**Rapport de Projet**

**Sous thème**

**« VITECARE »**

Filière : ILCS

Nom et prénom :

Reda Elbikri,

Yahya Es-salhi

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** et une base de données relationnelle **MySQL**, 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)BASE DE DONNEE :***

La base de données utilisée dans le projet VetCare est une base de données relationnelle SQL, implémentée avec **PHPMYADMIN** . Ce choix s’explique par la robustesse, la flexibilité et la facilité d’intégration qu’offre SQL, notamment grâce au langage de requête structuré qui permet de manipuler efficacement les données. La base est structurée autour de plusieurs tables principales, chacune représentant une entité clé du système.

Les principales tables sont les suivantes :

* **Vétérinaires** : cette table contient les informations personnelles et professionnelles des vétérinaires, telles que leur identifiant unique, nom, prénom, spécialité, et coordonnées.
* **Propriétaires** : elle regroupe les données des propriétaires d’animaux, incluant un identifiant, nom, prénom, adresse et numéro de téléphone.
* **Animaux** : cette table enregistre les informations spécifiques à chaque animal, comme son identifiant, nom, espèce, race, date de naissance, et elle contient une clé étrangère référant à la table Propriétaires pour associer chaque animal à son propriétaire.
* **Visites médicales** : cette table enregistre les consultations vétérinaires, avec des champs pour l’identifiant de la visite, la date, les observations, ainsi que des clés étrangères vers les tables Animaux et Vétérinaires pour relier chaque visite à l’animal concerné et au vétérinaire qui a effectué la consultation.

Ces tables sont liées entre elles par des relations de type **clé primaire / clé étrangère** qui assurent l’intégrité référentielle des données. Par exemple, la relation entre la table Animaux et Propriétaires garantit qu’un animal ne peut être associé qu’à un propriétaire existant. De même, la table Visites médicales fait le lien entre les vétérinaires et les animaux lors des consultations, ce qui permet de suivre précisément l’historique des soins.

Cette architecture relationnelle facilite la réalisation de requêtes complexes pour extraire des informations détaillées, comme la liste des visites d’un animal ou les patients suivis par un vétérinaire spécifique. Elle assure également la cohérence et la sécurité des données, tout en permettant une intégration fluide avec l’application JavaFX via JDBC.

***1)JDBC:***

Dans le cadre du projet VetCare, la communication entre l’application Java et la base de données MySQL est assurée par **JDBC** (Java Database Connectivity). JDBC est une API standard de Java qui permet aux programmes Java d’interagir avec des bases de données relationnelles via des requêtes SQL.

JDBC joue un rôle central en fournissant un pont entre la couche applicative développée avec JavaFX et la base de données SQL. Grâce à JDBC, il est possible d’établir une connexion sécurisée à la base, d’exécuter des commandes SQL pour créer, lire, mettre à jour ou supprimer des données (opérations CRUD), et de récupérer les résultats pour les afficher dans l’interface utilisateur.

L’architecture JDBC repose sur plusieurs composants essentiels :

* **Driver JDBC** : un pilote spécifique à la base de données utilisée (ici MySQL) qui traduit les appels Java en commandes compréhensibles par la base.
* **Connection** : représente la connexion active entre l’application Java et la base de données. Elle doit être ouverte avant d’envoyer des requêtes et fermée après usage pour libérer les ressources.
* **Statement / PreparedStatement** : objets utilisés pour exécuter les requêtes SQL. Le PreparedStatement est recommandé car il permet de préparer des requêtes paramétrées, améliorant la sécurité (prévention des injections SQL) et les performances.
* **ResultSet** : objet contenant les résultats d’une requête SELECT, qui peut être parcouru pour extraire les données nécessaires.

Dans VetCare, JDBC est utilisé pour gérer toutes les interactions avec la base de données : insertion de nouvelles visites, mise à jour des informations des animaux, récupération des listes de propriétaires ou vétérinaires, etc. L’utilisation de JDBC permet d’abstraire la complexité des échanges avec la base et de garantir une bonne gestion des transactions et des erreurs.

Par ailleurs, une attention particulière a été portée à la gestion des exceptions SQL afin d’assurer la stabilité de l’application en cas de problème lors des accès à la base. De même, la fermeture correcte des connexions et des ressources est systématiquement appliquée pour éviter les fuites mémoire et les blocages.

Enfin, l’intégration de JDBC avec JavaFX facilite la mise à jour dynamique des interfaces graphiques en fonction des données présentes dans la base, offrant ainsi une expérience utilisateur fluide et réactive.

Pour organiser efficacement le code et faciliter la maintenance, le projet VetCare est structuré en deux dossiers principaux : **entities** et **dao**.

Le dossier **entities** contient les classes représentant les différentes entités du domaine métier, telles que Veterinaire, Proprietaire, Animal ou VisiteMedicale. Ces classes modélisent les données stockées en base sous forme d’objets Java, avec leurs attributs et méthodes associées (getters, setters, etc.). Elles permettent de manipuler plus facilement les informations dans l’application en utilisant une approche orientée objet.

Le dossier **dao** (Data Access Object) regroupe les classes responsables de la gestion des interactions avec la base de données via JDBC. Chaque DAO est dédié à une entité spécifique et contient les méthodes permettant d’effectuer les opérations CRUD (Create, Read, Update, Delete) sur cette entité. Par exemple, VeterinaireDAO gère l’insertion, la mise à jour, la suppression et la récupération des vétérinaires dans la base. Cette séparation claire entre les entités et leur accès aux données améliore la lisibilité du code, facilite les tests et rend le projet plus modulaire.

Cette architecture respecte les bonnes pratiques en matière de développement logiciel en isolant la logique métier (entités) de la logique d’accès aux données (DAO), ce qui facilite également une évolution future, comme le remplacement de la base de données ou la modification des méthodes d’accès sans impacter le reste de l’application.

Pour valider le bon fonctionnement de l’architecture mise en place, notamment la communication entre les classes **entities**, les **DAO** et la base de données via **JDBC**, nous avons réalisé des tests dans une classe principale appelée Main.

Cette classe sert de point d’entrée à l’application et permet de vérifier les différentes opérations CRUD sur les entités. Par exemple, nous avons pu insérer de nouveaux vétérinaires, récupérer la liste des animaux, mettre à jour des informations et supprimer des visites médicales, le tout en s’assurant que les données étaient correctement enregistrées et extraites de la base MySQL.

Ces tests manuels réalisés dans la classe Main ont permis de s’assurer que les connexions à la base de données étaient bien établies, que les requêtes SQL étaient correctement exécutées via JDBC, et que les données circulaient de manière fluide.

**Alternatives à JDBC : Frameworks de gestion des données en Java**

Bien que **JDBC** soit une API puissante et standard pour interagir directement avec les bases de données, elle nécessite souvent d’écrire beaucoup de code SQL et de gérer manuellement les connexions, les transactions, et la conversion des résultats. Pour simplifier ce travail et améliorer la productivité, plusieurs frameworks et outils existent pour abstraire ces détails et automatiser la gestion des données.

Voici quelques alternatives populaires à JDBC :

* **Hibernate** : c’est un framework de mapping objet-relationnel (ORM) très utilisé en Java. Il permet de mapper automatiquement les classes Java aux tables de la base de données, et de manipuler les données via des objets plutôt que par des requêtes SQL. Hibernate gère aussi la génération automatique des requêtes, la gestion des transactions, et le cache. Il facilite la maintenance du code et améliore la productivité.
* **JPA (Java Persistence API)** : c’est une spécification standard pour la gestion de la persistance des données en Java. Hibernate est l’une des implémentations les plus connues de JPA. Grâce à JPA, le développeur peut utiliser des annotations pour définir le mapping entre les classes et la base de données, et bénéficier d’une interface commune pour différents fournisseurs de bases.
* **Spring Data JPA** : intégré au framework Spring, Spring Data JPA simplifie encore plus l’utilisation de JPA et Hibernate en fournissant des interfaces prêtes à l’emploi pour les opérations CRUD, des mécanismes de pagination, de tri, et de requêtes personnalisées. Il automatise la création des implémentations DAO, réduisant considérablement le code à écrire.
* **MyBatis** : c’est un framework de mapping SQL qui offre un contrôle plus fin que Hibernate. Contrairement à l’ORM, MyBatis permet d’écrire explicitement les requêtes SQL, tout en simplifiant le mapping entre les résultats et les objets Java. Il est apprécié quand les requêtes SQL complexes doivent être optimisées manuellement.

Ces frameworks apportent une couche d’abstraction qui facilite la gestion des données, améliore la lisibilité du code et accélère le développement. Leur utilisation est particulièrement recommandée pour des projets de moyenne à grande taille où la maintenance et l’évolution du code sont importantes.

***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 VetCare avec JavaFX et la connexion à la base de données via JDBC, nous avons rencontré plusieurs difficultés. La gestion des connexions à la base de données et la manipulation des requêtes SQL directement dans le code ont parfois rendu le développement fastidieux et source d’erreurs. De plus, l’intégration entre l’interface graphique JavaFX et la couche d’accès aux données n’a pas toujours été intuitive, notamment pour rafraîchir les vues après des opérations de modification ou d’ajout. Enfin, la prise en main de certains outils, comme Scene Builder, a nécessité un temps d’apprentissage pour maîtriser la conception des interfaces.

**Perspectives d’amélioration**

Pour renforcer la qualité et la performance de l’application VetCare, plusieurs axes d’amélioration peuvent être envisagés.

Tout d’abord, l’intégration d’un framework de persistance comme **Hibernate** ou **Spring Data JPA** permettrait de simplifier considérablement la gestion des interactions avec la base de données. Ces outils automatisent la génération des requêtes SQL et assurent une meilleure gestion des transactions, ce qui réduirait le risque d’erreurs et améliorerait la maintenabilité du code.

Ensuite, il serait pertinent de renforcer la **séparation des responsabilités** en appliquant un modèle architectural plus structuré, comme MVC (Modèle-Vue-Contrôleur). Cette organisation clarifierait la distinction entre la logique métier, la gestion des données et l’interface utilisateur, facilitant ainsi l’évolution et le débogage de l’application.

Par ailleurs, l’ajout de fonctionnalités avancées pourrait grandement enrichir l’expérience utilisateur. Par exemple, la mise en place de notifications automatiques pour les rappels de visites médicales, un système de recherche avancée pour filtrer les animaux ou les vétérinaires, ou encore la possibilité d’exporter des rapports au format PDF.

Enfin, améliorer l’ergonomie et le design de l’interface avec JavaFX en utilisant davantage les possibilités offertes par CSS et les animations contribuerait à rendre l’application plus agréable et intuitive à utiliser, ce qui est essentiel pour un usage quotidien par les vétérinaires et leur personnel.

Ces perspectives d’amélioration ouvrent la voie à une application plus robuste, évolutive et adaptée aux besoins réels des utilisateurs

***IIIII) Conclusion :***

En résumé, le projet VetCare a permis de développer une application complète et fonctionnelle pour la gestion des soins vétérinaires, en utilisant Java, JavaFX et une base de données SQL. Malgré certaines difficultés liées à la gestion directe des données avec JDBC et à l’intégration de l’interface graphique, ce projet a offert une bonne expérience pratique du développement d’applications desktop.

Les perspectives d’amélioration identifiées, notamment l’adoption de frameworks pour simplifier la gestion des données et l’amélioration de l’ergonomie, permettront de rendre l’application encore plus performante et adaptée aux besoins des utilisateurs. Ce projet constitue une base solide pour continuer à développer des solutions innovantes au service des vétérinaires.