**Rapport de Projet**

**Sous thème**

**« Vet-care-360»**

Filière : ILCS

Nom et prénom :

Mohamed Amine Ouazzou,

Soulayman Amadid

Encadre par :

Prof. Redouane Esbaie .

1. ***Remerciment :***

Nous tenons à remercier chaleureusement Monsieur Redouan Esbai pour son accompagnement et son soutien tout au long de la réalisation de ce projet. Sa disponibilité, ses conseils pertinents et sa patience ont été d’une grande aide pour notre équipe. Nous lui sommes reconnaissants pour l’encadrement rigoureux et bienveillant qu’il nous a apporté.

1. ***Introduction :***

Ce rapport détaille la conception et le développement de **VetCare**, une application destinée à simplifier et optimiser la gestion des soins vétérinaires. Grâce à une interface graphique conviviale développée avec **JavaFX** , VetCare permet de centraliser et d’organiser efficacement les informations relatives aux vétérinaires, aux propriétaires d’animaux, aux animaux eux-mêmes, ainsi qu’aux visites médicales. Ce projet, réalisé dans le cadre de notre projet universitaire encadres par monsieur Redouane Esbai ,répond à un besoin réel de gestion fluide et fiable dans les cabinets vétérinaires. L’objectif principal est de proposer une solution complète qui facilite le suivi des patients, améliore la planification des consultations, et offre un accès rapide aux données essentielles, tout en garantissant une interface intuitive et facile à utiliser. Ce rapport présente ainsi les choix techniques effectués, les étapes de développement, les résultats obtenus ainsi que les perspectives d’amélioration.

***III)REALISATION TECHNIQUE :***

***Java*** :

JAVA est un langage de programmation orienté objet, conçu pour être portable, robuste et sécurisé. Pour utiliser Java, plusieurs composants essentiels interviennent, chacun jouant un rôle clé dans le développement et l’exécution des applications.

* JDK (Java Development Kit) : c’est un ensemble complet d’outils nécessaires pour développer des applications Java. Il inclut le compilateur Java (javac), la machine virtuelle Java (JVM), des bibliothèques standard, ainsi que des outils pour déboguer et documenter le code. Le JDK est utilisé par les développeurs pour écrire, compiler et tester leurs programmes.
* JRE (Java Runtime Environment) : c’est un environnement d’exécution qui permet d’exécuter les applications Java. Il contient la JVM ainsi que les bibliothèques nécessaires pour faire fonctionner les programmes, mais il ne comprend pas les outils de développement comme le compilateur.
* JVM (Java Virtual Machine) : c’est la machine virtuelle qui interprète le bytecode Java (code compilé) et le transforme en instructions compréhensibles par le système d’exploitation. La JVM garantit la portabilité des applications Java, car elle adapte l’exécution au système sur lequel elle tourne.
* Bytecode Java : c’est un code intermédiaire généré par le compilateur Java à partir du code source. Le bytecode est indépendant de la plateforme et est exécuté par la JVM, ce qui permet la portabilité.

Ces composants forment ensemble l’écosystème Java qui facilite le développement, la compilation, la distribution et l’exécution d’applications multiplateformes.

Dans notre projet VetCare, nous utilisons le JDK pour développer l’application, la JVM pour exécuter le programme, et le JRE est ce que les utilisateurs finaux doivent avoir installé pour lancer l’application.

***1)STOCKAGE DE DONNEE :***

Dans le projet **VetCare360**, aucune base de données externe (comme MySQL ou PHPMyAdmin) n’a été utilisée. À la place, les données sont **stockées en mémoire** à l’aide de **structures de données Java**, principalement des **ArrayList** et parfois des **HashMap**, selon les besoins.

Ce choix a été fait pour simplifier l’architecture et se concentrer sur la logique métier et l’interface graphique. Il permet également de simuler une base de données relationnelle de manière légère sans configuration serveur.

Les principales structures de données sont les suivantes :

* **Liste des vétérinaires** : un tableau ou une liste d’objets Veterinaire contenant des informations comme le nom, prénom, spécialité et téléphone.
* **Liste des propriétaires** : une ArrayList<Proprietaire> pour stocker les identifiants, noms, prénoms, adresses, etc.
* **Liste des animaux** : chaque animal est représenté par un objet Animal contenant nom, espèce, race, date de naissance, et une référence vers son propriétaire.
* **Liste des visites médicales** : chaque visite contient la date, les observations, et des références vers le vétérinaire et l’animal concernés.

Même sans base de données réelle, ces **relations logiques entre objets** ont été soigneusement gérées dans le code. Par exemple, un Animal contient une référence vers son Proprietaire, et une Visite contient une référence vers un Animal et un Veterinaire, simulant ainsi les **relations entre tables** d’une base SQL classique.

Cette approche en mémoire permet d’obtenir un système fonctionnel, sans installation de serveur, idéal pour un projet pédagogique ou une application de démonstration.

***2)STOCKAGE DE DONNES ET ACCESSANS JDBC:***

Dans le cadre du projet **VetCare360**, aucune base de données externe ni connexion SQL n’a été utilisée. Par conséquent, l’API **JDBC (Java Database Connectivity)** n’est pas impliquée dans ce projet. À la place, toutes les données sont **stockées localement en mémoire**, à l’aide de **structures de données Java**, telles que des **ArrayList**, pour simuler les différentes entités manipulées (vétérinaires, propriétaires, animaux, visites médicales).

Ce choix permet de se concentrer sur la **logique métier** et la **gestion de l’interface utilisateur** avec JavaFX, sans complexifier le projet avec la configuration d’une base de données externe ou l’écriture de requêtes SQL.

L’organisation des données s’appuie sur une séparation claire entre :

* Des **classes entités** (Veterinaire, Proprietaire, Animal, VisiteMedicale) représentant les objets manipulés dans l’application, avec leurs attributs et méthodes.
* Des **structures de stockage internes** (par exemple : ArrayList<Veterinaire> ou ArrayList<Animal>) servant à contenir les objets en mémoire.
* Des **méthodes de gestion** dans la classe principale (main) ou dans des gestionnaires utilitaires, permettant d’effectuer les opérations CRUD (ajouter, modifier, supprimer, consulter).

Bien que cette approche soit limitée à une exécution locale sans persistance des données après la fermeture de l’application, elle reste parfaitement adaptée à un projet pédagogique. Elle permet de **modéliser les relations entre entités** (par exemple : un animal appartient à un propriétaire, une visite est liée à un vétérinaire) et de manipuler les données de manière dynamique.

**Pourquoi ne pas utiliser JDBC ici ?**

JDBC permet de connecter une application Java à une base de données relationnelle comme MySQL. Mais cela nécessite :

* Un serveur de base de données configuré (MySQL, PostgreSQL, etc.)
* Des requêtes SQL à écrire et à tester
* Une gestion manuelle des connexions, exceptions, transactions, etc.

Dans ce projet, l’objectif était avant tout de maîtriser :

* La programmation orientée objet
* La gestion de données dynamiques avec des collections (ArrayList)
* L'interaction avec une interface utilisateur via **JavaFX**

**Perspectives d’évolution**

Dans une version future de VetCare360, l’utilisation de JDBC ou d’un framework comme **Hibernate** ou **JPA** pourrait être envisagée pour **permettre la persistance réelle des données**, c’est-à-dire les conserver entre deux utilisations de l’application. Cela offrirait aussi la possibilité de travailler avec une base centralisée et multi-utilisateurs.

***2)JAVAFX:***

**JavaFX : Présentation et avantages par rapport à Swing**

**JavaFX** est une plateforme moderne développée par Oracle pour créer des applications graphiques riches (RIA - Rich Internet Applications) en Java, principalement destinées aux applications desktop mais également compatibles avec d’autres plateformes. Lancée pour succéder à **Swing**, qui est une bibliothèque graphique plus ancienne, JavaFX offre un cadre plus puissant, flexible et ergonomique pour concevoir des interfaces utilisateurs avancées.

**Caractéristiques et atouts de JavaFX :**

* **Interface moderne et attrayante** : JavaFX permet de créer des interfaces graphiques esthétiques grâce à son support natif des animations, effets visuels, transitions fluides et styles personnalisables via CSS. Cette capacité permet aux développeurs de concevoir des applications avec une expérience utilisateur enrichie, bien plus difficile à obtenir avec Swing.
* **FXML et Scene Builder** : JavaFX propose un langage déclaratif nommé FXML, basé sur XML, qui sépare clairement la conception graphique du code Java. Cela favorise la modularité et la collaboration entre développeurs et designers. L’outil **Scene Builder** permet de construire visuellement l’interface en drag-and-drop, ce qui accélère le développement et réduit les erreurs. Swing ne dispose pas d’un tel système intégré, rendant la conception d’interface plus laborieuse.
* **Support multimédia avancé** : JavaFX intègre facilement la gestion audio, vidéo, et les effets graphiques complexes. Cette intégration native facilite la création d’applications multimédia ou interactives, ce qui est beaucoup plus limité et compliqué à réaliser avec Swing.
* **Meilleure intégration avec les technologies modernes** : JavaFX est conçu pour fonctionner harmonieusement avec les frameworks Java récents, les bases de données via JDBC, et même le développement web avec des possibilités d’intégration à des moteurs web modernes. Swing, quant à lui, est plus ancien et moins flexible dans ce contexte.
* **Architecture évolutive et supportée** : JavaFX bénéficie d’un développement actif et d’une communauté croissante, ce qui assure une meilleure pérennité et des mises à jour régulières. Swing est aujourd’hui considéré comme stable mais en maintenance, sans évolution majeure prévue, ce qui peut poser problème pour des projets à long terme.
* **Facilité d’apprentissage et productivité accrue** : Grâce à ses outils (FXML, Scene Builder) et son approche plus intuitive, JavaFX permet aux développeurs de gagner en productivité et de réduire la complexité du code graphique. Le code JavaFX est souvent plus propre et plus facile à maintenir que le code Swing.

**En résumé**

JavaFX représente la solution recommandée pour le développement d’applications desktop modernes, offrant une interface utilisateur riche, des outils puissants et une meilleure intégration avec l’écosystème Java actuel. Pour un nouveau projet, JavaFX est souvent préféré à Swing, qui reste toutefois pertinent pour maintenir d’anciennes applications mais est moins adapté aux besoins actuels d’interactivité et de design.

Scene Builder est un outil graphique développé par Oracle qui facilite la création d’interfaces utilisateur pour les applications JavaFX. Il permet de concevoir visuellement des fenêtres, formulaires, boutons, tableaux et autres composants graphiques, sans écrire directement le code Java.

Avec Scene Builder, l’interface est construite en mode drag-and-drop (glisser-déposer) : il suffit de sélectionner les éléments souhaités dans une palette et de les placer sur la scène de conception. On peut également configurer leurs propriétés (taille, couleur, texte, événements) via des panneaux de paramètres. Le fichier de sortie est un fichier FXML, un format XML spécifique à JavaFX qui décrit la structure et les composants de l’interface.

L’utilisation de Scene Builder présente plusieurs avantages :

* Séparation claire entre design et logique : grâce au fichier FXML, la conception graphique est séparée du code Java qui gère la logique métier. Cela facilite la collaboration entre développeurs et designers.
* Gain de temps : créer une interface graphique à la main en Java peut être fastidieux et source d’erreurs. Scene Builder simplifie cette tâche et accélère le développement.
* Facilité d’apprentissage : même les développeurs débutants peuvent créer des interfaces complexes sans maîtriser toutes les subtilités du code JavaFX.
* Compatibilité : les fichiers FXML générés par Scene Builder sont directement utilisables dans une application JavaFX en les chargeant via le chargeur FXML.

En résumé, Scene Builder est un outil précieux qui rend la création d’interfaces JavaFX plus accessible, rapide et organisée, tout en améliorant la qualité et la maintenabilité du code.

Bien que JavaFX soit aujourd’hui le framework recommandé pour créer des interfaces graphiques modernes en Java, il existe plusieurs autres options que l’on peut envisager selon les besoins et contraintes du projet :

* Swing  
  Swing est la bibliothèque graphique historique de Java, largement utilisée depuis les années 1990. Elle offre un ensemble complet de composants graphiques (boutons, menus, tableaux, etc.) et permet de créer des applications desktop multiplateformes. Swing est stable et bien documenté, mais son design est parfois considéré comme vieillissant et moins moderne que JavaFX. Il ne supporte pas nativement les animations et styles CSS comme JavaFX.
* SWT (Standard Widget Toolkit)  
  Développé par Eclipse Foundation, SWT est un framework graphique qui utilise les composants natifs du système d’exploitation, ce qui garantit un rendu plus « natif » et performant. SWT est souvent utilisé pour créer des applications lourdes et réactives, comme l’IDE Eclipse lui-même. Cependant, il est moins portable que JavaFX et nécessite d’intégrer des bibliothèques natives spécifiques à chaque plateforme.
* Apache Pivot  
  Moins connu, Apache Pivot est un framework open source pour créer des applications Java desktop avec une approche basée sur XML et Java. Il propose des composants riches et une bonne gestion des styles, mais sa communauté est plus réduite et le projet moins actif comparé à JavaFX.
* Griffon  
  Inspiré de Grails (framework web), Griffon est un framework qui facilite le développement rapide d’applications desktop Java en combinant JavaFX, Swing, ou SWT avec une architecture MVC. Il simplifie la création d’applications complexes, notamment via des scripts Groovy, mais demande une certaine courbe d’apprentissage.

***IIII)*** ***Perspectives d’amélioration et difficultés :  :***

**🛠️ Difficultés rencontrées**

Lors du développement de l’application **VetCare360** en **Java avec JavaFX**, plusieurs défis ont été rencontrés. Le principal concernait la **gestion manuelle des données** sans base externe, en s’appuyant uniquement sur des structures telles que les ArrayList pour simuler une base de données. Cela a nécessité une rigueur particulière dans la gestion des relations entre objets (par exemple, associer correctement un animal à son propriétaire ou une visite à un vétérinaire).

L’autre difficulté résidait dans la conception des interfaces **entièrement en code Java**, sans passer par des outils visuels comme Scene Builder. Cela impliquait de gérer manuellement les composants graphiques (VBox, GridPane, TextField, Button, etc.), leur positionnement et leur style, ce qui augmentait la charge de travail et le risque d’erreurs.

Enfin, maintenir la cohérence de l’interface après des opérations d’ajout, de suppression ou de modification nécessitait de rafraîchir manuellement les composants, ce qui demandait une bonne organisation du code.

**🔭 Perspectives d’amélioration**

Pour améliorer la qualité, la maintenabilité et les fonctionnalités de l’application VetCare360, plusieurs pistes peuvent être envisagées :

* **Intégrer une base de données réelle** (comme SQLite ou MySQL) via **JDBC**, afin de permettre la **sauvegarde persistante** des données et leur réutilisation entre les sessions.
* Adopter un **modèle MVC** clair pour séparer les responsabilités entre les données (modèles), l’interface graphique (vues) et la logique de traitement (contrôleurs ou gestionnaires).
* Ajouter des fonctionnalités avancées, comme :
  + La **recherche dynamique** de propriétaires ou d’animaux,
  + Un **système de rappels** ou notifications pour les visites à venir,
  + L’**export des données** (ex : rapports PDF, tableaux CSV),
  + Une **pagination** pour faciliter l’affichage de longues listes.
* Améliorer l’ergonomie en utilisant **CSS avec JavaFX** pour moderniser le design et améliorer l’expérience utilisateur (UI/UX).

Ces évolutions rendraient l’application **plus robuste**, **plus professionnelle** et **plus proche des besoins réels** d’un cabinet vétérinaire.

***IIIII) Conclusion :***

En résumé, le projet **VetCare360** a permis de développer une application de bureau complète et fonctionnelle pour la gestion des soins vétérinaires, en utilisant **Java** et **JavaFX**, avec un stockage des données en mémoire via des **structures comme ArrayList**. L’objectif principal centraliser la gestion des propriétaires, animaux, vétérinaires et visites a été atteint à travers une interface claire et des fonctionnalités bien définies.

Malgré certaines difficultés liées à la gestion manuelle des données et à la création d’interfaces entièrement en JavaFX sans outil graphique, ce projet a offert une solide expérience pratique en **programmation orientée objet**, en **organisation logicielle** et en conception d’interfaces utilisateur.

Les perspectives d’amélioration identifiées comme l’intégration future d’une base de données via JDBC ou Hibernate, ou encore l’optimisation de l’ergonomie permettront de rendre l’application encore plus complète et plus professionnelle. Ce projet constitue une **base stable et extensible**, sur laquelle pourront s’ajouter de nouvelles fonctionnalités pour répondre plus largement aux besoins des structures vétérinaires.