

Après la connexion, au lieu de de récupérer les informations stockées dans le *JSON* sur le téléphone, on récupère les *ToDo Lists* de l'utilisateur grâce à l'*API*, on recréer notre modèle à partir des informations et on affiche les *ToDo Lists* comme dans la séquence précédente.

La fonctionnalité principale à développer est de faire fonctionner le cochage des tâches avec *l'API*. Pour cela, on réalise la requête *PUT* dans la fonction *toggleCheckbox()* des tâches.

```
String url = "http://tomnab.fr/todo-api/lists/" + this.toDoListJsonId +
"/items/" + this.JsonId + "?check=" + (this.isDone ? 1 : 0);

StringRequest request = new StringRequest(Request.Method.PUT, url, null,
null) {
    @Override
    public Map<String, String> getHeaders() {
        Map<String, String> params = new HashMap<String, String>();
        params.put("hash", hash);

        return params;
    }
};
```

La seconde fonctionnalité principale consiste à pouvoir ajouter une tâche. Ici, la requête est un peu plus complexe car il s'agit également de récupérer l'*id* de la tâche en réponse de la requête.

Voici la réponse en *JSON* d'une requête : *POST* http://tomnab.fr/todo-api/lists/237/items?label=test


```
{
  "version":1,
  "success":true,
  "status":201,
  "item":{
    "id":"981",
    "label":"test",
    "checked":"0",
    "url":null
  }
}
```


On voit que l'*id* se trouve dans l'objet *item* de la réponse.

```
String url = "http://tomnab.fr/todo-api/lists/" + this.JSONid +
"/items?label=" + name;

JsonObjectRequest request = new JsonObjectRequest
        (Request.Method.GET, url, new Response.Listener<JSONObject>() {

            @Override
            public void onResponse(JSONObject response) {
                try {

task.setJsonId(Integer.valueOf(((JSONObject)response.get("items")).get("id"
).toString()));
                } catch (JSONException e) {
                    e.printStackTrace();
                }

            }
        }, new Response.ErrorListener() {

            @Override
            public void onErrorResponse(VolleyError error) {
                // TODO: Handle error

            }
        }) {
    @Override
    public Map<String, String> getHeaders() {
        Map<String, String> params = new HashMap<String, String>();
        params.put("hash", hash);

        return params;
    }
};
```

Séquence 3

Pour cette séquence, nous avons réalisé plusieurs tâches : Tout d'abord,

nous avons créé deux classes afin d'interagir facilement avec la base de données :

- User,
- UserDao

User :

```
@Entity
public class User {
    @PrimaryKey(autoGenerate = true)
    private int uid;

    @ColumnInfo(name = "username")
    private String username;

    @ColumnInfo(name = "todolists")
    private String toDoLists;


    public User(String username) {
        this.username = username;
    }

    public User(Profile profile) {
        this.username = profile.getUsername();
//          this.uid = profile.getId();
        Gson gson = new GsonBuilder().create();
        this.toDoLists = gson.toJson(getToDoLists());
    }
```

Cette classe représente l'utilisateur courant. Il possède un constructeur qui prend un paramètre profil (utilisé pour tout le reste de l'application) De même, un constructeur avec comme paramètre User a été créé dans la classe Profile.

UserDao :

```
@Dao
public interface UserDao {
    @Query("SELECT * FROM user")
    List<User> getAll();

    @Query("SELECT * FROM user WHERE uid IN (:userIds)")
    List<User> loadAllByIds(int[] userIds);

    @Query("SELECT * FROM user WHERE username LIKE :username LIMIT 1")
    User findByName(String username);

    @Insert(onConflict = OnConflictStrategy.REPLACE)
    void replaceAll(User... users);

    @Insert
    void insertAll(User... users);

    @Delete
    void delete(User user);
}
```

Cette interface permet de passer de la base de données à l'utilisateur. Elle possède les requêtes SQL permettant de consulter la base de données.

Ensuite, nous devions créer l'objet permettant d'interagir avec la base de données. Elle a été créée dans la classe *CustomApplication*, ce qui nous permet d'y avoir accès depuis n'importe quelle classe et à n'importe quel moment du cycle de vie de l'application.

Pour commencer,

```
final Context context = getApplicationContext();
executor.execute(new Runnable() {
    @Override
    public void run() {
        database = Room.databaseBuilder(context, AppDatabase.class,
"users").build();
    }
});
```

L'executor permet d'exécuter nos fonctions en dehors du *thread* principal.

Ensuite, lors de chaque appuie sur une *checkbox,* le profil est enregistré dans la base de données, qu'on soit en ligne ou hors ligne :

```
executor.execute(new Runnable() {
    @Override
    public void run() {
        CustomApplication.database.userDao().replaceAll(new User(profile));
    }
});
```

En plus de cela, si on est hors ligne, une *HashMap* définie dans *CustomApplication* est modifée : nous lui ajoutons la valeur de l'id de la tâche cochée, de l'id de la liste contenant cette tâche et de l'état de la case.

```
CustomApplication.changedCheckboxes.put(this.JsonId,new
Pair<>(this.toDoListJsonId,this.isDone));
```

Enfin, il ne nous reste plus qu'a vérifier toutes les 10 secondes si le réseau est revenu, et si c'est le cas, nous faisons une requête à l'API pour chaque ligne de la *HashMap.* Une fois cela fait, nous vidons cette *HashMap* :

```
final Handler handler = new Handler();
final int delay = 10000; //milliseconds

handler.postDelayed(new Runnable(){
    public void run(){
        if(hash!=null){
            if (checkNetwork(getApplicationContext())) {
                for (Map.Entry iterator: changedCheckboxes.entrySet()) {
                    String url = "http://tomnab.fr/todo-api/lists/" +
                    ((Pair)iterator.getValue()).first + "/items/" +
                    iterator.getKey() + "?check=" +
                    ((Boolean)((Pair)iterator.getValue()).second ? 1 : 0);

                    StringRequest request = new
                    StringRequest(Request.Method.PUT, url, null, null) {
                        @Override
                        public Map<String, String> getHeaders() {
                            Map<String, String> params = new
                                    HashMap<String, String>();
                            params.put("hash", hash);

                            return params;
                        }
                    };
                    RequestQueueInstance instance =
RequestQueueInstance.getInstance(getApplicationContext());
                    instance.addToRequestQueue(request);

                }

                changedCheckboxes = new HashMap<>();
            }

        }
```

De la même manière nous avons codé la fonction pour ajouter une tâche en ligne, mais Nous ne pouvons pas à la fois en créer une et la cocher tout en étant hors ligne, car nous n'avons pas encore l'ID correspondant à la tâche dans l'API. Il faut donc sortir de l'application et se reconnecter avant de pouvoir la valider.