

|  |
| --- |
| Rapport Technique – Vetcare 360  Version : java. |
|  |
| 17 mai  ILCS  Créé par : SAFAA MOUFLIH & SIHAM BOUZID |



Table of Contents

[Table des Matière 1](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698866)

[Introduction 2](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698867)

[Partie 1 : Objectif du projet 3](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698868)

[Technologies utiliseés 3](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698870)

[1.2 Architecture du projet(MVC) 4](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698872)

[Modèle (Model) 4](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698873)

[Vue (View – FXML & CSS) 4](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698874)

[Contrôleur (Controller) 4](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698874)

[1.3 Fonctionnement de Main.java 4](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698875)

[Les Contrôleurs : MainController,OwnerController,PetController,MedicalRecordControlle 5](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698876)

Services (logique métier)………………………………………………………………………………………………………….6

[2.2 Utilitaires (Utils) : DatabaseUtil, AlertUtil, NavigationUtil, ValidationUtil 6](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698880)

[2.3Configuration du projet 6](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698881)

[2.4 Gestion des images 6](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698881)

[2.5 Navigation entre les vues 6](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698881)

[2.6 Base de données (JDBC) 6](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698881)

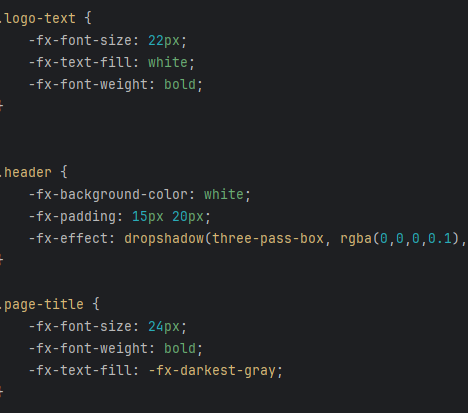
[2.7Améliorations possibles 6](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698881)

[Conclusion 8](file:///C:\Users\dell\Downloads\suham.docx#_Toc197698882)

# 1. Introduction

|  |
| --- |
| 1.1\_Ce rapport présente une documentation technique approfondie du système de gestion « Pet Clinc ».  Développée en Java avec Java FX, cette application permet de gérer les opérations d’une clinique vétérinaire, telles que : - la gestion des animaux (ajout, modification, suppression), - la gestion des propriétaires, - l’historique médical des animaux, - la planification de rendez-vous. |
| Elle suit une architecture logicielle propre (MVC), permettant séparation des responsabilités, évolutivité, et maintenabilité. |
| 1.2\_Objectif du projet L’application permet de centraliser les informations médicales des animaux et facilite le travail administratif du vétérinaire. Elle évite les erreurs manuelles et propose une interface simple à utiliser. 1.3\_ Technologies utilisées - Java 17 : langage de programmation principal. - JavaFX : pour créer l’interface graphique. - FXML : pour structurer les vues. - CSS : pour le style graphique. - Maven : pour gérer les dépendances. - IntelliJ IDEA : comme environnement de développement.  2. **Comment c’est construit ?**  On utilise une méthode appelée architecture MVC :   * M pour Modèle → c’est les données (ex : Pet.java = animal). * V pour Vue → c’est l’interface graphique (ex : les fichiers. Fxml). * C pour Contrôleur → c’est le lien entre les deux (ex : PetController.java).   L'application repose sur le modèle MVC (Modèle-Vue-Contrôleur) : Modèle (Model) : classes représentant les données (Pet, Owner, MedicalRecord). Chaque modèle correspond à une table dans la base de données.  Vue (View): fichiers `.fxml` définissant l’interface utilisateur. Ils sont stylisés avec des fichiers CSS pour améliorer l’expérience visuelle.  Contrôleur (Controller) : fichiers `.java` qui relient la vue et la logique métier.  D'autres composants incluent : Services : ils contiennent la logique métier, souvent liée aux opérations CRUD. Utilitaires: classes d’aide pour la gestion des dates, des connexions, des validations, etc. 3. Classe Principale – Main.java Cette classe est le point d’entrée de l’application. Elle : - Initialise la base de données via `DatabaseUtil.initDatabase()`. - Charge l'interface utilisateur principale avec `FXMLLoader.load()`. - Applique les styles avec un fichier CSS (`application.css`). - Définit les propriétés de la fenêtre principale (`primaryStage`) : titre, icône, taille minimale. - Libère les ressources à la fermeture (`stop()` -> `DatabaseUtil.closeConnection()`). 4. Contrôleurs Les contrôleurs JavaFX permettent d’interagir avec les éléments définis dans les fichiers FXML.    MainController : gère l’écran d’accueil et la navigation générale. OwnerController : permet d’ajouter, modifier, supprimer des propriétaires. PetController: gère les informations relatives aux animaux. MedicalRecordControlle: permet la gestion des dossiers médicaux des animaux. DashboardController : centralise les accès rapides à toutes les sections.  Chaque contrôleur interagit avec ses services respectifs pour récupérer ou modifier les données. **5. Modèle (Model) :** Les classes du modèle représentent les entités principales du système comme : - Pet : contient le nom, type, âge, etc. - Owner : contient le nom, adresse, téléphone. - MedicalRecord : contient l’historique des soins. **6. Vue (View – FXML & CSS) :** Les fichiers FXML définissent l’interface (boutons, tableaux, zones de texte). Le fichier CSS applique le style (couleur, taille, police, etc.) à l’interface. 7. Interfaces Utilisateur (FXML & CSS)A. ****Qu’est-ce que le FXML ?**** Le ****FXML**** est un ****langage basé sur XML**** utilisé avec JavaFX pour ****décrire l’interface graphique**** d’une application.  C’est un peu comme le ****HTML d’un site web****, mais pour JavaFX. Il permet de ****séparer le design de l’interface du code Java****, ce qui rend le projet :   * Plus lisible, * Plus facile à modifier, * Plus organisé.   Exemple simple de FXML :    **B. Pourquoi utiliser FXML ?**   * Tu te concentres sur l’interface dans un fichier. Fxml * Tu mets la logique métier dans un fichier .java (le contrôleur)   **C**. **Et le CSS dans tout ça ?**  Le CSS (Cascading Style Sheets) est utilisé pour styliser ton interface. C’est exactement comme pour un site web.  Tu peux modifier :   * Les couleurs * Les polices de caractères * Les tailles * Les bordures * Les icônes * Les animations, etc. |

Exemple de fichier application.css :



### D. Comment le FXML et le CSS sont liés au Java ?

1. Le fichier. Fxml définit l’**interface**.
2. Il est **chargé dans le code Java** avec FXMLLoader :

« FXMLLoader.load(getClass().getResource("/fxml/MainView.fxml")) ;»

1. Le CSS est **appliqué au Scène** :

« scene.getStylesheets().add(getClass().getResource("/css/application.css").toExternalForm());»

# 8. Services Métier

Les services sont des classes intermédiaires entre les contrôleurs et la couche d’accès aux données.  
Ils regroupent la logique métier.  
  
Exemples :

PetService : vérifie si un animal peut être ajouté/modifie/supprime un animal (pas de doublons).  
OwnerService : recherche un propriétaire par nom, ID ou adresse.  
MedicalRecordService: ajoute un diagnostic, un traitement, etc.  
  
Cela permet de réutiliser la logique sans la dupliquer dans chaque contrôleur.

# 9. Outils et Classes Utilitaires

Ces classes permettent de simplifier la gestion d’actions fréquentes :

DatabaseUti: ouvre, initialise et ferme la connexion à la base de données.

AlertUtil : affiche des boîtes de dialogue personnalisées.

DateUtil : convertit les dates locales et formatées.

ValidationUtil: vérifie que les champs requis ne sont pas vides.

NavigationUtil : charge et affiche une nouvelle vue FXML dans la fenêtre.

# 10. Configuration du Projet

JDK : version 17

IDE utilisé : IntelliJ IDEA avec configuration Maven

Module-path JavaFX: utilisé via `--module-path` pour ajouter les modules

`javafx.controls` et `javafx.fxml`.

Organisation en packages : controller, model, service, util

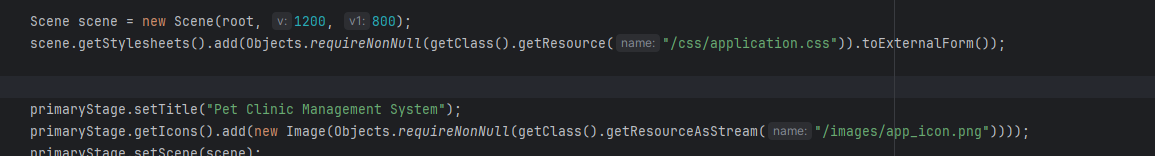
Structure Maven: `src/main/java`, `src/main/resources` contiennent le code source et les vues.

# 11. Ressources et Gestion des Images

Les images de l'application sont stockées dans `src/main/resources/images`.

Exemple :

`/images/app\_icon.png` est chargé comme icône principale.  
  
Dans le code :



Les images peuvent être utilisées dans les vues FXML pour les boutons, arrière-plans, avatars, etc.

# 12. Navigation entre Vues

Le changement de vues est centralisé via un utilitaire comme `NavigationUtil`, ou intégré directement dans les contrôleurs avec `FXMLLoader`.  
  
Cela permet :

- De passer d’une interface à une autre sans dupliquer le code de navigation.

- De garder une scène unique et la mettre à jour avec différentes vues selon le contexte.

Quand tu veux passer d’un écran à un autre (ex. cliquer sur "Propriétaires"), on utilise :

«FXMLLoader.load(getClass().getResource("/fxml/OwnerView.fxml")); »

Mais au lieu de recopier ce code partout, on peut créer une **méthode utilitaire** dans NavigationUtil.

# 13. Base de Données et Persistance.

* On utilise une base locale (probablement MySQL).
* Les données sont sauvegardées : animaux, propriétaires, dossiers…
* DatabaseUtil s’occupe de la connexion.
* À chaque lancement, si la base est vide, elle est créée automatiquement (crée les tables si elles n'existent pas).
* Toutes les opérations (select, insert, update, delete) passent par les services.

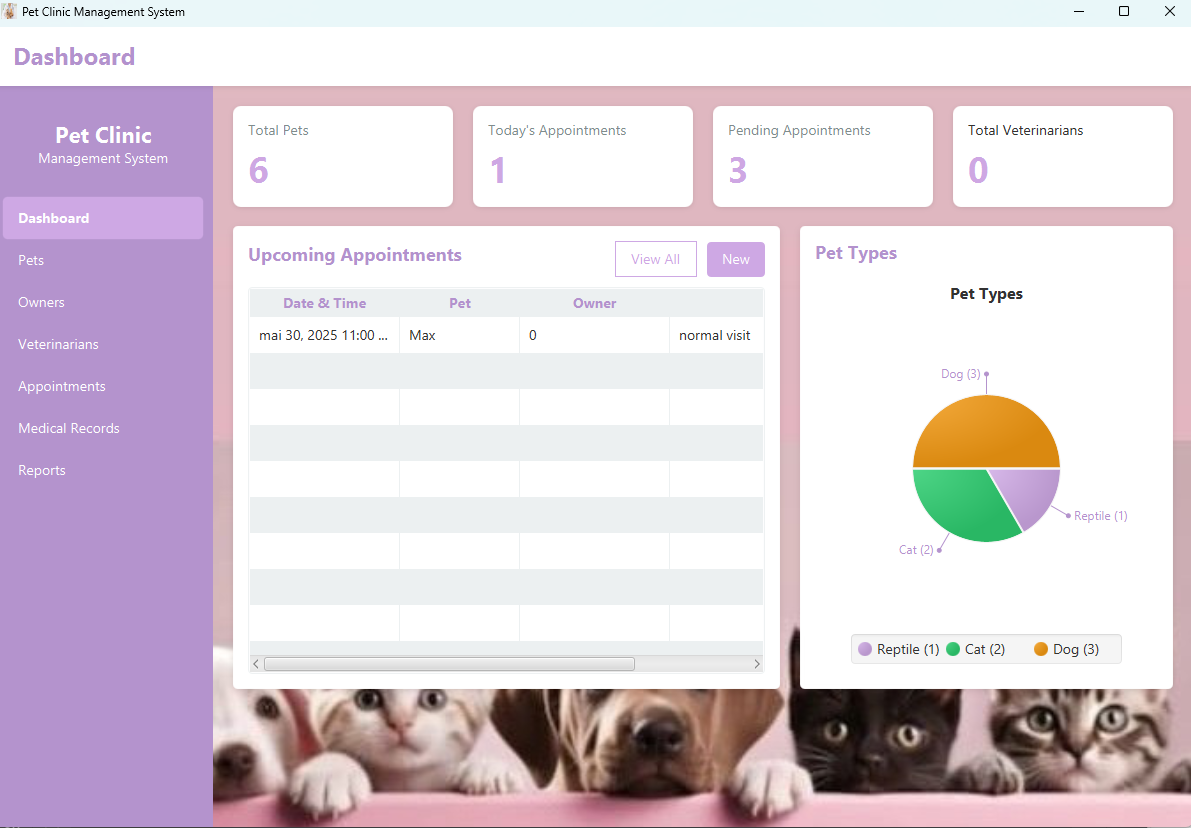
Avantage : les données sont persistantes (elles ne disparaissent pas quand tu fermes l’application).

# 

14. Annexes – Captures d’écran de l’application

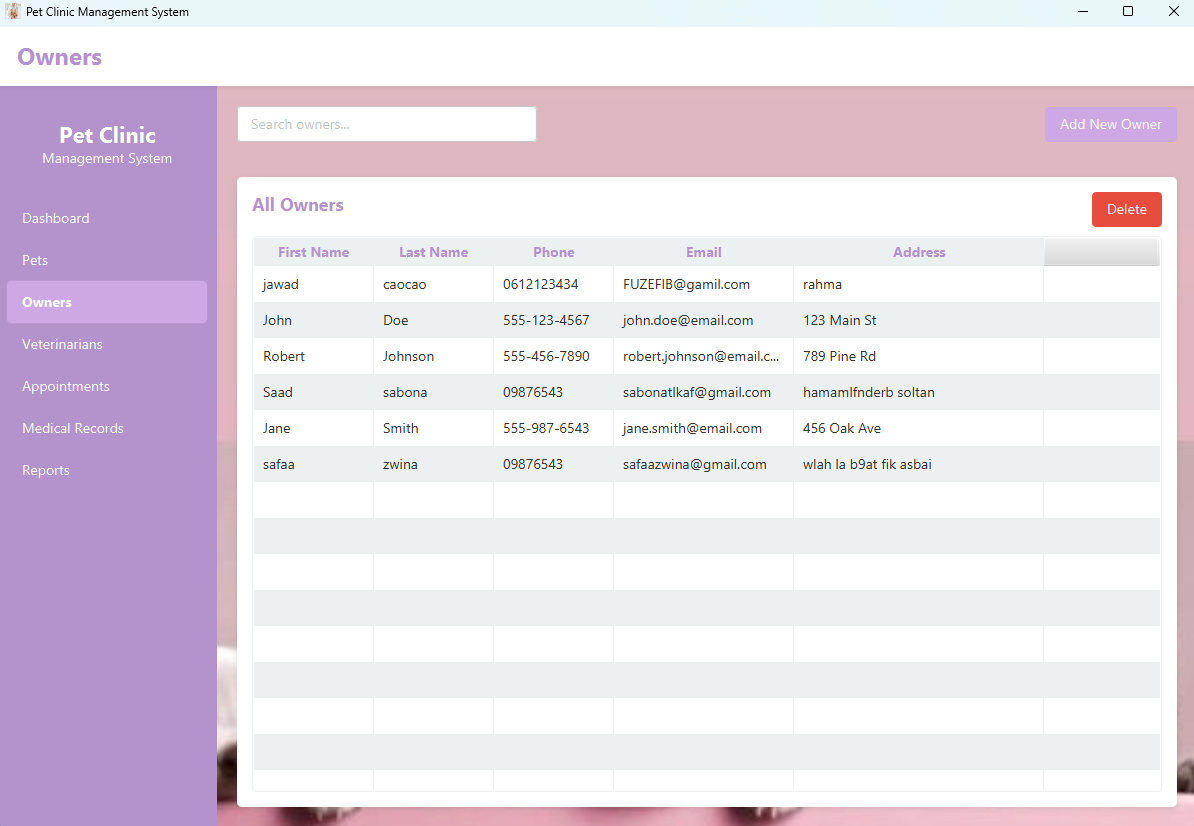
– Interface principale :

Cette image montre la fenêtre d’accueil de l’application Pet Clinc.  
On y voit les boutons de navigation permettant d'accéder aux différentes sections (animaux, propriétaires, dossiers médicaux...).



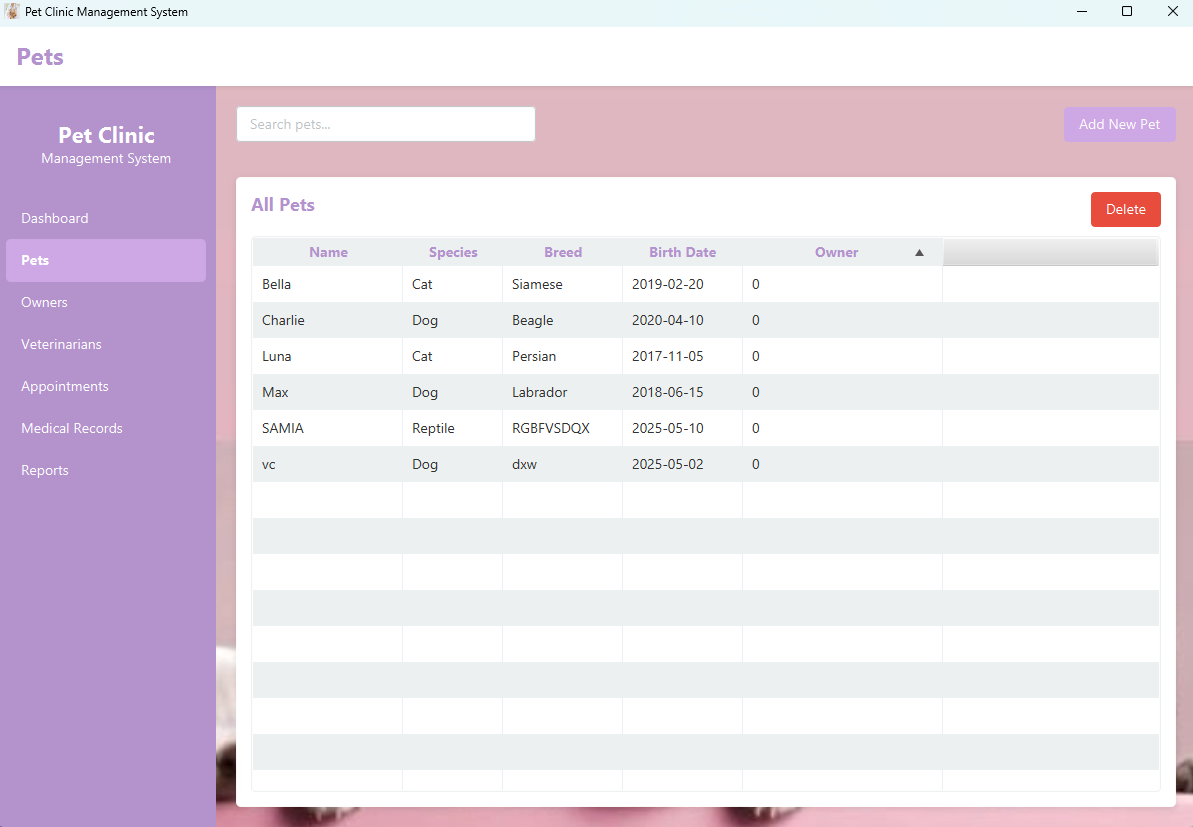
– Gestion des propriétaires :

Ici, l'utilisateur peut ajouter, modifier ou supprimer un propriétaire d'animal.  
Les champs obligatoires sont validés grâce à ValidationUtil.



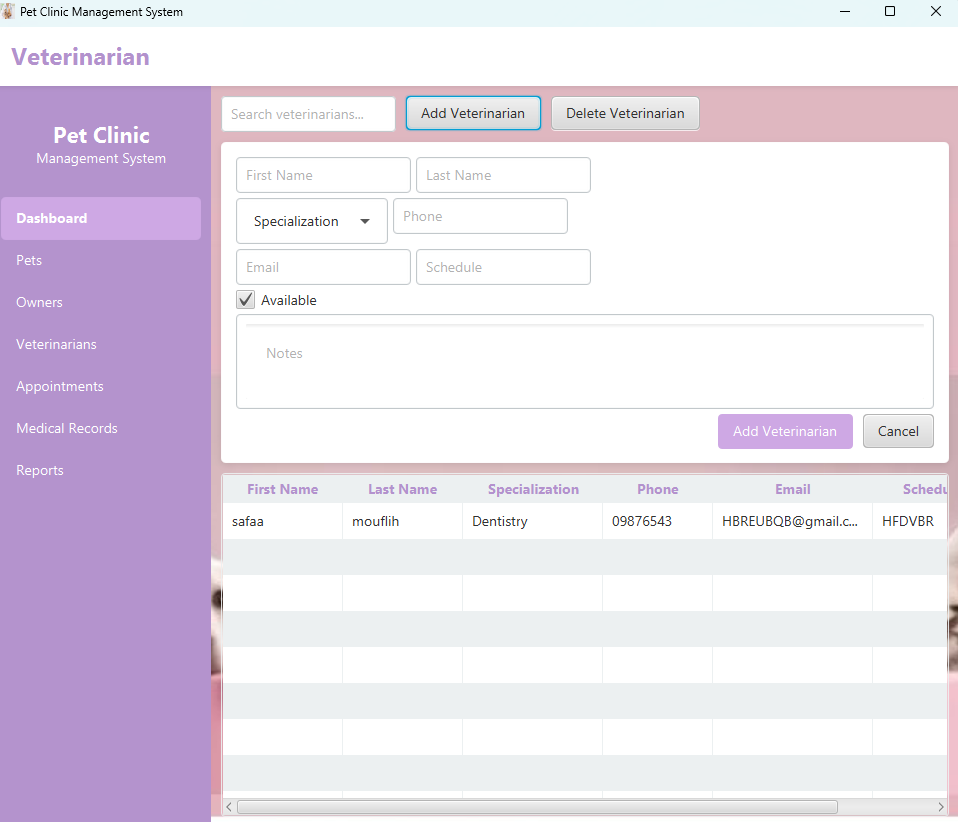
– Liste des animaux :

Cette capture présente la table contenant les animaux enregistrés.  
Chaque ligne correspond à un animal avec son nom, type, race, propriétaire et date de naissance.

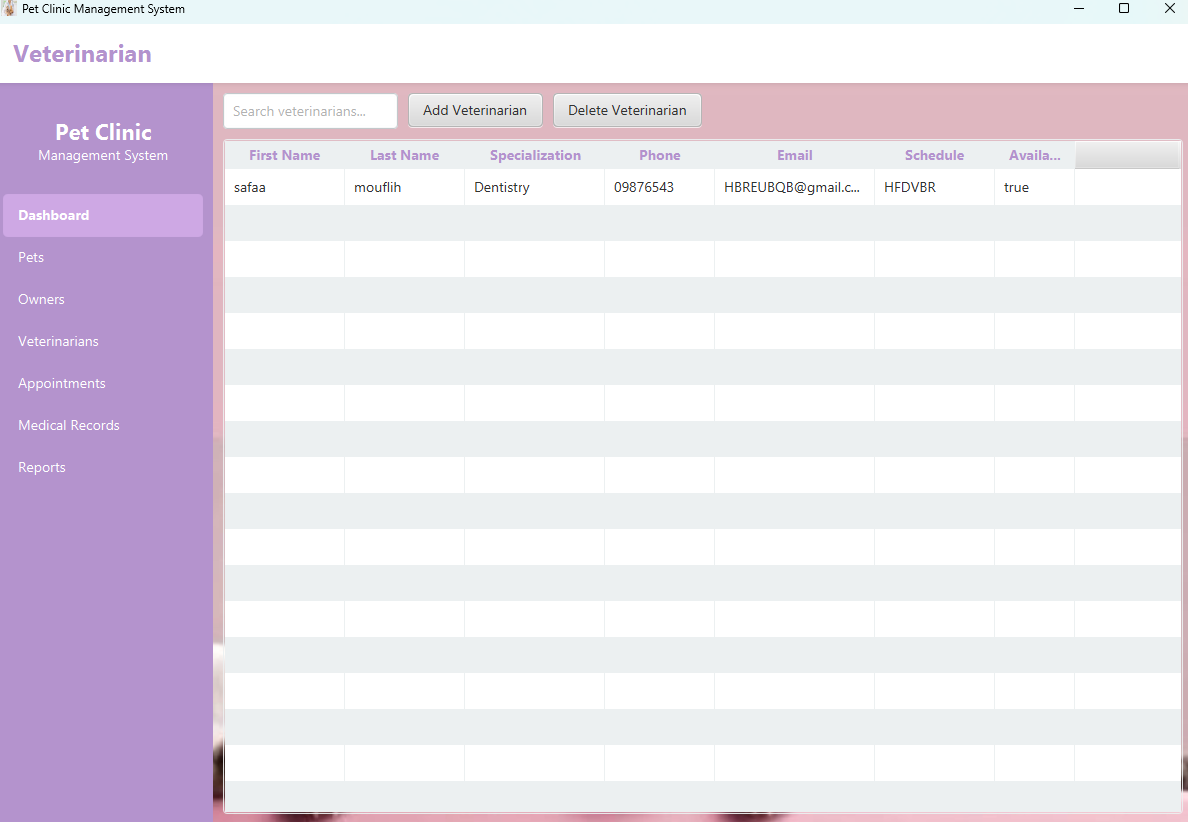


\_Vétérinaires :

1.formulaire a remplire :



2.afichage de la liste :



Notre clinique s’appuie sur une équipe de vétérinaires hautement qualifiés, chacun spécialisé dans des domaines tels que la médecine générale, la chirurgie, la dentisterie ou encore l’oncologie. Grâce à leur expertise et à leur disponibilité, nous garantissons à nos patients à quatre pattes des soins personnalisés et de qualité, dans un environnement sécurisé. Chaque praticien suit un planning rigoureux pour répondre rapidement aux urgences et assurer un suivi régulier des animaux confiés à notre établissement.

– Dossier médical :

cette interface représente la section Dossiers Médicaux (*Médical Records*) de l’application de gestion d’une clinique vétérinaire.

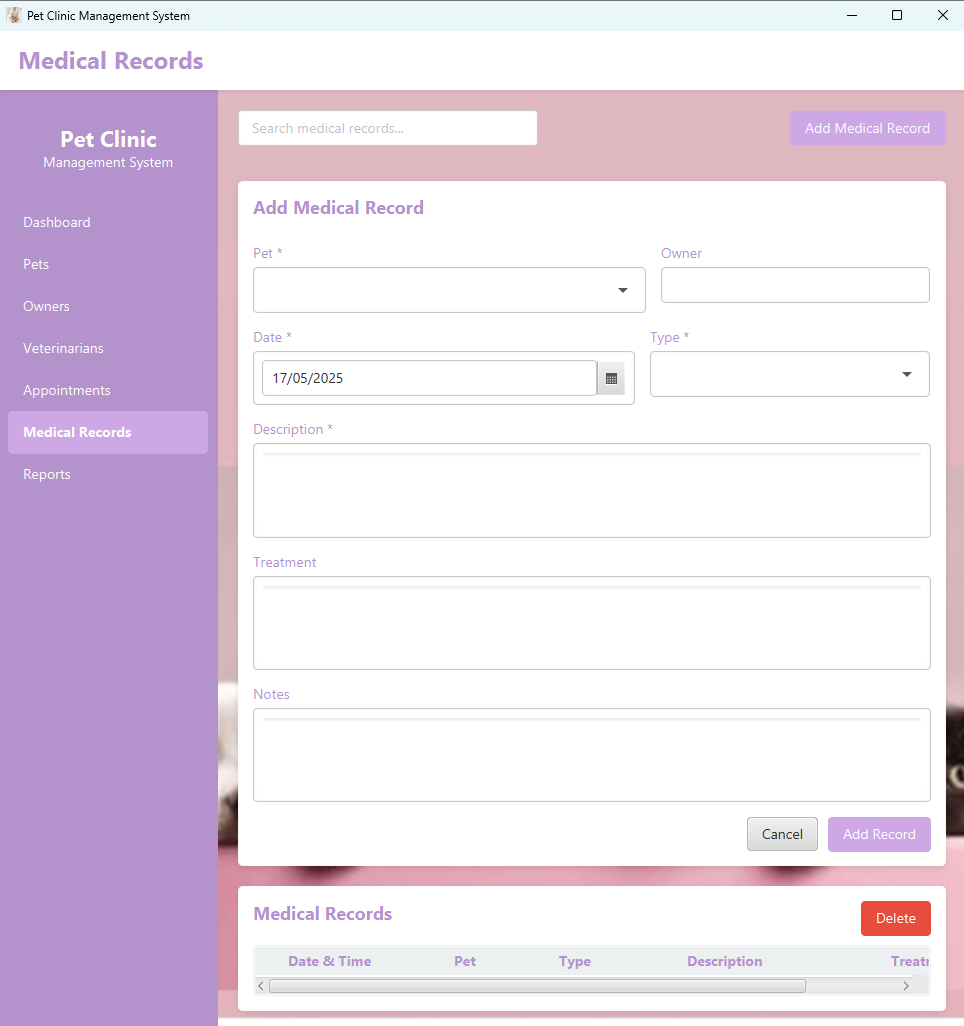
Elle permet à l’utilisateur de :

* Chercher des dossiers médicaux à l’aide de la barre de recherche.
* Ajouter un nouveau dossier en cliquant sur le bouton Add Médical Record.
* Supprimer un dossier sélectionné via le bouton Delete.

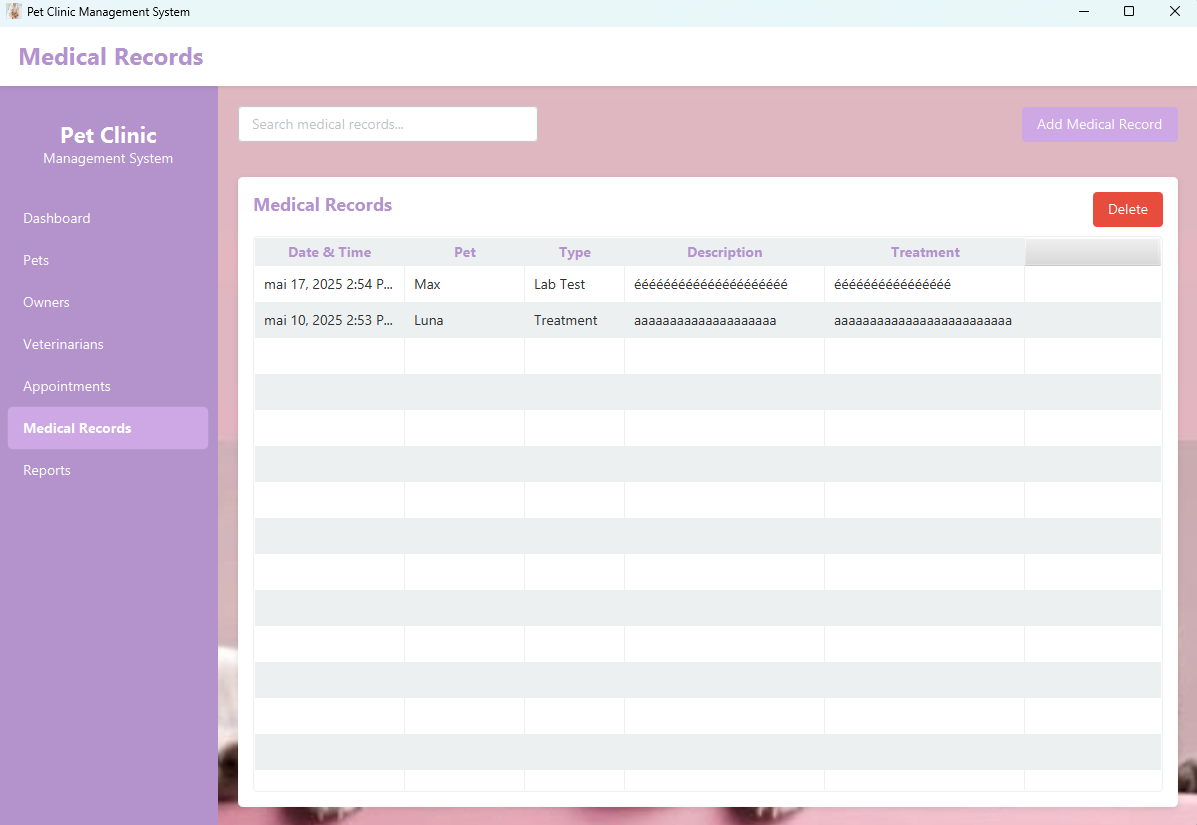
La table affiche les données suivantes :

* Date & Heure de l’enregistrement.
* Nom de l’animal (Pet).
* Type de traitement (ex. : *Lab Test*, *Treatment*).
* Description du problème ou diagnostic.
* Traitement prescrit ou effectué

1.formulaire a remplir :



2.afichage de la liste :



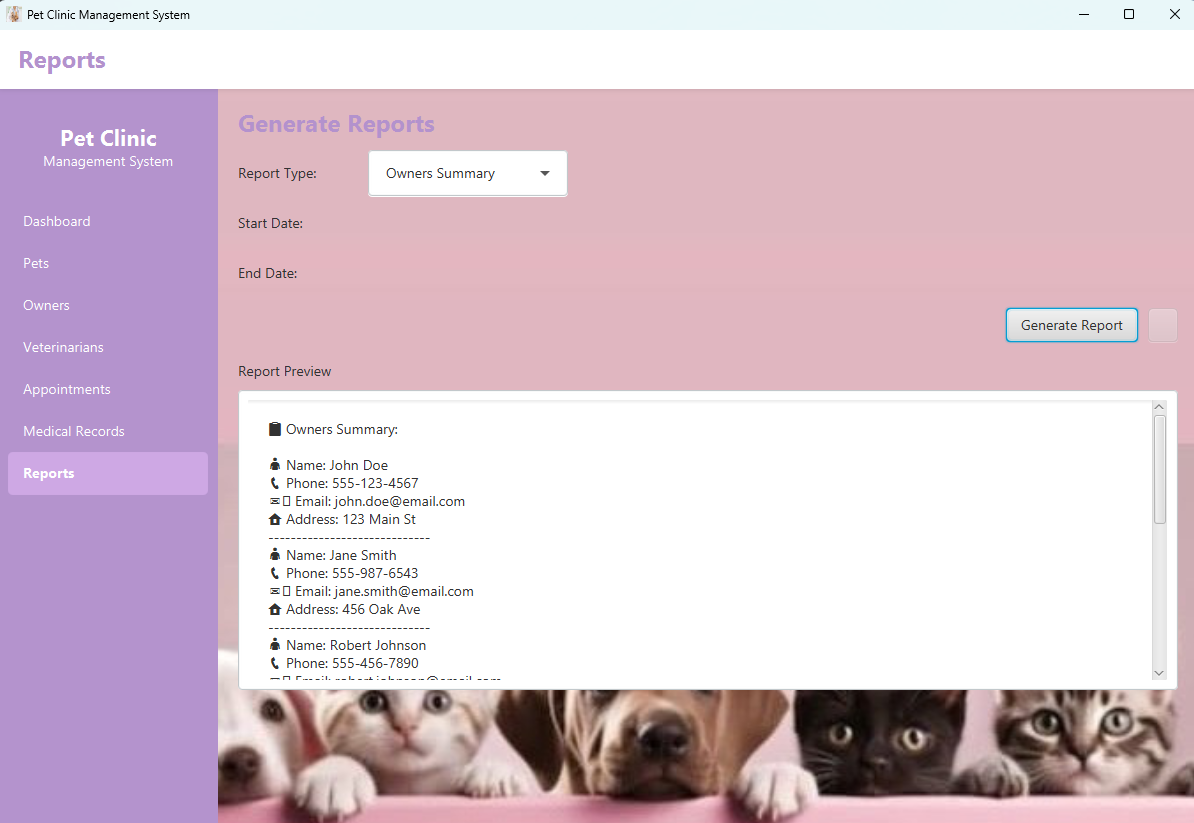
\_ Reports :

Cette interface montre le système de génération de rapports dans l’application de gestion de la clinique vétérinaire.

L’interface est conçue avec des tons rose pastel et un arrière-plan décoré de photos d’animaux mignons, ce qui lui donne un style doux et féminin.

L’utilisateur peut choisir un type de rapport depuis le menu déroulant (par exemple : *Résumé des propriétaires*), sélectionner une date de début et une date de fin, puis cliquer sur le bouton Generate Report.

Le résultat s’affiche dans la zone de texte située en dessous, sous forme de résumé contenant les informations des propriétaires : nom, numéro de téléphone, email et adresse.

15.Avantages de l'application Pet Clinc :

# 1. Portabilité de Java :

# L'application est développée en Java, un langage reconnu pour sa portabilité. Grâce à la machine virtuelle Java (JVM), le même fichier exécutable peut fonctionner sur plusieurs systèmes d’exploitation (Windows, Linux, macOS) sans modifications du code source. Cela facilite la distribution de l’application sur divers postes de travail dans un environnement professionnel.

# 2. Interfaces graphiques modernes avec JavaFX :

# JavaFX permet la création d’interfaces utilisateur modernes, interactives et attrayantes. Grâce à l’utilisation de FXML pour la structure et CSS pour le style,

# l’interface est séparée de la logique, ce qui améliore la lisibilité et la maintenabilité du projet. JavaFX permet aussi :

# L’ajout d’animations,

# L’intégration de médias (images, icônes),

# Une expérience utilisateur fluide et professionnelle.

# 3.Fiabilité de MySQL :

# MySQL est un système de gestion de base de données solide, largement utilisé et bien documenté. Son intégration avec JDBC dans le projet Pet Clinic permet :

# La persistance des données (elles restent sauvegardées même après la fermeture),

# Une gestion structurée des informations (animaux, propriétaires, dossiers médicaux…),

# La possibilité d’évoluer vers des bases plus complexes ou distantes (ex : PostgreSQL, MariaDB).

# 4.Sécurité de l’application :

# Même en environnement local, la sécurité des données est une priorité. Plusieurs mécanismes sont mis en place ou prévus :

# Connexion sécurisée à la base de données : L’accès à la base MySQL se fait via JDBC avec des identifiants stockés dans un fichier de configuration non exposé dans l’interface. Il est possible de chiffrer ces identifiants pour plus de sécurité.

# Isolation des responsabilités : L’architecture MVC limite les accès directs à la base depuis l’interface. Toute action passe par les services, où des règles de sécurité peuvent être centralisées.

# Fermeture propre des connexions : La classe DatabaseUtil garantit que chaque connexion est correctement ouverte et fermée, évitant ainsi les fuites de ressources ou les attaques de type "denial of service" internes.

# Fiabilité du Système :

# Gestion des erreurs : Des blocs try-catch sont présents dans toutes les classes critiques pour intercepter les erreurs système et informer l'utilisateur sans interrompre brutalement l’application.

# Validation des Données :

# L'application Vetcare360 intègre une couche de validation robuste afin de garantir la cohérence et la fiabilité des informations saisies par l'utilisateur. Cela comprend :

# Contrôle des champs requis : Les champs essentiels comme le nom, email, numéro de téléphone, date de naissance ou description médicale sont obligatoires. L'oubli d’un champ déclenche un message d’erreur clair.

# Messages d’erreur explicites : Chaque champ incorrect ou manquant affiche un message personnalisé (ex. : "Veuillez entrer un email valide"), ce qui améliore l’expérience utilisateur.

# Validation de format : Utilisation de ValidationUtil pour vérifier le format des emails, des numéros, ou la cohérence des dates (ex. : pas de date future pour une naissance).

# 16. Suggestions d’Amélioration

Pour le rendre encore plus professionnel, on peut ajouter :

* 🔐 Connexion avec identifiants (admin / utilisateur)
* 🧠 Recherche intelligente (filtrage en temps réel)
* 📤 Exportation des données (en PDF ou Excel)
* ✅ Tests automatiques avec JUnit
* - Ajouter une couche DAO (Data Access Object) pour séparer encore mieux la logique.
* ☁️ Connexion à une base en ligne (PostgreSQL distante)
* 🔔 Notifications ou rappels de rendez-vous

# 17. Conclusion

Ce projet est une base robuste pour toute application de gestion.  
La séparation MVC, l’organisation modulaire, la clarté du code et la configuration Maven permettent une extension facile.  
Il est recommandé d’enrichir ce projet par des fonctionnalités avancées comme : authentification, messagerie, export de données.