DAVID Loïc

PetCare

Projet de bachelor Systèmes d'Informations et Sciences des Services,
CUI 2021

02/09/2021

1 Table des matières

2 Problématique

Quand on adopte un animal de compagnie, on devient responsable de son bien-être et de sa santé.
Il faut lui donner à manger et à boire, mais ils ont aussi d'autres besoins : un chat a besoin d'une litière, un chien doit être promenée, les oiseaux peuvent avoir besoin d'os de seiche (pour l'apport de calcium), etc...

L'inexpérience ou simplement un planning chargé peut parfois nous faire oublier de dédier un peu de temps à notre/nos animaux.

Une application permettrait d'éviter ces oublies.

3 État de l'art

3.1 Articles scientifiques

3.1.1 Applications de coaching

Beaucoup d'articles et de papiers scientifiques parlent de l'utilisation d'applications mobiles et de leur effet sur la santé.

Une étude de 2020 (So Mi Jemma Cho, 2020) compare les résultats de la baisse de poids pour trois groupes d'individu : un groupe de contrôle, un groupe utilisant une application de coaching et un groupe utilisant l'application en plus d'un coaching personnalisés, les chiffres indiquent que les services de coaching ont un impact sur la baisse de poids, même si l'ajout d'un coaching personnalisé améliorent la baisse l'application seule présente tout de même une utilité.

Certains travaux (Monteiro-Guerra F, 2020) établissent que les activités sportives telles que la marche aident les personnes ayant survécu à un cancer du sein et que nombre de ces personnes ne font pas autant d'activité physique que nécessaire pour vraiment se remettre.
Les auteurs travaillent donc sur le design d'une application spécialement conçu avec l'aide d'experts en interface personne-machine et avec les utilisateurs pour s'assurer d'obtenir un résultat qui répondra vraiment aux besoins des utilisateurs.

J'ai trouvé un autre papier (Bogaerts A, 2020) qui dit qu'il n'existe pas d'applications dédiées pour aider les femmes ayant donné naissance récemment à perdre le poids pris pendant la grossesse.
Ce papier pour but de concevoir une application dédiée pour aider ces femmes à atteindre un poids optimal et à adopter un mode de vie plus sain.

Les auteurs de cet autre papier (Cueto V, 2019) ont mené une étude où ils concluent qu'une application aide à garder l'engagement des enfants obèses et en surpoids pour un programme de perte de poids par rapport à des rendez-vous classique chez un praticien.

Ces articles montrent donc qu'il y a un réel intérêt à utiliser des applications mobiles pour aider les individus à être en meilleure santé et à améliorer leur bien-être et qu'elles peuvent augmenter la motivation des utilisateurs à exécuter des tâches journalières nécessaire à leur bien-être.

3.1.2 Trouver articles liant informatique et animaux

L'informatique et les téléphones peuvent aussi être utilisés pour aider à prendre soin des animaux qu'il s'agisse du bétail ou d'animaux de compagnie.

Dans cet article de journal (S. O. Ogunlere, 2018), les auteurs se concentrent sur l'utilisation des puces RFID, qui sont utilisées pour retrouver du bétail perdus ou volés, pour savoir quand les animaux ont besoin d'être nourris et ne soient pas négligés par les fermiers et pour différencier les animaux entre eux.
Les auteurs proposent que des systèmes automatisés pourraient récupérer ses informations pour nourrir le bétail automatiquement ou pour recueillir des informations sur les animaux.

D'après certains chercheurs de Tanzanie (Esron D. Karimuribo, 2016), les services vétérinaires dans les zones rurales de leur pays sont quasiment inexistants, les auteurs ont trouvé que 96% des fermiers possédaient des téléphones portables mais seulement 10% avait des smartphones. Ils proposent donc que le développement d'un système de consultation vétérinaire par téléphone pourrait aider à diagnostiquer les problèmes du bétail, et fournir un service vétérinaire à ces zones très mal desservies.

Certains travaux ((Alexis Anne A Luayon, 2019) et (Junior, 2020)) parlent de l'utilisation de smart sensors et de l'Internet of Things pour monitorer la santé des animaux de compagnie et informer leur propriétaire de leur état afin qu'ils s'occupent mieux d'eux.
Cela permet aussi de connaître leur comportement ou de les retrouver, mais peut aussi permettre l'implémentation d'objets s'activant automatiquement pour l'animal, comme une chatière automatique ou un distributeur de nourriture automatique.

3.2 Services existants

Il existe plusieurs autres applications sur le Google Play Store faites pour aider les propriétaires dans leurs différentes tâches

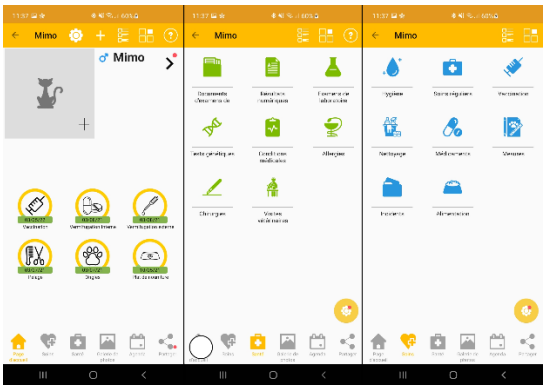

Figure 1: Interface d'11pets

11pets par 11Pets **[11pets]** est une application très complète (Figure 1), voir même complexe, qui permet de définir un nombre impressionnant de rappels de type précis et de stocker de nombreuses informations médicales et permet de gérer plusieurs animaux.

Au premier lancement l'application demande quel type d'utilisateur on est entre « Famille avec animaux de compagnie », « coiffeur professionnel » et « Organisation du bien-être animal », ce qui explique l'interface et la complexité du logiciel : l'application est destinée à

des professionnels qui doivent prendre soin de plusieurs animaux en même temps et peuvent avoir besoin d'un suivi complet pour savoir comment prendre soin d'un animal en particulier. L'application permet aussi d'utiliser un compte ou de récupérer des données partagées, permettant de l'utiliser sur plusieurs appareils ou de la partager entre plusieurs personnes.

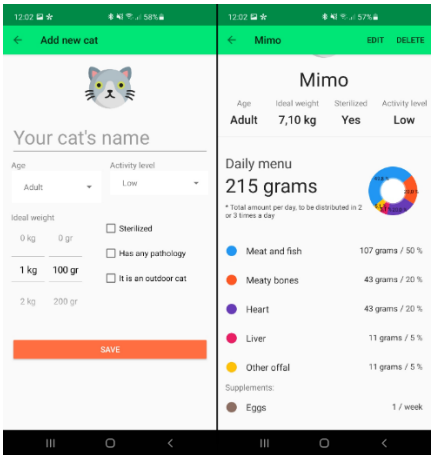

Figure 2: Interface de Barftastic

Barftastic par refactorizado **[Barftastic]** (Figure 2) permet de saisir les informations de son animal tel que la tranche d'âge, le poids idéal, le niveau d'activité, s'il est stérilisé ou non, etc. puis donne grâce à ces informations le type et la quantité de nourriture dont l'animal a besoin chaque jour. Cette application fonctionne pour les chats, les chiens et les furets.

L'application propose aussi une liste des différents types de nourritures pour aider à les identifier et une section qui essaie de vendre des livres sur l'alimentation des chats et des chiens.

Elle permet donc de savoir quelle alimentation donner à son animal mais ne permet pas de suivre tous ces autres besoins.

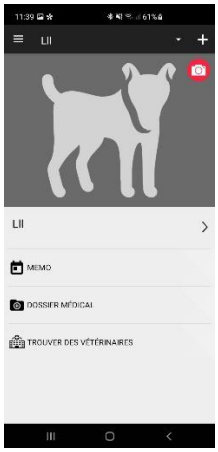

Figure 3: Interface de Dog Health

Dog Health par Luca Biasotto **[Dog Health]** (Figure 3) permet de créer un carnet de santé numérique pour ses chiens en enregistrant leurs vaccinations, leurs visites et interventions

vétérinaires et leurs traitements médicaux et antiparasitaires. Toutes ces informations créent des rappels dans la section « Memo » permettant de rappeler aux propriétaires de mieux s'occuper de la santé de leurs animaux en leur disant quand les animaux doivent prendre leur traitement et quand il faut retourner chez le vétérinaire.

L'application permet aussi de trouver un vétérinaire à proximité en utilisant la position du téléphone, mais cette fonctionnalité ne marchait pas sur mon appareil.

Le logiciel permet de s'occuper de plusieurs animaux et de suivre la croissance de leur taille et de leur masse, mais uniquement dans la version PRO qui est payante.

Pet's Diary par Whisper Arts, Pet Diary par Behrang Javaherian, et Pet Diary par Pet Diary sont des applications permettant de créer des rappels pour ses animaux et ne proposent pas beaucoup plus de fonctionnalités (sauf Pet's Diary par Whisper Arts qui propose des rappels par défaut, mais a une interface très confuse avec plusieurs bandeaux de textes, des calendriers et des listes de « tâches » répétant presque les mêmes informations)

4 Analyses

4.1 Modèle des use-cases

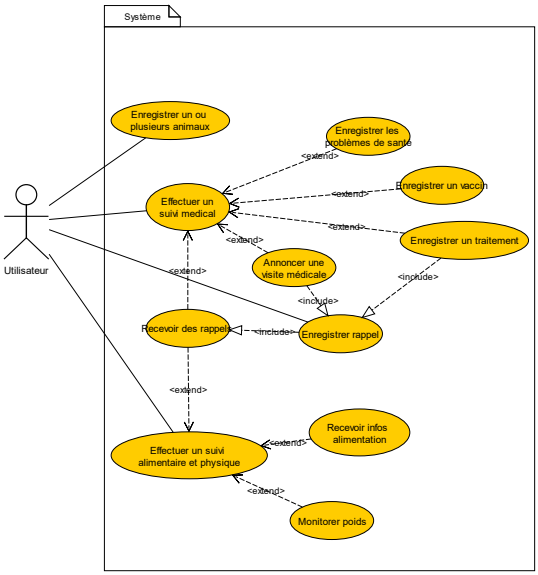

Figure 4: Use-cases

Je souhaite que le système puisse permettre à l'utilisateur d'enregistrer plusieurs animaux différents.

Le système doit offrir un suivi médical des animaux, pour se faire il doit pouvoir enregistrer les problèmes de santé, les vaccins et les traitements des animaux mais aussi enregistrer une visite médicale qui générera un rappel.

En plus du suivi médical, je souhaite que le système effectue un suivi alimentaire et physique de l'animal notamment en proposant un suivi de l'évolution du poids de l'animal et en proposant des informations sur la quantité de nourriture nécessaire à l'animal.

Le système devrait pouvoir rappeler à l'utilisateur les différentes tâches qu'il a à réaliser pour chacun de ces animaux en temps réel.

4.2 Modèles des objectifs

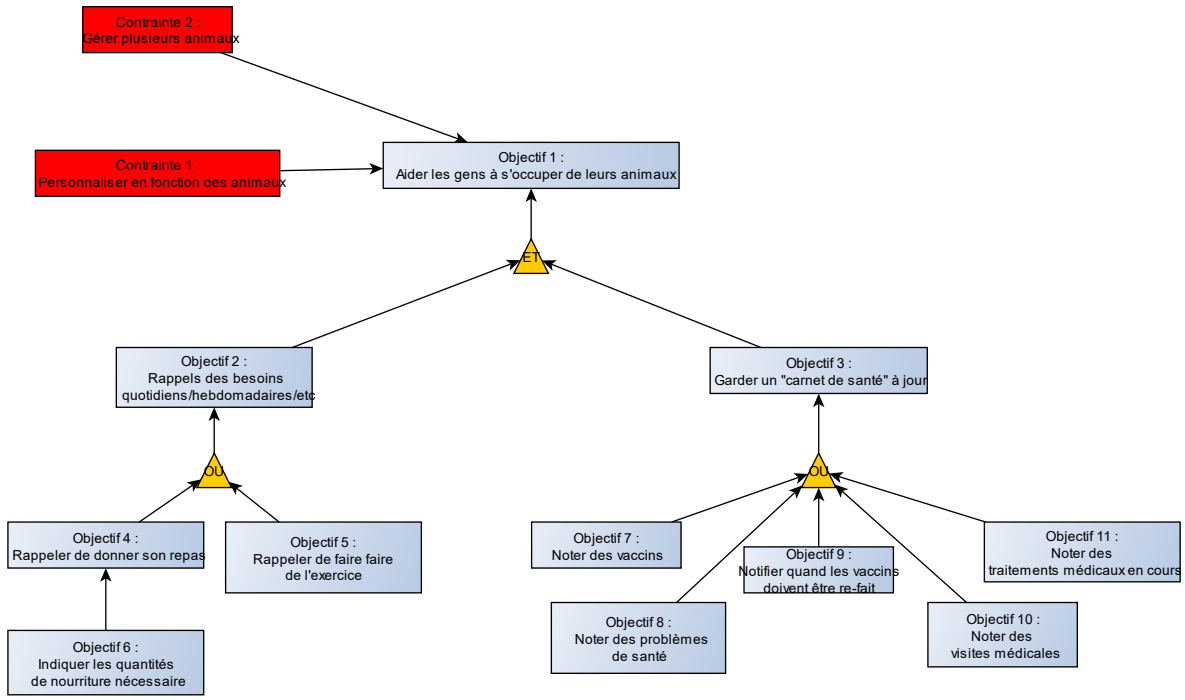

Figure 5: Modèle des objectifs

L'objectif principal de mon service est d'aider les propriétaires d'animaux à prendre soin de leur compagnon, les contraintes sont que mon service doit pouvoir gérer plusieurs animaux et que l'application puissent être personnalisées selon le « type » d'animal, en effet il n'est pas rare de posséder plusieurs animaux et/ou d'avoir plusieurs « types » d'animaux différents simultanément comme des chats ou des chiens ou des oiseaux.

Pour assurer cet objectif principal, le service doit pouvoir rappeler à l'utilisateur ce qu'il doit faire (que ce soit de donner son repas à l'animal ou autre choses) et doit permettre de garder un carnet de santé permettant d'avoir un suivi médical et de noter les différents traitements et vaccins à venir de l'animal.

4.3 Modèle des exigences

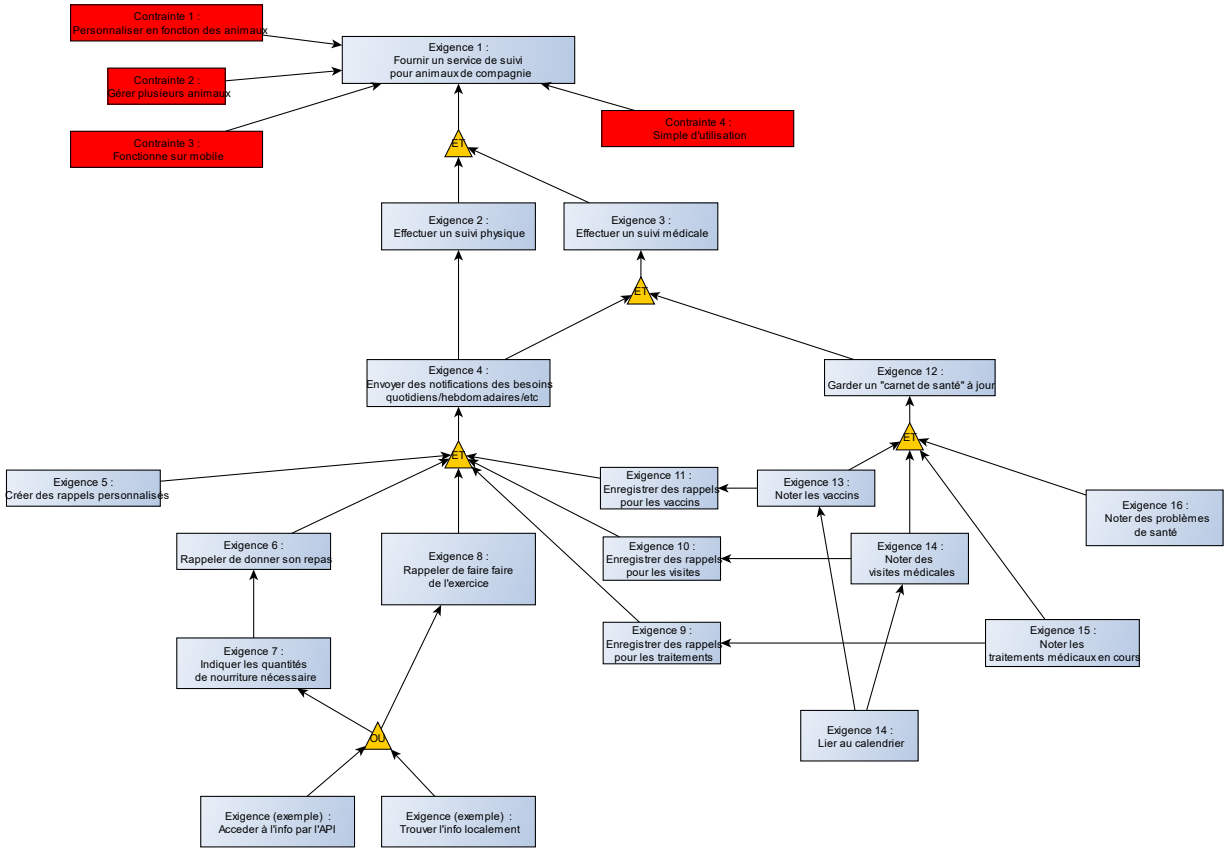

Figure 6 : Modèle des exigences

Dans le modèle des exigences (Figure 6), on ajoute les contraintes que le service fonctionne sur un appareil mobile et qu'elle soit simple d'utilisation.

La contrainte que l'application fonctionne sur mobile est motivée par le fait que presque tout le monde a son smartphone à portée de main pendant presque toute la journée et l'emmène partout, cela permet donc de s'assurer que l'utilisateur voit les notifications au bon moment et que l'utilisateur ajoute plus facilement de nouveaux rappels ou notes les visites médicales/vaccins/traitements.

La contrainte qu'elle soit simple d'utilisation est pour augmenter les chances que l'utilisateur utilise vraiment l'application : la complexité des applications telle que 11Pets est nécessaire pour leur public cible, mais pour des utilisateurs qui veulent seulement un peu d'aide, une interface ou un user-flow complexe peut être dissuasif et ils peuvent ne plus utiliser l'application du tout

On souhaite fournir un service de suivi des animaux de compagnie, qui permette d'effectuer un suivi médical et physique (plus précisément de connaître l'évolution du poids de l'animal, un peu comme ce que propose Dog Health), pour faire ça, il faut pouvoir envoyer des notifications à l'utilisateur et de garder un carnet de santé à jour.

Les notifications peuvent être de à propos de l'alimentation ou des autres besoins de l'animal mais aussi être des rappels personnalisés. Avoir des rappels personnalisés permet

de couvrir des besoins spécifiques à chaque individu ou de couvrir des besoins que l'on aurait pu oublier pendant la programmation.

Il serait préférable de pouvoir fournir des informations précises sur les besoins alimentaires,etc. des animaux plutôt que de simplement donner un rappel générique (par exemple : « Donner à manger à X »).

5 Design d'interface

5.1 Mockups

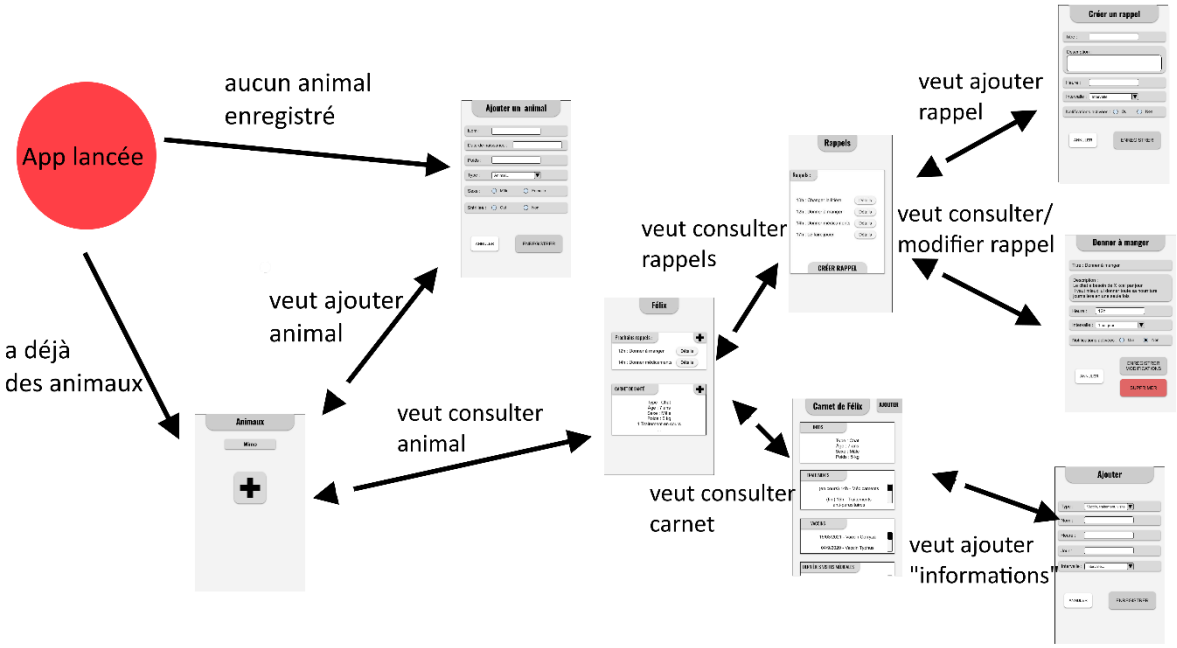

Figure 7: Première version du userflow et mockup de mon application

Dans mon premier mockup (Figure 7), j'avais réalisé une interface très grise car je voulais faire une application très sobre, mais elle avait aussi une apparence un peu désuète.

Pour redesigner mon interface et effectuer des tests du userflow, j'ai utilisé Figma.

Figma permet de designer des mockups mais aussi de pouvoir créer des boutons pour passer d'une page à l'autre sur le mockup.

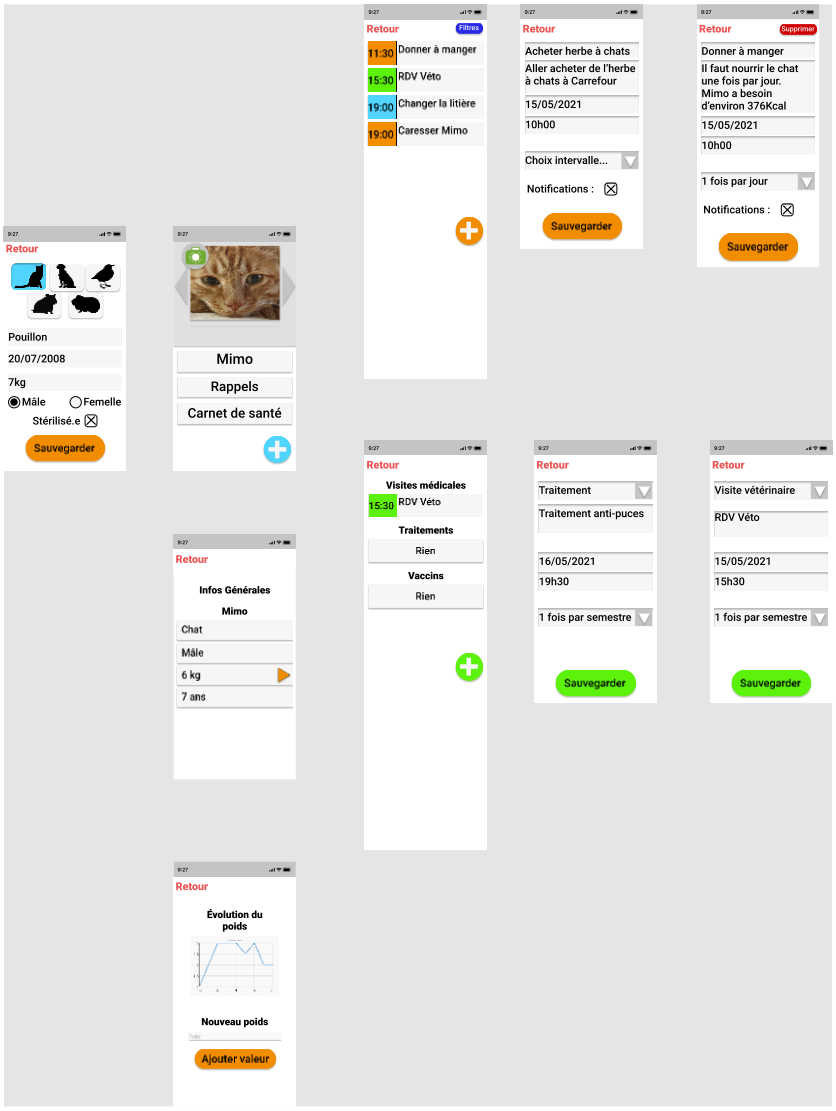

Figure 8: Deuxième mockup

Ma deuxième interface (Figure 8) a changé d'apparence au fur et à mesure des tests AB avec des utilisateurs.
Au final, ce mockup est principalement blanc et gris clair avec des touches de couleurs pour indiquer les éléments importants.

La structure des mockups est globalement la même.
L'une des différences majeures entre mes 2 mockups est l'apparence de la page principale.
Dans mon premier mockup, je pensais faire une liste avec un bouton pour ajouter de nouveaux animaux, puis il faudrait cliquer sur leur nom pour avoir accès à leurs informations.
Dans mon deuxième mockup, il suffit de cliquer sur les flèches présentes sur les côtés de l'image de l'animal pour sélectionner le suivant ou le précédent, ce qui permet d'avoir une page en moins à naviguer pour l'utilisateur *(Figure 9)*.

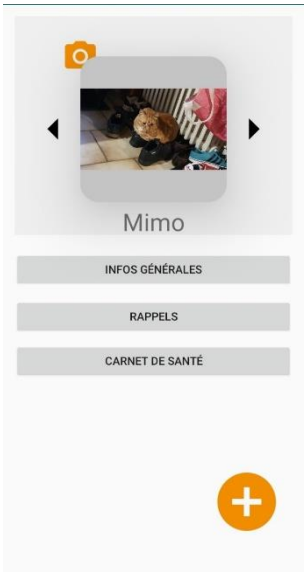

Figure 9: Screenshot de l'écran principal

5.2 Tests d'interface

Les résultats des test d'interfaces indiquaient que les utilisateurs préfèraient plus de touches de couleurs et l'application est devenue blanche et grise avec du vert ou du orange selon la section de l'application (la section de création des rappels personnalisés est orange alors que la section « Carnet de santé » est verte, cela a pour but d'aider l'utilisateur à se repérer plus facilement dans ces menus à l'apparence assez similaire).

J'ai designé l'application pour que toutes les pages soient atteignables en maximum 2 clics à partir de la première page

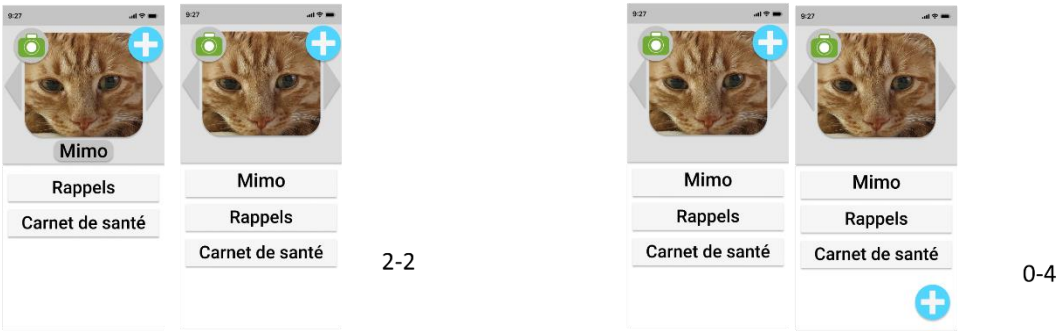

Figure 10: Tests AB de la page principale

Ces tests AB *(Figure 10)* avaient pour but de décider de l'apparence de la page principale, tout d'abord où fallait-il mettre le bouton contenant le nom de l'animal pour que les utilisateurs comprennent que ce soit un bouton. Le résultat étant une égalité j'ai choisi de mettre le bouton à côté des autres car je trouvais ce positionnement plus clair et esthétiquement plaisant.
Le deuxième test visait à déterminer où placer le bouton d'ajout d'animal, tous les

participants l'ont préféré positionné en bas, certains des retours disaient que la position superposé à l'animal faisait penser que le bouton était lié à l'animal sélectionné, aussi le bouton était plus dur à atteindre en haut de l'écran qu'en bas.

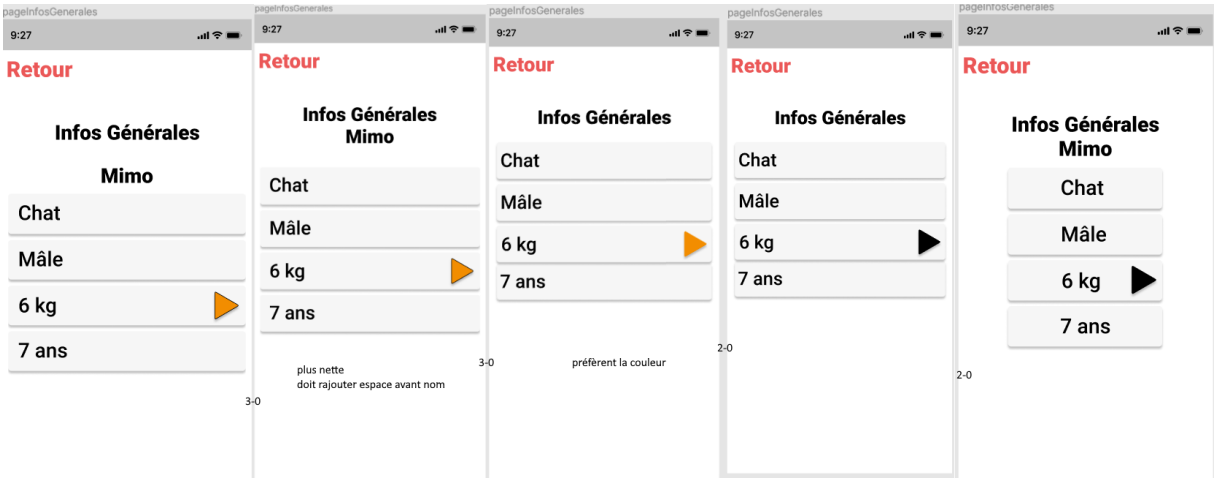

Figure 11: Tests AB de la page Infos. Générales

Pour l'apparence de la page contenant les informations sur l'animal sélectionné, j'ai effectué une série de test *(Figure 11)*. Les utilisateurs préféraient que les zones de textes couvrent toutes la largeur de la page et que le bouton soit coloré et surligné pour augmenter le contraste.

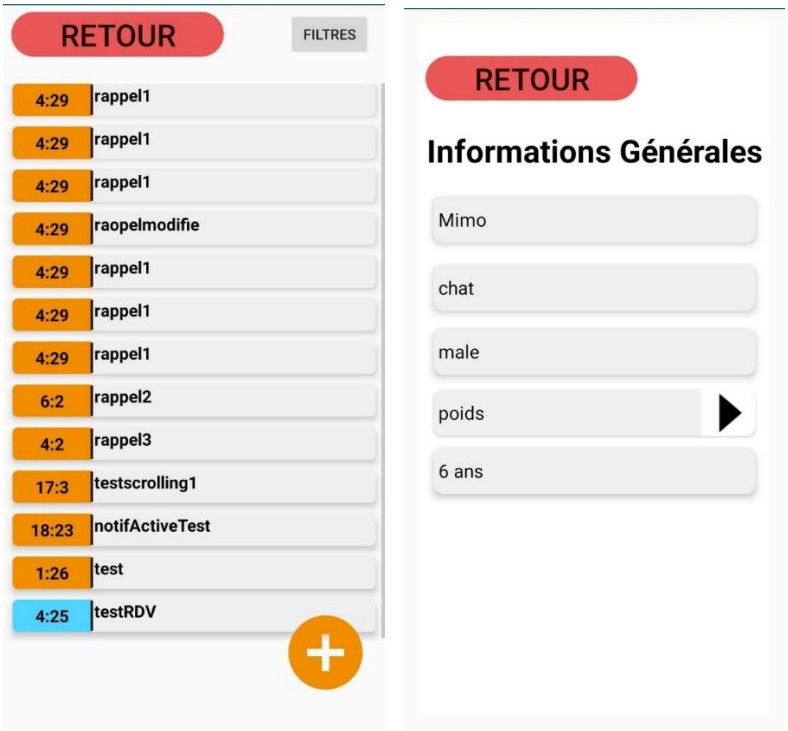

Figure 12: Screenshots des rappels et info. générales

Certains utilisateurs pensait aussi qu'il faudrait séparer le nom de l'animal du titre de la page pour plus de clarté.

Figure 13: Tests AB de la page des rappels

Ce test-ci *(Figure 13)* avait pour but de choisir l'apparence des rappels.
Les utilisateurs préféraient la version colorée et la version blanche et j'ai choisi de garder la version blanche, car je trouvais que la version colorée détonnais avec les autres pages qui ont relativement peu de couleurs.

6 Design du service

6.1 Modèle de la base de données local

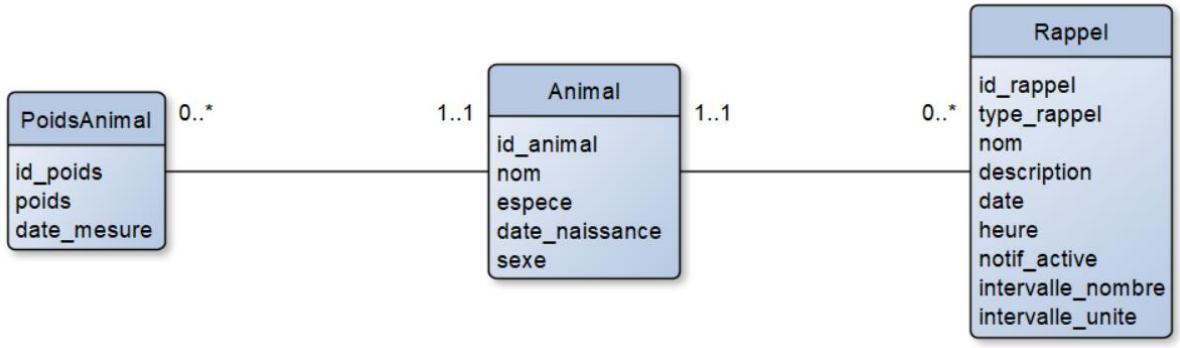

Figure 14: Schéma logique des données

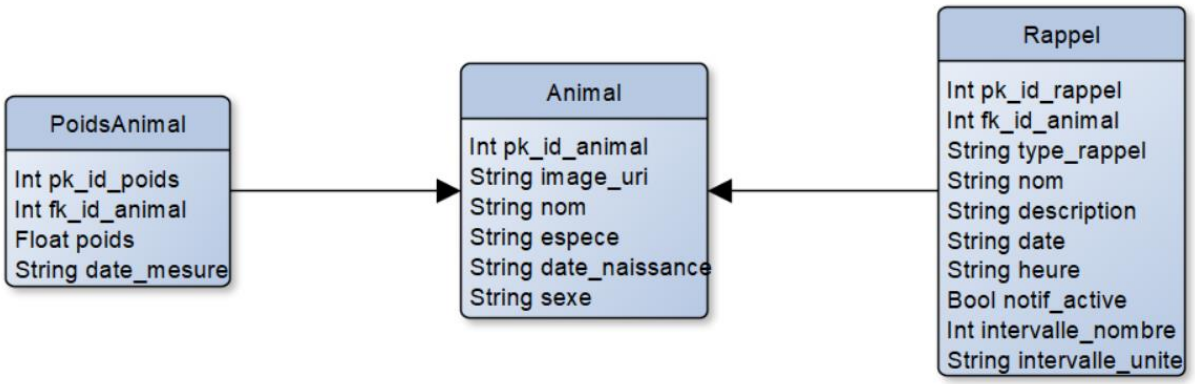

Figure 15: Schéma relationnel des données

Pour stocker les données de chaque animal et chaque rappel, j'ai choisi d'utiliser la bibliothèque Room qui permet d'utiliser une base de données SQLite dans l'application, car cela permettait de gérer les données très facilement et car je connaissais déjà le SQL.

La base de données doit comporter une table pour les animaux, une table pour enregistrer chaque mesure du poids de chaque animal, et une table pour enregistrer chaque rappel de chaque animal.

La table Poids doit donc posséder un id pour identifier la mesure, l'id de l'animal auquel elle correspond, la valeur du poids elle-même et la date de la mesure qui serait utile pour visualiser l'évolution du poids de l'animal.

La table Rappel doit posséder un id pour identifier le rappel, l'id de l'animal auquel il correspond, le type de rappel dont il s'agit, le nom du rappel, sa description, sa première date et heure d'exécution, ainsi que l'intervalle (la valeur et « l'unité » par exemple heure ou jours), et finalement un Boolean permettant de désactiver les notifications si l'utilisateur le souhaite.

La table Animal possède un id pour identifier l'animal, l'URI de l'image représentant l'animal, son nom, sa date de naissance, son sexe, et son espèce.

À cause de certaines limitations de la bibliothèque Room, les dates, les heures et les URI (les identifiant des images) doivent être transformées en un autre type de données pour pouvoir être stockées.
Room permet seulement de stocker variables de type basique tel que les nombres entiers (int), les nombres décimaux (float), les chaines de caractères (String), etc.
J'ai donc décider de transformer ces variables de type incompatible en String puisqu'il s'agit du seul type pouvant stocker cette quantité d'informations, de plus Java dispose de convertisseur pour la plupart de ces types permettant de les transformer en String ou des les retransformer en leur type d'origine à partir d'une String.

6.2 Algorithmes

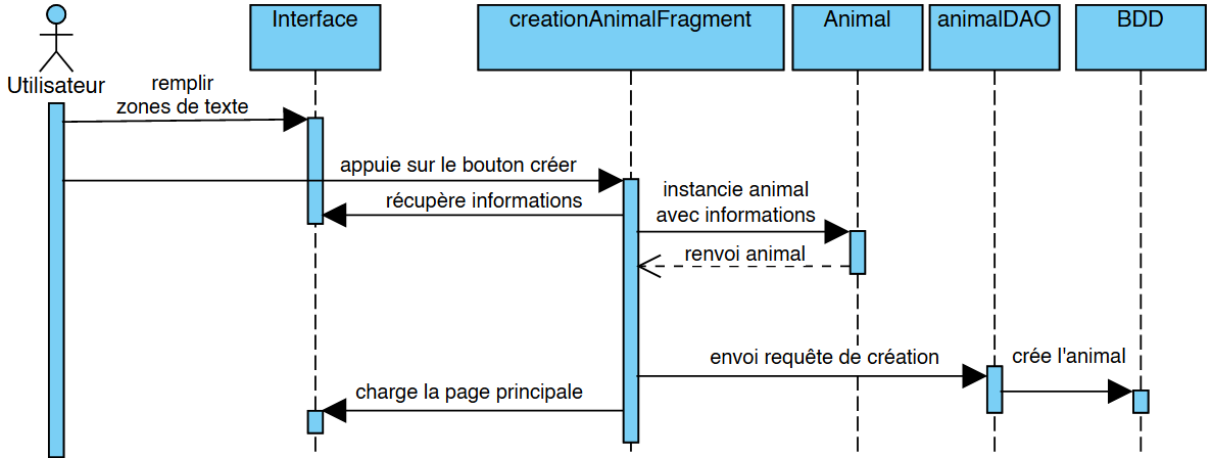

Figure 16: Diagramme de séquences de la création d'animal

Pour la création d'un nouvel animal, l'utilisateur rempli tous les champs.
Quand il a fini, il appuie sur le bouton sauvegardé et la classe Java AnimalCreationFragment récupère les informations contenues dans les zones de texte et l'état des boutons, ensuite la classe instancie un objet Animal, en utilisant les informations récupérées, qui sera transmis au animalDAO dans une requête INSERT, qui s'occupera d'ajouter le nouvel animal dans la base de données *(Figure 16)*.

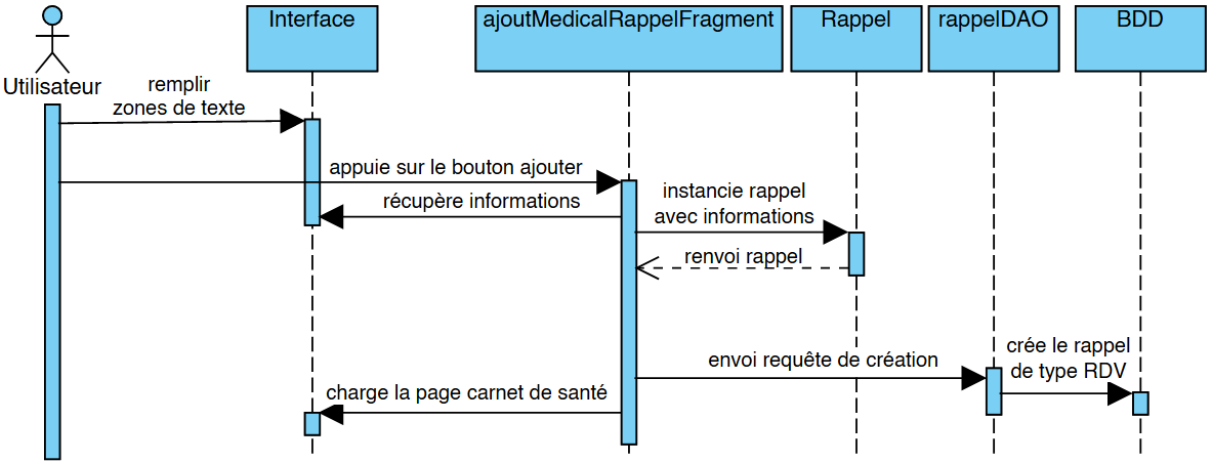

Figure 17: Diagramme de séquence de l'ajout d'un rendez-vous

Pour l'ajout d'un nouveau rendez-vous, l'utilisateur doit remplir tous les champs, puis appuyer sur le bouton ajouter. Après ça le code Java va chercher les valeurs des éléments graphiques puis s'en servir pour créer un objet Rappel qui sera passé à RappelDAO en

paramètres de la fonction insertAll() qui effectue la query et ajoute le rappel à la base de donnée, puis on retourne sur le carnet de santé *(Figure 17)*.

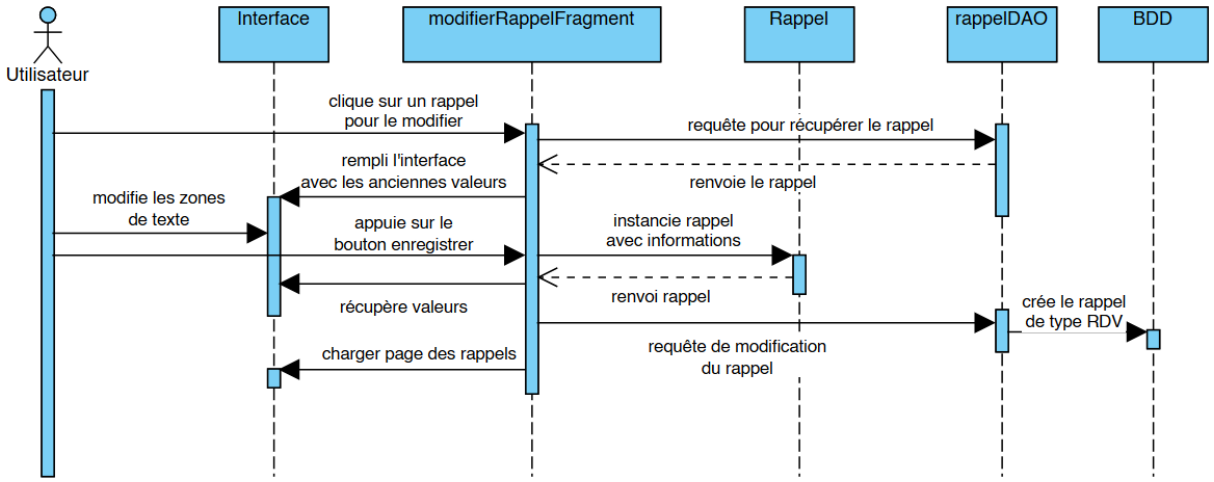

Figure 18: Diagramme de séquence de la modification d'un rappel

Pour la modification d'un rappel, le code Java demande au rappelDAO de lui fournir les informations du rappel grâce à son identifiant obtenu sur la page précédente. Après avoir reçu les informations le code modifie les valeurs des objets de l'interface.
L'utilisateur peut alors les modifier comme il le désire.
Après avoir fini de les modifier, l'utilisateur clique sur Enregistrer, le code de ModifierRappelFragment récupère alors les valeurs contenus dans l'interface puis crée un objet Rappel avec l'ID du rappel et les informations modifiées, puis utilise la méthode updateAll() pour mettre à jour le rappel.
Pour finir, on recharge la page des rappels *(Figure 18)*.

Figure 19: Screenshot de la page de modification d'un rappel

J'ai décidé d'organiser les rappels en plusieurs « types » que sont défaut, custom, vaccin, traitement et rdv.
L'intérêt de ses types est de pouvoir différencier les rappels entre eux : en un clin d'œil l'utilisateur peut différencier des rappels médicaux de ceux proposés par défaut et ces propres rappels personnalisés.
Pour aider à les différencier, je leur attribut une couleur dans la page affichant tous les rappels en fonction de leur type.
Ces types sont aussi utiles pour l'affichage des différentes sections dans le carnet de santé pour séparer les vaccins des traitements ou des rendez-vous vétérinaires.

7 Implémentation

7.1 Technologies utilisées

Étant familier avec Android Studio et Java et ayant un téléphone Android, j'ai décidé de réalisé mon application sur Android Studio.

Android studio permet de créer des applications assez facilement car il créé tout les fichiers nécessaire à une application tout seul et est doté d'outils d'aides tels que l'auto-complétion et la suggestion d'actions en cas erreurs dans le code en utilisant Alt+Entrée.
De plus Android Studio est plutôt bien documenté en grande partie grâce à sa grande communauté.

J'ai utilisé plusieurs bibliothèques pour pouvoir faire ce que je souhaitais : Room pour pouvoir créer et utiliser une base de données locale, jjoe64's graphview pour pouvoir créer un graphique et Junit pour pouvoir réaliser des tests sur le code.

Pour pouvoir afficher des listes dynamiques j'ai utilisé un ViewHolder, un ListAdapter, un RecyclerView et un layout.
Le layout contient l'apparence de l'élément
Le ViewHolder contient le code de l'élément et a accès au layout.
Le ListAdapter permet de gérer la création des ViewHolder et leur attribuant un rappel.

Dans le Fragment, dans la méthode onCreateView() (la méthode ne s'effectuant que pendant que la page se charge), on instancie le ListAdapter, on attribut au RecyclerView le layout et le ListAdapter.
Ensuite il suffit d'utiliser submitList() avec une List<> contenant les rappels pour que des ViewHolders soient créés, contenant chacuns les informations d'un rappel.

7.2 Implémentation de la base de données

Pour implémenter une base de données locale, j'ai utilisé la bibliothèque Room qui utilise SQLite.

Pour utiliser Room, il faut ajouter quelques lignes dans le build.gradle qui spécifient quelle version de Room on souhaite utiliser, ensuite il faut créer plusieurs fichiers :

- Une classe *(Figure 20)* précisant le contenu de la table et les contraintes sur chaque attributs, que se soit les clés primaires et étrangères, si la valeur doit être auto-généré à la création. Ce fichier contient aussi des constructeurs pour créer de nouveaux objets et des accesseurs (getters et setters). Ayant 3 tables dans mon modèle, mon projet contient 3 de ces fichiers.

```
@Entity(tableName = "Poids", foreignKeys = @ForeignKey(entity = Animal.class,
        parentColumns = "id_animal",
        childColumns = "id_animal",
        onDelete = CASCADE,
        onUpdate = CASCADE))
public class Poids {

    //id pour reconnaitre le poids
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name="id_poids")
    public int id_poids;
```

Figure 20: Exemple de classe

- Une interface *(Figure 21)* définissant les requêtes utilisables en Java sur les tables, cette interface s'appelle un DAO (Data Access Object). Ainsi on peut simplement utiliser une méthode dans le code en Java qui renverra le résultat de la requête définie dans le DAO. Un DAO peut être utilisé pour plusieurs tables différentes mais j'ai choisi d'en créer un par table pour que le code soit un peu plus clair.

```
@Dao
public interface AnimalDAO {
    @Query("SELECT * FROM Animal")
    List<Animal> getAll();

    @Query("SELECT * FROM Animal WHERE animal_name LIKE :first LIMIT 1")
    Animal findByAnimalName(String first);

    @Query("SELECT * FROM Animal WHERE id_animal = :id LIMIT 1")
    Animal findByAnimalId(int id);

    @Query("SELECT * FROM Animal ORDER BY id_animal DESC LIMIT 1")
    Animal findLatestAnimal();

    @Query("SELECT id_animal, date_naissance FROM Animal WHERE id_animal LIKE :id LIMIT 1")
    Animal getAnimalBirthday(int id);
```

Figure 21: Exemple de DAO

- Et une classe abstraite *(Figure 22)* précisant quelles tables sont utilisées dans la base de données. Cette classe peut être utilisée pour créer plusieurs bases de données de noms différents mais qui auront les mêmes tables.

```
@Database(entities = {Animal.class, Poids.class, Rappel.class}, version = 1)
public abstract class AppDatabase extends RoomDatabase {
    public abstract AnimalDAO animalDAO();
    public abstract PoidsDAO poidsDAO();
    public abstract RappelDAO rappelDAO();
}
```

Figure 22: Exemple de classe abstraite

Voici un exemple de l'utilisation de ces classes dans le code :

Figure 23: Exemple de l'utilisation de Room

Il faut instancier une base de données en précisant la classe et le nom de la base de données, si une base de données de ce nom n'existe pas, elle est créée automatiquement sinon la base de données existante sera utilisée.
On instancie ensuite le DAO ce qui nous permet d'accéder au contenu de la base de données.
On crée un rappel, dont on utilise l'accesseur pour remplir des données souhaitées.
Pour finir on utilise la méthode insertAll du DAO qui permet de créer un ou plusieurs nouveaux objets.
La méthode insertAll (ainsi que updateAll et delete) est un peu spéciale car il n'y a pas besoin de définir sa requête SQL, elle est gérée automatiquement par Room *(Figure 23)*.

7.3 Front end

L'interface des applications Android Studio est programmée en XML un langage à base de balise.
Dans mon code, chaque Fragment correspond à une page et possède donc son propre fichier XML.
Mon application présente 10 pages différentes dont certaines partagent du code similaire (les pages de création et de modification de rappel).

Android propose plusieurs types de layout, ceux que j'ai utilisé dans mon projets sont les LinearLayout, qui permettent de mettre des éléments les uns à la suite des autres horizontalement ou verticalement en précisant la place qu'ils prennent ou l'écartement entre les éléments, et les ConstraintLayout, pour lesquelles il faut préciser des parents à chaque éléments (et ce pour les 4 coins appelés top, bottom, left, right) ce qui permet de les positionner par rapport à ces parents.
Le LinearLayout est beaucoup plus simple à utiliser, mais le ConstraintLayout permet de placer des éléments à n'importe quel endroit.

Les balises que j'ai utilisés pour permettre à l'utilisateur de saisir toutes les informations qu'ils souhaitent sont les Spinner (qui permettent de créer des menus déroulants, idéal pour faire choisir parmi une liste), les RadioButtons (qui permettent de créer des boutons liés qui se désactivent si un autre est pressé mais est assez dur à implémenter car il faut ajouter du code dans la classe Java), les EditText (qui sont des zones de textes très versatiles : on peut décider de modifier le clavier à l'aide d'une ligne dans le code Java, on peut remplacer le clavier par un calendrier pour faire choisir une date ou pour faire choisir une heure).

7.4 Back end

Pour implémenter le processus décrit dans le diagramme de séquence de la modification d'un rappel *(Figure 18)* :
Quand l'utilisateur clique sur le rappel (sur la page précédente), on stocke l'id dudit rappel dans un ViewModel, il s'agit d'une classe Java dont il n'existe qu'une seule instance dans l'application et que l'on peut uniquement grâce à ces accesseurs, elle permet de transférer des valeurs d'un Fragment à l'autre.
La page de modification est chargée et on récupère l'id du rappel dans le ViewModel.
On utilise le rappelDAO pour récupérer toutes les informations du rappel et les mettre dans les EditText, cocher ou décocher la case pour l'activation des notifications et sélectionner la bonne unité dans le Spinner.
À ce moment-là, l'utilisateur peut librement modifier les informations et quand il clique sur le bouton « Sauvegarder », le Fragment récupère toutes les informations et crée un nouvel objet Rappel en précisant l'id (de base l'id n'est pas créé quand on crée un rappel car celui-ci est généré quand la requête d'ajout est traité par le DAO et la base de données).

Après cette requête, on peut utiliser getFragmentManager().popBackStack() pour retourner sur le Fragment précédent.
L'une des spécificités des Fragments et que l'ont peut les mettre dans une pile quand on charge le suivant. Ce qui permet de très facilement pouvoir retourner en arrière en prenant le Fragment au sommet de la pile.

Le fait que l'id du rappel ne soit pas généré quand on crée l'objet Rappel mais quand la requête est éxécutée peut poser problème : pour l'ajout de l'animal, on demande à l'utilisateur de saisir immédiatement une première valeur de poids.
Pour pouvoir créer le Poids, il faut donc effectuer la requête de création d'animal puis effectuer une requête pousr trouver l'id de l'animal avec l'id le plus élevé (les id commençant à 0 et étant incrémenté à chaque ajout de valeurs, on est sûr que l'animal avec l'id le plus élevé est le plus récent),
Après ça on peut enfin ajouter la valeur de Poids avec l'id de ce nouvel animal.

7.5 Composants et déploiement

L'application ne communique pas avec internet, toutes les données sont stockées en interne dans une base de données SQLite, les seules ressources qui ne soit pas stockées en interne sont les images représentant les animaux : on stocke l'URI de l'image afin de pouvoir l'appeler au moment de l'affichage, cela permet de ne pas stocker les images dans l'application (ce qui l'alourdirait considérément selon la définition de l'image).

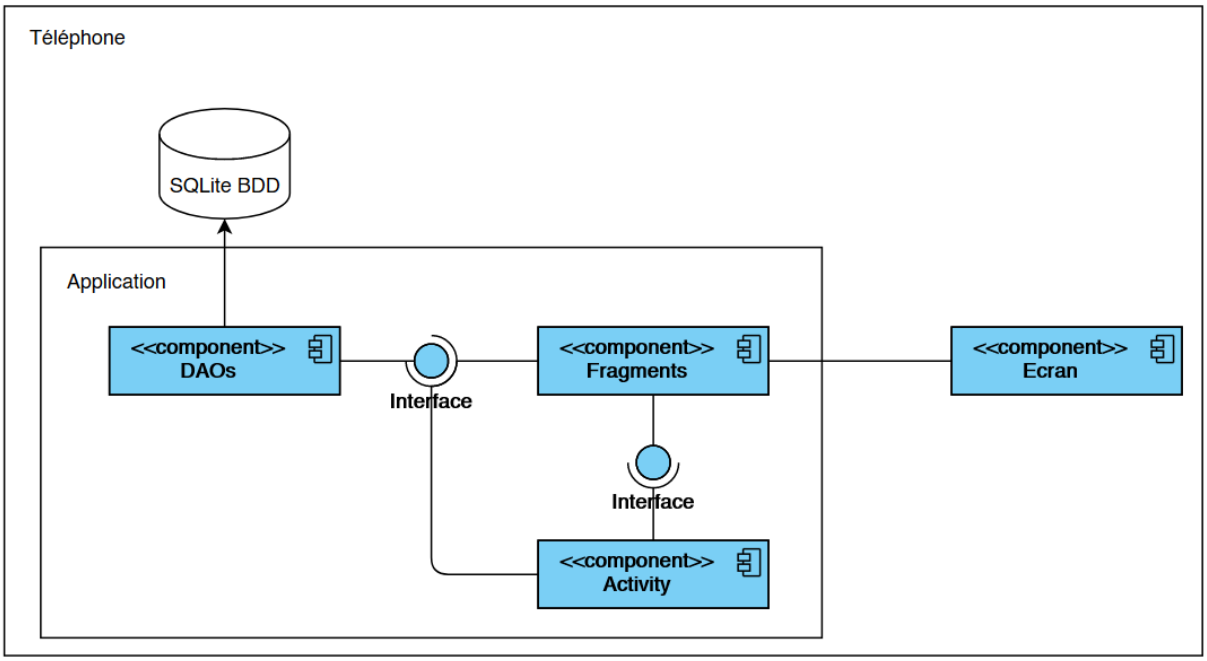

Figure 24: Diagramme des composants

La plupart de l'application fonctionne comme suit :

Au lancement, une Activity se charge qui utilise un DAO pour vérifier si la base de données contient un animal. S'il n'y en a pas l'Activity charge le Fragment qui permet la création d'un animal, sinon elle charge le Fragment qui donne accès au reste de l'application.
Le reste de l'application se passe dans des fragments qui ont des interfaces et communiquent avec la base de données grâce aux DAOs

7.6 Algorithme

Stocker la date de naissance des animaux

Algorithme Calcul âge

```
Procédure getAge(date_naissance)
        anneeNaissance <- année de naissance
        moisNaissance <- mois de naissance
        jourNaissance <- jour de naissance
        anneeActuelle <- année actuelle
        moisActuel <- mois actuel
        jourActuel <- jour actuel
        ageAn <- anneeActuelle - annéeNaissance
        SI moisActuel < moisNaissance OU (moisActuel == moisNaissance ET jourActuel < jourNaissance)
ALORS
                ageAn <- ageAn - 1
        SI ageAn < 0 ALORS
                retourner « Date invalide »
        SINON SI ageAn == 0 ALORS
                SI moisActuel < moisNaissance OU moisActuel = moisNaissance ET jourActuel <
jourNaissance ALORS
                        ageMois <- 12 + moisActuel – moisNaissance
                SINON
                        ageMois <- moisActuel – moisNaissance
                SI jourActuel < jourNaissance ALORS
                        ageMois <- ageMois -1
                SI moisActuel < moisNaissance ET anneeActuelle < annéeNaissance ALORS
                        retourner « Date invalide »
                SINON SI ageMois == 0 ALORS
                        SI moisActuel == moisNaissance ALORS
                                ageJour = jourActuel – jourNaissance
                        SINON SI moisNaissance == février ALORS
                                ageJour = 28 + (jourActuel – jourNaissance)
                        SINON SI mois == mois de 31 jours ALORS
                                ageJour = 31 + (jourActuel – jourNaissance)
                        SINON
                                ageJour = 30 + (jourActuel – jourNaissance)
                        retourner ageJour +  « jours »
                SINON
                        retourner ageMois + « mois »
        SINON SI ageAn == 1 ALORS
                retourner ageAn + « an »
        SINON
                retourner ageAn + « ans »
```

---

8 Évaluation

8.1 Fonctionnalités

Une grosse partie de l'application fonctionne comme je le désirais : l'application permet de créer plusieurs animaux et de créer pour chacun d'entre eux des rappels et de stocker leur poids.
Aussi l'interface affiche tous les rappels et il est possible de les modifier.
On peut aussi consulter le poids de l'animal sur un graphique.

La partie la plus importante qu'il manque à l'application est à mon avis les notifications, j'avais commencé à les implémenter mais n'ai pas eu le temps de les finir : le AlarmManager (qui doit envoyer des messages à un certain intervalle) et le BroadcastReceiver (qui envoie des notifications quand il reçoit le message de l'AlarmManager) sont implémentés mais le AlarmManager n'envoie pas de messages au BroadcastReceiver.

8.2 Bugs

J'ai corrigé tous les bugs que j'ai trouvé, par exemple retourner en arrière pendant l'ajout d'une image faisait crasher l'application car l'application ne recevait aucune valeur, le graphique de poids apparaissait parfois derrière les autres pages et la rotation du téléphone faisait revenir sur la première page.
Le seul bug qui existe encore et dont j'ai connaissance est que l'affichage des photos peut faire planter l'application sur les versions moins récentes d'Android.

8.3 Tests unitaires

Pour faire des tests dans Android Studio, cet outil permet de tester si les fonctions et méthodes renvoient les bonnes valeurs : il suffit de créer un objet de la classe possédant cette méthode, puis de comparer la valeur attendue avec la valeur retourner avec assertsEquals().

Mon application ne dispose pas de beaucoup de méthodes renvoyant des valeurs, il y a la fonction getAge(), les DAOs, et les ViewModels, mais le reste de l'application consistent de méthodes Void (ne renvoyant rien).

Les DAOs et les ViewModels ont besoin de fonctionner pour que l'application fonctionne, les rappels chargés grâce aux ViewModel et les requêtes n'ayant jamais montré produits de résultats inattendus, je pense que ces classes fonctionnent correctement.

Sur getAge(), j'ai réalisé des tests sur des cas me semblant probable et pouvant créer des problèmes en testant une date future, le jour même, 1 jour après la naissance, 2 semaines après la naissance (avec un changement de mois entre les 2), 11 mois après la naissance puis 2 ans.

Ces tests dépendent de la date et ont été réalisés le 2 septembre 2021 (03/09/2021).

Dans 1 an (02/09/2022) : le résultat est bien celui escompté, « Date de naissance invalide ».

Pour le jour même (02/09/2021) : le résultat est le bon, « 0 jours ».

Pour 1 jour après la naissance (01/09/2021) : le résultat est le bon, « 1 jours ».

Pour 2 semaines après la naissance (19/08/2021) : le résultat est le bon, « 14 jours ».

Pour 2 ans après la naissance (02/09/2019) : le résultat est le bon, « 2 ans ».

Pour 11 mois après la naissance (02/10/2020) : le test a révélé une erreur, l'algorithme ne retournait pas la bonne valeur si c'était l'année suivante mais tout de même moins d'un an. J'ai corrigé l'algorithme dans le code et dans ce rapport.

Après avoir vu cette erreur j'ai décidé d'essayer d'autres âges de quelques mois :

Pour 10 mois après la naissance (02/11/2020) : le résultat est bon, « 10 mois ».

Pour 6 mois après la naissance (02/3/2021) : le résultat est bon, « 6 mois ».

Pour 364 jours après la naissance (03/09/2021) : j'ai trouvé un autre problème où le mois étant le même, la fonction renvoyait -1, j'ai ajouté une condition vérifiant le jour en plus du mois dans le calcul du mois pour corriger ce problème.
Maintenant ce test renvoie bien « 11 mois ».

9 Conclusion

Le projet contient des aspects non finis :

Je considérais l'aspect fonctionnel prioritaire à l'aspect, par conséquent l'apparence de l'application n'est pas la même que celle des mockups, sauf dans les parties où l'apparence était liée à la fonctionnalité.

La bibliothèque que j'utilise pour dessiner le graphe de poids n'affiche que les valeurs du graphe et je n'ai pas trouvé de moyen d'afficher la date sur l'axe des abscisses. Aussi je n'ai pas trouvé de fonctionnalités permettant de nommer les axes rendant la compréhension du graphe plus compliquée.

Pour une prochaine version, il faudrait faire fonctionner le bouton de « filtre » sur la page des rappels, il permettrait de choisir quel type de rappel afficher.

On prévoit de faire des notifications grâce à un AlarmManager et d'un BroadcastReceiver, j'avais commencé à les implémenter mais les notifications ne fonctionnent pas encore.

Dans la version actuelle de l'application, les animaux qui viennent d'être créés n'ont pas d'images, il faudrait ajouter des images par défaut pour les animaux.

J’avais prévu d’implémenter des rappels par défaut pour les animaux qui auraient été créés en même temps que l’animal pour répondre aux besoins génériques de chaques espèces telles que « Changer la litière » pour les chats ou « Promener » pour les chiens.

Dans une deuxième version, il faudrait retravailler le système de rappels, en l’état actuel on peut choisir de ne pas recevoir de notification d’un rappel en particulier, mais il faudrait pouvoir choisir le nombre de fois qu’un rappel doit être répété (en l’état le rappel à besoin d’un intervalle pour être créé de manière valide). Il faudrait aussi ajouter un bouton pour supprimer les rappels qui pourraient s’avérer utile si on ajoute les rappels terminés.

Actuellement je n’ai pas mis en place de tests pour s’assurer que les champs soient tous remplis et valides, il faudrait en mettre en place pour s’assurer qu’un utilisateur ne puisse pas faire crasher l’application.

Aussi il faudrait une section supplémentaire qui contienne des « post-it » qui seraient des informations sur l’animal, elle pourrait permettre de stocker ce que dit le vétérinaire sur la santé de l’animal, pour stocker les problèmes de santé de l’animal ou d’autres informations plus spécifiques comme ces aliments favoris, son activité favorites, etc.
Elles pourraient appartenir à des catégories différentes permettant de les différencier si elles sont nombreuses.

10 Bibliographie

Alexis Anne A Luayon, G. F. (2019). PetCare: a smart pet care IoT mobile application. (A. f. Machinery, Éd.) *IC4E '19: Proceedings of the 10th International Conference on E-Education, E-Business, E-Management and E-Learning*. Récupéré sur https://dl.acm.org/doi/10.1145/3306500.3306570

Bogaerts A, B. M. (2020). *Development and Field Evaluation of the INTER-ACT App, a Pregnancy and Interpregnancy Coaching App to Reduce Maternal Overweight and Obesity: Mixed Methods Design.* JMIR Formative Research. Récupéré sur http://formative.jmir.org/2020/2/e16090/

Cueto V, W. C. (2019). *Impact of a Mobile App–Based Health Coaching and Behavior Change Program on Participant Engagement and Weight Status of Overweight and Obese Children: Retrospective Cohort Study.* JMIR Mhealth and Uhealth. Récupéré sur http://mhealth.jmir.org/2019/11/e14458/

Esron D. Karimuribo, E. K. (2016). Potential use of mobile phones in improving animal health service delivery in underserved rural areas: Experience from Kilosa and Gairo districts in Tanzania. *Spring Nature*. Récupéré sur https://www.researchgate.net/publication/309515243_Potential_use_of_mobile_phones_in_improving_animal_health_service_delivery_in_underserved_rural_areas_Experience_from_Kilosa_and_Gairo_districts_in_Tanzania

Junior, R. L. (2020). IoT applications for monitoring companion animals: A systematic literature review. *2020 14th International Conference on Innovations in Information Technology*. Récupéré sur https://ieeexplore.ieee.org/document/9299045

Monteiro-Guerra F, S. G.-L. (2020). *A Personalized Physical Activity Coaching App for Breast Cancer Survivors: Design Process and Early Prototype Testing.* JMIR Mhealth Uhealth . Récupéré sur https://mhealth.jmir.org/2020/7/e17552

S. O. Ogunlere, S. E. (2018). Animal Care Automated System - Software Development and Implementation. *International Journal of Computer Applications*. Récupéré sur https://www.semanticscholar.org/paper/Animal-Care-Automated-System-Software-Development-Ogunlere-Ebiesuwa/72e3dfdc3880b7a48c5a5a04a05cf25dcfaf675b

So Mi Jemma Cho, J. H.-S. (2020). Effect of Smartphone-Based Lifestyle Coaching App on Community-Dwelling Population With Moderate Metabolic Abnormalities: Randomized Controlled Trial. *Journal of Medical Internet Research*. Retrieved from https://pubmed.ncbi.nlm.nih.gov/33034564/

[11pets] https://www.11pets.com/en

[Barftastic] https://www.naturzoo.com/barfastic

[Dog Health] http://doghealthapp.it/