

Fiche Model- View – ViewModel

1 Objectifs hauts-niveaux

- Découvrir l'implémentation du pattern MVVM en Android
- Utiliser une RecyclerView
- Première approche de la gestion des bases de données en Android
- Consommation asynchrone d'un service web en Android (optionnel)

2 Consignes générales

Pour cette fiche, vous devez :

- Implémenter l'exercice 1 : c'est un tutoriel à suivre pas à pas.
- Implémenter un exercice de votre choix parmi l'exercice 2 de suppression de données ou l'exercice 3 de filtrage de données.
- Soumettre votre code (**ZIP du main uniquement**) et un **lien** vers votre **vidéo** expliquant comment vous estimez valider les objectifs associés à l'exercice de votre choix (exercice 2 ou 3 : voir §3 pour les détails associés à la validation).
- Veillez à ce que votre **vidéo** ait une **durée d'environ 5 minutes**.
- **Introduire votre vidéo en donnant vos motivations pour avoir réalisé 0, 1 ou 2 exercice(s) optionnel(s) et via une très courte démo de votre application fonctionnelle.**

NB : si vous avez un ennui technique majeur vous stoppant dans la réalisation d'une vidéo, vous pouvez utiliser un fichier Word reprenant vos explications, des références à votre code et des captures d'écran.

Pour cette fiche, vous pouvez implémenter tant l'exercice 2 que l'exercice 3. Cela ne vous vaudra pas plus de points, mais vos compétences seront au top !

Vous pouvez aussi choisir de faire l'exercice 4 associé à la consommation asynchrone d'un web service. Là encore, vous n'aurez pas plus de points. Mais vous allez pouvoir acquérir des compétences incroyablement intéressantes, ou solidifier des compétences probablement déjà mises en œuvre dans votre PAE (mais via d'autres librairies).

N'hésitez donc pas à en faire un peu plus que ce qui est obligatoire. Pour vous y encourager, nous avons créé un délai de soumission spécialement long : la date butoir pour la **soumission** étant le **mardi 5 mai à 23h59**.

3 Objectifs à faire valider

Objectif	Priorité	Comment valider ?
Utiliser la BD ROOM et son DAO pour traiter des données	2	Expliquer via votre vidéo le code associé

Objectif	Priorité	Comment valider ?
		<ul style="list-style-type: none"> - soit à la suppression de données (exercice 2) - soit à l'interrogation de données (exercice 3)
Mettre en œuvre un ViewModel pour traiter des données	1	Expliquer via votre vidéo le code associé <ul style="list-style-type: none"> - soit à la suppression de données (exercice 2) - soit à l'interrogation de données (exercice 3)
Mettre en œuvre un repository pour faire une opération asynchrone vers la BD ROOM	2	Expliquer via votre vidéo le code associé à une opération asynchrone : <ul style="list-style-type: none"> - Soit à la suppression de données (exercice 2) - Soit d'interrogation de données (exercice 3)
Gérer des LiveData et observer leurs changements	1	Soit via l'exercice 2 : Veuillez répondre à cette question au sein de votre vidéo : comment se fait-il que votre RecyclerView est automatiquement mis à jour lorsque vous effacer un mot ? Soit via l'exercice 3 : Expliquer via votre vidéo comment vous avez géré la création de LiveData, où vous observez les changements et ce que vous faites en cas de changement.

4 Objectifs optionnels

- Découvrir comment consommer un web service via Android
- Approfondir la compréhension d'une gestion asynchrone de données

L'exercice 4 vous permet d'acquérir ou solidifier ces top compétences :

Objectif	Priorité	A valider ?
Rendre abstrait un web service en une interface Java via la librairie Retrofit	3	Aucun de ces objectifs optionnels n'est à faire valider.
Mettre en œuvre un repository pour interroger en asynchrone un web service	3	
Gérer des MutableLiveData pour notifier des changements de données	3	

5 Exercices

5.1 Exercice 1 : Codelab Room with a View

5.1.1 Instructions

Vous allez construire une application gérant une liste de mots dans une base de données. Cette application sera structurée selon l'architecture de référence proposée par Google.

Suivez pas à pas toute les instructions du codelab:

<https://codelabs.developers.google.com/codelabs/android-room-with-a-view/#0>

5.1.2 Objectifs à faire valider

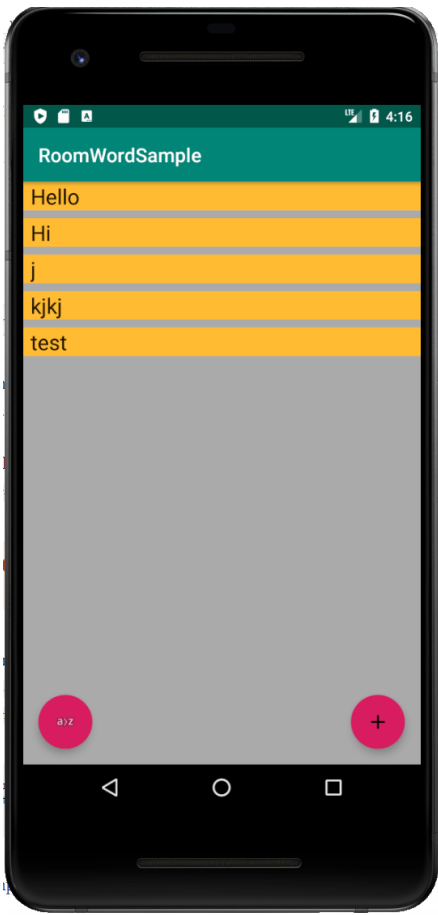
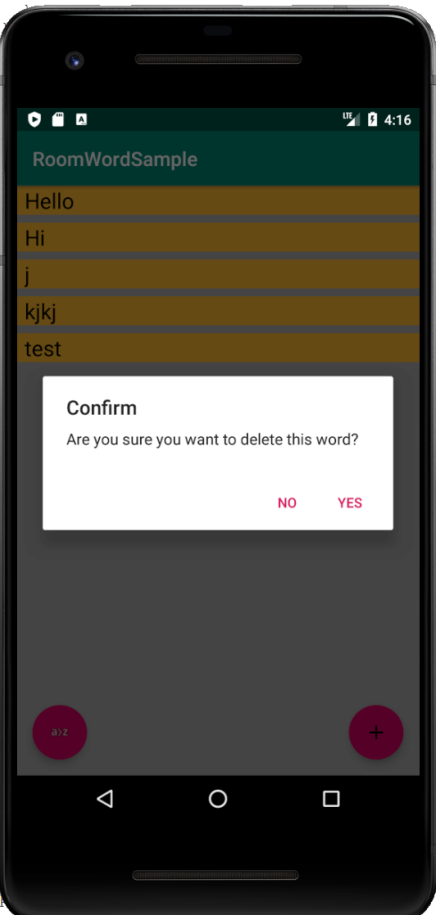
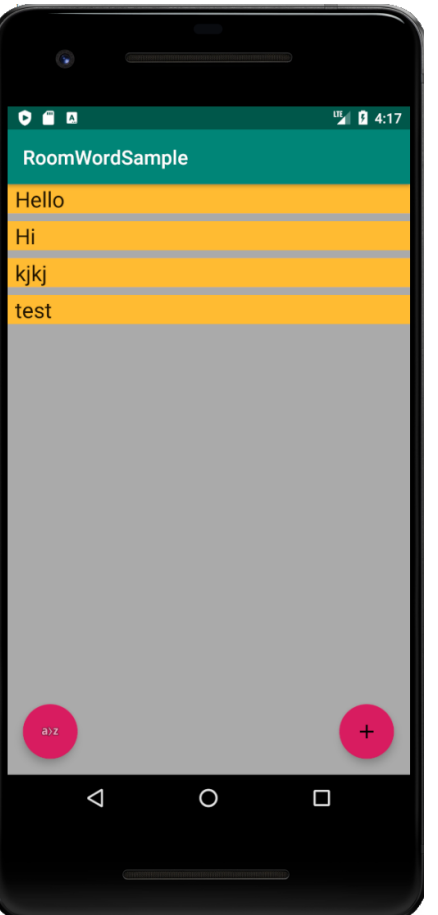
Aucun objectif n'est à faire valider pour ce tutoriel. Celui-ci sert comme base aux exercices qui suivent.

5.2 Exercice 2 : Ajout de la fonctionnalité de suppression d'un mot

5.2.1 Introduction

Ajoutez la suppression d'un mot lors d'un clic long. À la suite d'un long clic sur un mot, affichez un message « Voulez-vous supprimer ce mot ? » et deux boutons « NON » et « OUI » pour initier les traitements.

Exemple : Voici une succession de snapshots montrant comment on supprime le mot « j ».

		
<p>1) Long clic sur le mot « j »</p>	<p>2) Confirmation de l'effacement</p>	<p>3) Le mot est supprimé de la DB et la RecyclerView est mise à jour</p>

5.2.2 Notions

Vous pouvez choisir comment implémenter la fonctionnalité d'effacement, sans vous soucier des notions qui suivent. Ces notions sont là pour vous aider dans votre développement.

- Pour construire une requête de type **delete** avec paramètre(s) via votre DAO : <https://developer.android.com/reference/androidx/room/Delete>

- Utilisation d'un AlertDialog.Builder (pour un Yes/No Dialog Box.

- Tous sur les Dialogs :

<https://developer.android.com/guide/topics/ui/dialogs>

- Exemple de code :

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);

builder.setTitle("Confirm");
builder.setMessage("Are you sure?");

builder.setPositiveButton("YES", new
DialogInterface.OnClickListener() {

    public void onClick(DialogInterface dialog, int which) {
        // Do nothing but close the dialog
        dialog.dismiss();
    }
});

builder.setNegativeButton("NO", new
DialogInterface.OnClickListener() {

    @Override
    public void onClick(DialogInterface dialog, int which) {
        // Do nothing
        dialog.dismiss();
    }
});

AlertDialog alert = builder.create();
alert.show();
```

- Pour gérer une opération de type **delete** asynchrone via votre Repository :
inspirez-vous de la méthode **insert** donnée dans le tutoriel pour votre Repository
- Gestion d'un observateur de clics au niveau de votre Adapter et de la RecyclerView :
 - soit sera géré selon le même genre de code que dans la fiche précédente (voir vidéo <https://youtu.be/PP5x4wg-818>).
 - soit l'observateur de clics peut être directement une instance de la MainActivity :
au sein de l'adaptateur, utilisation de **WordListAdapter(Context**

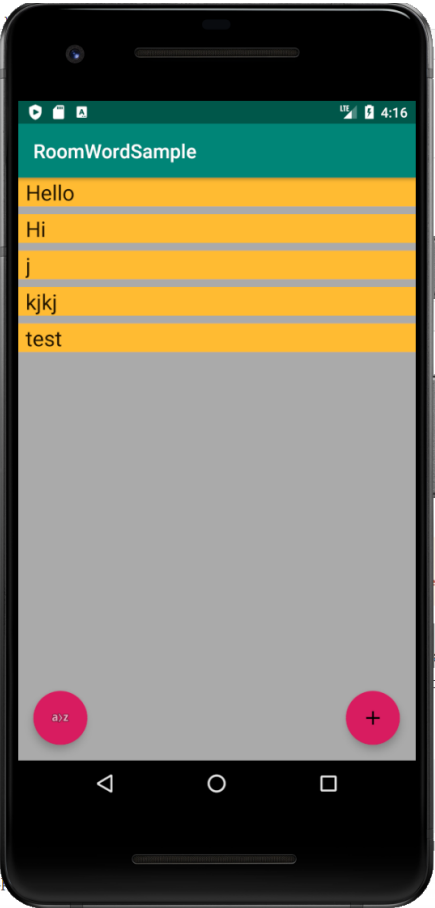
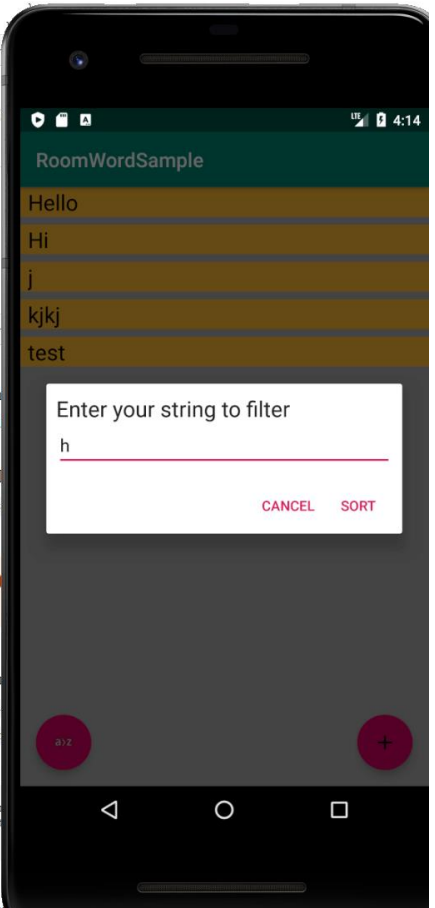
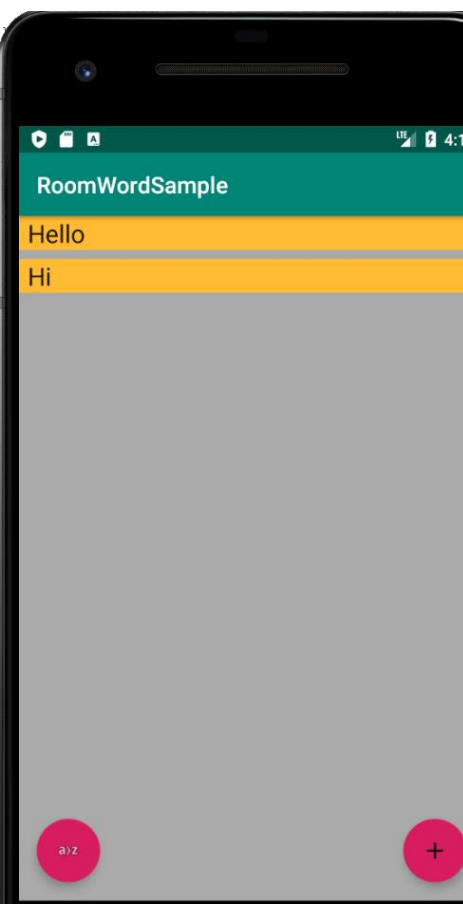
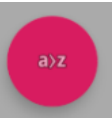
context) pour enregistrer l'observateur en utilisant le paramètre **context** que l'on caste en **MainActivity** pour le passage de l'observateur de clics (instance de MainActivity) à l'Adapter.

5.3 Exercice 3 : ajout de la fonctionnalité de filtrage

5.3.1 Instructions

Filtrez tous les mots sur base de l'icône « Filtre ». Au clic sur l'icône « Filtre », un « Prompt » demande l'introduction d'une chaîne de caractère pour filtrer la liste de mots.

Exemple : Voici une succession de snapshots montrant comment on filtre les mots commençant par la lettre « h ».

		
1) Clic sur le bouton pour filtrer 	2) Confirmer l'effacement à l'aide d'un « Prompt »	3) Il n'y a plus que les mots filtrés qui sont affichés

5.3.2 Notions

Vous pouvez choisir comment implémenter la fonctionnalité de tri, sans vous soucier des notions qui suivent. Ces notions sont là pour vous aider dans votre développement.

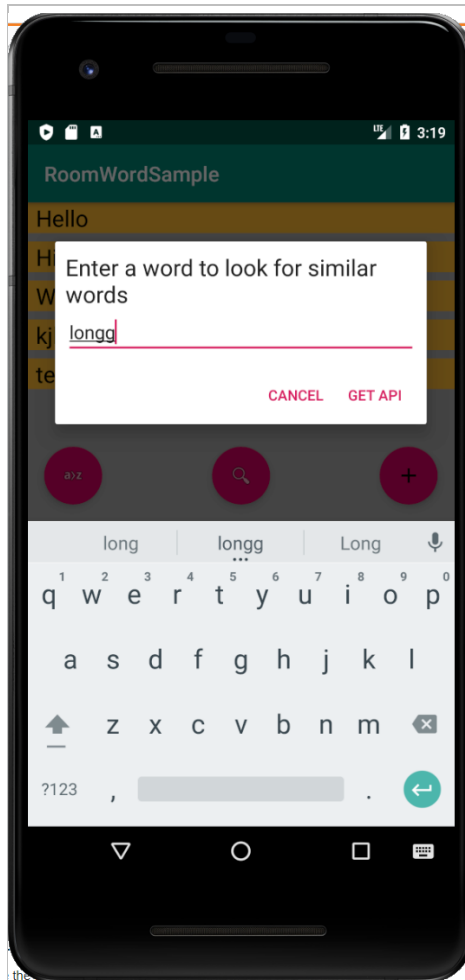
- Utilisation d'un AlertDialog.Builder pour le prompt d'introduction d'une chaîne de caractères :
 - Tous sur les Dialogs :
<https://developer.android.com/guide/topics/ui/dialogs>
 - Exemple avec un prompt : <https://mkyong.com/android/android-prompt-user-input-dialog-example/>
- Il vous est conseillé de travailler avec des LiveData :
 - D'ajouter une nouvelle méthode du style « LiveData<List<Word>> getAllWordsFiltered(String startOfWords) » là où approprié (pensez à votre DAO, Repository, ModelView).
 - D'écouter les changements de votre LiveData lors du clic sur le bouton « Filtre » de votre « Prompt » permettant de filtrer la liste de mots.
 - De mettre à jour le data set de votre Adapter lors d'un changement de votre LiveData.
- Pour construire une **query** avec paramètre(s) via votre DAO :
<https://developer.android.com/reference/androidx/room/Query>

5.4 Exercice 4 : ajout de la fonctionnalité de trouver des mots similaires

Pour ceux qui ont envie de développer des compétences optionnelles très intéressantes, vous pouvez réaliser l'exercice suivant : connectez-vous à un web service fournissant une liste de mots similaires à un mot donné.

Nous allons interroger le service web <https://www.datamuse.com/api/> lors d'un clic sur le bouton « Search » et de l'introduction de critères au sein d'un Prompt.

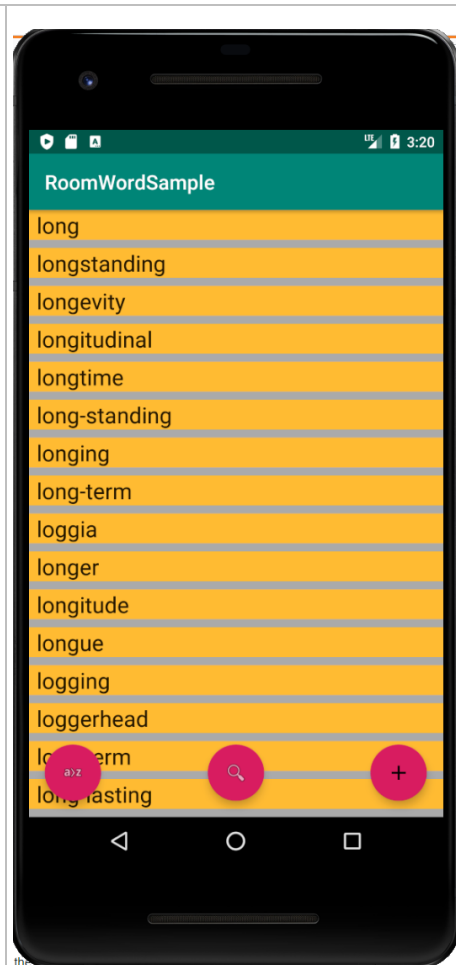
Exemple : Voici une succession de snapshots montrant comment on cherche une liste de mots – en anglais - similaires à la chaîne « longg ».



1) Clic sur le bouton « Search »



2) Introduction du mot à chercher via un « Prompt » et confirmation en cliquant sur GET API



3) Affichage de la liste de mots similaires renvoyés par l'API

Notons ici que l'appel fait à l'API correspond à l'URL :

<https://api.datamuse.com/sug?max=100&s=longg>

N'hésitez pas à accéder à cette URL via votre browser pour voir le JSON associé.

5.4.1 Notions

Vous pouvez choisir comment implémenter la fonctionnalité de recherche de mots similaires, sans vous soucier des notions qui suivent. Ces notions sont là pour vous aider dans votre développement.

- Pour découvrir tous les paramètres pouvant être passés au web service (ou API), lire : <https://www.datamuse.com/api/>

Par exemple, pour pouvoir récupérer tous les mots en anglais qui pourrait s'apparenter à une recherche sur « hi » (par défaut, un maximum de 10 mots sont retournés par l'API) <https://api.datamuse.com/sug?s=hi>

- Rendre abstrait une API en une interface Java :
Utilisation de la librairie Retrofit et de Gson (pour désérialiser le JSON renvoyé par l'API datamuse) : <https://square.github.io/retrofit/>
 - Ajout dans Gradle des librairies, modification de build.gradle (Module :app) :

```
implementation 'com.squareup.retrofit2:retrofit:2.8.1'  
implementation "com.squareup.retrofit2:converter-gson:2.8.1"
```

- « Solution type » pour la création d'une interface permettant de faire une requête GET à l'API DataMuse :

```
import java.util.List;  
  
import retrofit2.http.GET;  
import retrofit2.http.Path;  
import retrofit2.Call;  
import retrofit2.http.Query;  
  
public interface DataMuseService {  
    @GET("sug?max=100")  
    Call<List<WordFromAPI>> listSimilarWords(@Query("s") String  
        querySimilarWords);  
}
```

- « Solution type » pour la création d'une classe permettant de désérialiser le JSON renvoyé de l'API vers une liste de Word (vous pouvez voir ce qu'est renvoyé par l'API en tapant <https://api.datamuse.com/sug?s=hi> dans votre browser) :

```

public class WordFromAPI {
    private String word;
    private int score;
    WordFromAPI(){
    }
    public int getScore() {
        return score;
    }
    public String getWord() {
        return word;
    }
}

```

- La difficulté de cet exercice est de gérer des données de manière asynchrone en Java. En effet, dans une application Android, il n'est pas possible d'effectuer une opération réseau dans le « main thread ».
- NB : il en est de même pour des requêtes Room, celles-ci ne peuvent pas être exécutées sur le thread principal.
- C'est le rôle du Repository de faire la requête vers le Web service :
 - L'idée est de s'inspirer d'un exemple détaillé pour un appel synchrone à une API (exemple officiel de la librairie Retrofit) : <https://github.com/square/retrofit/blob/master/samples/src/main/java/com/example/retrofit/SimpleService.java>
 - Puis de transformer le code de votre appel synchrone (qui génèrerait une exception au sein de votre environnement) vers un appel asynchrone (vers votre API). Voici un lien en guise d'inspiration : <https://howtodoinjava.com/retrofit2/retrofit-sync-async-calls/>
- Pour utiliser le résultat de votre appel asynchrone vers le web service, afin de nourrir votre Adapter de données :
 - Il est proposé de créer des LiveData.
Pour ce faire, vous pouvez découvrir le concept de **MutableLiveData** et de la méthode **setValue()** permettant de distribuer une valeur à tous les observateurs actifs.
Voici un site donnant des infos intéressantes au sujet de la création et de la gestion de LiveData :

<https://medium.com/@taman.neupane/basic-example-of-livedata-and-viewmodel-14d5af922d0>

- Attention que pour nourrir votre Adapter, celui-ci attend un objet de type List<Word>. Pour ce faire, dans MainActivity, vous pourriez avoir un observateur de LiveData qui, une fois un changement de données, fait appel à adapter.setWords(...).
- Donner la permission internet (AndroidManifest.xml) :
`<uses-permission android:name="android.permission.INTERNET" />`