Rapport Application Mobile

MAPEZA

Membres:
MEJDOUBI Othman
BERCKMANS Yannick
RUBENS Thibaut
BOUHADDI Zakariya
COOLS Hadrien

Professeur:
M. LURKIN

ECAM
2016-2017

1. Introduction

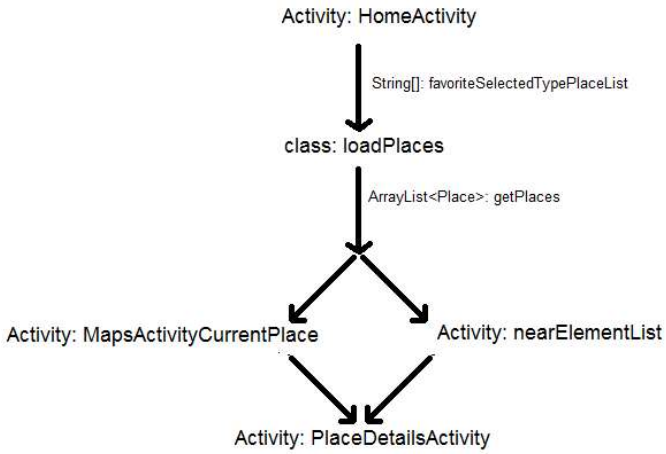
Dans le cadre du Laboratoire d'Application Mobile, il est demandé de réaliser une application mobile via Android Studio en Java.
Les objectifs sont les suivants :
— Création d'une application Android avec Android Studio;
— Organisation efficace d'un projet de développement;
— Gestion du groupe.

Le choix de l'application est libre mais elle doit au minimum comporter :
— plusieurs Activités,
— des préférences,
— une Recycler View,
— une base de donnée locale,
— récupérer des données d'internet,
— fonctionner aussi bien en portrait qu'en paysage et se comporter convenablement lors du passage de l'un à l'autre.

Notre application, Mapeza, a pour but d'aider les gens à trouver le lieu qu'ils cherchent de manière rapide en utilisant l'API Google Maps. Ils peuvent filtrer les lieux qu'ils veulent trouver parmi ceux présent dans un certain rayon autour d'eux et enregistrer leurs lieux favoris, ainsi que leurs préférences de filtrage.
Pour cela, nous avons découpé l'application en plusieurs "activités" :

1. HomeActivity (Thibaut)
2. nearElementList (Yannick)
3. MapsActivityCurrentPlace (Zakariya)
4. PlaceDetailsActivity (Othman)

L'utilisateur pourra naviguer entre ces différentes activités au moyen d'un menu de navigation réalisé par Hadrien.
Voici le cheminement entre les différentes activités :



Nous allons donc expliquer le fonctionnement de l'application activité par activité, ainsi que le fonctionnement du menu de navigation.

2. HomeActivity

Le but de cette activité est d'enregistrer dans les préférences les types de lieux (Place) que veut trouver l'utilisateur dans un certain rayon (représenté par un tableau de String favoriteSelectedTypePlaceList, enregistré sous forme d'un JSON via la bibliothèque GSON), ainsi que convertir ces types de lieux (qui sont intelligibles par l'utilisateur et donc en français) en les types de lieux utilisés dans les requêtes Google Map.
Voici la liste des types de lieux que l'on peut choisir :
"Bars", "Stade", "Centres commerciaux","Restaurants", "Parcs", "Magasin d'électronique", "Casino", "Parc d'attraction", "Boites de nuit", "Magasin d'alcool"
Et voici leurs codes API dans l'API Google Maps :
"bar", "stadium", "shopping_mall", "restaurant", "park", "electronics_store", "casino", "amusement_park", "night_club", "liquor_store"
J'ai donc créé deux classes pour faire cette correspondance : PlaceType et PlaceList. PlaceType représente le type d'un lieu (Place) et contient trois variables : le nom en français (String), le code API (String) et un booléan qui représente si le type a été checké ou pas dans l'activité (selected). La classe PlaceList encapsule une ArrayList de PlaceType et le principe est que lorsque l'utilisateur va checker les types de lieux qu'il veut rechercher, les lieux checkés vont avoir leurs variable "selected" mises à "True". Ensuite, lorsque l'utilisateur cliquera sur le bouton "OK" (après avoir checké les lieux qui l'intéressaient), l'ArrayList de String favoriteSelectedTypePlaceList va être remplie par les APICode des types de places qui ont leur variable "selected" à "True". L'objet favoriteSelectedTypePlaceList va par après être enregistré dans les SharedPreference sous forme d'un JSON, via la bibliothèque GSON, qui va par après être utilisée dans la classe loadPlaces dans l'activité nearElementList afin de construire la requête Google Maps pour récuperer uniquement les lieux (Place) qui correspondent aux types checkés dans HomeActivity.
Cette activité n'a pas pu être implémentée avec des fichiers XML (Settings) car cette dernière ne peut être réalisée qu'avec des radioBox (où un seul élément peut être choisi). Cela peut être expliqué par le fait que les Settings impliquent par exemple de choisir 1 élément parmi N différents possibles (avec $N > 1$), mais pas pour choisir X éléments parmi N éléments (avec $1 < X < N$).
La vue a, elle, été implémentée au moyen d'une vue qui rajoute autant de checkBox qu'il n'y a de lieux possibles à checker.

3. loadPlaces

La classe loadPlaces est une classe indispensable dans le bon fonctionnement de notre application. Cette classe est constituée d'une méthode principale: getPlaces. Elle prend en paramètre d'un côté le rayon dans lequel nous voulons récupérer les lieux proches qui nous intéressent, et de l'autre un tableau de String contenant l'ensemble des types de lieux que l'utilisateur a choisi d'afficher. Nous devons donc récupérer cette liste des SharedPreferences avant d'appeler cette méthode.

Le fonctionnement de la méthode est le suivant:

- En premier lieu, nous récupérons la position de l'utilisateur grâce à son GPS
- Ensuite, nous initialisons une ArrayList d'objets "place" qui est au départ vide.
- Nous bouclons sur l'ensemble des lieux types de lieux de que l'utilisateur a coché (tableau de string "myPlaceName") et nous créons le string "requete" sous le format donné par l'API google, à savoir

```
String nearByPlaceSearchURL = "https://maps.googleapis.com/maps/api/place/nearbysearch/json?"
        + "location=" + myLatPosition + "," + myLonPosition
        + "&radius=" + myPlaceDistanceMeters
        + "&types=" + myPlaceName[i]
        + "&key=" + "AIzaSyBLkj8WSx2AjfJNN0-BEYp1ZQWwNs8mY8U";
```

- Nous envoyons ensuite le string créé à la classe "NetworkUtils" qui va se charger de réaliser la transaction.
- Toujours dans la boucle, nous envoyons le string récupéré (à savoir l'ensemble des lieux proches, renvoyés par l'API Google) à la méthode "Place.parse()" qui se charge de créer des objets "Place" à partir de cette liste et de les stocker dans une variable statique interne à cette classe.
- Enfin, nous renvoyons en sortie l'ArrayList de l'ensemble des places stockée statiquement dans la classe "Place".

Avant de continuer dans la hiérarchie de notre application, il me semble important d'expliquer comment sont constitués les objets "places".

4. Classe Places

La classe Place est constituée d'une méthode principale "parse" et d'un ensemble d'accesseur (en plus du constructeur). Les accesseurs et le constructeur sont présentés de manière classique, nous allons donc uniquement développer la méthode "parse" dans cette partie.

La méthode parse possède 2 paramètres: le premier est un String, au format renvoyé par l'API Google. Le second est un string décrivant le type des lieux renvoyés par l'API.

Cette méthode parse possède un but principal: parcourir le JSON renvoyé par l'API, et récupérer les éléments intéressants. Nous avons donc étudié la structure du JSON récupéré, et nous avons isolé les paramètres qui nous intéressent pour former nos objets, à savoir:

- l'ID du lieu
- Le nom du lieu
- L'adresse du lieu
- Le type du lieu
- Les coordonnées du lieu

Ces éléments sont d'abord placés dans des Strings temporaire, pour au final créer de nouveaux objets "Place" en appelant le constructeur. Ce lieu est ensuite ajouté à la variable statique "array", qui garde l'ensemble des lieux en mémoire.

L'ensemble des accesseurs permettent simplement l'accès aux différents paramètres des objets "place", ainsi qu'à l'ArrayList "array".

5. nearElementList

L'activité nearElementList est une des deux pages qui permettent à l'utilisateur d'accéder aux lieux présents dans l'ArrayList de la classe "Place". Une série d'actions est réalisée pour permettre le bon fonctionnement du système.

- Premièrement, lors de la création de l'activité, deux choses principales sont initialisées. La première est "l'itemAdapter" qui se charge de remplir les textviews de la recyclerView avec de nouveaux éléments de l'array. La seconde est le lancement de "l'AsyncLoader" qui va permettre un chargement asynchrone des données (appel de la fonction "getPlaces" de la classe LoadPlaces).
- L'itemAdapter remplit les textViews de seulement quelques informations basiques sur les lieux. Le reste des informations seront accessible dans l'activité "placeDetailsActivity" qui sera expliquée par après.
- Le changement d'orientation du mobile ne pose pas de problème au bon fonctionnement de cette activité.
- Un bouton permettant le passage à l'autre manière d'afficher les données (sous forme de carte, dans l'activité "MapsActivityCurrentPlace") est présent pour permettre à l'utilisateur de rapidement switcher entre les différentes façons de consulter les données.

Lors du chargement asynchrone des données, plusieurs choses sont réalisées dans la méthode "loadInBackground". Premièrement, il y a la récupération des SharedPreferences pour récupérer l'ensemble des types de lieux que l'utilisateur veut afficher. Ensuite, nous supprimons l'ensemble des RECORDS présents dans l'array reprenant l'ensemble des lieux pour ne pas risquer d'avoir de doublons. Enfin, nous faisons le chargement asynchrone des données depuis l'API Google.

6. MapsActivityCurrentPlace

L'activité MapsActivityCurrentPlace est la seconde page permettant d'accèder aux lieux présents dans l'Arraylist de la classe « Place ».
Lors de sa création :

- Nous créons notre design visuel à partir de notre xml activity_maps.
- Nous construisons notre GoogleApiClient en ajoutant les API dont nous avons besoin (LocationServices, Places.GEO_DATA_API et Places.PLACE_DETECTION_API)
- Nous nous connectons à notre instance de GoogleApiClient.

Cette activité implémente quatre interfaces :

1. **LoaderManager.LoaderCallbacks< ArrayList<Place>>**
2. **GoogleApiClient.ConnectionCallbacks**
3. **OnMapReadyCallback**
4. **GoogleApiClient.OnConnectionFailedListener**

1.
La méthode loadInBackground a été expliquée plus haut. Pour cette activité, les valeurs qui sont chargées servent à construire les marqueurs sur la map.
Dans la méthode onLoadFinished, nous créeons nos marqueurs. Cela se fait dans cette méthode afin de nous accorder suffisamment de temps pour charger nos données. (Éviter ainsi le null pointer exception)
Rem : La construction des marqueurs des lieux à proximité se fait à l'aide des valeurs de Latitude/Longitude fournies par la classe "Places" avec ses méthodes getLat() et getLng.
Dans le cas des lieux à proximité, il suffit de boucler sur l'ArrayList que nous avons récupéré.

```
import static be.ecam.mapeza.mapeza.loadPlaces.getPlaces;
```

/ ! \Pour utiliser la méthode static de la classe loadPlaces qui est utile à la récupération de l'ArrayList.
2. / 4.
La construction de la map se fait dans la méthode onConnected(…) qui s'exécute à la fin de notre onCreate() suite à la connexion de notre instance mGoogleApiClient.
Les méthodes onConnectionFailed(…) et onConnectionSuspended fournissent des messages sur la console.
3.
La méthode onMapReady() contient le cœur de l'activité : récupération de la position actuelle du device et repositionnement de la caméra sur celle-ci.
Nous avons tout d'abord la méthode updateLocationUI() qui comme son nom l'indique va mettre à jour l'interface si l'utilisateur donne son **autorisation**.
N.B. : Nous devons absolument demander les autorisations à l'utilisateur avant d'accéder à la position actuelle. Un pop-up sera donc affiché sur l'écran lors du lancement de l'activité. Si la permission est accordée :

```
mMap.setMyLocationEnabled(true);
```

Ensuite, la méthode getDeviceLocation() (après confirmation des permissions) va fournir notre position (dernièrement connue/cf. connexion récente) + reposition de la caméra:

```
mLastKnownLocation = LocationServices.FusedLocationApi.getLastLocation(mGoogleApiClient);
```

Rem : Des valeurs de LatLng ont été données par défault si notre position s'avérait nulle. (Permission refusée, mauvaise connexion, etc.)

La position "mLastKnownLocation" que nous récupérons est de type Location. Nous allons donc utiliser les méthodes getLatitude() et getLongitude() pour ensuite instancier un objet de type LatLng utile à la création de notre marqueur de position actuelle.

```
MyCurrentLat = mLastKnownLocation.getLatitude();
MyCurrentLong = mLastKnownLocation.getLongitude();
LatLng MyCurrentLocation = new LatLng(MyCurrentLat,MyCurrentLong);
mMap.addMarker(new MarkerOptions().position(MyCurrentLocation).title("Ma
Position Actuelle"));
```

Finalement, lorsque nous cliquons sur un marqueur, nous avons "mMap.setOnMarkerClickListener()" qui va lier chaque marqueur à son détail correspondant dans PlaceDetailsActivity.
Le seul marqueur qui est exempt du ClickListener est celui de la position actuelle car elle se trouve dans la condition "else" de cette affirmation :

```
if(!arg0.getId().equals("m0"))
```

N.B. : Différenciation de la position actuelle par rapport aux autres marqueurs via la coloration.

```
arg0.setIcon(BitmapDescriptorFactory.defaultMarker(BitmapDescriptorFactory.HUE_ORANGE));
```

7. PlaceDetailsActivity

Cette activité s'ouvre lorsqu'on clique sur un élément de la liste (RecyclerView) qui est affichée dans l'activité "nearElementList" ou lorsqu'on clique sur un marqueur sur la carte affichée dans l'activité "MapsActivityCurrentPlace". Cette page permet d'avoir plus d'information sur le lieu sélectionné et permet également à l'utilisateur d'enregistrer le lieu dans ses favoris (base de données locales). L'activité "Favorites" permet également d'accéder aux détails des lieux mis en favoris.

Nous allons expliquer le cheminement de cette activité.

D'abord, à partir de l'activité "nearElementList", "MapsActivityCurrentPlace" ou "Favorites", l'utilisateur clique sur un lieu (élément de la liste ou marqueur sur la carte). Cela a pour effet de lancer l'activité PlaceDetailsActivity. L'objet "Place" qui correspond au lieu sur lequel l'utilisateur a cliqué est récupéré du Bundle sous forme de JSON via la bibliothèque GSON, et est converti en objet Place. De cette objet nous récupérons un identifiant de la place qui nous permettra de faire une requête vers l'API Google Places pour récupérer toutes les informations sur le lieu en question : nom, adresse, numéro de téléphone, site web, latitude et longitude.

Ce dernier est ensuite affiché sur la carte GoogleMap à l'aide des informations de localisation que sont la latitude et la longitude.

Pour se connecter à la base de données, une classe DBHelper a été créée afin d'encapsuler les connexions/enregistrements/récupérations des lieux favoris à partir de la base de données locale. Cette dernière s'appuie sur la base de données SQLite, et les données enregistrées dans la base de données respectent la structure écrite dans la classe DBContract. Les lieux enregistrés en base de données possèderont donc tous les champs suivants : id, adresse, nom, type, latitude et longitude.

Lorsqu'on affiche la vue, on vérifie d'abord si le lieu est présent dans la base locale. Selon le cas, le bouton a une certaine couleur ce qui permet de différencier un lieu favori d'un autre lieu. Lorsqu'on clique sur le bouton, celui-ci est ajouté aux favoris ou est supprimé des favoris selon l'état de celui-ci.

8. FavoritesActivity

Cette activité permet d'afficher des lieux favoris. Celle-ci récupère les lieux favoris en base de données grâce à la méthode getPlaces() implémentée dans le DBHelper. Le reste de l'activité est identique à l'activité nearElementList expliquée plus haut et exploitant une RecyclerView pour afficher le nom des lieux.

9. Navigation

Dans un premier temps, nous avons voulu implémenter un menu latéral.
Ce menu apparaît grâce à un bouton sur l'action bar ou bien est déclenché par un événement de slide horizontal par l'utilisateur .
Une fois déclenché, le menu affiche les différentes vues auxquels l'utilisateur peut accéder.

Un premier problème s'est posé:
Ce type de menu marche en symbiose avec des fragments.
Or notre projet ne comporte que des activités différentes car le menu a été implémenté à posteriori.
L'idée pour y remédier est donc de créer une première fois le template de notre menu pour ensuite faire hériter ses méthodes et son layout aux autres activités.
Forçant donc l'application à 'recréer le menu' lors de chaque changement d'activité induit par la bar de menu elle-même!
Ceci manque d'optimisation dans la perspective de créer une application la plus performante possible.

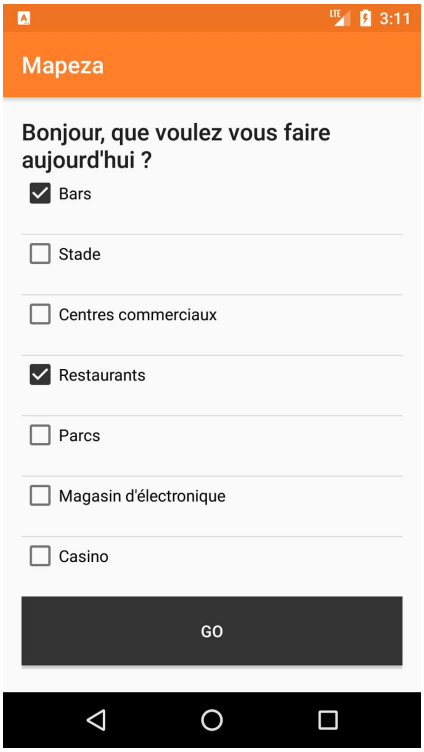
Les essais avec différentes activités "test" sont concluants, le passage de l'une à l'autre par le biais d'intent est fonctionnel.
Mais lors de l'intégration dans le projet, il a été impossible de créer un drawable menu ou de l'importer à l'intérieur du projet malgré plusieurs tentatives pour rendre compatible ces activités.
La solution est de simplement placer un bouton sur l'actionbar elle-même.

L'événement onClick ("onCreateOptionsMenu()") de notre bouton fait inflate un menu précédemment déclaré dans nos ressources, offrant la liste des activités dans lesquels l'utilisateur peut se rendre.
Une fois ce menu inflated, on a plus qu'à identifier sur quel élément l'utilisateur a cliqué en récupérant l'id de l'item.
En fonction de l'id, on redirigera l'utilisateur vers la page souhaitée par le biais d'un Intent.
Voici un aperçu :

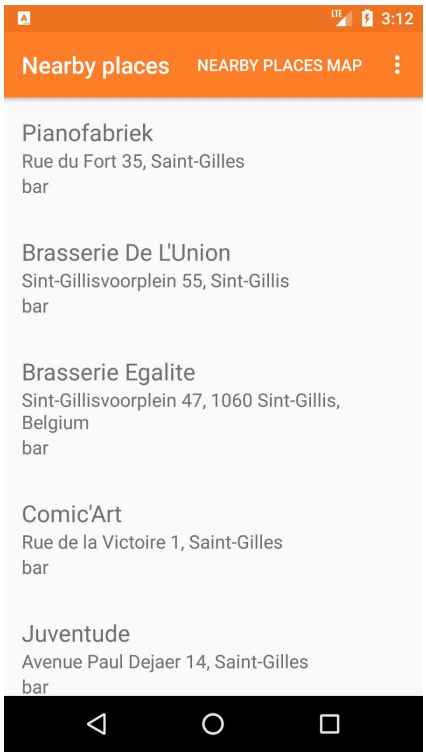
```
@Override
public boolean onOptionsItemSelected(MenuItem item){
    int id = item.getItemId();
    if (id == R.id.MnearElementList) {
        Intent intent = new Intent(this, nearElementList.class);
        startActivity(intent);
    }
```

10. Screens

HomeActivity

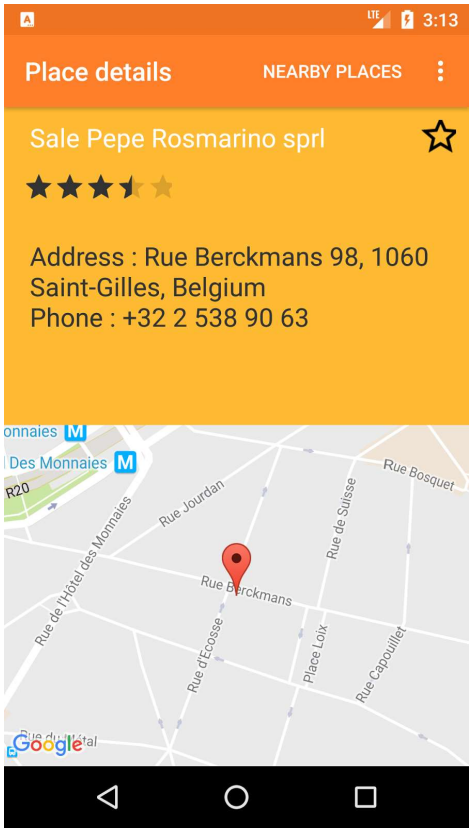


Nearby places

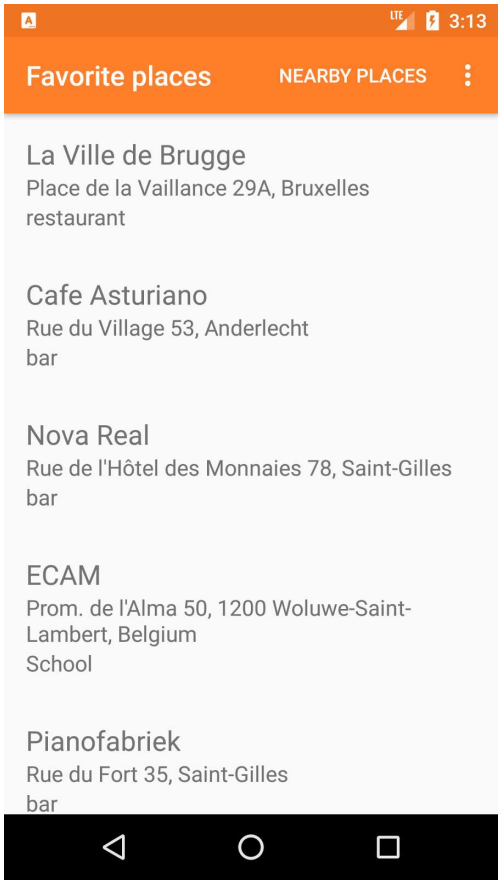


Nearby places map

Place Details



Favorite places



Exemple de menu