Programmation Mobile et Réalité Augmentée
FAN Shiyu
TANG Qing

Compte-Rendu du TP2 Todo List API

Introduction:

Les travails déjà réalisés:

- MainActivity
    1. permettre de **saisir le pseudo** et **le mot de passe**, si les deux se correspondent, on peut accéder à la liste.
    2. **Le dernier pseudo** saisi est automatiquement renseigné dans le champ de saisie.
    3. Un bouton de “**préférences**” nous permet d’accéder aux “préférences”.
    4. Un bouton **OK** nous permet d’accéder à des listes-todo liés à pseudo choisi.

- SettingActivity
    1. **Afficher des pseudos** sauvegardé auparavant
    2. **Sauvegarder** et **Permettre à l’édition de l’url** de base de l’API

- ListActivity
    1. **Afficher la list des listes-todo** de l’utilisateur dont le pseudo a été saisi
    2. Lors du clic sur une liste, des **items** de cette liste **s’affichent**.
    3. Permet d’**ajouter une nouvelle liste** pour cet utilisateur
    4. Permet de **supprimer une liste** par glissant l’écran à gauche

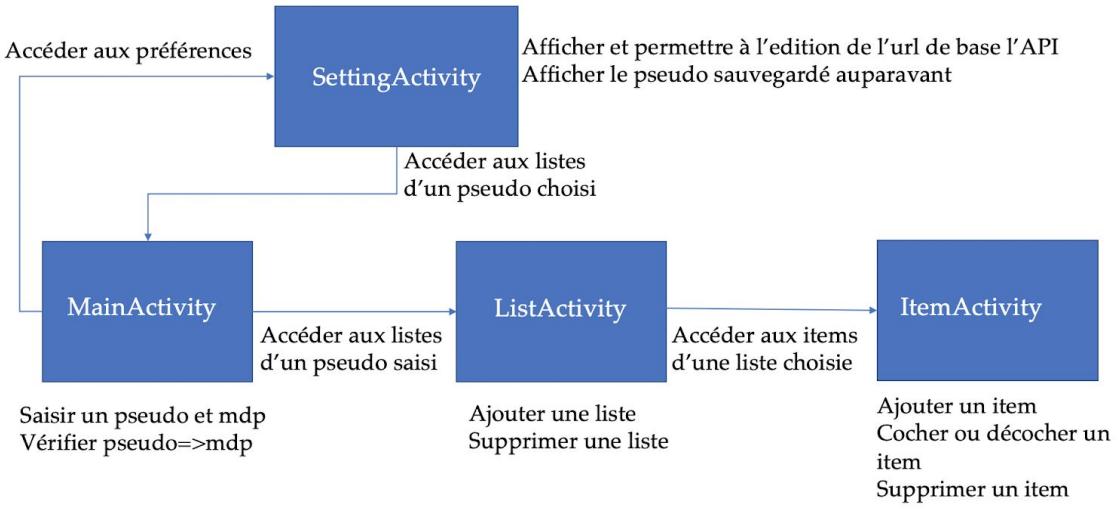
- ItemActivity
    1. **Afficher les items des listes** de l’utilisateur dont le pseudo a été saisi
    2. Permet de **cocher ou décocher** un item
    3. permet d’**ajouter un nouvel item** dans cette liste
    4. Permet de **supprimer un item** par glissant l’écran à gauche

Analyses:

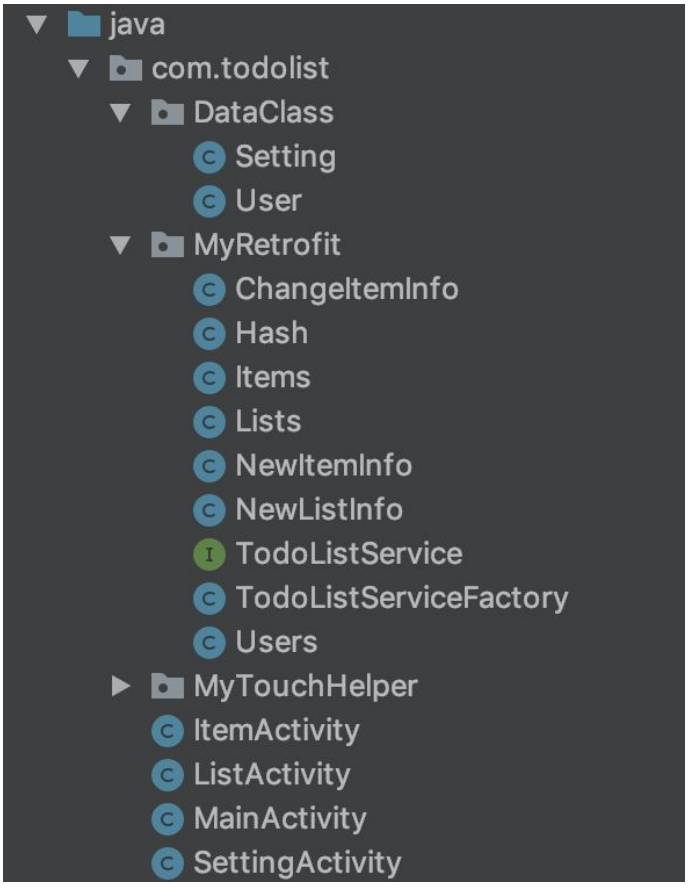
1. Analyse de TP:

On analyse d'abord des relations et connections entre chaque activité, et on fait une figure comme montré ci-dessous. Pour réaliser chaque travail demandé, on va devoir écrire et faire fonctionner plusieurs fonctions.

Pour faire des requêtes asynchrones, on utilise la librairie **Retrofit**.



2. Analyse de fonction:



Structure de class:

Activities :
- MainActivity
- ListActivity
- ItemActivity
- SettingActivity

Data :
- Setting
- User

TouchHelper :
- ItemTouchHelperAdapter
- ItemTouchHelperCallback

MyRetrofit :
- TodoListService
- TodoListServiceFactory
- Hash
- Users
- Lists
- Items
- NewListInfo
- NewItemInfo
- ChangeItemInfo

Les activity class gèrent les 4 écrans, les data class construisent la structure de donnée, et la class TouchHelper sert à supprimer et ré-ordonner les items, les classes et interface MyRetrofit gèrent le changement de données avec l'api.

MyRetrofit

```java
import retrofit2.Retrofit;
import retrofit2.converter.gson.GsonConverterFactory;

public class TodoListServiceFactory {
    public static <T> T createService(String url, Class<T> type) {
        Retrofit retrofit = new Retrofit.Builder()
                .baseUrl(url)
                .addConverterFactory(GsonConverterFactory.create())
                .build();
        return retrofit.create(type);
    }
}
```

TodoListServiceFactory

Elle initialize le Call de Retrofit à partir de l'url et l'interface TodoListService.

```java
public interface TodoListService {
    @FormUrlEncoded
    @POST("authenticate")
    Call<Hash> authenticate(@Field("user") String user,
                            @Field("password") String password);

    @GET("users")
    Call<Users> getUsers(@Query("hash") String hash);

    @POST("users")
    Call<ResponseBody> signin(@Header("hash") String hash,
                              @Query("pseudo") String pseudo,
                              @Query("pass") String password);

    @GET("lists")
    Call<Lists> getLists(@Query("hash") String hash);

    @POST("users/{id}/lists")
    Call<NewListInfo> addList(@Header("hash") String hash,
                              @Path("id") String id,
                              @Query("label") String label);

    @DELETE("users/{user_id}/lists/{list_id}")
    Call<ResponseBody> deleteList(@Header("hash") String hash,
                                  @Path("user_id") String user_id,
                                  @Path("list_id") String list_id);

    @GET("lists/{id}/items")
    Call<Items> getItems(@Header("hash") String hash,
                         @Path("id") String id);

    @POST("lists/{id}/items")
    Call<NewItemInfo> addItem(@Header("hash") String hash,
                              @Path("id") String id,
                              @Query("label") String label);

    @PUT("lists/{list_id}/items/{item_id}")
    Call<ChangeItemInfo> changeItem(@Header("hash") String hash,
                                    @Path("list_id") String list_id,
                                    @Path("item_id") String item_id,
                                    @Query("check") String check);

    @DELETE("lists/{list_id}/items/{item_id}")
    Call<ResponseBody> deleteItem(@Header("hash") String hash,
                                  @Path("list_id") String list_id,
                                  @Path("item_id") String item_id);
}
```

TodoListService

L'interface TodoListService définit les différentes requêtes utilisés pour interagir avec l'API.

@Header peut ajouter hash code dans l'entête de la requête.

@Query peut générer la partie "?key=val&key=val" de la requête.

@Path peut modifier le chemin "users/{id}/…".

Call<Type> définit auquel format les données json vont transférer.

```java
public class Lists {

    /**
     * version : 1
     * success : true
     * status : 200
     * lists : [{"id":"35","label":"test2"},{"id":"36","label":"test1"}]
     */

    private int version;
    private boolean success;
    private int status;
    private List<ListsBean> lists;

    public int getVersion() {
        return version;
    }

    public void setVersion(int version) {
        this.version = version;
    }

    public List<ListsBean> getLists() { return lists; }

    public static class ListsBean {
        /**
         * id : 35
         * label : test2
         */

        private String id;
        private String label;

        public String getId() { return id; }

        public void setId(String id) { this.id = id; }

        public String getLabel() { return label; }

        public void setLabel(String label) { this.label = label; }
    }
}
```

Lists etc.

Les classes hash,users,lists,items … sont pour définir la structure de données. À l'aide de GsonConverter, les json données récupérées de l'API peuvent transférer au format prédéfinie.

Pour simplifier le travail, nous avons utilisé un plugin de Android Studio qui s'appelle GsonFormat. Ce plugin sert à générer ces classes à partir du retour de l'API automatiquement.

MainActivity

```
// Save setting into sharedpreferences
public void savePreference(Setting setting) {
    SharedPreferences preferences = getSharedPreferences( name: "setting", MODE_PRIVATE);
    SharedPreferences.Editor editor = preferences.edit();

    final GsonBuilder builder = new GsonBuilder();
    final Gson gson = builder.create();
    String setting_serialized = gson.toJson(setting);
    editor.putString("setting", setting_serialized);

    editor.apply();
    editor.commit();
}

// Return setting class
public Setting readPreference() {
    SharedPreferences preferences = getSharedPreferences( name: "setting", MODE_PRIVATE);

    final GsonBuilder builder = new GsonBuilder();
    final Gson gson = builder.create();
    String setting_serialized = preferences.getString( key: "setting", gson.toJson(new Setting()));
    return gson.fromJson(setting_serialized, Setting.class);
}
```

savePreference et readPreference

Elle servent à stocker les données de comptes et l'url de l'API dans un fichier xml en utilisant SharedPreferences. Le fichier xml contient l'url de l'API, les pseudos et les mots de passe des utilisateurs.

```
if (setting.hasUser(pseudo)) {
    User user = setting.getUser(pseudo);
    if (user.verify(password)) {
        Intent i = new Intent( packageContext: MainActivity.this, ListActivity.class);
        i.putExtra( name: "hash", user.getHash());
        i.putExtra( name: "url", setting.getUrl());
        i.putExtra( name: "pseudo", pseudo);
        startActivity(i);
    } else {
        Toast.makeText( context: this, text: "Incorrect username or password", Toast.LENGTH_SHORT).show();
    }
} else {
    TodoListService todoListService = TodoListServiceFactory.createService(setting.getUrl(), TodoListService.class);

    Call<Hash> call = todoListService.authenticate(pseudo, password);

    call.enqueue(new Callback<Hash>() {
        @Override
        public void onResponse(Call<Hash> call, Response<Hash> response) {
            if (response.isSuccessful()) {
                setting.addUser(new User(pseudo, password, response.body().hash));
                savePreference(setting);
                Intent i = new Intent( packageContext: MainActivity.this, ListActivity.class);
                i.putExtra( name: "hash", response.body().hash);
                i.putExtra( name: "url", setting.getUrl());
                i.putExtra( name: "pseudo", pseudo);
                startActivity(i);
            } else {
                Toast.makeText( context: MainActivity.this, text: "Incorrect username or password", Toast.LENGTH_SHORT).show();
            }
        }

        @Override
        public void onFailure(Call call, Throwable t) {
            Toast.makeText( context: MainActivity.this, text: "No internet connection", Toast.LENGTH_SHORT).show();
        }
    });
}
```

login

Vérifier si le compte existe ou pas dans le setting.
Si il existe , et le mot de passe est correcte, ouvrir l'activité list et transmettre les paramètres url, hash code et pseudo de l'utilisateur saisie par Intent.
Si il n'existe pas au local, utiliser la requête authenticate pour vérifier s'il existe ou pas dans le serveur. Si le requête réussit, stocker pseudo, mot de passe et hash code au local, et passer à l'activité list.

```
if (setting.hasUser(pseudo)) {
    Toast.makeText( context: this,  text: "User already exist", Toast.LENGTH_SHORT).show();
    return;
}

TodoListService todoListService = TodoListServiceFactory.createService(setting.getUrl(), TodoListService.class);

Call<ResponseBody> call = todoListService.signin(setting.getLastUser().getHash(), pseudo, password);

call.enqueue(new Callback<ResponseBody>() {
    @Override
    public void onResponse(Call call, Response response) {
        if (response.isSuccessful()) {
            Toast.makeText( context: MainActivity.this,  text: "Success to sign in", Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText( context: MainActivity.this,  text: "Fail to sign in", Toast.LENGTH_SHORT).show();
        }

    }

    @Override
    public void onFailure(Call call, Throwable t) {

    }
});
```

signin

Vérifier si le compte existe ou pas dans le setting.
SI il n'existe pas, faire une requête à l'API pour créer un compte.

ListActivity

```
private void recyclerViewConfig(String hash) {
    Call<Lists> call = todoListService.getLists(hash);

    call.enqueue(new Callback<Lists>() {
        @Override
        public void onResponse(Call<Lists> call, Response<Lists> response) {
            if (response.isSuccessful()) {
                for (Lists.ListsBean l : response.body().getLists()) {
                    listAdapter.addData(new DataEntity(l.getLabel(), l.getId()));
                }
            }
        }
```

recyclerViewConfig

Faire une requête à l'API pour récupérer la liste de Todolist à partir du hash code qui représente l'utilisateur courant. Et afficher dans l'écran à l'aide de recyclerViewAdapter.

```
// Function bind to the "OK" button as onClick event for a new list
public void newList(View v) {
    final String list_name = edt_list.getText().toString();
    if (list_name.isEmpty()){
        Toast.makeText( context: this,  text: "Please enter list name", Toast.LENGTH_SHORT).show();
    } else {
        Call<NewListInfo> call = todoListService.addList(hash, userId, list_name);

        call.enqueue(new Callback<NewListInfo>() {
            @Override
            public void onResponse(Call<NewListInfo> call, Response<NewListInfo> response) {
                if (response.isSuccessful()) {
                    listAdapter.addData(new DataEntity(list_name, response.body().getList().getId()));
                    edt_list.setText("");
                }
            }
```

newList

Faire une requête à l'API pour ajouter une nouvelle liste pour l'utilisateur courant et actualiser l'écran.

ItemActivity

```
private void recyclerViewConfig(String hash, String listId) {
    Call<Items> call = todoListService.getItems(hash, listId);

    call.enqueue(new Callback<Items>() {
        @Override
        public void onResponse(Call<Items> call, Response<Items> response) {
            if (response.isSuccessful()) {
                for (Items.ItemsBean i: response.body().getItems()) {
                    itemAdapter.addData(new DataEntity(i.getLabel(),i.getId(), i.getChecked()));
                }
            }
        }
```

recyclerViewConfig

Faire une requête à l'API pour récupérer la liste de Item à partir du hash code qui représente l'utilisateur courant. Et afficher dans l'écran à l'aide de recyclerViewAdapter.

```
@Override
public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
    String item_selected = data.get(getAdapterPosition()).getItemId();
    Call call;

    if (isChecked) {
        call = todoListService.changeItem(
                hash, listId, item_selected, check: "1");
    } else {
        call = todoListService.changeItem(
                hash, listId, item_selected, check: "0");
    }
```

onCheckedChanged

Faire une requête à l'API pour changer l'état checked de Item.

SettingActivity

```
public void newURL(View v) {
    final String url = edt_url.getText().toString();
    if (url.isEmpty()){
        Toast.makeText( context: this,  text: "Please enter list name", Toast.LENGTH_SHORT).show();
    } else {
        setting.setUrl(url);
        savePreference(setting);
    }
}
```

newURL

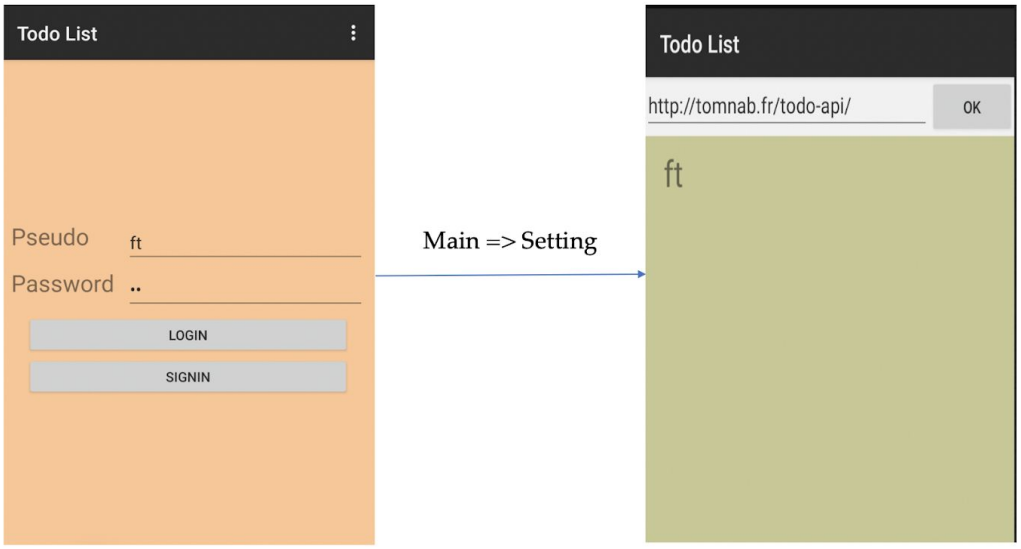
Sauvegarder l'url de l'API au local.

Conclusion:

Avec notre travail réalisé, on a réussi à créer l'application "ToDo List". Voici quelques images pour montrer le processus de l'application.

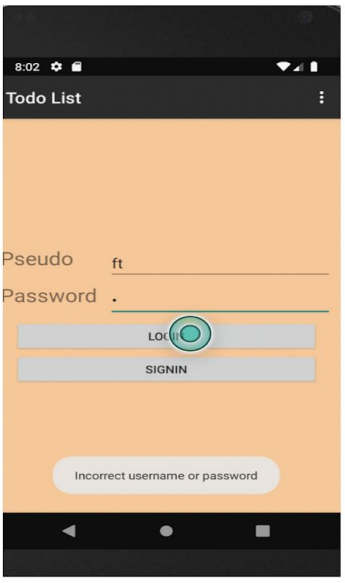
- Pour MainActivity et SettingActivity, on peut accéder aux préférences, éditer la base de l'url de l'API.

MainActivity et SettingActivity



Si le pseudo et le mot de passe ne correspondent pas, un message s'affiche.

MainActivity et SettingActivity

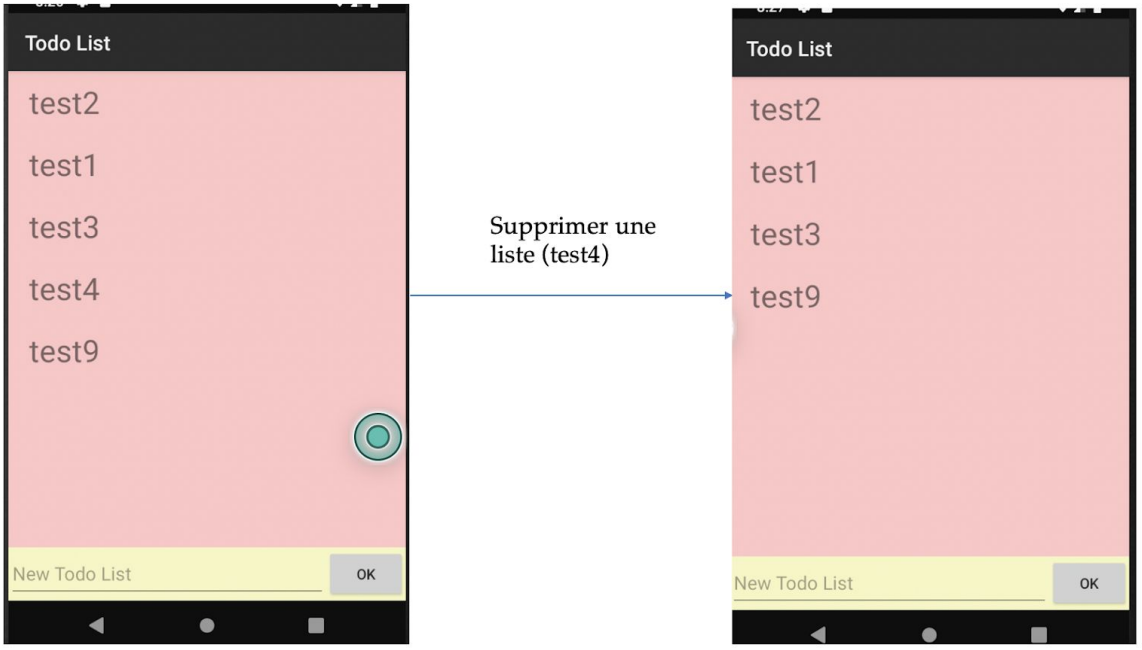


Si le pseudo et le mot de passe ne correspondent pas:

Incorrect username or password

- Pour ListActivity, on peut ajouter, supprimer une liste et on peut changer la position de deux listes.

ListActivity



- Pour ItemActivity, on peut ajouter ou supprimer un item. On peut aussi cocher ou décocher un item.

ItemActivity