Université de Lille

Technologies pour les Applications Connectées

Compte rendu projet final
Développement d'une application Android

Application :
MEALIST

Binôme :
CARLIER Annelise
BRICOUT Gaël

Année :
2020 - 2021

Sommaire

INTRODUCTION

Cahier des charges

→ Dans un premier temps, nous devons, via des requêtes, récupérer des données d'une API de notre choix.
→ Ensuite, nous devons en déduire et afficher des items en liste et en grille
→ Il faut pouvoir passer d'un affichage en liste, ou en grille
→ Chaque items affiché doit comporter des images, et du texte
→ Nous devons stocker des données en local
→ Il faut donner à l'application un look professionnel et esthétique, qui respecte les standards d'Android.

Pour avancer dans la programmation de l'application, nous suivrons une répartition des tâches afin de pouvoir nous organiser. Si celui-ci change, nous indiquerons à droite les raisons de changement.

<table>
<tr><th></th><th colspan="2">Missions</th><th colspan="2">Réponses / changements / remarques</th></tr>
<tr><th>Semaine du</th><th>Gaël</th><th>Annelise</th><th colspan="2"></th></tr>
<tr><td>05/10/2020</td><td colspan="2" rowspan="2">Déterminer l'API</td><td colspan="2">Marmiton ?</td></tr>
<tr><td>12/10/2020</td><td colspan="2">Spoonacular ?</td></tr>
<tr><td>19/10/2020</td><td colspan="2">Déterminer l'architecture</td><td colspan="2">MVVM</td></tr>
<tr><td>26/10/2020</td><td>Programmer le squelette / tabLayout / viewPager2</td><td>Recherche d'utilisation d'API pour application Android (Volley)</td><td rowspan="2"><-- Changement, Gaël était en vacances ce qui lui a permis d'avancer bien plus que prévu sur l'application</td><td>Volley ? Retrofit ?</td></tr>
<tr><td>02/11/2020</td><td>Programmer l'API / recycler view du premier fragment</td><td>Relecture et correction de petits détails</td><td></td></tr>
<tr><td>09/11/2020</td><td>Programmer le passage de SearchFragment à DetailsFragment</td><td>Récupérer l'ID & affichage des Ingrédients dans DetailsFragment</td><td colspan="2"></td></tr>
<tr><td>16/11/2020</td><td colspan="2">Programmer changement de vue de liste à grille (et vice versa)</td><td colspan="2">Changement : Avancement + rapide que prévu sur le DetailsFragment</td></tr>
<tr><td>23/11/2020</td><td colspan="2" rowspan="2">Mise en place de la base de données</td><td colspan="2"></td></tr>
<tr><td>30/11/2020</td><td colspan="2">SQLlite ?</td></tr>
<tr><td>07/12/2020</td><td>Récupérer éléments dans la base de données</td><td>Insérer éléments dans la bdd</td><td colspan="2">Room</td></tr>
<tr><td>14/12/2020</td><td>Créer les requêtes afin d'augmenter le nombre d'ingrédients</td><td>Affichage des données avec Recycler view de la bdd</td><td colspan="2">Changement, ces tâches ont été gérées avant la gestion des plats favoris</td></tr>
<tr><td>21/12/2020</td><td colspan="2" rowspan="2">Gestion des plats favoris</td><td colspan="2"></td></tr>
<tr><td>28/12/2020</td><td colspan="2"></td></tr>
<tr><td>04/01/2021</td><td colspan="2" rowspan="2">Rédaction des rendus / Finalisation des détails</td><td colspan="2"></td></tr>
<tr><td>11/01/2021</td><td colspan="2"></td></tr>
</table>

Figure 1 : Diagramme de répartition des tâches

Présentation de l'application Mealist

Notre projet est de créer une application qui respecte un certain cahier des charges (voir ci-dessus).

Nous avions l'envie de programmer une application qui pourrait nous servir dans la vie de tous les jours, et qui pourrait vraiment nous apporter un plus dans notre quotidien.

Une des problématiques que nous avons tous les deux, est de trouver des idées de repas dans la vie de tous les jours. Nous voulions donc une application qui serait capable de nous afficher des idées de repas.

De ce fait, nous avons eu l'idée de créer une application de recettes de cuisine. Plus précisément, à partir d'une recherche (d'un ingrédient/d'un nom de plat), une liste s'affiche à l'écran montrant tous les plats possibles à partir de cette recherche. Lors d'un clic sur un des éléments de la liste, une page s'affiche avec inscrit tous les ingrédients nécessaires à la réalisation de ce plat.

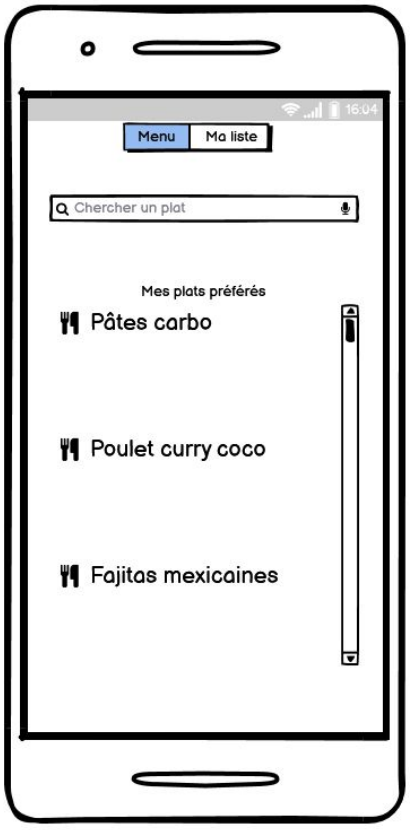
Afin de faciliter encore plus la vie quotidienne, l'utilisateur pourrait alors choisir les ingrédients dont il a besoin, et en faire une liste de courses. Les ingrédients choisis seraient stockés dans une liste annexe, qui est en fait gardée dans une base de données locale. Celui-ci pourrait aussi supprimer et ajouter des éléments de sa liste, qui agissent directement sur la base de données locale.

L’application serait utilisable sans connexion internet. En effet, nous stockons en base de données locale les plats choisis par l’utilisateur. Autrement dit, chaque repas cliqué serait stocké dans un historique. En plus de pouvoir afficher à l’utilisateur les repas les plus choisis, cela permettrait d’utiliser l’application sans connexion internet.

Enfin, l’utilisateur sera en mesure de choisir des repas qui seront “favoris” et seront affichés en premier dans la liste de choix de la première page.

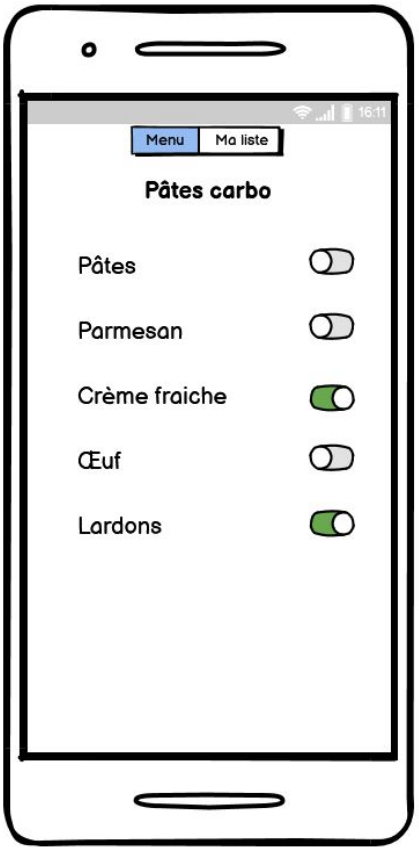
Afin de mieux comprendre nos idées, voici la modélisation de notre application :

Modélisation de l’application



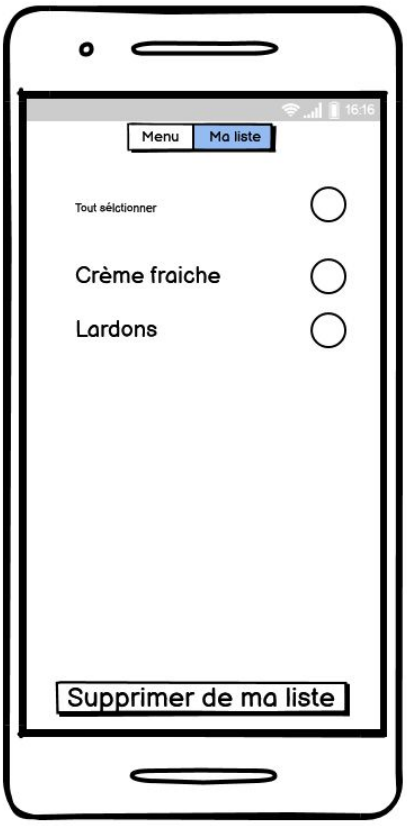
- Notre idée est de faire une application, qui, à partir d’une recherche, nous retourne et affiche tous les plats possibles à partir de cette recherche.
- Les premiers éléments affichés sont les plats favoris / les plats les plus cliqués.

Figure 2 : Écran d’accueil



- Pour chaque clic sur un des éléments retournés, une page s’affiche et nous montre les ingrédients de cette recette.
- Des petits boutons à côté de chaque ingrédient sont alors disponibles afin de l’ajouter dans une liste, une liste de course.

Figure 3 : Lors d’un clic sur pâtes carbo (image précédente)



- Cette liste est stockée en interne dans une base de données locale.
- Il sera aussi possible d’ajouter et de réduire les quantités des ingrédients affichés dans la liste de course.
- Nous pourrons gérer le nombre d'ingrédients, et les quantités via des boutons.

Figure 4 : Liste de course finale, stockée en local

MANUEL D'UTILISATION

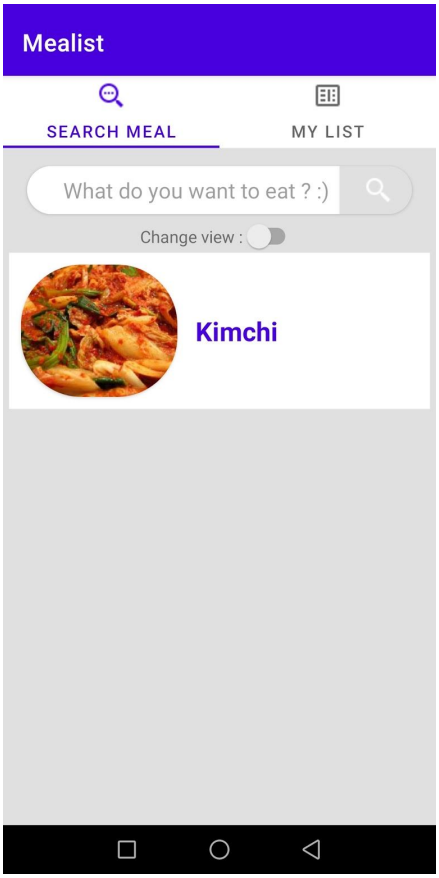
Comment installer l'application

Ajouter l'APK fourni dans le rendu sur votre téléphone personnel.
Ou cloner le repository de notre Git :
https://gitlab-etu.fil.univ-lille1.fr/carliera/bricout_carlier_tac

Pour une meilleure utilisation de l'application, connectez vous en WiFi via routeur ou connexion mobile puis ouvrez l'application sur votre téléphone.

Comment l'utiliser ?

Vous tomberez directement sur une page composée d'un champ de recherche



Le champ de recherche permet de chercher, via notre API des milliers de plats. Il est possible de rechercher des ingrédients, des plats, des nutriments etc…

Une recherche est envoyée lors du clic sur le bouton **Search**, ou via la pression sur la touche **Enter**

Le bouton Change View permet de modifier l'affichage, d'échanger entre l'affichage en "liste" ou en "grille".

Lors du clic sur un des éléments affichés, une page s'ouvrira avec tous les ingrédients nécessaires à la réalisation. Il sera alors possible, via le clic sur le bouton "+", de les ajouter dans votre, présente dans l'onglet **MY LIST**, qui fera office de "liste de course".

Dans l'onglet **MY LIST**, vous pourrez ajouter ou supprimer des ingrédients, afin de mettre à jour votre liste à votre guise.

Figure 5 : premier affichage de l'application.

Spoonacular API - contrat d'interface

Afin de réaliser cela, nous voulions utiliser une API en particulier, celle du site marmiton. Cependant, aucune API n'était fournie par le site. Nous avons alors cherché une alternative.

Nous avons trouvé spoonacular, un site qui, comme Marmiton, propose énormément de recettes. Cependant, à l'inverse de Marmiton, Spoonacular possède une API très étoffée et très accessible, qui nous permettra d'aller chercher les données facilement.

Pour avoir accès aux données fournies par Spoonacular, il nous a tout d'abord fallu générer une clé. Cette clé est à insérer dans toutes nos requêtes vers l'API sous la forme :

```
https://api.spoonacular.com/{information}?apiKey=OUR-API-KEY
```

Il y a plusieurs URL qui permettent d'aller chercher des données. Le site propose une documentation pour chaque requête possible sur leur API. Seulement deux nous serviront. Ces requêtes sont de type "GET". La première est celle qui nous permettra de trouver des repas lors d'une recherche : par requête, par ingrédients et par nutriments (voir **Figure 6**). Les réponses sont en JSON. Pour notre application nous récupérons uniquement l'ID des plats, leurs noms ainsi que l'image que nous afficherons. La deuxième requête sera la recherche d'ingrédients d'une recette, via son id (voir **Figure 7**). Ici nous récupérons le nom et l'image de l'ingrédient.

Figure 6 : documentation de Search Recipes de Spoonacular

Figure 7 : documentation de Get Recipe Ingredients de Spoonacular

Afin d’utiliser cette API, nous avons voulu utiliser la bibliothèque **Volley** afin de faire les requêtes, tout d’abord pour la facilité d’utilisation de ses fonctions. Les réponses de l’API étant en Json, la bibliothèque Volley permet de récupérer facilement les éléments JSON des réponses de l’API. De plus, l'objectif de Volley est de gérer tous les besoins en matière de réseaux pour Android en particulier.

DESCRIPTION

Architecture

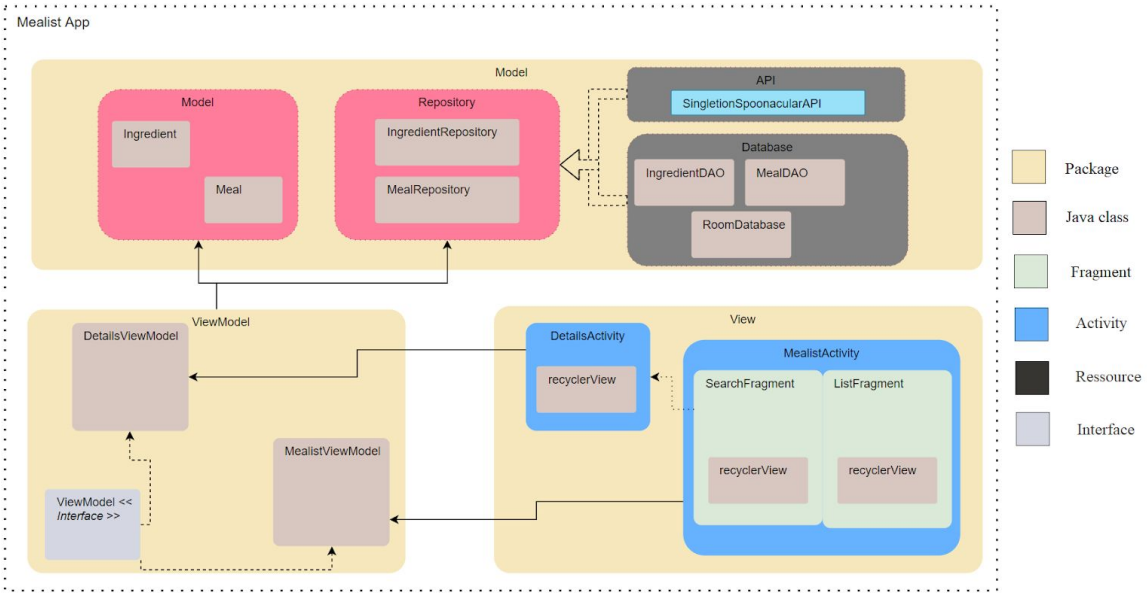


Figure 8 : Diagramme de présentation de l'architecture choisie

Nous avons choisi de nous orienter vers une architecture en **Model View ViewModel**.

Tout d'abord, la structure MVC a été écartée tout de suite, car celle-ci est efficace seulement pour des applications simples avec un ou deux écrans.

De plus, contrairement au MVP, le MVVM utilise un mécanisme de liaison de données, le DataBinding & LiveData, qui sont des concepts assez important qu'on retrouve beaucoup dans les métiers du E-Services (notamment en ReactJS / Angular etc…), ce qui nous permet de mieux comprendre ce modèle de programmation.

Ce modèle à été privilégié car il paraît être bien plus utilisé en entreprise. En dehors de son "ancienneté", ce modèle reste très utilisé, ce qui peut être prouvé grâce à l'implémentation d'un objet ViewModel, que nous utiliserons, sur Android Studio.

En plus d'être fortement poussé par Google, cette structure permet plusieurs choses :

- Possibilité d'interchanger plusieurs package, si on souhaite changer de modèles, views etc, grâce à ses composants.
- Les tests unitaires sont donc plus faciles, car rien ne dépend de la vue.
- Une maintenance plus facile.

Quelques inconvénients quand même :

- Comprendre et maîtriser le concept de DataBinding
- Selon le créateur du patron MVVM, John Gossman, si une application est très grande, généraliser le MVVM peut devenir très complexe.
- Beaucoup de contraintes liées à cette architecture.

Cependant, ces inconvénients n'ont pas su nous faire changer d'avis. Nous sommes donc partis sur une application d'architecture Model View ViewModel.

Notes de conception de l'affichage (View)

- Afin de gérer l'affichage, la partie **View** de l'application, nous utilisons **ViewPager2** lié à un **Adapter**. Plus précisément, ce sont deux fragments **SearchElementFragment** et **ShoppingListFragment** qui alternent à l'affichage, le fragment qui s'occupe de l'affichage de la recherche et le fragment qui affiche la liste de courses. Les deux sont gérées par une activité **MealistActivity**, qui communique avec le **ViewModel MealistViewModel**. Un clic sur un des éléments du recycler view, une nouvelle activité **DetailsActivity** est lancée avec tous les ingrédients affichés. Cette seconde activité est liée à son propre **ViewModel DetailsViewModel** (qui nous permet de retrouver facilement des fonctions, et donc de rendre l'application plus maintenable).

- Chaque Fragment possède un **dataBinding**, qui le lie au **ViewModel** de l'activité. C'est le **viewModel** qui gère les **LiveData** et récupère les données. Les ViewModel sont des objets liés à l'outil **AndroidViewModel** proposé par Android Studio.

- Lors d'une recherche d'aliment, nutriment ou de plat dans l'onglet recherche, la commande est directement envoyée du **ViewModel** au **Repository** qui est en contact direct avec l'API. Via une requête **Volley**, l'application va répondre à la requête avec une réponse sous forme d'un objet JSON. La réponse est redirigée via un **Callback** (**VolleyCallBack**) vers le **ViewModel**, qui va mettre à jour la liste affichée par le **RecyclerView,** sans être gêné par le stock des données (impossible à cause de la nature asynchrone de la requête **Volley**).

- Il est possible, via un bouton sur le premier affichage de l'application (Change view) il est possible de changer le mode d'affichage des éléments de la liste (Liste ou grille)

- Pour la liste d'ingrédients dans l'onglet **MY LIST** un petit numéro est indiqué (voir **Figure 9**) dans le cas où l'utilisateur souhaite ajouter ou retirer des ingrédients à la main. La quantité est affichée via le recyclerViewAdapter, où on multiplie la quantité de base par le compteur.

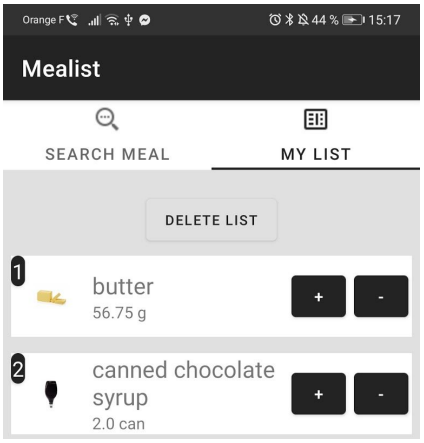


Figure 9 : affichage du fragment ListShoppingFragment

- La ligne de code "*android:configChanges="keyboardHidden|orientation|screenSize"* permet de ne pas être embêté par un changement d'orientation du téléphone.

Description détaillée des fonctionnalités

Pour plus de précisions, la Javadoc est disponible dans le dossier **Doc** de l'application présente sur le git.

→ Version sans internet

Chaque clics sur des repas sont stockés dans une base de données locale, afin de créer un historique. A chaque lancement de l'application, l'historique est affiché. Cela permet d'une part de permettre à l'utilisateur de retrouver les repas qu'ils ont cherché précédemment, et d'autre part de créer une version "sans internet" dans le cas où l'utilisateur n'a pas accès à un réseau. Les éléments sont affichés par ordre de nombre de clics sur le repas. Cependant, un utilisateur sans connexion ne pourra rechercher de repas, ni consulter les ingrédients des repas. Dans **DetailsActivity**, il sera affiché les ingrédients si connexion internet il y a, sinon, une EditText sera affiché signifiant à l'utilisateur que cela nécessite une connexion internet.

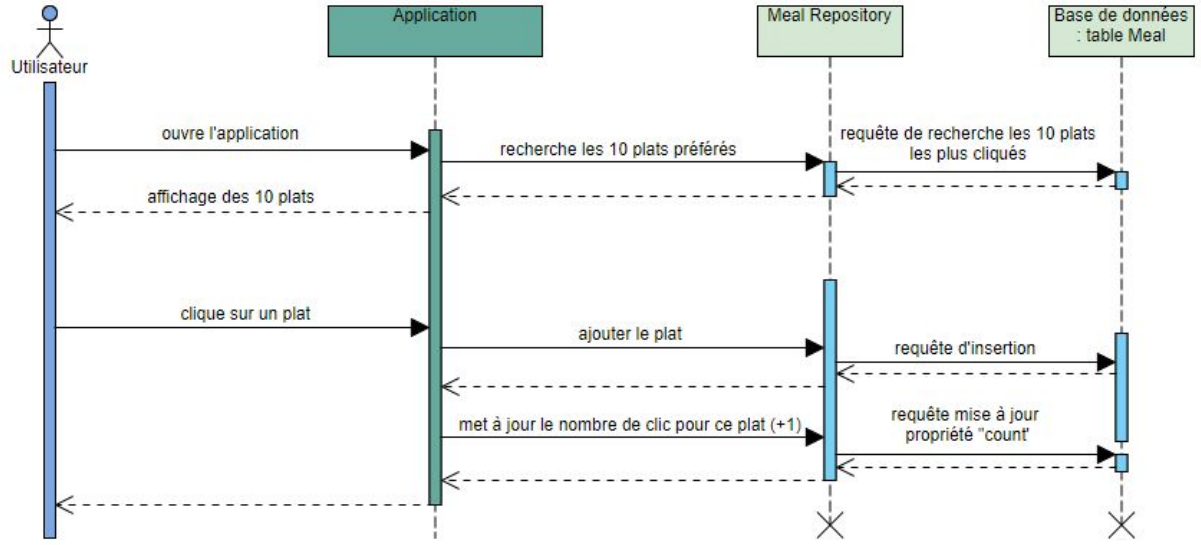


Figure 10 : diagramme de séquence des interactions entre l'utilisateur et la base de données lorsque l'utilisateur se trouve sur la page principale

→ Ajout d'un ingrédient dans la BDD

Lors d'un clic sur le bouton "+" dans l'activité des détails, l'aliment est ajouté à la base de données locale, qui est affichée dans le second fragment (**ShoppingListFragment**). Dans ce fragment, il est possible d'ajouter de la quantité de chaque ingrédients. En fait, on ajoute la quantité de base de l'aliment à la quantité affichée.

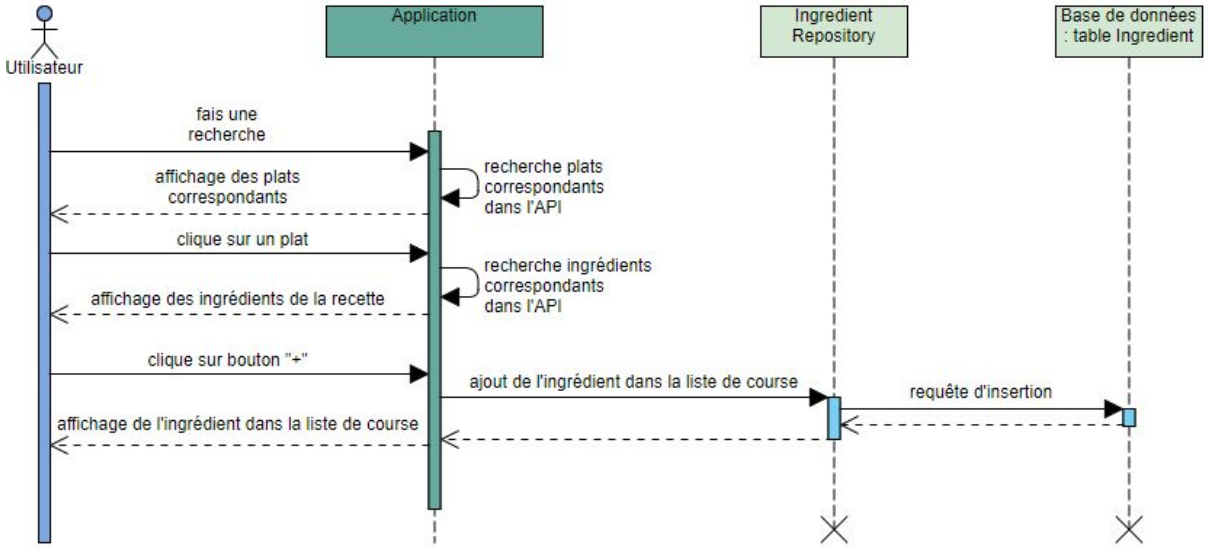


Figure 11 : diagramme de séquence des interactions entre l'utilisateur, l'application et la base de données lorsque l'utilisateur veut ajouter un ingrédient à sa liste

→ Gestion de la quantité d'ingrédient dans la BDD

Dans le fragment **ShoppingListFragment** (voir **Figure 9**) se trouve deux boutons ("+" & "-"). Lors d'un clic sur l'un des deux, on incrémente ou décrémente un compteur, qui est affiché directement sur l'ingrédient en question (et stocké dans la base de données). La quantité affichée sera en fait la quantité de base stockée multipliée par ce compteur. Une fois arrivé à 0, le compteur supprime l'élément dans la base de données.

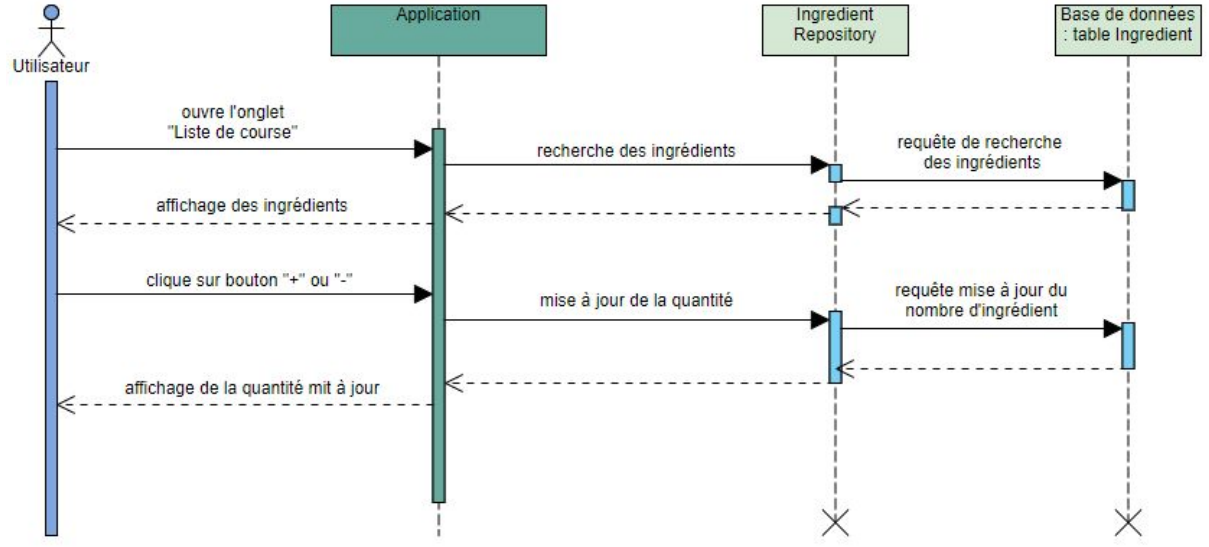


Figure 12 : diagramme de séquence des interactions entre l'utilisateur et la base de données lorsque l'utilisateur se trouve sur l'onglet "Liste de course"

→ Système de base de donnée

La base de données est gérée par des DAO (voir **Figure 13**) et l'outil **Room**. Chaque DAO gère les requêtes sur la base de données (SQLlite) et la classe **MealistRoomDatabase** la dirige. Toutes les données récupérées ou stockées transitent par le **Repository**, qui s'occupe de la gestion de celles-ci.

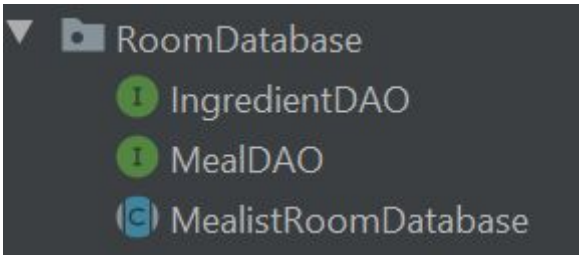


Figure 13 : organisation de l'accès à la base de données via l'application

BILAN ET PERSPECTIVES

Nous avons terminé l'application en temps voulu et avons respecté chaques champs du cahier des charges. De plus, nous avons su respecter notre planning de répartition des tâches, qui a été pensé afin que chacun de nous puisse travailler sur différents aspects de la programmation de l'application.

La réalisation de cette application nous a permis d'en apprendre bien plus sur les contraintes du développement d'applications avec une architecture. En effet, les architectures sont le squelette qui permet de bien programmer, d'une part afin de retrouver facilement du code, de le maintenir. D'autre part pour les mises à jour qui sont grandement simplifiées.

De plus, ce projet a pu nous initier à l'Android, qui inclut le fonctionnement, les méthodes, et aussi les outils (Room, APIs, Fragments etc…). Personnellement, j'ai beaucoup apprécié travailler sur Android. Etant fan du développement Web via Angular/React, j'ai pu retrouver des marques, et des outils supplémentaires qui rendent le développement bien plus agréable. Si l'avenir me le permet, il est possible que je me spécialise en développement mobile, suite à cette expérience.

Enfin, il est possible que nous implémentions d'autres fonctionnalités dans notre application, notamment la possibilité de filtrer les recherches selon certains paramètres (nutriments, sucre etc…), ou encore l'accès à la recette précise de chaque repas. De plus, nous aimerions la publier sur l'app store.