جامعة محمد الأول بوجدة
UNIVERSITÉ MOHAMMED PREMIER OUJDA
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵎⵓⵃⴰⵎⵎⴷ ⵜⴰⵎⴻⵡⴰⵔⵜ ⵏ ⵡⵓⵊⴷⴰ

المدرسة العليا للتكنولوجيا
École Supérieure de Technologie
ⵜⴰⵙⵙⵓⵔⵜ ⵏ ⵜⵓⵙⵙⵏⵉⵡⵉⵏ ⴰⵏⴰⵎⵓⵔ ⵏ ⵜⵓⵊⴷⴰ

Diplôme Universitaire de Technologie (DUT)

Ingénierie Logicielle et Cybersécurité (ILCS)

Rapport du Projet
JAVA

VetCare-360 -- Application web d'une clinique vétérinaire

Réalisé par :

ASTITOU fatima zzahrae

AIT ADDI tithrite

Encadrement pédagogique :
Pr . Redouane esbai

Année universitaire : 2024 - 2025


رَبَّنَآ ءَاتِنَا مِن لَّدُنكَ رَحْمَةً
وَهَيِّئْ لَنَا مِنْ أَمْرِنَا رَشَدًا

Remerciements

Nous tenons à exprimer notre profonde gratitude à Monsieur Esbai Redouane, notre professeur de la matière "Modélisation et Programmation Orientée Objet en Java", pour son précieux enseignement et son accompagnement tout au long de ce projet.

Ses explications claires, sa pédagogie et sa patience nous ont permis de comprendre les fondamentaux de la programmation orientée objet et de les appliquer concrètement dans ce projet. Son expertise et son soutien ont été déterminants dans l'acquisition de ces compétences essentielles en développement Java.

Je lui suis sincèrement reconnaissant pour le temps qu'il nous a consacré et pour son engagement à transmettre son savoir avec passion.

Table des matières

I. INTRODUCTION

Dans le cadre de notre projet de développement d'applications de bureau concernant la matière de **Programmation Orientée Objet en Java**, j'ai choisi de réaliser une version desktop de **VetCare 360**, une application dédiée à la gestion d'une clinique vétérinaire. Ce projet vise à faciliter l'administration des dossiers des animaux, la planification des rendez-vous et la gestion des vétérinaires, en appliquant les concepts étudiés lors des séances de travaux pratiques.

L'objectif principal est de maîtriser la création d'applications professionnelles en **Java et JavaFX**, tout en renforçant nos compétences en développement logiciel. Ce rapport mettra en lumière l'importance de **JavaFX** pour concevoir des interfaces utilisateur dynamiques et ergonomiques .

2. BESOINS ET OBJECTIFS DU PROJET

Le projet VetCare 360 répond à un besoin croissant d'optimisation et de digitalisation des processus de gestion dans les cliniques vétérinaires. À travers cette application desktop développée en Java avec JavaFX, nous visons à offrir une solution complète et intuitive pour les professionnels du secteur vétérinaire.

Besoins identifiés

Notre analyse préliminaire a identifié plusieurs besoins cruciaux auxquels l'application doit répondre :

- Centralisation des informations relatives aux propriétaires d'animaux et à leurs compagnons
- Suivi médical détaillé et historique des visites pour chaque animal
- Gestion efficace des rendez-vous et optimisation du planning des vétérinaires
- Interface intuitive permettant une prise en main rapide par le personnel de la clinique .
- Sécurisation des données sensibles des clients et de leurs animaux

Objectifs principaux

Développer une interface utilisateur ergonomique exploitant pleinement les capacités de JavaFX pour créer une expérience utilisateur fluide et moderne .

Implémenter un système de gestion des propriétaires permettant d'ajouter, modifier et rechercher les informations des animaux et des vétérinaires et des propriétaires .

1. Concevoir un module de suivi des animaux avec la possibilité d'associer plusieurs animaux à un même propriétaire et de documenter leur historique médical
2. Créer un système de gestion des visites médicales offrant la possibilité d'enregistrer les diagnostics, traitements et observations
3. Intégrer une fonctionnalité d'affichage des vétérinaires disponibles dans la clinique
4. Appliquer les principes de la programmation orientée objet pour créer une architecture logicielle modulaire, extensible et maintenable

VetCare 360: Système de Gestion pour Clinique Vétérinaire

II. Fonctionnalités Principales

1. Gestion des propriétaires d'animaux

- **Ajouter des propriétaires:** Interface permettant d'enregistrer les informations personnelles des propriétaires (nom, prénom, adresse, numéro de téléphone, email).
- **Modifier des propriétaires:** Possibilité de mettre à jour les informations des propriétaires existants.
- **Supprimer des propriétaires:** Fonctionnalité pour retirer des propriétaires de la base de données (avec confirmation).
- **Rechercher des propriétaires:** Système de recherche par nom de famille pour retrouver rapidement un propriétaire.
- **Afficher les détails:** Vue détaillée des informations du propriétaire avec liste de ses animaux associés.

2. Gestion des animaux

- **Ajouter des animaux:** Enregistrement des animaux avec leurs caractéristiques (nom, espèce, race, date de naissance, sexe, poids).
- **Associer aux propriétaires:** Lier chaque animal à son propriétaire dans la base de données.
- **Modifier des informations:** Interface pour mettre à jour les données des animaux.
- **Supprimer des animaux:** Option pour retirer un animal de la base de données.
- **Historique médical:** Accès à l'historique complet des visites médicales pour chaque animal.

3. Gestion des visites médicales

- **Programmer des visites:** Système de prise de rendez-vous pour les animaux.
- **Enregistrer des consultations:** Saisie des informations médicales lors des visites (symptômes, diagnostic, traitement).

- **Consulter l'historique:** Visualisation chronologique des visites passées pour chaque animal.
- **Ajouter des prescriptions:** Enregistrement des médicaments prescrits et instructions de traitement.
- **Suivi de santé:** Tableau de bord pour visualiser l'évolution de la santé de l'animal.

4. Liste des vétérinaires

- **Afficher les vétérinaires:** Présentation de tous les vétérinaires exerçant à la clinique.
- **Détails des vétérinaires:** Informations sur chaque vétérinaire (spécialité, horaires, contact).
- **Associer aux visites:** Lier chaque visite médicale au vétérinaire qui l'a effectuée.

III. Installation et Configuration

1. Prérequis techniques

- JDK 17 ou version supérieure
- Maven 3.9.9ou version supérieure
- ArrayList pour la structure de données
- IDE recommandé: IntelliJ IDEA .

2. Installation du JDK

- Télécharger le JDK depuis le site officiel d'Oracle
- Configurer les variables d'environnement JAVA_HOME
- Vérifier l'installation avec la commande java -version

3. Installation de Maven

- Télécharger Maven depuis le site apache.maven.org
- Configurer les variables d'environnement MAVEN_HOME

- Vérifier l'installation avec la commande mvn -version

4. Configuration de la structure de données

- Utilisation des ArrayList pour stocker les données en mémoire
- Implémentation des classes de modèle avec des relations entre objets
- Système de sauvegarde et chargement des données via sérialisation/désérialisation

5. Configuration du projet

- Cloner le dépôt Git: git clone [URL_DU_REPO]
- Naviguer vers le répertoire du projet: cd vetcare360
- Installer les dépendances: mvn install
- Configurer les fichiers de gestion des ArrayList

6. Lancement de l'application

- Compiler le projet: mvn compile
- Exécuter l'application: mvn javafx:run
- Accéder à l'application via l'interface JavaFX qui s'ouvre

Technologies utilisées

- Frontend: JavaFX avec SceneBuilder pour les interfaces graphiques (fxml)
- Backend: Java 17+ avec gestion des données via ArrayList
- Structure de données: ArrayList pour stocker les propriétaires, animaux, visites et vétérinaires
- Persistance: Sérialisation/désérialisation des objets Java
- Build et gestion de dépendances: Maven
- Gestion des versions: Git/GitHub

Cette version du système VetCare 360 utilise des ArrayList pour la gestion des données au lieu d'une base de données traditionnelle.

- Pour développer l'application VetCare 360, nous avons mis en place un environnement de développement complet nécessitant plusieurs composants essentiels. L'installation de Java JDK 17 (ou version ultérieure) constitue la base de notre environnement, fournissant les outils fondamentaux pour le développement Java. Pour la gestion des dépendances et l'automatisation du build, nous avons intégré Apache Maven 3.9.9, un outil essentiel qui simplifie considérablement la gestion du cycle de vie du projet et l'intégration des bibliothèques externes. La création d'interfaces utilisateur modernes et interactives est assurée par JavaFX SDK 17 (version x64), offrant des fonctionnalités avancées pour concevoir une expérience utilisateur optimale.

```
PS C:\Users\user> mvn -version
Apache Maven 3.9.9 (8e8579a9e76f7d015ee5ec7bfcdc97d260186937)
Maven home: D:\Maven\apache-maven-3.9.9
Java version: 17.0.14, vendor: Microsoft, runtime: C:\Users\user\.jdks\ms-17.0.14
Default locale: en_US, platform encoding: Cp1252
OS name: "windows 11", version: "10.0", arch: "amd64", family: "windows"
```

Configuration de l'environnement

La configuration de notre environnement de développement a nécessité plusieurs étapes cruciales. Tout d'abord, nous avons défini les variables d'environnement système, notamment JAVA_HOME pointant vers le répertoire d'installation du JDK, et ajouté les chemins nécessaires à la variable PATH pour accéder aux exécutables Java et Maven depuis n'importe quel emplacement. Pour JavaFX, nous avons configuré la variable PATH_TO_FX pointant vers le répertoire lib du SDK JavaFX.

L'IDE IntelliJ IDEA a été choisi pour sa puissance et son support natif des technologies Java. Nous avons configuré l'environnement en installant le plugin JavaFX qui offre une intégration complète pour le développement d'interfaces graphiques. L'utilisation d'IntelliJ nous a permis de bénéficier de fonctionnalités

avancées comme la prévisualisation des interfaces FXML en temps réel et l'auto-complétion contextuelle pour les composants JavaFX.

Pour intégrer Maven à notre projet, nous avons créé un fichier pom.xml personnalisé incluant les dépendances

JavaFX et d'autres bibliothèques requises. Nous avons également configuré les options VM appropriées (--module-path %PATH_TO_FX% --add-modules javafx.controls,javafx.fxml) dans la configuration d'exécution d'IntelliJ pour assurer la bonne exécution de l'application. Cette configuration rigoureuse nous a permis de disposer d'un environnement stable et performant pour le développement de notre application de gestion vétérinaire.

L'instalation de javaFx :

https://gluonhq.com/products/javafx/

le type SDK , Architecture : x 64 , Version : 17

L'instalation du VM :

https://openjfx.io/openjfx-docs/

- Pour Windows

On copier cette commande :

--module-path "\path\to\javafx-sdk-17\lib" --add-modules javafx.controls,javafx.fxml

On la coller dans : Run -> Edit Configurations -> VM options.

IV. Développement du Backend avec JavaFX et ArrayList

1. Création des modèles (Java Classes)

Pour la version JavaFX avec ArrayList, les modèles sont implémentés comme des classes Java standard:

- **Owner:** Classe représentant les propriétaires d'animaux avec attributs comme nom, prénom, adresse, téléphone, email.
- **Pet:** Classe pour les animaux de compagnie avec attributs tels que nom, espèce, race, date de naissance, sexe, poids et référence au propriétaire.
- **Visit:** Classe pour les visites médicales contenant date, raison, diagnostic, traitement et références à l'animal et au vétérinaire.
- **Veterinarian:** Classe pour les vétérinaires avec nom, spécialité, et coordonnées.

Ces classes incluent des méthodes getters/setters et des constructeurs appropriés pour l'encapsulation des données.

2. Implémentation des contrôleurs

En JavaFX, les contrôleurs gèrent l'interaction entre l'interface utilisateur et les données:

- HomeController: Gère la page d'accueil et la navigation principale.
- NavbarController: Contrôleur pour la barre de navigation commune.
- FindOwnersController: Gère la recherche de propriétaires par nom.
- OwnerFormController: Traite l'ajout et la modification des propriétaires.
- OwnerListController: Affiche la liste des propriétaires.
- OwnerDetailsController: Présente les détails d'un propriétaire et ses animaux.

- PetFormController: Gère l'ajout et la modification des animaux.
- PetListController: Affiche la liste des animaux.
- VisitFormController: Traite l'ajout des visites médicales.
- AppointmentsViewController: Gère la visualisation des rendez-vous.
- VeterinarianFormController: Pour l'ajout et la modification des vétérinaires.
- VeterinarianListController: Affiche la liste des vétérinaires.

3. Définition des services de données

La gestion des données est centralisée dans un service qui manipule les ArrayList:

- DataService: Service central qui:
  - Maintient des ArrayList pour chaque type d'entité (owners, pets, visits, veterinarians)

- Fournit des méthodes CRUD pour chaque entité
- Gère les relations entre les objets
- Implémente des méthodes de recherche et de filtrage
- Assure la persistance des données via sérialisation/désérialisation

Exemple de fonctionnalités:

- dataService.addOwner(firstName, lastName, phone, email, address);
- veterinarians.add(vet);
- getPetsForOwner(Owner owner)
- addVisitToPet(Pet pet, Visit visit)

public Visit findVisitById(int id)

4. Tests des fonctionnalités

Pour tester l'application avec JavaFX et ArrayList:

- **Tests unitaires:** Validation des modèles et de la logique métier
- **Tests d'intégration:** Vérification des relations entre objets et de la persistance des données
- **Tests d'interface:** Validation des interactions utilisateur et de l'affichage
- **Tests de performances:** Évaluation des temps de réponse avec des volumes de données variables
- **Tests de scénarios:** Simulation de flux utilisateur complets (ex: ajout d'un propriétaire, puis de son animal, puis d'une visite)

V. Développement du Frontend

1. Structure du projet JavaFX

La structure du frontend de l'application VetCare 360 s'organise autour de l'architecture MVC (Modèle-Vue-Contrôleur) de JavaFX:

- Structure des ressources:
  - **Views (FXML):** Fichiers de définition des interfaces utilisateurs
  - **Images:** Ressources graphiques, logos et icônes
  - **Styles:** Feuilles de style CSS pour personnaliser l'apparence
- Composants réutilisables:
  - Navigation commune à travers l'application
  - Formulaires standardisés (saisie des animaux, propriétaires, visites)

    - Tableaux de données avec fonctionnalités de tri et filtrage
- Internationalisation:
    - Support multilingue via fichiers de ressources
    - Adaptation automatique des formats de date et nombres

2. Pages principales et leurs fonctionnalités

- **Page d'accueil:** Dashboard avec raccourcis vers les fonctions principales et statistiques
- Liste des vétérinaires: Affichage de tous les vétérinaires disponibles
    - Interface de visualisation simple
    - Détails sur spécialités et disponibilités
- Recherche de propriétaires:
    - Champ de recherche par nom de famille
    - Résultats affichés en temps réel

    - Bouton pour ajouter un nouveau propriétaire
- Formulaire de propriétaire: Interface pour ajouter/modifier les informations des propriétaires
    - Validation des champs en temps réel
    - Gestion des erreurs intégrée
- Détails du propriétaire:
    - Affichage des informations personnelles
    - Liste des animaux associés
    - Historique des visites pour chaque animal
    - Options pour modifier le propriétaire ou ajouter un animal
- Formulaire d'animal: Interface pour ajouter/modifier les informations des animaux
    - Champs adaptés aux différentes espèces
    - Association automatique au propriétaire
- Détails de l'animal:

    - Informations complètes sur l'animal
    - Historique médical chronologique
    - Option pour ajouter une nouvelle visite
- Formulaire de visite: Interface pour enregistrer une nouvelle consultation
    - Sélection du vétérinaire
    - Champs pour diagnostic et traitement
    - Affichage des visites précédentes pour référence

VI. Technologies Utilisées

1. Frontend:

- **JavaFX:** Framework d'interface graphique moderne pour Java
- **FXML:** Langage de description d'interface basé sur XML
- **CSS:** Personnalisation avancée de l'apparence des composants
- Scene Builder: Outil de conception visuelle pour les interfaces FXML

2. Backend: Java avec services de données

- Java 17+: Langage de programmation moderne avec fonctionnalités avancées
- JavaFX Controllers: Gestion de la logique d'interface utilisateur
- EventHandlers: Gestion des événements utilisateur et de l'interface

3. Base de données: ArrayList

- Collections Java: Utilisation d'ArrayList pour stocker les données en mémoire
- Serialization/Deserialization: Persistance des données entre les sessions
- Relations d'objets: Mise en œuvre des associations entre entités
- Data service patterns: Implémentation des services d'accès aux données

4. Gestion des versions: Git/GitHub

- Versioning de code: Suivi des modifications et collaboration
- Branches de développement: Organisation du travail par fonctionnalités
- Pull requests: Revue de code et intégration
- Issues tracking: Suivi des problèmes et améliorations

VII. Architecture de l'Application

1. Architecture JavaFX

L'architecture de VetCare 360 s'appuie sur les principes MVC (Modèle-Vue-Contrôleur) de JavaFX:

- **Modèle:** Classes Java représentant les entités métier (Owner, Pet, Visit, Veterinarian)
- **Vue:** Fichiers FXML définissant l'interface utilisateur
- **Contrôleur:** Classes Java gérant les interactions entre les vues et les modèles

Composants architecturaux:

- **Application principale:** VetCare360App.java initialise l'application et charge la scène principale
- **Gestionnaire de scènes:** Contrôle la navigation entre les différentes vues

- DataService: Service centralisé de gestion des données avec pattern Singleton
- Contrôleurs spécifiques: Un contrôleur dédié pour chaque vue/fonctionnalité

2. Flux de données

Le flux de données dans VetCare 360 suit un cycle clair:

1. **Saisie utilisateur:** Interaction avec les formulaires et contrôles JavaFX
2. **Traitement événementiel:** Les contrôleurs captent les événements et les traitent
3. Manipulation des données:
    - Les contrôleurs appellent les méthodes du DataService
    - Le DataService modifie les ArrayList appropriées

4. Mise à jour de l'interface:

- Les contrôleurs mettent à jour les composants d'affichage (TableView, ListView, etc.)
- Utilisation de l'observation de propriétés JavaFX pour les mises à jour dynamiques

VIII. Base de Données (ArrayList)

1. Structure des collections (ArrayList)

Au lieu de collections MongoDB, le système utilise des ArrayList typées:

- ArrayList<Owner>: Stockage des propriétaires d'animaux
- ArrayList<Pet>: Stockage des animaux de compagnie
- ArrayList<Visit>: Stockage des visites médicales
- ArrayList<Veterinarian>: Stockage des vétérinaires

Chaque ArrayList est gérée par le DataService central.

2. Relations entre les class

Les relations sont implémentées par des références d'objets en mémoire:

- **Owner → Pet:** Un propriétaire peut avoir plusieurs animaux (relation 1:n)
- **Pet → Owner:** Un animal appartient à un seul propriétaire (relation n:1)
- **Pet → Visit:** Un animal peut avoir plusieurs visites médicales (relation 1:n)
- **Visit → Pet:** Une visite concerne un seul animal (relation n:1)
- **Visit → Veterinarian:** Une visite est effectuée par un vétérinaire (relation n:1)

3. Exemple de données initiales

Le système est préchargé avec des données de démonstration:

- Propriétaires: 5-10 propriétaires avec informations complètes
- Animaux: 10-15 animaux répartis entre les propriétaires

- Vétérinaires: 3-5 vétérinaires avec différentes spécialités

IX. Développement du Backend

1. Création des modèles (Classes Java)

Définition des classes de modèle avec attributs, relations et comportements:

- **Owner.java:**
  - **Attributs: id, firstName, lastName, phone, email, address**
  - **Méthodes: getters/setters, getFullName()**
- **Pet.java:**
  - **Attributs: id, name, species, breed, age, ownerName**
  - **Méthodes: getters/setters**
- **Visit.java:**
  - **Attributs: id, petName, vetName, dateTime, reason, diagnosis**
  - **Méthodes: getters/setters**
- **Veterinarian.java:**

- Attributs: id, firstName, lastName, specialization, phone, email
- Méthodes: getters/setters, toString(),getFullName()

2. Implémentation des services de données

Développement du DataService central:

- DataService.java:
  - ArrayLists pour chaque type d'entité
  - Méthodes CRUD complètes pour chaque entité
  - Fonctions de recherche et filtrage
  - Gestion des relations entre objets
  - Mécanismes de persistance

3. Implémentation des contrôleurs

Les contrôleurs JavaFX gèrent l'interaction avec l'utilisateur:

- HomeController.java: Gestion de la page d'accueil

- OwnerController.java: Gestion des propriétaires (liste, ajout, modification)
- PetController.java: Gestion des animaux (ajout, modification)
- VisitController.java: Gestion des visites médicales
- VeterinarianController.java: Gestion des vétérinaires

4. Tests des fonctionnalités

Approche de test complète:

- Tests unitaires: Validation des modèles et des services
- Tests d'intégration: Test des flux complets
- Tests manuels: Validation de l'interface utilisateur
- Tests de charge: Comportement avec un grand nombre d'entités

X. Développement du Frontend

1. Structure du projet JavaFX

Organisation des ressources et composants:

- /resources:
  - /views: Fichiers FXML définissant les interfaces
  - /styles: Feuilles de style CSS
  - /images: Ressources graphiques
- /controllers: Classes contrôleur JavaFX
- /models: Classes de modèle
- /utils: La class VetCare360App.java qui contient data

2. Pages principales et leurs fonctionnalités

- Page d'accueil (Home.fxml):
  - Dashboard avec accès rapide aux fonctionnalités

- Affichage des statistiques (nombre de propriétaires, animaux, visites)
- Navigation principale vers les autres sections

- Liste des vétérinaires (VeterinarianList.fxml):
  - TableView affichant tous les vétérinaires
  - Informations: nom, spécialité, contact
  - Tri et filtrage par différents critères
- Recherche propriétaire (FindOwners.fxml):
  - Formulaire de recherche par nom
  - Affichage des résultats en temps réel
  - Bouton d'ajout de nouveau propriétaire
- Formulaire propriétaire (Ownerlist.fxml):
  - Champs pour toutes les informations du propriétaire
  - Validation des données saisies
  - Boutons de sauvegarde ou annulation

- Détails propriétaire (OwnerDetails.fxml):
  - Informations complètes du propriétaire
  - Liste de ses animaux
  - Boutons pour modifier les informations ou ajouter un animal
- Formulaire animal (NewPetForm.fxml):
  - L'ajoute d'un animal
- Détails animal et visites (PetList.fxml):
  - Informations complètes sur l'animal
  - Historique chronologique des visites
- Formulaire visite (AddVisitForm.fxml):
  - L'ajoute d'une visite
  - Sélection du vétérinaire
  - Champs pour diagnostic et traitement
- Détails visite (AppointmentsView.fxml):
  - Affichage des visites précédentes en référence

XI. Cahier des charges

1. Présentation générale du projet

VetCare 360 est une application desktop destinée à la gestion d'une clinique vétérinaire. Cette solution logicielle, développée en Java avec JavaFX, vise à optimiser et centraliser les processus administratifs et médicaux liés aux soins animaliers. L'application s'adresse principalement au personnel des cliniques vétérinaires, incluant les vétérinaires, les assistants et le personnel administratif.

2. Fonctionnalités requises

Gestion des propriétaires

- Enregistrement des informations complètes
- Recherche de propriétaires par nom.
- Modification et mise à jour des informations personnelles
- Visualisation de la liste des propriétaires
- Association avec leurs animaux de compagnie

Gestion des animaux

- Enregistrement des informations détaillées.

- Association d'un animal à son propriétaire
- Suivi de l'état de santé général
- Historique médical complet
- Modification des informations de l'animal

Gestion des visites médicales

- Programmation de nouvelles visites
- Enregistrement des diagnostics et traitements prescrits
- Documentation des observations médicales

  Consultation de l'historique des visites par animal
- Synthèse des visites antérieures

Gestion des vétérinaires

- Affichage de la liste des vétérinaires de la clinique
- Consultation des informations de contact et spécialités

Contraintes techniques

Technologies imposées

- Langage de programmation : Java

- Interface graphique : JavaFX
- Gestion des dépendances : Maven
- IDE recommandé : IntelliJ IDEA
- Architecture : Modèle-Vue-Contrôleur (MVC)

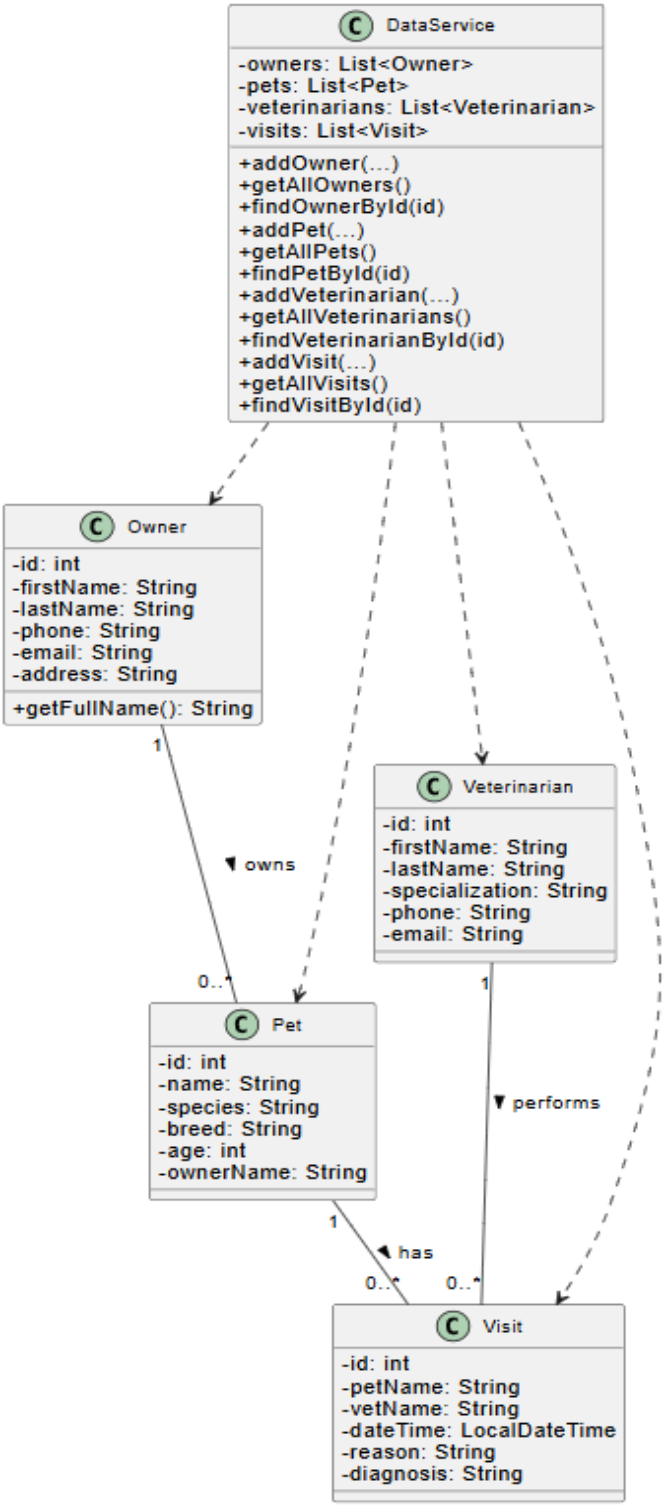
3. Structure du projet

- Organisation en packages distincts pour le modèle, les vues et les contrôleurs
- Séparation claire des responsabilités entre les différentes couches
- Documentation du code selon les conventions Java
- Utilisation des principes SOLID pour une conception orientée objet robuste

4. Interfaces utilisateur

- Design moderne et épuré utilisant les capacités de styling de JavaFX
- Formulaires de saisie avec validation des données
- Tableaux interactifs pour l'affichage des listes
- Messages de confirmation et d'erreur appropriés

XII. DIAGRAMME DE CLASS



XIII. Développement Technique / La Stratégie

1. Architecture MVC

L'application VetCare 360 sera développée selon le modèle d'architecture MVC (Modèle-Vue-Contrôleur) pour assurer une séparation claire des responsabilités et faciliter la maintenance :

Couche Modèle

- **Classes d'entité** : Proprietaire, Animal, Visite, Veterinaire
- **Classes de service** : GestionProprietaire, GestionAnimal, GestionVisite, GestionVeterinaire

Couche Vue

- Interfaces FXML pour chaque écran de l'application
- Styles CSS pour personnaliser l'apparence
- Elements JavaFX (TableView, TextField, Button, etc.)

Couche Contrôleur

- Contrôleurs JavaFX pour chaque vue FXML
- Gestion des événements utilisateur

- Communication entre l'interface utilisateur et les services métier

XIII. PLANIFICATION DE PROJET

La réalisation de l'application VetCare 360 a nécessité une planification rigoureuse pour assurer la coordination efficace des différentes phases de développement et respecter les délais impartis. Nous avons adopté une approche méthodique, divisée en plusieurs étapes clés, pour structurer notre travail et optimiser notre progression.

1. Phases du projet

- Analyse et conception

Étude approfondie du cahier des charges

- Identification des fonctionnalités essentielles
- Modélisation et Élaboration du domaine avec diagrammes UML
- Conception de l'architecture logicielle

- Mise en place de l'environnement

- Installation des outils de développement (JDK, JavaFX, Maven)
- Configuration de l'IDE IntelliJ IDEA

- Création du projet Maven avec les dépendances requises
- Configuration du système de contrôle de version Git

- Développement

- Développement des interfaces FXML pour chaque page d'application
- Programmation des contrôleurs JavaFX

Validation des fonctionnalités par rapport au cahier des charges .

- Finalisation et documentation

- Optimisation des performances
- Amélioration de l'interface utilisateur
- Préparation du rapport final
- Création du support pour la présentation du projet

2. Répartition des tâches

ASTITOU Fatima Zzahrae :

- Responsable de la conception du modèle de données

Développement des fonctionnalités de gestion des propriétaires et des animaux

- Implémentation des interfaces utilisateur principales
- Coordination générale du projet

AIT ADDI Tithrit :

- Responsable de l'architecture technique
- Développement des fonctionnalités de gestion des visites et des vétérinaires
- Implémentation du système de recherche

Cette planification structurée nous a permis de maintenir une progression constante tout au long du développement du projet, en assurant une communication efficace entre les membres de l'équipe et une gestion optimale des ressources disponibles.

XV. Conclusion

Le projet VetCare 360 représente une application complète et fonctionnelle dédiée à la gestion d'une clinique vétérinaire, développée dans le cadre de notre formation en Ingénierie Logicielle et Cybersécurité. Ce projet nous a permis de mettre en pratique les concepts fondamentaux de la programmation orientée objet en Java, tout en explorant les capacités de JavaFXpour créer des interfaces utilisateur modernes et intuitives.

À travers ce développement, nous avons réussi à implémenter les fonctionnalités principales demandées, notamment la gestion des propriétaires, des animaux, des visites médicales, et des vétérinaires. L'utilisation d'ArrayList pour la gestion des données en mémoire a été un choix judicieux pour simplifier l'architecture tout en garantissant une manipulation efficace des données. L'approche MVC (Modèle-Vue-Contrôleur) a permis une séparation claire des responsabilités, facilitant ainsi la maintenance et l'évolutivité de l'application.

Ce projet a été une opportunité précieuse pour renforcer nos compétences en développement logiciel, en gestion de projet, et en travail d'équipe. Les défis rencontrés,

tels que la gestion des relations entre les entités ou l'optimisation des interfaces utilisateur, ont été autant d'occasions d'apprentissage et d'amélioration.

Enfin, nous tenons à remercier une fois de plus notre encadrant, Monsieur Esbai Redouane, pour son soutien, ses conseils avisés, et son accompagnement tout au long de ce projet. Son expertise a été déterminante dans la réussite de cette réalisation.