AVIGNON UNIVERSITÉ

Rapport final du projet : 'le petit musée du CERI'

Tommy Anfosso

25 mai 2020

Licence d'Informatique
Ingénierie Logicielle
UE Applications mobiles

Responsable
Stéphane HUET

UFR
SCIENCES
TECHNOLOGIES
SANTÉ

AVIGNON UNIVERSITÉ
CERI

CENTRE
D'**ENSEIGNEMENT**
ET DE **RECHERCHE**
EN **INFORMATIQUE**
ceri.univ-avignon.fr

Sommaire

1 Introduction

Le but du projet est de concevoir et programmer une application Android permettant d'accéder de manière simple aux informations des divers objets de la vitrine du CERI. Ces informations sont accessibles en interrogeant un service web hébergé sur un serveur du CERI.
Il nous a été demandé dans un premier temps de réaliser le design des différents écrans constituants l'application , puis de faire son implémentation avec AndroidStudio, tout en satisfaisant les fonctionnalités répertoriées dans le cahier des charges.

1.1 Rappel du cahier des charges

- Afficher la liste des objets présents dans la collection, faisant apparaître pour chacun son image en miniature, son nom (et, le cas échéant, sa marque), ainsi que les catégories auxquelles il appartient. Cette liste devra pouvoir être triée par ordre alphabétique ou par ordre chronologique.

- Présenter la liste des objets découpée en sections correspondant aux diverses catégories d'objets. Chaque section affichera en titre le nom de la catégorie, puis la liste des objets appartenant à cette catégorie.

- Filtrer le catalogue via une requête textuelle simple fouillant dans toutes les caractéristiques des objets.

- Régler le niveau de difficulté. Un utilisateur pourra changer ce niveau entre deux parties

- Afficher une vue détaillée d'un objet en particulier : la sélection d'une entrée dans les vues précédentes amènera à une vue dédiée à la présentation détaillée de l'objet correspondant, affichant toutes les informations disponibles au sujet de cet objet.Le choix du style de présentation est laissé libre, en particulier concernant l'affichage des images liées à l'objet

[Optionnel] Amélioration de l'affichage par ordre chronologique :

- Afficher les entrées du catalogue sous forme d'une frise chronologique.

1.2 Lien vers le projet Gitlab

`https://gitlab.com/Motyak/App_Projet`

2 Présentation de l'interface

L'application possède trois activités différentes : 'MainActivity' représente l'écran d'accueil présentant les objets et des méthodes de tri, 'SearchActivity' représente la vue de recherche d'un objet retournant une liste de résultats filtrés à partir d'une requête textuelle, et enfin 'ItemActivity' est la vue plus en détails d'un objet sélectionné.

2.1 MainActivity

Lors du premier lancement de l'application, un chargement automatique permettra à l'activité de récupérer toutes les informations concernant les objets du musée. Vous pourrez par la suite mettre à jour vos données manuellement en swipant l'écran vers le bas.

Lorsqu'on récupère de nouvelles données, elles sont stockées localement, ce qui veut dire que vous pourrez continuer de consulter le catalogue même sans connexion à Internet (y compris les images).

(a) Une icône en forme de loupe, située en haut à droite de l'écran, permet de lancer une recherche. (1)

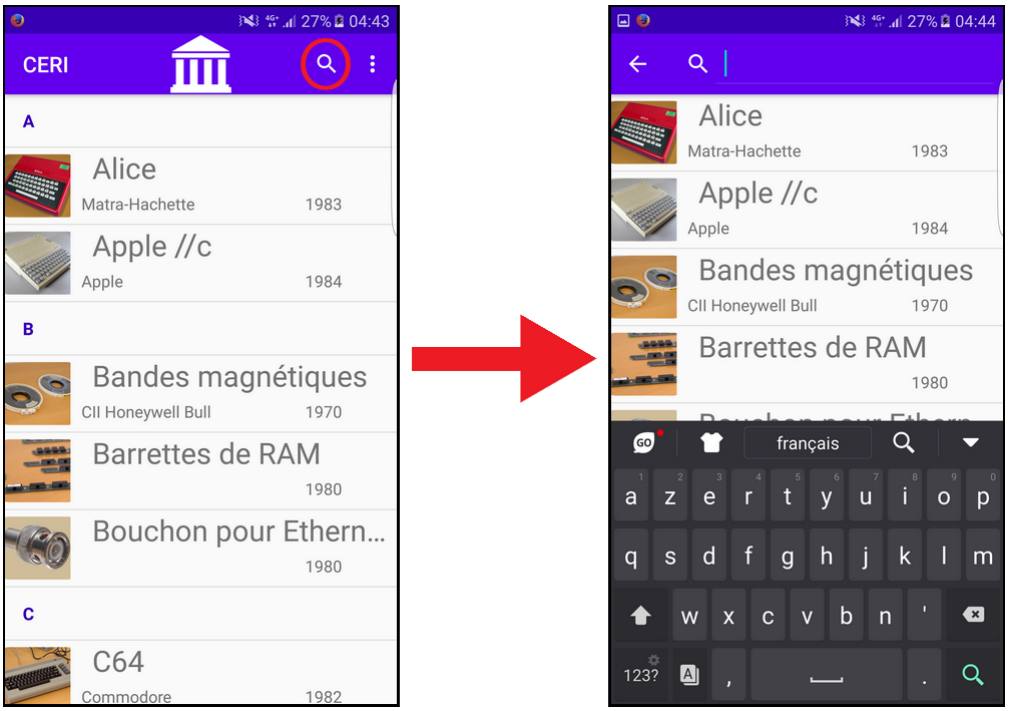

Figure 1. Clic sur l'icône loupe => Lancement d'une recherche

(b) A côté de celle-ci, un menu déroulant propose trois méthodes de tri de la liste. (2)

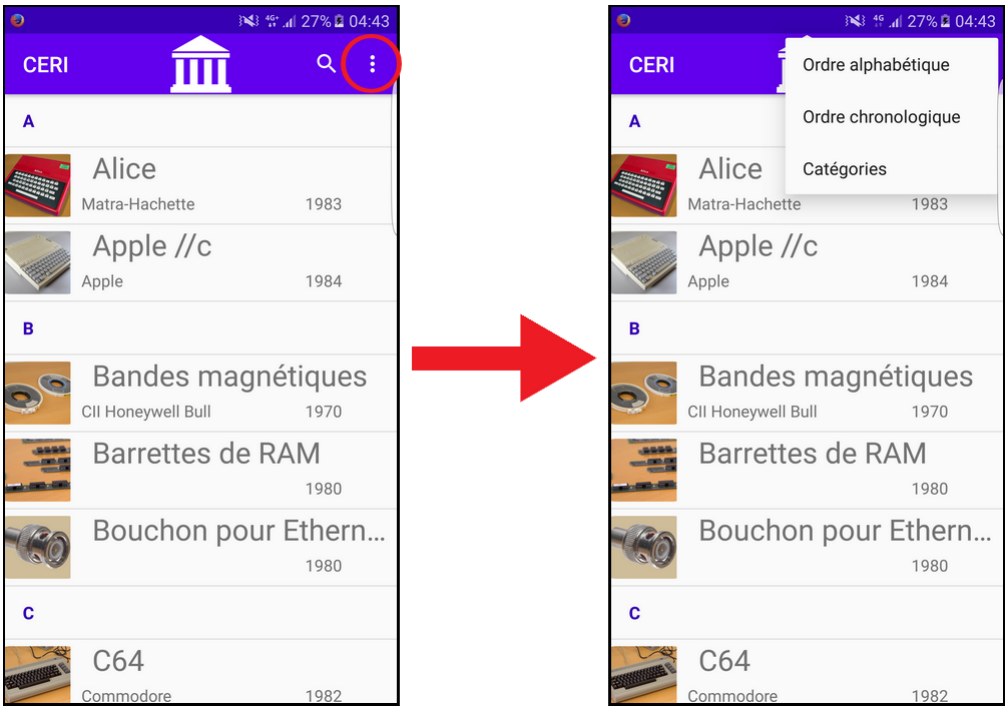

Figure 2. Clic sur le menu déroulant => Affichage de propositions de tri

(c) Les trois tris sont : par ordre alphabétique des noms, par ordre chronologique des dates de production ou par catégories (elle-mêmes triées alphabétiquement). (3)

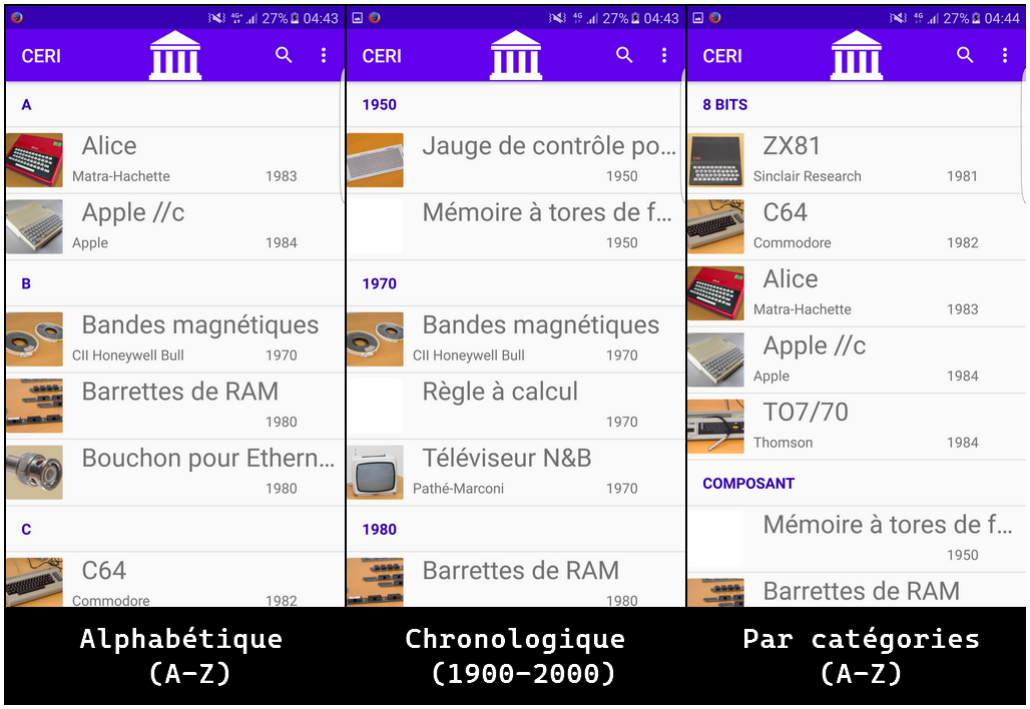

Figure 3. Les différentes méthodes de tri de la liste d'objet

(d) En cliquant sur un objet de la liste, une vue plus détaillée de l'objet en question est affichée. (4)

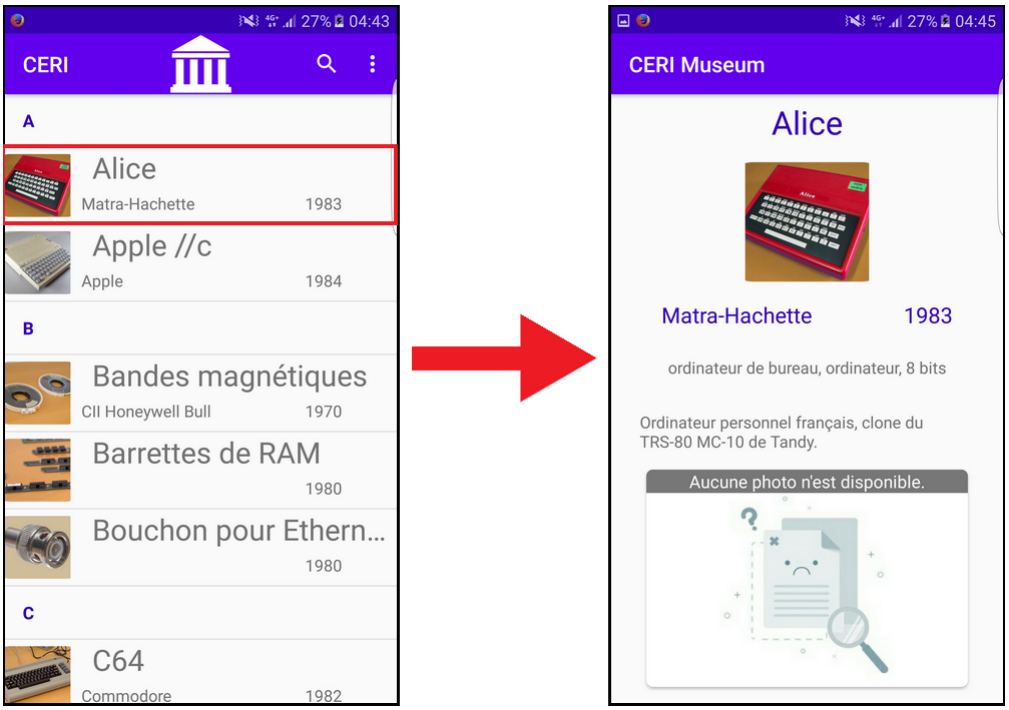

Figure 4. Clic sur un objet de la liste => Affichage de celui-ci

(e) Cette vue, comme toutes les vues de l'interface, peut s'adapter horizontalement en cas de changement d'orientation de l'appareil. (5)

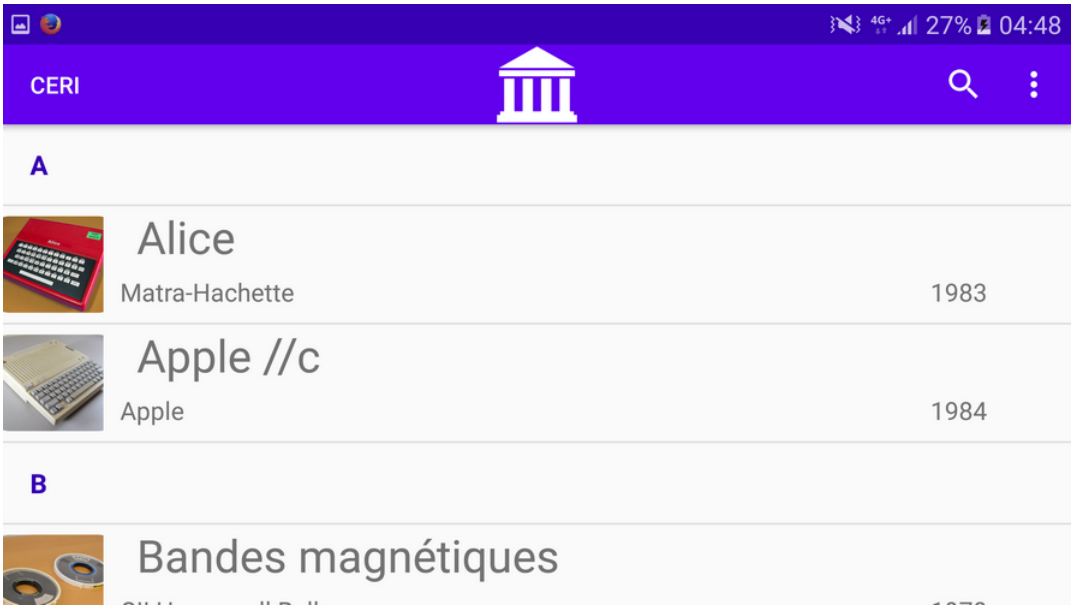

Figure 5. Vue horizontale de la liste

2.2 ItemActivity

Cette vue présente toutes les informations possibles sur un objet en particulier. Il s'agit d'une vue scrollable qui affichera en premier le nom de l'objet, sa marque, son année de production, ses catégories et une description. Puis en dessous des images (si disponibles), une description et enfin la date de dernière mise à jour de toutes ces informations.

(a) Lorsqu'un objet ne possède pas d'image, une indication est affichée à la place. (6)

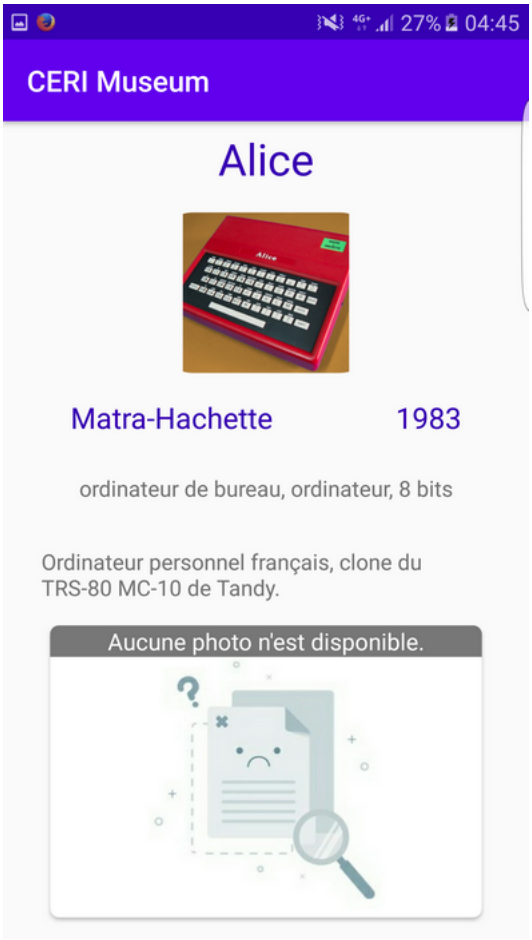

Figure 6. Présentation d'un objet ne contenant aucune photo (vue supérieure)

(b) Sinon, on affiche un slideshow de toutes les images que l'on possède. (7)

Figure 7. Présentation d'un objet avec des photos (vue inférieure)

(c) La vue peut s'adapter horizontalement ce qui peut s'avérer être utile lorsqu'une image possède une longue description par exemple. (8)

Figure 8. Vue horizontale d'un objet

2.3 SearchActivity

Cette vue propose une barre de recherche afin de pouvoir rechercher certains objets à partir de leurs caractéristiques.

La recherche retournera en priorité les objets dont le nom correspond, elle cherchera également parmi les autres informations à disposition. (9)

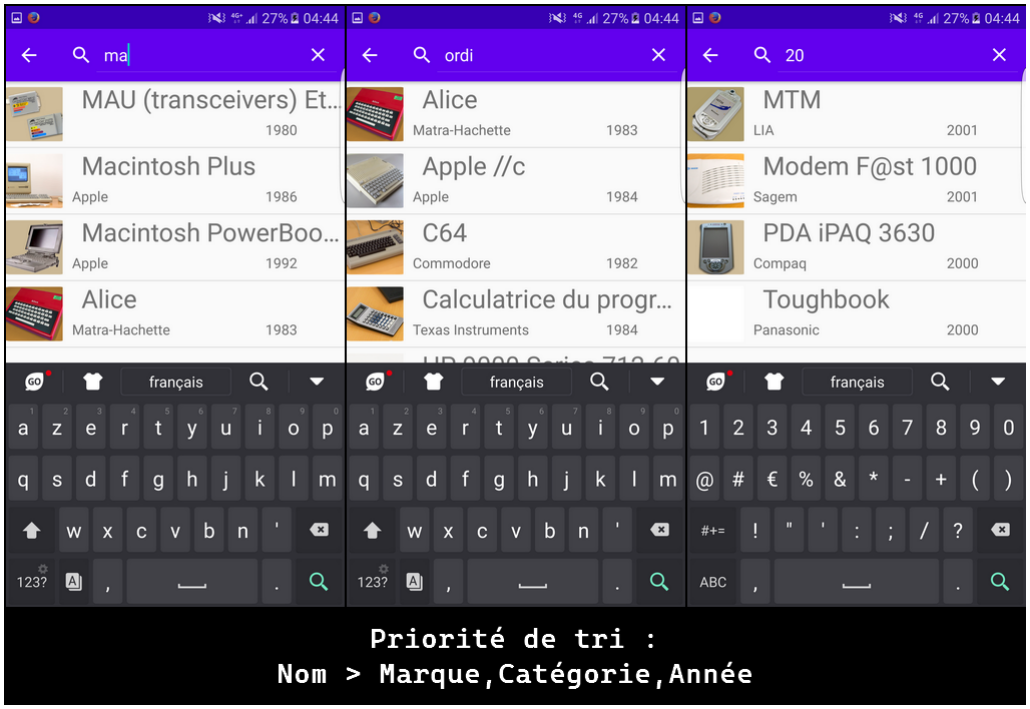

Figure 9. Différentes recherches basées sur des caractéristiques différentes

3 Choix techniques

3.1 Version minimale du SDK Android

J'ai choisi de créer un projet avec comme version minimale du SDK l'API de niveau 15 (pour Android 4.0.3). C'est une configuration que j'utilisais déjà lors des TPs précédents, elle permet d'être compatible avec un maximum d'appareil Android différents et est selon moi largement suffisante pour des projets simples comme les notres. J'utilise les librairies d'Android Jetpack (androidx.*) visant à remplacer les librairies natives (android.support.v7.*).

3.2 Modélisation de la base de données

Pour la base de données j'ai choisi de n'utiliser qu'une seule table pour stocker les objets du musée. Ma table 'catalog' contient les colonnes suivantes :

- **'_id'** (entier) : nom générique SQLite pour avoir des valeurs uniques qui s'auto-incrémentent, permet d'identifier un objet dans notre table.
- **'web_id'** (chaîne de caractères) : identifiant de l'objet sur l'API web du CERI
- **'name'** (chaîne de caractères) : le nom

- **'thumbnail'** (chaîne de caractères) : l'URL de la vignette de l'objet. Je peux sauvegarder l'image correspondante dans le cache à partir de son URL.
- **'brand'** (chaîne de caractères) : marque
- **'time_frame'** (chaîne de caractères) : représente un tableau de dates sérialisé au format JSON.
- **'categories'** (chaîne de caractères) : représente une liste de catégories (string) sérialisée au format JSON.
- **'description'** (chaîne de caractères) : description
- **'pictures'** (chaîne de caractères) : représente une liste d'URL (string) sérialisée au format JSON.
- **'technical_details'** (chaîne de caractères) : les détails techniques
- **'last_update'** (chaîne de caractères) : date de dernière mise à jour

La clé primaire est évidemment '_id', on dira que chaque Item aura une paire (nom,marque) unique (en cas de conflit la requête SQL rollback). J'ai choisi d'utiliser les chaînes de caractères en grande majorité car il n'est pas prévu au cours du projet de faire des opérations complexes directement sur les données de la base et qu'il est très simple de sérialiser en JSON n'importe quelle classe avec la libraire 'gson'.

3.3 Stockage des images

Pour la mise en cache et le chargement des images, j'utilise la librairie 'Glide' (`https://github.com/bumptech/glide`). Pour avoir expérimenté la gestion manuelle de la sauvegarde d'image, je trouve que Glide est beaucoup plus simple d'utilisation. Auparavant pour pouvoir stocker des fichiers sur la carte SD il me fallait demander la permission à l'utilisateur, désormais je n'en ai plus besoin. Lorsque j'ai besoin de charger une image à partir d'une URL, je la récupère depuis le cache, en cas d'erreur (image introuvable), je la télécharge depuis le web et la sauvegarde dans le cache.

3.4 Affichage des images d'un objet

Pour afficher les images d'un objet du musée, j'utilise Android Image Slider (`https://github.com/smarteist/Android-Image-Slider`) qui permet d'ajouter un élement de type slideshow à une vue et de le personnaliser à souhait. Il permet notamment d'ajouter une description à chaque image.

3.5 Tri des objets dans la RecyclerView

Pour pouvoir ajouter des sections à ma RecyclerView j'ai récupéré une classe adaptateur en ligne qui fonctionne en parallèle à l'adaptateur déjà existant, ce qui m'a permis de le faire fonctionner sans avoir à changer mon code. Je n'ai fait que créer une classe pour générer des adaptateurs avec sections à partir d'une liste d'Item et de Section.

3.6 Barre de recherche

Pour la barre de recherche j'utilise une SearchView et pour l'adaptateur de RecyclerView j'ai créé un adaptateur filtrable. Ma méthode de filtre prend en considération le nom, la marque, l'année de production et les catégories, elle va chercher les informations qui commencent par ce qu'a tapé l'utilisateur et elle retournera les objets dont le nom correspond en priorité par rapport aux autres caractéristiques. La requête textuelle est insensible à la casse, l'ensemble de la liste est retournée si rien n'est écrit.

4 Rôle des classes

4.1 Les activités

- **MainActivity** : Représente l'activité principale de l'application, permet à l'utilisateur d'accéder aux autres activités.
- **ItemActivity** : Représente l'activité qui affiche des détails supplémentaires sur un objet en particulier.
- **SearchActivity** : Représente l'activité qui permet d'effectuer une recherche parmi la liste d'objets via une barre de recherche.

4.2 Les entités

- **Item** : Représente un objet du musée.
- **ItemImage** : Représente une image d'un objet du musée (image avec description).
- **SearchActivity** : Représente l'activité qui permet d'effectuer une recherche parmi la liste d'objets via une barre de recherche.

4.3 Les adaptateurs

- **RecyclerViewAdapter** : Représente l'adaptateur de RecyclerView qui possède comme données une liste d'Item (objet du musée).
- **SimpleSectionedRecyclerViewAdapter** : Représente l'adaptateur de RecyclerView qui possède comme données une liste d'Item ET des sections.
- **SearchRecyclerViewAdapter** : Représente l'adaptateur de RecyclerView qui possède comme données une liste d'Item ET une liste filtrée d'Item.
- **SliderAdapter** : Représente l'adaptateur de ImageSlider (slideshow pour affichage des images d'un objet) qui possède comme données une liste d'ItemImage (image d'objet du musée)

4.4 Les parseurs JSON

- **JSONResponseHandlerItem** : Permet de parser la réponse d'une requête HTTP retournant les informations d'un objet en JSON.
- **JSONResponseHandlerCatalog** : Permet de parser la réponse d'une requête HTTP retournant un catalogue d'objets en JSON.

4.5 Les classes utilitaristes

- **MuseumDbHelper** : Représente le gestionnaire de la base de données.
- **WebServiceUrl** : Permet de construire les URL de l'API web que l'on interroge au travers des requêtes HTTP entre autre.
- **AdapterCreator** : Permet de construire des SimpleSectionedRecyclerViewAdapter à partir d'une liste d'Item.
- **ApiComBny** : Permet d'éxecuter les différents scénarios avec l'API web.
- **GlideBny** : Permet de gérer le chargement d'images prises sur le web, leur mise en cache, etc..
- **Mk.HttpCon** : Permet d'envoyer des requêtes HTTP