Rapport final NF18

RAGOT Nils, VITAL Simon, PONTOIRE Julien, BIFFE Simon

Présentation du problème

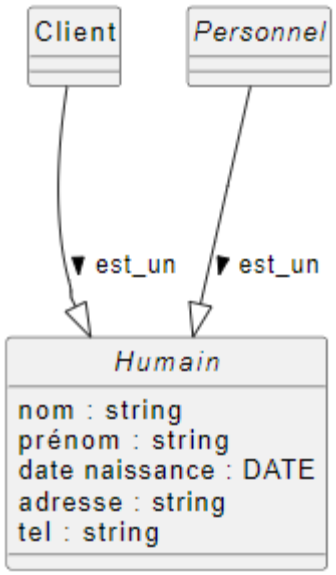
Dans le cadre du projet de ce semestre, nous avons été amenés à développer une base de données pour aider un gérant de clinique vétérinaire (fictif). Notre architecture était destinée dès le départ à être exploitée par une application python à destination de plusieurs profils d'utilisateurs. Nous avons jugé bon d'en distinguer quatre :

- administrateur (le gérant de la clinique)
- vétérinaire
- assistant vétérinaire
- client

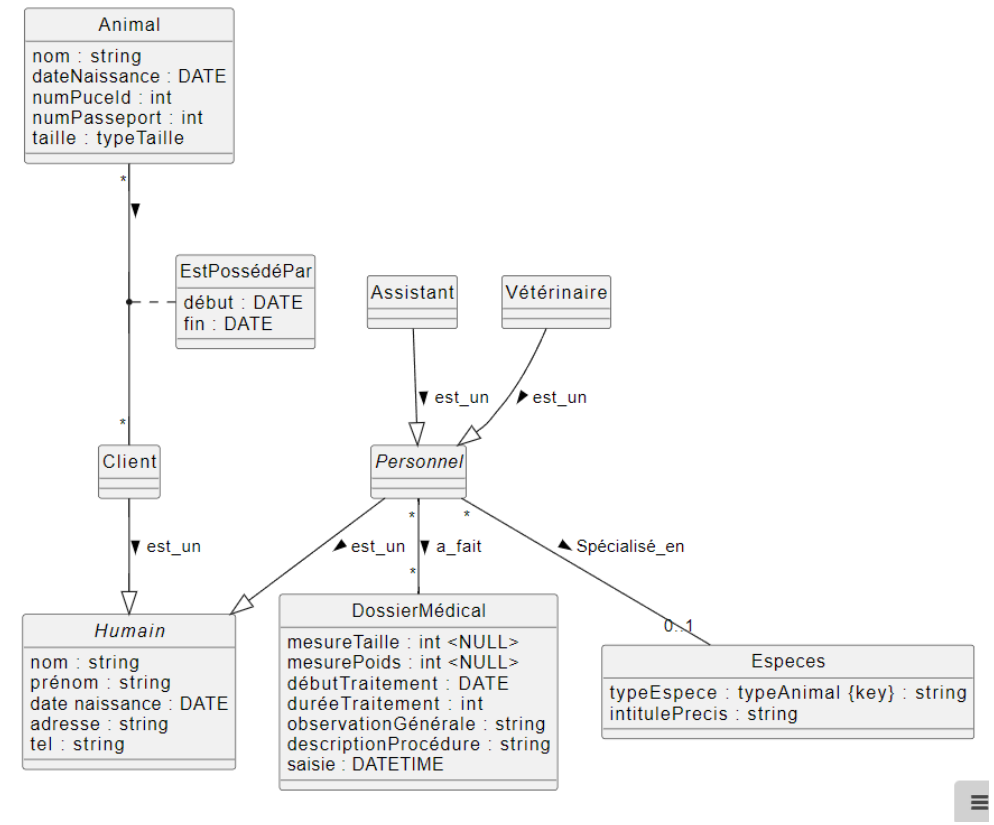
De plus, plusieurs utilisations de notre base de données étaient demandées par l'utilisateur final. On devait lui permettre de gérer le personnel de la clinique, les différents clients et leurs animaux. Le personnel de la clinique devait quant à lui pouvoir créer des dossiers médicaux, ajouter des médicaments / traitements. On devait aussi permettre au gérant de la boutique d'obtenir différentes statistiques sur sa clinique, notamment :

- le nombre total de médicaments consommés (et donc vendus)
- le nombre de traitements en cours
- les rapports d'activités entre vétérinaire

Présentations de l'architecture proposée

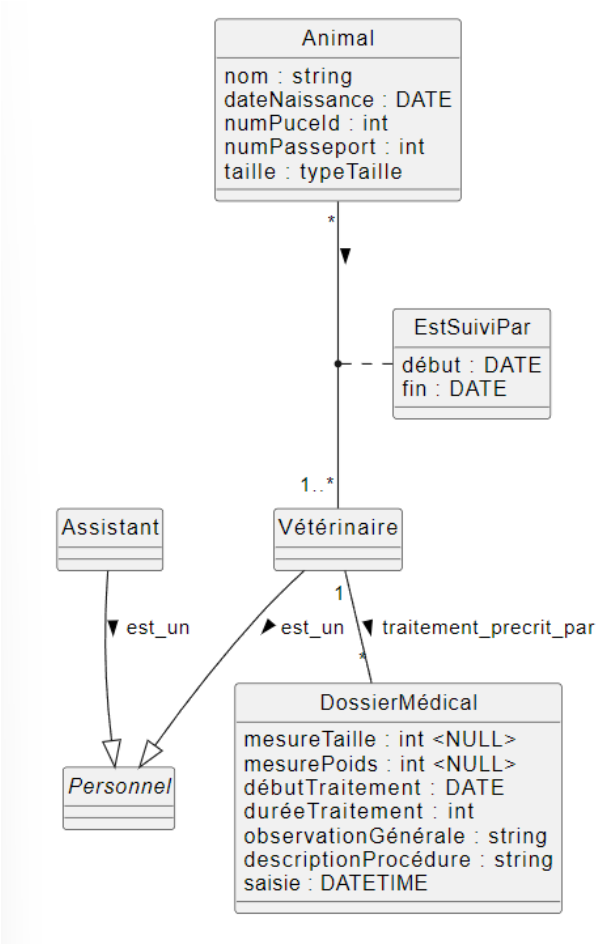


Nous avons créé une classe abstraite *Humain* dont héritent les classes Client et Personnel. Aucun attribut supplémentaire et spécifique aux classes filles n'a été ajouté, la distinction était cependant nécessaire car celles-ci n'entretiennent pas les mêmes relations avec les autres tables.



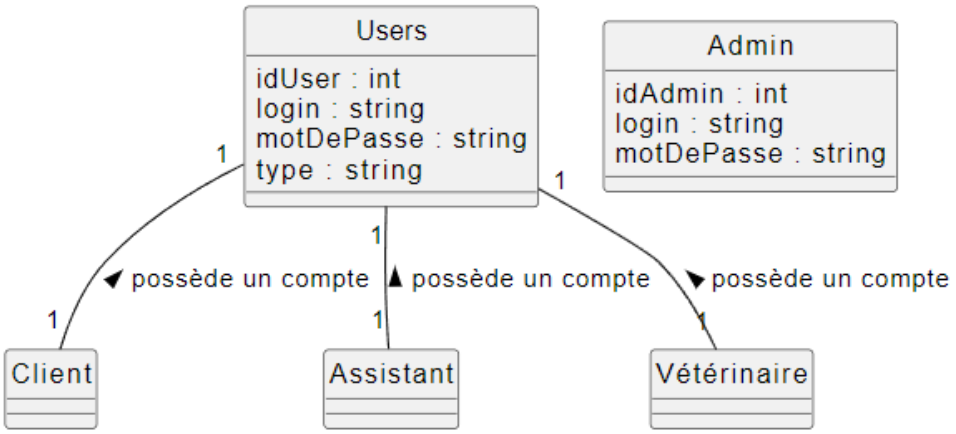
L'unique relation du client et celle qu'il a avec ses animaux : "possède". Le Personnel peut quant à lui faire des dossiers médicaux et il est spécialiste d'une Espece. Assistant et vétérinaire héritant tout deux de Personnel, ils héritent aussi de ses relations. Certaines relations spécifiques à chacun ont été ajoutées, d'où la distinction une nouvelle fois.

Mentionner dans la partie nosql que la vérification des dates se faisait bien dans la partie applicative.



La classe Vétérinaire a plus d'association que la classe Assistant. En effet, un vétérinaire peut créer un ou plusieurs dossiers médicaux, et est suiveur d'un ou plusieurs animaux. On note pour chaque animal dont le vétérinaire est suiveur les dates de début et de fin de suivi

user client véto et assistant + admin.



Pour pouvoir faire des modifications de la base de données depuis l'application python, nous avons créé un système de compte. Nous avons créé deux types de compte : utilisateurs (Users) et administrateurs (Admin). Les classes Users et Admin retiennent les identifiants de connexion des utilisateurs. La classe Users contient en plus un attribut "type" qui permet d'indiquer si l'utilisateur est un Client, un Assistant ou un Vétérinaire.

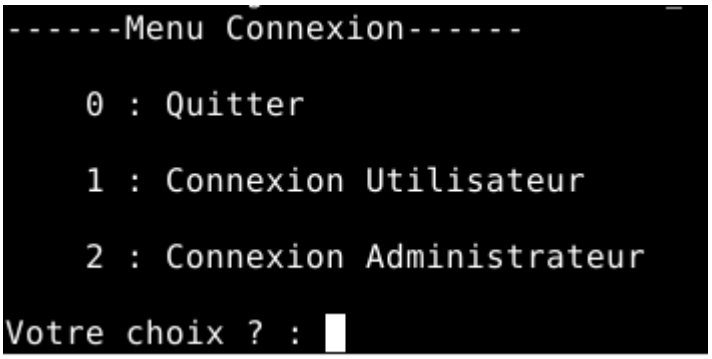
Pour pouvoir associer chaque Client à son compte utilisateur, nous avons mis le même id dans les deux classes. Ce qui veut dire que le client dont l'id est 5 a pour compte utilisateur le User dont l'id est 5. De même pour Assistant et Vétérinaire, nous avons fait correspondre leur clé primaire à celle de leur compte associé dans la table Users.

En fonction du type du compte que l'on est, on a accès à plus ou moins de fonctionnalités :

- Le client peut uniquement consulter les informations sur son animal.
- L'assistant peut en plus ajouter un dossier médical et un médicament.
- Le vétérinaire peut encore en plus ajouter et modifier un animal.
- L'administrateur a les pleins pouvoirs, il peut créer et modifier des clients, assistants et vétérinaires, il peut créer de nouveaux administrateurs. Il peut également consulter un grand nombre de statistiques sur la base de données comme détaillées dans l'introduction.

Notice d'utilisation de l'application

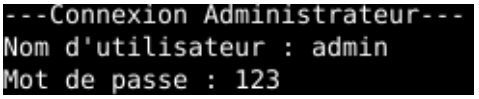
Lorsque l'on lance l'application python. Le menu suivant s'ouvre :



Vous avez donc le choix du mode de connexion. Les clients, vétérinaires et assistants sont ici considérés comme des utilisateurs. Ainsi, l'application sera capable de reconnaître le type d'utilisateur en fonction de ses informations de connexion.

Après avoir choisi votre mode de connexion, vous serez amenés à entrer votre ensemble : identifiant / mot de passe. À la livraison de l'application pour la clinique, seul un compte administrateur sera créé, ses identifiants étant transmis au gérant de la clinique. Ce compte

administrateur permettra d'en créer d'autres et de créer les comptes des différents utilisateurs.



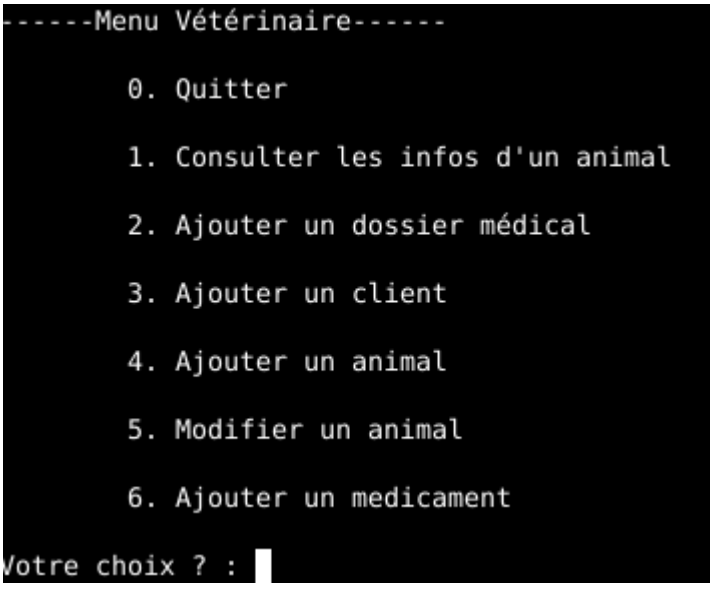
Pour pouvoir tester l'application, nous avons cependant créé les différents types de comptes, dont voici les identifiants de connexion :

- client : user1 / 123456789
- assistant : user7 / 123456789
- vétérinaire : user10 / 123456789
- admin : admin / 123

Une fois connectés, les utilisateurs se voient proposer par un menu les différentes options à leur disposition, avec le même type de choix (par numéro). L'application est assez intuitive par la suite.

Voyons cependant un peu plus en détail deux cas d'utilisation de l'application, par l'administrateur et par un vétérinaire.

Cas d'utilisation : vétérinaire



Ci-dessus se trouve le menu d'actions proposées à un vétérinaire, on peut voir que celles-ci répondent aux besoins précédemment décrits. Le vétérinaire peut effectivement ajouter un animal, consulter et/ou modifier ses informations, ajouter un médicament ou bien créer un dossier médical contenant un traitement.

Exemple d'affichage des informations d'un animal :

À des fins de praticité, nous avons décidé de lui proposer deux types de recherche, dans le cas d'une recherche par nom, si plusieurs animaux correspondent, on les affiche et lui demande de choisir lequel il veut consulter parmi eux.

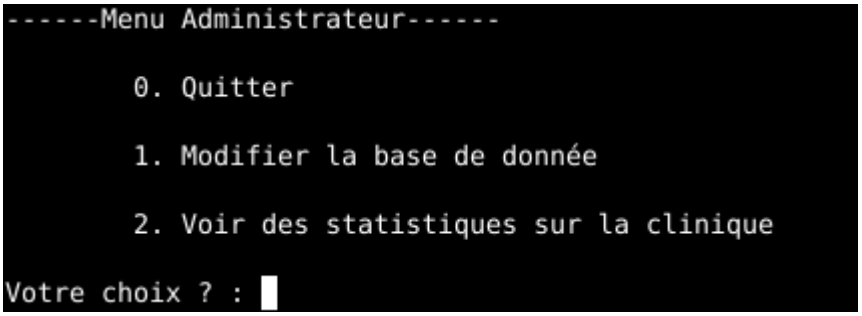
```
Rechercher un animal par :
1) Nom
2) Id
2
Entrez l'id de l'animal : 1
Dossiers médicaux de Nemo:

    --------------------
    Dossier 1 :
        Date de saisie : 2003-12-03
        Nom de l'animal : None

        Mesure de la taille : 10

        Mesure du poids : 5

        Début du traitement: 2003-12-04

        Durée du traitement: 20

        Observation générale:

                Blessure à la patte

        Description Procédure :

                Guérir la patte
```

n définie, on lui propose d'en l'espèce n'existe pas encore, on lui propose de la créer afin qu'il puisse correctement enregistrer son animal.

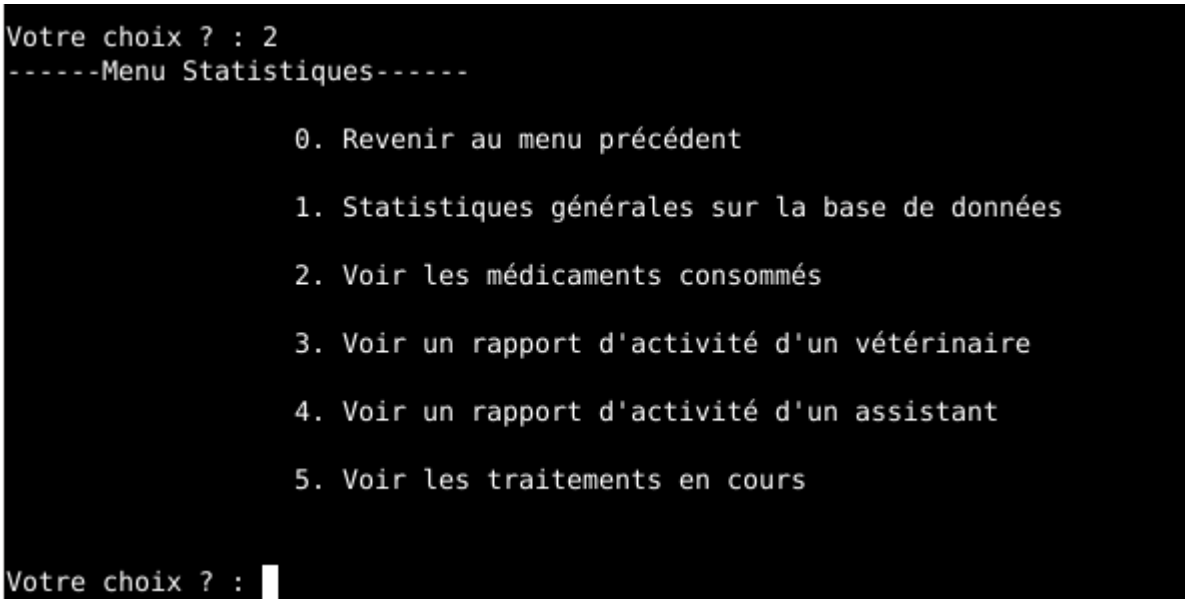
```
------Menu Vétérinaire------

       0. Quitter

       1. Consulter les infos d'un animal

       2. Ajouter un dossier médical

       3. Ajouter un client

       4. Ajouter un animal

       5. Modifier un animal

       6. Ajouter un medicament

Votre choix ? : 4
Entrez le nom de l'animal : Mich
Entrez le numéro de puce de l'animal (opt. entrez 'non' pour ne pas spécifier)non
Entrez le numéro de passport de l'animal (opt. entrez 'non' pour ne pas spécifier)1243
Tailles possibles de l'animal :

         0. petite

         1. moyenne

         2. autre
Votre choix ? (0,1,2)0
```

Cas d’utilisation 2 : Administrateur

Après s’être connecté, l’administrateur a accès à un menu où chaque option représente un sous-menu.

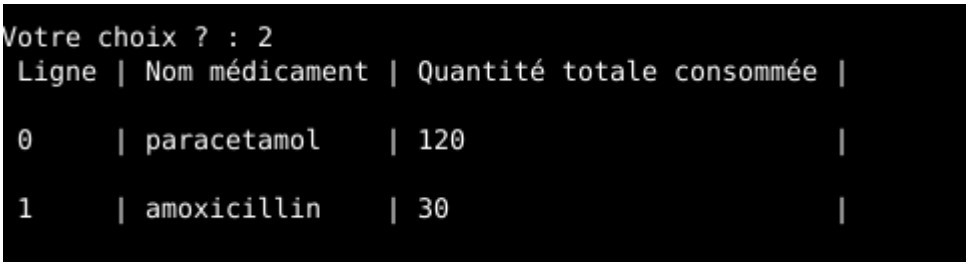


Exemple d’utilisation du sous menu statistiques



Pour revenir au menu précédent, il suffit à l'administrateur de taper 0. Ainsi, il peut effectuer autant d'actions liées aux statistiques de suite qu'il veut sans avoir à ré-entrer dans le sous-menu à chaque fois.

Exemple : voir les médicaments consommés



Pour permettre cet affichage, nous avons utilisé une vue se chargeant du calcul de la quantité totale de médicaments consommés.

L'autre sous-menu "Modifier la base de donnée" est celui qui permettra à l'administrateur de faire toutes sortes d'insertion et de modifications dans celle-ci.

Extension NoSQL

Pour l’extension NoSQL de notre projet, nous avons choisi de modifier deux parties de notre modèle.

Tout d'abord nous avons remplacé la classe resultatsAnalyse par un JSON incorporé directement dans la classe DossierMedical. Nous avons fait ce choix en raison de l’optimisation que cela apporte en lecture puisqu’on souhaite en général lire tous les résultats d’analyse liés à un dossier donné ensemble et de la simplicité que ce changement apporte à notre implémentation. En effet elle permet de supprimer la classe ResultatsAnalyse mais également le classe d’association ContientResultDoss qui permettait de faire le lien entre ResultatsAnalyse et DossierMedical.

Nous avons également choisi de modifier la classe *Humain* pour regrouper toutes les caractéristiques en un champ JSON “infos”.
Puisque nous avons adapté l’héritage par un héritage en classe fille, nous avons dû dans la couche SQL modifier les classes Client, Assistant et Veterinaire.
A noter qu’en raison de cette transformation, le vérification du type des données doit se faire au niveau de la couche applicative, notamment pour le format du champ dateNaissance.

```
postgres=# -- Assistants
postgres=#
xpostgres=# INSERT INTO Assistant (idAssist, infos, specialite) VALUES (7, '{"nom" : "Renaud", "prenom" : "Augustin", "dateNaissance" : "1965-
02-24", "adresse" : "124 Rue de Paris", "tel" : "0685159675"}', 1);
INSERT 0 1
postgres=#
tpostgres=# INSERT INTO Assistant (idAssist, infos, specialite) VALUES (8, '{"nom" : "Eberhardt", "prenom" : "Alexandre", "dateNaissance" : "1
965-02-25", "adresse" : "12 rue des fleurs", "tel" : "0684559671"}', 4);
INSERT 0 1
postgres=#
Npostgres=# INSERT INTO Assistant (idAssist, infos, specialite) VALUES (9, '{"nom" : "Fouinat", "prenom" : "Quentin", "dateNaissance" : "1965-
02-26", "adresse" : "34 avenue du Port", "tel" : "0645627891"}', 2);
INSERT 0 1
```

Exemple de trois instructions INSERT qui ajoutent des assistants

```
 idassist |                                                          infos
          | specialite
----------+-----------------------------------------------------------------------------------------------------------------------------------
----------+------------
        7 | {"nom" : "Renaud", "prenom" : "Augustin", "dateNaissance" : "1965-02-24", "adresse" : "124 Rue de Paris", "tel" : "06851596
75"}      |          1
        8 | {"nom" : "Eberhardt", "prenom" : "Alexandre", "dateNaissance" : "1965-02-25", "adresse" : "12 rue des fleurs", "tel" : "068
4559671"} |          4
        9 | {"nom" : "Fouinat", "prenom" : "Quentin", "dateNaissance" : "1965-02-26", "adresse" : "34 avenue du Port", "tel" : "0645627
891"}     |          2
(3 rows)
```

Affichage des données résultant des instructions précédentes dans la table Assistant

```
postgres=# SELECT vet.infos->>'prenom' AS prenom, vet.infos->>'nom' AS nom, ani.
idAnimal, ani.nom from Animal ani
postgres-# JOIN EstSuiviPar esp ON esp.animal = ani.idAnimal
postgres-# JOIN Veterinaire vet ON esp.veterinaire = vet.idVet
postgres-# WHERE vet.idVet=11;
 prenom |  nom  | idanimal | nom
--------+-------+----------+------
 Simon  | Biffe |        3 | Bubu
(1 row)

postgres=# SELECT infos->>'prenom' AS prenom, infos->>'nom' AS nom, idDossier, s
aisie from DossierMedical dm
postgres-# JOIN Veterinaire vet ON vet.idVet = dm.veterinairePrescripteur
postgres-# WHERE vet.idVet=12;
 prenom |  nom  | iddossier |   saisie
--------+-------+-----------+------------
 Simon  | Vital |         3 | 2013-12-03
(1 row)
```

Exemple de deux instructions SELECT qui renvoient respectivement les animaux suivis par un vétérinaire et les dossiers où il a été vétérinaire prescripteur

Annexe

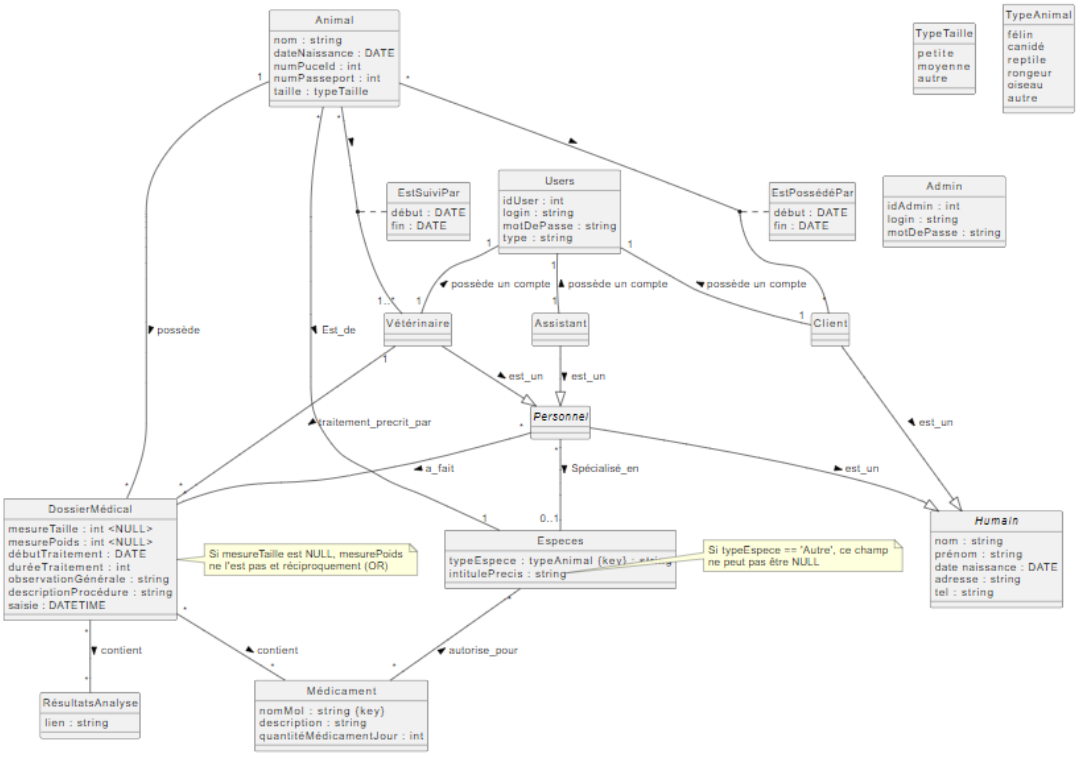


Schéma UML de notre projet

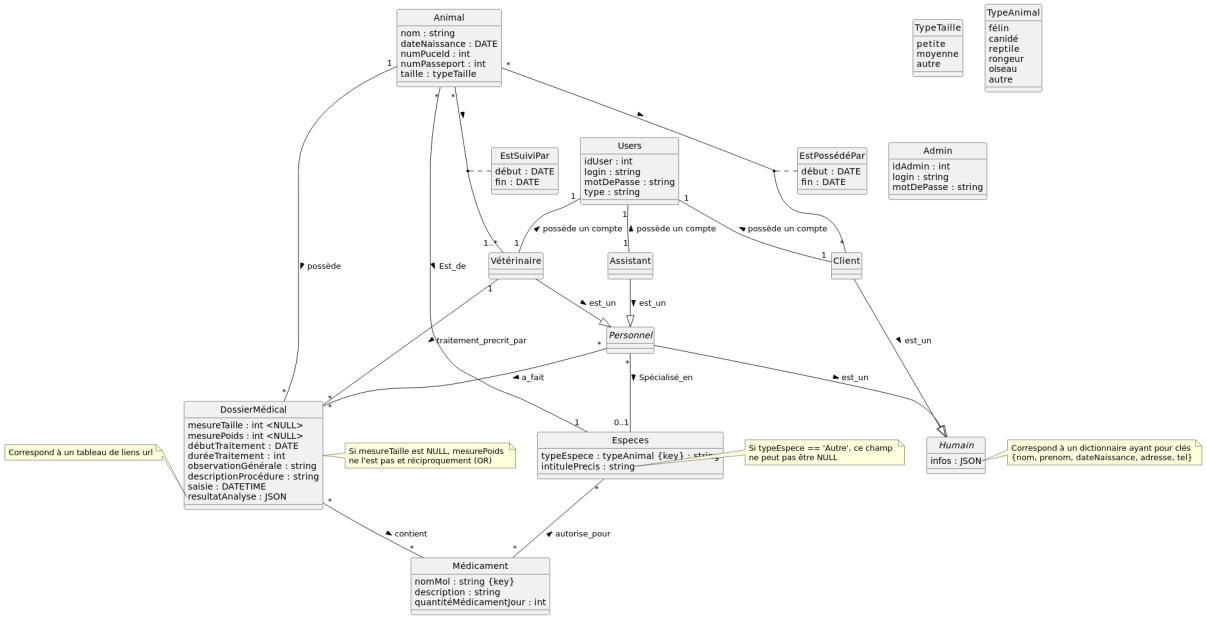


Schéma UML du projet en NoSQL (disponible à l'adresse https://md.picasoft.net/s/p9aqzUgHq)