**AVANT PROPOS**

**FISAORANA**

**REMERCIEMENTS**

Je rends grâce à Dieu Tout-Puissant pour la volonté, la santé et la patience qu’Il m'a accordées tout au long de ces trois années universitaires. Sans Son soutien, la réalisation de ce mémoire n’aurait pas possible. Je tiens également à exprimer ma profonde reconnaissance à toutes les personnes qui, par leur soutien, leur disponibilité et leurs conseils, ont grandement contribué à l’aboutissement de travail.  
Je remercie tout particulièrement :

* Monsieur **RAVELOMANANA Mamy Raoul**, Professeur titulaire et Président de l’Université d’Antananarivo, pour ses responsabilités dans la gestion de l'université ;
* Monsieur **ANTSONANTENAINARIVONY Ononamandimby**, Maître de conférences et Directeur de l’Institut d’Enseignement Supérieur Antsirabe-Vakinankaratra, pour m'avoir autorisé à accomplir cette soutenance ;
* Madame **RANAIVOSOA Mamitiana Lalaonirina Olivette**, Maître de conférences, Responsable de la mention Automatisme et Informatique et Présidente du jury, pour ses conseils précieux et son engagement malgré ses nombreuses responsabilités ;
* Monsieur **RAKOTONIRINA Andriamitantsoa Soloarinala**, Assistant d’Enseignement Supérieur et encadreur pédagogique, pour son aide précieuse et son accompagnement tout au long de l’élaboration de ce mémoire ;
* Monsieur **RADRIAMIARISON Heritiana Zilga**, Assistant d’Enseignement Supérieur, examinateur de ce mémoire, pour avoir accepté d’évaluer mon travail ;
* Monsieur **RALAHIARIJAONA Zandrison Richard**, Assistant d’Enseignement Supérieur, examinateur de ce mémoire, pour son implication dans l’évaluation de ce travail;
* L’ensemble des enseignants de l’**IESAV**, pour la qualité de leur enseignement et pour les bases théoriques indispensables qu’ils m'ont transmises ;
* Ma famille et mes amis, pour leur soutien moral, leur encouragement et leur présence constante.

À tous, je vous exprime ma gratitude la plus sincère.

AVANT PROPOS i

FISAORANA ii

REMERCIEMENTS iii

NOTATIONS viii

LISTE DES TABLEAUX ET FIGURES ix

INTRODUCTION GENERALE 1

Chapitre 1 : PRESENTATION GENERALE 2

1.1. Introduction 2

1.2. Etude de l’existant 2

1.2.1. Vetup 2

1.2.2. Vétocom 4

1.2.3 Provet Cloud 6

1.2.4 Animana 8

1.2.5 Recapitulation 10

1.3 Problématique 10

1.4 Solutions 11

1.5 Fonctionnalités 12

1.6 Conclusion 12

Chapitre 2 CHAPITRE 2 : CONCEPTION DU PROJET 14

2.1 Introduction 14

2.2 Notation UML 14

2.2.1 Définition 14

2.2.2 Objectif d’UML 14

2.2.3 Avantages 15

2.3 Conception 15

2.3.1 Diagramme de cas d’utilisation 16

2.3.2 Description textuelle 18

2.3.3 Diagramme de séquence 21

2.3.4 Diagramme d’activité 26

2.3.5 Règle de gestion 30

2.3.6 Dictionnaire de données 30

2.3.7 Diagramme de classe 32

2.4 Conclusion 35

Chapitre 3 CHAPITRE 3 : REALISATION DU PROJET 36

3.1 Introduction 36

3.2 Langages utilisés 36

3.2.1 HTML 36

3.2.2 JavaScript 37

3.3 Technologies utilisés 39

3.3.1 CSS 39

3.3.2 Node.js 40

3.4 Framework 42

3.4.1 Express.js 42

3.4.2 React avec Vite 43

3.4.3 Tailwind CSS 44

3.5 Logiciels utilisés 45

3.5.1 Visual Studio code 45

3.5.2 Postman 46

3.5.3 MongoDB Compass 48

3.5.4 Visual Paradigm 49

3.5.5 Brave Browser 50

3.6 Présentation des interfaces graphiques 51

3.6.1 Planning de disponibilités 51

3.6.2 Profile d’utilisateur 52

CONCLUSION ET PERSPECTIVE 54

ANNEXES xi

REFERENCES xii

FICHE DE RENSEIGNEMENTS xiii

**NOTATIONS**

**LISTE DES TABLEAUX ET FIGURES**

1. **Liste des tableaux**
2. **Liste des figures**

[Figure 1.01.1: Interface graphique de vetup.com 3](#_Toc204251957)

[Figure 1.01.2: Interface graphique de vétocom 5](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251958)

[Figure 1.01.3 : Interface graphique de provetCloud 7](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251959)

[Figure 1.01.4 : Page d'accueil d'Animana 9](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251960)

[Figure 2.02.1: Diagramme de cas d'utilisation de Vetika 17](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251961)

[Figure 2.02.2: Diagramme de sequence de cas d'utilisation n°1 23](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251962)

[Figure 2.02.3: Diagramme de séquence de cas d'utilisation n°2 24](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251963)

[Figure 2.02.4: Diagramme de séquence de cas d'utilisation n°3 25](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251964)

[Figure 2.02.5: Diagramme de séquence de cas d'utilisation n°4 26](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251965)

[Figure 2.6: Diagramme d'activité de cas d'utilisation n°1 28](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251966)

[Figure 2.7: Diagramme d'activité de cas d'utilisation n°2 28](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251967)

[Figure 2.8: Diagramme d'activité de cas d'utilisation n°3 29](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251968)

[Figure 2.9: Diagramme d'activité de cas d'utilisation n°4 29](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251969)

[Figure 2.10: Representation du classe 33](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251970)

[Figure 2.11: Diagramme de classe de l'application 35](#_Toc204251971)

[Figure 3.03.1: Logo du HTML 36](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251972)

[Figure 3.2: Logo de JavaScript 38](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251973)

[Figure 3.03.3: Logo du CSS 39](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251974)

[Figure 3.03.4: Logo de Node.JS: 40](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251975)

[Figure 3.03.5: Logo de Express.js 42](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251976)

[Figure 3.6: Logo de React avec Vite 43](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251977)

[Figure 3.07: Logo de TailwindCSS 44](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251978)

[Figure 3.08: Logo de VSCode 45](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251979)

[Figure 3.09: Logo officiel de Postman 47](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251980)

[Figure 3.10: Logo de MongoDB Compass 48](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251981)

[Figure 3.11: Logo de Visual Paradigm 49](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251982)

[Figure 3.12: Logo de Brave 50](file:///D:\\Boky%20Soutenance\\Boky%20Soutenance.docx" \l "_Toc204251983)

[Figure 3.13: Planning de disponibilités 52](#_Toc204251984)

[Figure 3.14: Profile d'utilisateur 53](#_Toc204251985)

[Figure A:3.15: Message xi](#_Toc204251986)

**INTRODUCTION GENERALE**

Le monde change rapidement, porté par les avancées technologiques qui touchent presque tous les domaines. Dans cet élan de modernisation, la santé animale mérite qu'on y accorde une attention particulière. Les cabinets vétérinaires, qui jouent un rôle clé dans le bien-être des animaux, travaillent encore souvent avec des méthodes assez classiques. Beaucoup utilisent encore le papier ou des outils informatiques très limités. Cela peut compliquer l’organisation au quotidien, entraîner des retards dans les soins ou des oublis importants, comme des rendez-vous ou des suivis de traitement.

En passant un peu de temps dans certains cabinets, on se rend vite compte des difficultés rencontrées. La gestion des dossiers, les consultations, les rappels de vaccins, la facturation ou encore les prises de rendez-vous demandent une attention constante. Sans un système centralisé, tout devient plus compliqué, aussi bien pour le vétérinaire que pour ses collaborateurs. Et forcément, quand l’organisation est lourde, cela finit par affecter la qualité des soins.

C’est à partir de ces constats qu’est née l’idée de créer une application web dédiée à la gestion vétérinaire. L’objectif est d’apporter une solution simple, pratique et réellement adaptée aux besoins du terrain. Un outil qui permettrait non seulement de gagner du temps, mais aussi de mieux suivre chaque animal, de planifier les tâches plus facilement et de renforcer la relation avec les propriétaires.

Ce mémoire s’inscrit donc dans une démarche tournée vers l’innovation, mais avec une volonté très concrète de répondre à un besoin réel. Il ne s’agit pas seulement de créer un logiciel, mais de proposer un véritable soutien aux professionnels de la santé animale. Le travail s’organise en trois grandes chapitre : d’abord, une présentation du contexte et des difficultés rencontrées dans les cabinets ; ensuite, un chapitre consacré à la conception de l’application ; et enfin, une dernière étape qui racontera le développement et l’utilisation de l’outil dans un cadre réel.

# : PRESENTATION GENERALE

* 1. **Introduction**

Avec l’évolution constante des besoins dans le domaine de la santé animale, les structures vétérinaires sont aujourd’hui confrontées à de nouveaux défis, notamment en matière de gestion et d’organisation. La technologie, qui joue un rôle central dans l’amélioration des services dans de nombreux secteurs, s’impose également comme une solution incontournable dans le milieu vétérinaire. La gestion numérique permet non seulement d’optimiser les tâches quotidiennes, mais aussi de renforcer le suivi médical des animaux et la relation entre le cabinet et les propriétaires.

Dans ce chapitre, nous allons explorer différentes solutions et outils numériques existants qui ont été développés pour répondre aux besoins des cabinets vétérinaires. Nous analyserons leurs principales caractéristiques, les services qu’ils proposent, ainsi que leurs interfaces et modes de fonctionnement. Cette étude de l’existant nous permettra de mettre en évidence les limites des solutions actuelles et d’identifier les points à améliorer dans le cadre de notre propre projet.

* 1. **Etude de l’existant**

Plusieurs solutions numériques ont vu le jour ces dernières années pour aider les vétérinaires à gérer leurs activités. Ces plateformes offrent des fonctionnalités variées allant de la gestion des dossiers médicaux des animaux à la facturation en passant par la prise de rendez-vous. Cependant, toutes ne répondent pas forcément aux attentes spécifiques de certains cabinets, en particulier ceux qui recherchent une interface plus adaptée à leur environnement de travail, une personnalisation poussée ou une meilleure accessibilité.

Dans cette optique, nous allons analyser quelques plateformes existantes afin de comprendre leurs avantages et leurs limites. Cette analyse critique nous servira de base pour concevoir une application mieux adaptée aux réalités du terrain.

* + 1. **Vetup**

Vetup est une solution en ligne de gestion vétérinaire qui propose une interface accessible via un navigateur web. Elle permet d'enregistrer les dossiers médicaux des animaux, de gérer les rappels de vaccination, les prescriptions, les rendez-vous, et même la facturation [1.01]. Vetup se distingue par sa simplicité d'utilisation et sa couverture fonctionnelle assez large. Toutefois, certains utilisateurs rapportent des limites en termes d'ergonomie, notamment sur les petits écrans ou en l'absence de connexion internet. La personnalisation de l’interface reste également limitée selon les besoins spécifiques des cabinets.

La figure 1.01 illustre la page d’accueil de vetup.com

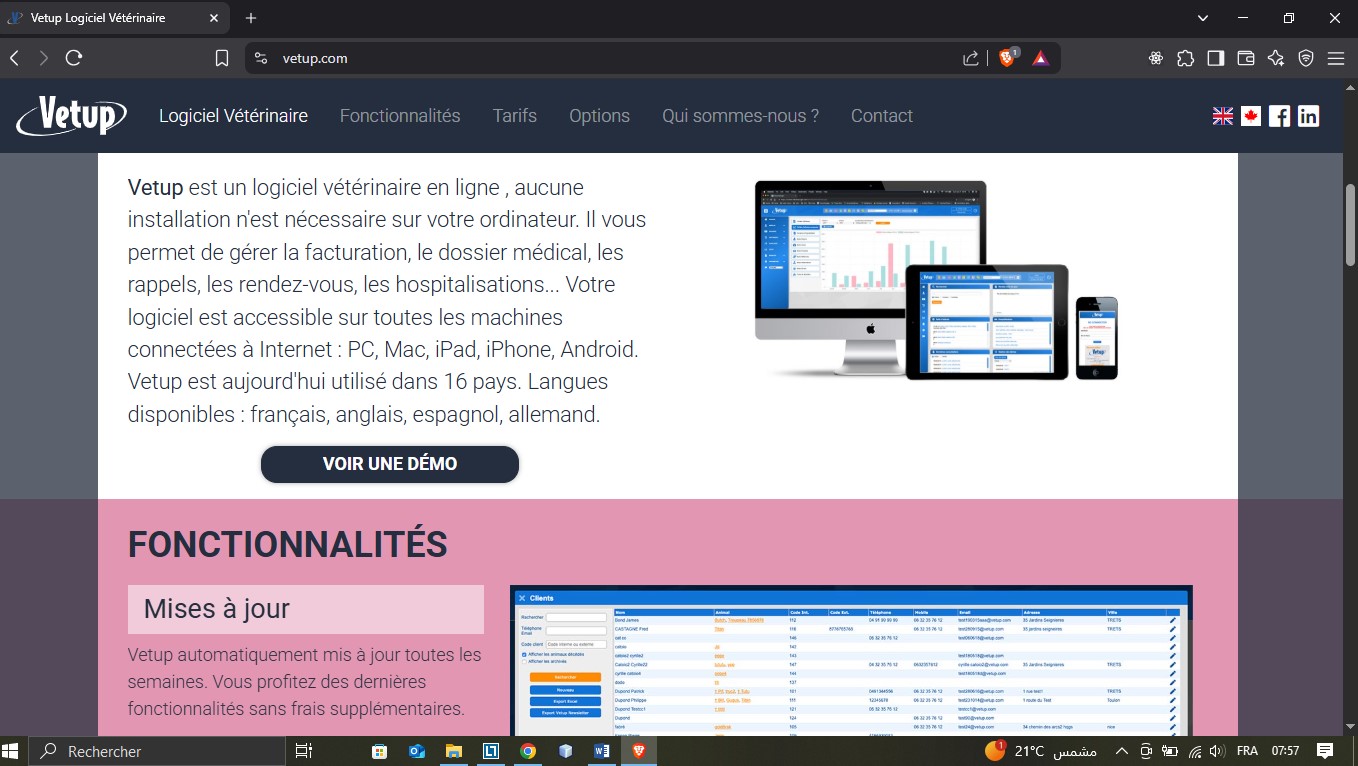


Figure 1.01.1: Interface graphique de vetup.com

**Avantages**

**Fonctionnalités diversifiées**

Il couvre plusieurs aspects de la gestion d’un cabinet : dossiers patients, gestion des propriétaires, facturation, stock, agenda des rendez-vous, et même gestion multi-utilisateurs.

**Interface multilingue**

La plateforme prend en charge plusieurs langues, ce qui la rend accessible à une clientèle internationale ou multilingue.

**Solution éprouvée**

Vetup est utilisé depuis plusieurs années dans divers pays, ce qui lui confère une certaine fiabilité et une base d'utilisateurs solide.

**Incovénients**

**Dépendance à une connexion internet**

Étant une application en ligne, Vetup nécessite une connexion stable. En cas de panne d’Internet, l’accès aux données devient impossible.

**Interface peu moderne**

L’interface utilisateur, bien qu’utile, peut sembler vieillissante ou moins intuitive comparée à des plateformes plus récentes. Cela peut ralentir la prise en main pour les nouveaux utilisateurs.

**Fonctionnalités limitées en personnalisation**

Certains utilisateurs trouvent que les options de personnalisation sont limitées, notamment lorsqu’il s’agit d’adapter le logiciel à des besoins très spécifiques.

**Fonctionnalités avancées payantes**

L’accès à certaines fonctions avancées peut nécessiter un abonnement ou des frais supplémentaires, ce qui peut être un frein pour les petites structures.

**Support technique parfois lent**

Des retours d’utilisateurs indiquent que le service d’assistance peut parfois prendre du temps à répondre, surtout en dehors des heures ouvrables européennes.

**Manque d'intégration externe**

Le système est relativement fermé, avec peu de possibilités d’intégration avec d'autres logiciels ou systèmes tiers (comme des systèmes de comptabilité ou outils d’analyse).

* + 1. **Vétocom**

Vétocom est une autre solution répandue en France. Elle se distingue par sa stabilité et son historique de développement solide. Elle intègre les outils nécessaires pour gérer efficacement un cabinet vétérinaire: dossiers patients, rappels de vaccination, suivi des traitements, etc. Toutefois, l’interface graphique commence à paraître vieillissante et moins intuitive comparée aux solutions plus modernes. De plus, elle est fortement dépendante d’un environnement Windows, ce qui peut poser problème en termes de compatibilité.

La figure 1.02 suivante illustre la page d’accueil de Vétocom

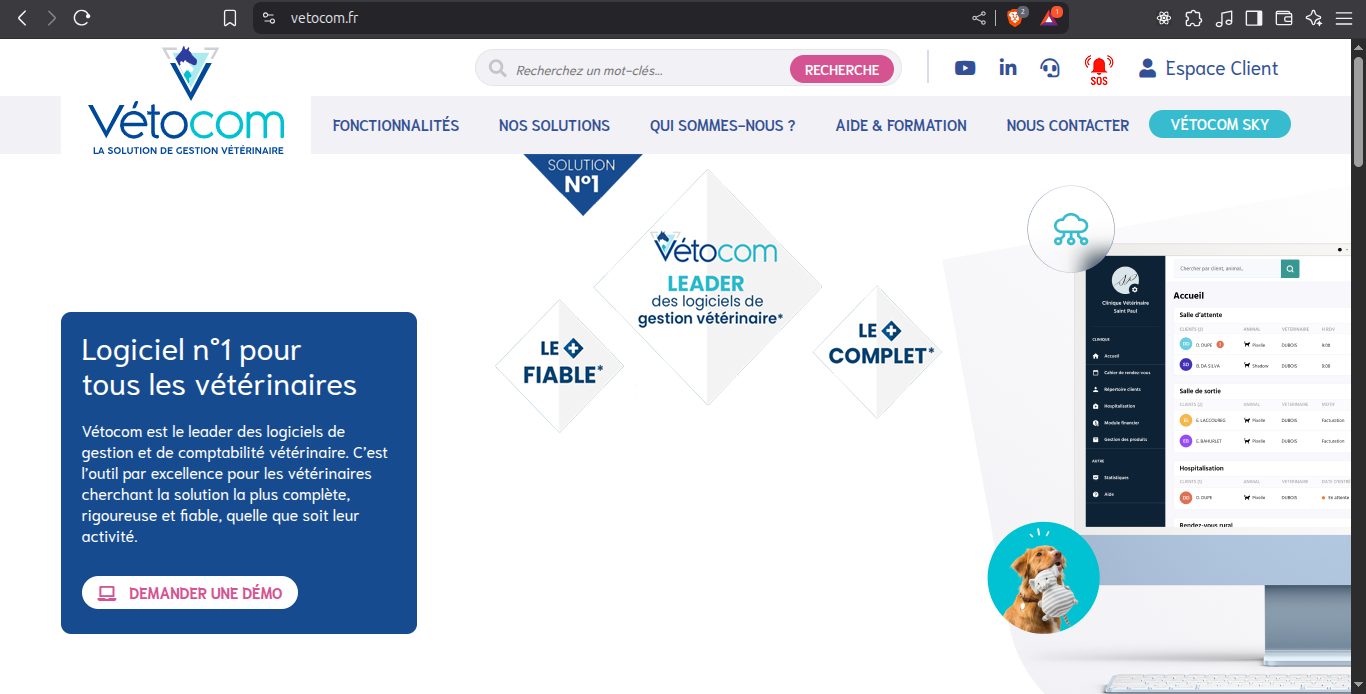


Figure 1.01.2: Interface graphique de vétocom

**Avantages**

**Personnalisation des actes et produits**  
Le cabinet peut personnaliser les actes médicaux, les tarifs, les fiches produits, ou encore les modèles de documents, ce qui permet une adaptation aux habitudes de travail de chaque structure.

**Sauvegarde des données sécurisée**  
Le logiciel propose un système de sauvegarde automatique qui préserve les données importantes en cas de panne ou d’incident technique.

**Service client réactif**

Le support technique est réputé pour sa disponibilité et sa réactivité. Les utilisateurs bénéficient d’une assistance rapide en cas de difficulté ou de question sur le logiciel.

**Inconvenients**

**Coût relativement élevé**  
Vétocom est une solution payante, avec un coût qui peut être élevé pour les petits cabinets ou les structures débutantes. Le modèle de tarification dépend généralement du nombre de postes ou de fonctionnalités activées.

**Interface peu moderne**  
L’interface utilisateur, bien qu’utile, peut sembler vieillissante ou moins intuitive comparée à des plateformes plus récentes. Cela peut ralentir la prise en main pour les nouveaux utilisateurs.

**Fonctionnalités limitées en personnalisation**  
Certains utilisateurs trouvent que les options de personnalisation sont limitées, notamment lorsqu’il s’agit d’adapter le logiciel à des besoins très spécifiques.

**Fonctionnalités avancées payantes**  
L’accès à certaines fonctions avancées peut nécessiter un abonnement ou des frais supplémentaires, ce qui peut être un frein pour les petites structures.

**Support technique parfois lent**  
Des retours d’utilisateurs indiquent que le service d’assistance peut parfois prendre du temps à répondre, surtout en dehors des heures ouvrables européennes.

**Manque d'intégration externe**  
Le système est relativement fermé, avec peu de possibilités d’intégration avec d'autres logiciels ou systèmes tiers (comme des systèmes de comptabilité ou outils d’analyse).

**1.2.3 Provet Cloud**

Provet Cloud est un logiciel de gestion vétérinaire basé sur le cloud. Il met l’accent sur l’intégration entre les différents services (rendez-vous, dossiers médicaux, communication avec les clients, paiements, etc.). Il offre une interface fluide et moderne ainsi que des fonctionnalités avancées comme la gestion des stocks, les statistiques financières, et les accès multi-utilisateurs. Néanmoins, certaines fonctionnalités ne sont accessibles qu’en version payante, ce qui peut limiter son adoption par de petites structures ou dans un contexte à budget restreint.

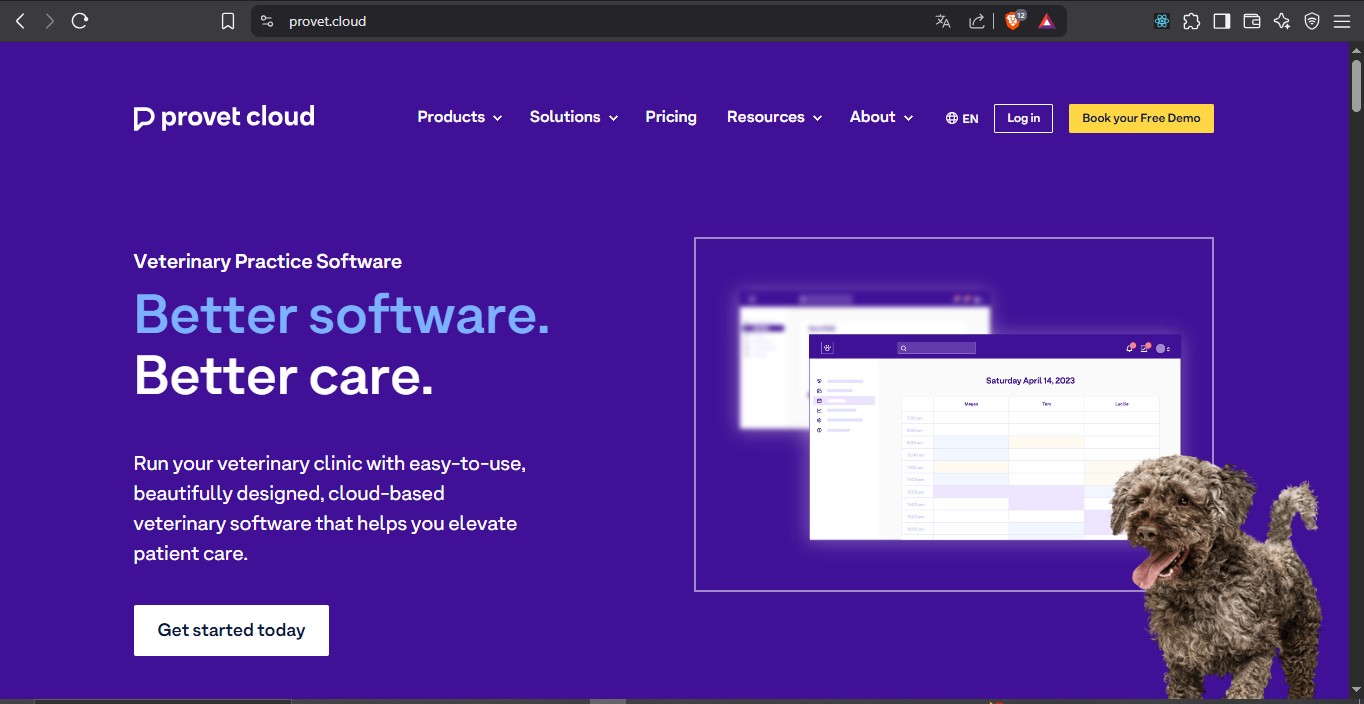
La figure 1.03 suivante illustre la page d’accueil de Provet Cloud

Figure 1.01.3 : Interface graphique de provetCloud

**Avantages**

**Interface moderne et intuitive**  
L’interface utilisateur est ergonomique, claire et bien structurée. Elle facilite la navigation et permet une prise en main rapide, même pour les personnes peu expérimentées avec les outils numériques.

**Fonctionnalités avancées de gestion**  
Le logiciel couvre de nombreux aspects de la gestion d’un cabinet : planification des rendez-vous, gestion des patients et propriétaires, facturation, gestion du stock, envoi de rappels automatisés, suivi des paiements, etc.

**Mises à jour automatiques**  
En tant que solution cloud, les mises à jour sont effectuées automatiquement sans intervention de l’utilisateur, ce qui garantit un accès constant aux dernières améliorations et fonctionnalités.

**Support multilingue et international**  
Le logiciel est utilisé dans plusieurs pays et disponible en plusieurs langues, ce qui le rend adapté aux cabinets situés dans des environnements multilingues ou internationaux.

**Inconvénients**

**Dépendance à la connexion Internet**  
L’un des principaux inconvénients d’un logiciel basé sur le cloud est la nécessité constante d’une connexion stable. En cas de panne d’Internet, l’accès à toutes les données et fonctionnalités est bloqué.

**Coût récurrent**  
Provet Cloud fonctionne sur un modèle d’abonnement mensuel ou annuel. Cela représente une charge continue pour le cabinet, qui peut devenir importante selon la taille de l’équipe et les fonctionnalités utilisées.

**Complexité de certaines configurations**  
Bien que l’interface soit intuitive, certaines fonctions plus techniques (comme l'intégration de systèmes tiers ou la configuration des modèles) peuvent demander un accompagnement technique initial.

**Moins adapté aux très petites structures**  
Pour les très petits cabinets, certaines fonctionnalités peuvent sembler superflues ou trop complexes. De plus, le coût peut ne pas toujours être justifié pour des structures avec peu de patients ou de personnel.

**Confidentialité des données**  
Comme pour toute solution cloud, certaines inquiétudes peuvent exister quant à la localisation et la protection des données, même si Provet Cloud respecte les normes de sécurité en vigueur.

### **1.2.4 Animana**

Développée par IDEXX, Animana est une plateforme de gestion vétérinaire très complète et orientée vers l’automatisation. Elle est accessible depuis n’importe quel appareil connecté et propose de nombreuses fonctionnalités, notamment l’intégration avec les appareils de diagnostic. Son principal inconvénient reste sa complexité pour les nouveaux utilisateurs, ce qui nécessite un certain temps d’adaptation. Par ailleurs, elle reste une solution coûteuse pour les petites structures vétérinaires.

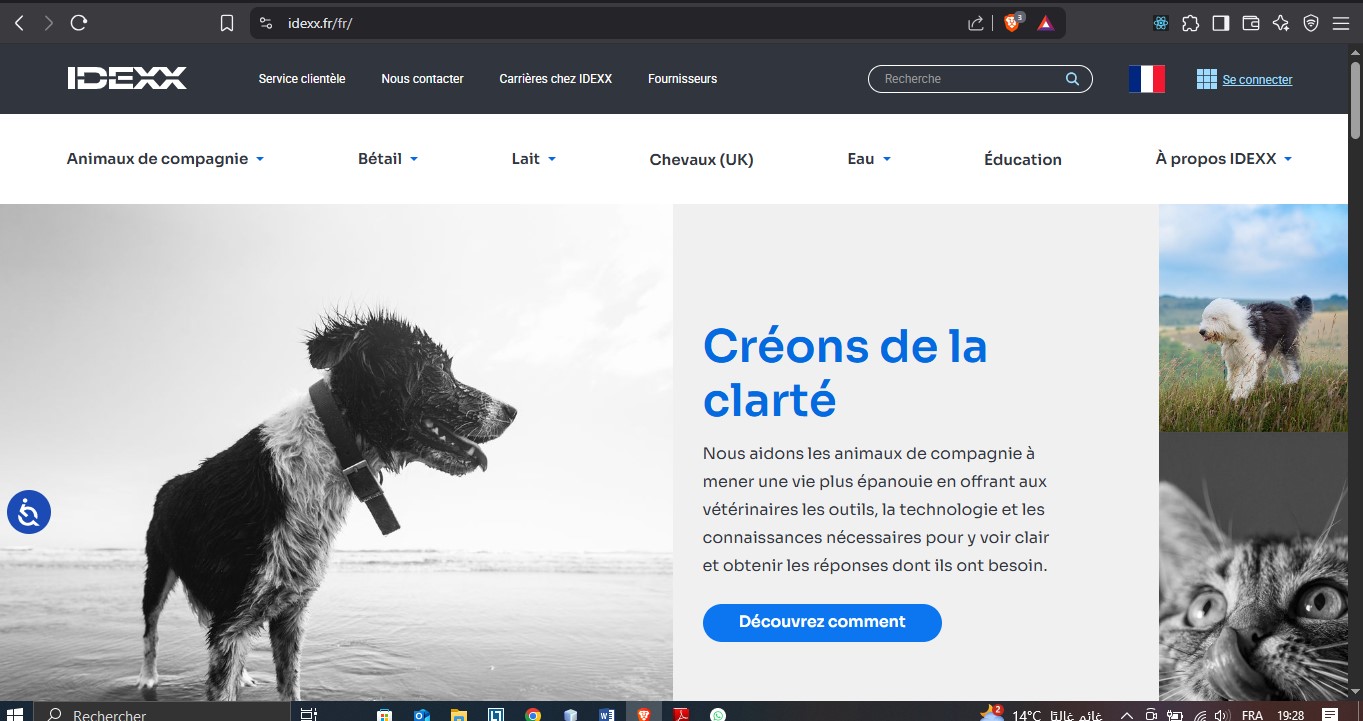
La figure 1.04 suivante affiche la page d’accueil d’Animana

Figure 1.01.4 : Page d'accueil d'Animana

**Avantages**

**Historique médical détaillé**  
Chaque animal bénéficie d’un dossier médical bien structuré où toutes les interventions, vaccinations, traitements et résultats sont enregistrés, assurant un excellent suivi vétérinaire.

**Envoi automatique de rappels**  
La plateforme permet d’envoyer automatiquement des rappels par email ou SMS pour les vaccins, consultations, ou traitements, renforçant la fidélisation des clients et le respect des protocoles de soin.

**Stock et facturation intégrés**  
La gestion du stock est synchronisée avec les actes médicaux et la facturation, permettant un suivi précis des produits utilisés et une réduction des erreurs administratives.

**Inconvénients**

**Nécessité d'une bonne connexion Internet**  
Comme Animana est une solution cloud, toute coupure ou instabilité du réseau Internet peut empêcher l’accès aux dossiers ou ralentir le travail quotidien du cabinet.

**Courbe d’apprentissage pour certaines fonctions**  
Certaines fonctionnalités avancées nécessitent un temps d’adaptation. Les utilisateurs novices peuvent rencontrer quelques difficultés au début, notamment pour la configuration initiale.

**Coût de l'abonnement**  
Le logiciel fonctionne par abonnement, ce qui implique des frais mensuels ou annuels. Ce modèle peut représenter un coût non négligeable, surtout pour les petites structures vétérinaires.

**Manque de personnalisation avancée**  
Certaines options de personnalisation sont limitées par rapport à d’autres solutions concurrentes, ce qui peut poser problème aux cabinets ayant des besoins très spécifiques.

**Interface parfois trop épurée**  
L’interface, bien qu’ergonomique, peut être perçue comme trop minimaliste. Certains utilisateurs déplorent le manque de raccourcis ou d’actions rapides pour des tâches récurrentes.

**Support technique variable selon la région**  
La qualité et la réactivité du service client peuvent varier selon le pays ou la langue de l’utilisateur, ce qui peut ralentir la résolution de certains problèmes.

**1.2.5 Recapitulation**

Bien que plusieurs plateformes de gestion vétérinaire soient déjà disponibles sur le marché, certaines lacunes persistent encore, impactant l’efficacité du suivi médical, la relation client ou encore la fluidité de l’expérience utilisateur. Notre étude des solutions existantes a permis d’identifier un ensemble de points faibles récurrents qui méritent d’être corrigés dans une future application. Les paragraphes suivants détailleront les pistes de solutions proposées dans notre plateforme afin de répondre aux limites observées, notamment :

* La complexité de prise en main pour les nouveaux utilisateurs
* Le manque d’adaptabilité aux besoins spécifiques des petites structures vétérinaires
* L’absence de personnalisation de certaines fonctionnalités comme les rappels clients ou les dossiers médicaux
* Le coût d’acquisition ou d’abonnement élevé de certaines solutions
* La dépendance à une connexion internet stable pour le bon fonctionnement des solutions cloud
* La gestion parfois rigide des stocks ou des fichiers clients

**1.3 Problématique**

Une bonne organisation constitue toujours la clé d’un fonctionnement efficace, quel que soit le domaine d’activité. Dans le secteur vétérinaire, cette organisation devient primordiale lorsqu’il s’agit de gérer les dossiers médicaux des animaux, les rendez-vous, les traitements, la facturation ainsi que le suivi clientèle. Pourtant, dans de nombreux cabinets, la gestion se fait encore de manière traditionnelle, avec des enregistrements sur papier ou dans des fichiers dispersés.

Dans notre contexte, à l’image de nombreux cabinets vétérinaires à Madagascar, les données concernant les animaux sont souvent consignées manuellement dans des registres ou sur des fiches physiques. Ce mode de fonctionnement, bien qu'encore utilisé, comporte plusieurs risques: perte de documents, erreurs de transcription, difficulté à retrouver rapidement les informations, ou encore manque de suivi structuré sur les traitements et les antécédents médicaux des animaux. Ces contraintes posent de réels problèmes d’efficacité et de traçabilité, tout en ralentissant la qualité des soins et du service offert aux clients.

**1.4 Solutions**

Pour remédier aux problèmes identifiés dans la gestion traditionnelle des cabinets vétérinaires, nous avons choisi d’effectuer une étude et une réalisation d’une application web destinée à améliorer l’organisation, le suivi médical et administratif des animaux. Cette plateforme offrira une solution centralisée permettant aux vétérinaires de mieux gérer les dossiers des animaux, les rendez-vous, les consultations ainsi que la communication avec les propriétaires. Elle vise également à faciliter la recherche rapide d’informations médicales et historiques pour chaque animal. Voici donc les principales solutions que notre application propose face aux problématiques et aux inconvénients soulevés par l’analyse des plateformes existantes :

* Mise en place d’un système numérique de gestion des dossiers médicaux pour assurer la traçabilité des consultations, vaccinations, traitements et antécédents sans risque de perte d’informations.
* Simplification de la gestion quotidienne grâce à une interface intuitive et ergonomique, permettant une prise en main rapide par le personnel vétérinaire.
* Intégration de modules adaptés à l’usage vétérinaire, respectant les bonnes pratiques d’ergonomie et de lisibilité entre l’utilisateur et le système.
* Disponibilité d’une interface dédiée aux vétérinaires pour gérer les consultations, les dossiers des animaux, les ordonnances et les stocks de médicaments.
* Présence d’un espace destiné aux propriétaires d’animaux, leur permettant de consulter les informations concernant leur animal, recevoir des rappels de rendez-vous, et suivre les soins prodigués.

**1.5 Fonctionnalités**

Pour offrir un meilleur système de gestion dans un cabinet vétérinaire, nous avons planifié de mettre en place quelques fonctionnalités essentielles en poursuivant les besoins nécessaires constatés lors de l’étude de l’existant.

Pour le propriétaire d’ animal :

* **Une page d’accueil permettant de visualiser les informations générales liées à son ou ses animaux**
* **Une page de gestion du profil personnel et des coordonnées**
* **Une page de suivi des consultations, vaccins, traitements et rendez-vous programmés**
* **Une page pour prendre un rendez-vous chez le vétérinaire**

Pour le vétérinaire :

* **Une page de gestion des dossiers médicaux de chaque animal avec historique des consultations**
* **Une page de gestion des rendez-vous avec vue calendrier (jour, semaine, mois)**
* **Une page de prescription pour générer et gérer les ordonnances**
* **Une page de gestion des propriétaires et de leurs animaux**

Pour l’administrateur :

* **Une page de gestion de tous les utilisateurs (vétérinaires et propriétaires)**
* **Une page de gestion des consultations et activités globales du cabinet**
* **Une page de gestion du contenu de la plateforme, y compris les informations visibles sur l’accueil**
* **Une page de statistiques (nombre d’animaux, consultations, rappels vaccinaux, etc.)**

**1.6 Conclusion**

Face aux différentes problématiques relevées dans la gestion traditionnelle des dossiers médicaux d’animaux et des rendez-vous vétérinaires, notre application propose une solution moderne, accessible et bien organisée. Elle permet à chaque utilisateur, qu’il soit vétérinaire, administrateur ou propriétaire, de naviguer dans un environnement simple, rapide et fonctionnel. Dans le chapitre suivant, nous allons aborder la phase de conception du système en tenant compte de ces fonctionnalités pour garantir une expérience utilisateur optimale.

# CHAPITRE 2 : CONCEPTION DU PROJET

**2.1 Introduction**

La conception de notre application commence par une phase d’analyse visant à comprendre les besoins des utilisateurs : vétérinaires, administrateurs et propriétaires d’animaux. Cette étape permet d’identifier les fonctionnalités clés attendues, comme la gestion des consultations, des dossiers médicaux et des rendez-vous. Une fois les besoins clarifiés, la phase de conception permet de proposer une solution adaptée. Pour cela, nous utilisons la méthode UML afin de modéliser les cas d’utilisation et les interactions entre les différentes entités du système. Cette approche structurée facilite la visualisation du fonctionnement de l’application avant sa mise en œuvre.

**2.2 Notation UML**

**2.2.1 Définition**

L’UML (Unified Modeling Language) est un langage de modélisation standardisé qui permet de représenter visuellement les éléments structurels et comportementaux d’un système logiciel. Il a été développé dans les années 1990 par trois experts en modélisation orientée objet : Grady Booch, James Rumbaugh et Ivar Jacobson. Chacun d’eux avait auparavant conçu sa propre méthode. Ces approches ont été fusionnées pour donner naissance à une notation unifiée, dans le but de standardiser les pratiques de conception logicielle.

Cette unification a été officialisée par l’Object Management Group (OMG), qui assure aujourd’hui la gestion et l’évolution du standard UML. Ce langage est devenu incontournable dans la modélisation des systèmes informatiques, car il permet de mieux analyser, concevoir et communiquer autour d’un projet logiciel, comme c’est le cas pour notre application de gestion vétérinaire [2.01].

**2.2.2 Objectif d’UML**

Les objectifs principaux d’UML sont les suivants :

* Proposer un langage visuel standardisé pour modéliser les systèmes logiciels de manière claire et structurée ;
* Être indépendant des langages de programmation, afin de pouvoir être utilisé dans tout type de projet informatique ;
* Offrir des mécanismes d’extension et de spécialisation pour adapter les concepts UML aux besoins spécifiques d’un projet ;
* Fournir une base formelle permettant de mieux comprendre, analyser et communiquer autour des modèles de conception.

**2.2.3 Avantages**

L'utilisation de l’UML (Unified Modeling Language) présente de nombreux avantages pour la conception et le développement de systèmes logiciels. Voici quelques-uns des principaux avantages que procure son adoption dans un projet informatique :

**Communication**  
L’UML permet de représenter visuellement les différents aspects d’un système logiciel, ce qui facilite grandement la communication entre les membres de l’équipe de développement, les chefs de projet et les parties prenantes non techniques.

**Compréhension**  
Grâce à ses diagrammes explicites, l’UML offre une meilleure compréhension globale du système, de ses composants et de leurs interactions. Cela rend plus facile l’analyse des besoins, la prise de décision technique, ainsi que la détection d’éventuelles incohérences dès les premières étapes.

**Cohérence**  
En utilisant un langage de modélisation unifié et standardisé, UML assure une cohérence entre les différents modèles produits tout au long du cycle de développement, qu’il s’agisse de diagrammes de classes, de cas d’utilisation ou de séquences.

**Gestion de la complexité**  
L’UML aide à mieux gérer la complexité d’un système logiciel en fournissant des représentations abstraites et hiérarchisées. Cela permet de diviser un système complexe en parties plus simples et compréhensibles, facilitant ainsi son développement progressif.

**2.3 Conception**

Pour cette partie de la conception, nous allons utiliser la notation UML qui utilise plusieurs diagrammes. Les diagrammes qu’on va utiliser pour concevoir notre logiciel :

- Diagramme de cas d’utilisation ;

- Diagramme de séquence ;

- Diagramme d’activité ;

- Diagramme de classe ;

**2.3.1 Diagramme de cas d’utilisation**

**2.3.1.1 Définition**

Le diagramme de cas d’utilisation est un moyen simple et efficace pour identifier ce que doit faire un système en fonction des besoins des utilisateurs. En modélisant les besoins, on liste toutes les fonctionnalités attendues et on organise leur relation entre elles, ce qui aide à mieux comprendre comment tout doit fonctionner ensemble. Avec UML, on utilise justement ce type de diagramme pour représenter clairement les différents acteurs qui utilisent le système, ainsi que les actions qu’ils peuvent faire. C’est un outil très utile pour poser les bases du projet et s’assurer que toutes les attentes sont bien prises en compte avant de commencer le développement.

**2.3.1.2 Elément du diagramme de cas d’utilisation**

Les éléments principaux qui composent un diagramme de cas d’utilisation sont : **le système, les acteurs, les cas d’utilisation** et **les relations** entreeux**.**

**Système**

Le système définit les limites de ce que l’on souhaite modéliser. Il représente l’ensemble des fonctionnalités que notre application doit proposer, et il sert à délimiter clairement ce qui fait partie du projet et ce qui est extérieur.

**Acteur**  
Un acteur représente une entité extérieure (humaine, organisationnelle ou technique) qui interagit avec le système. Il joue un rôle précis dans l’utilisation du système, sans en faire partie. Les acteurs peuvent être classés en quatre catégories :

* **Les acteurs principaux** : ce sont les utilisateurs qui profitent directement des fonctionnalités principales du système (ex. : vétérinaire, propriétaire d’animal).
* **Les acteurs secondaires** : ils sont impliqués dans la gestion ou l’administration du système (ex. : administrateur de la plateforme).
* **Le matériel externe** : cela désigne les dispositifs matériels qui interagissent avec le système (dans certains cas, par exemple des scanners de puces électroniques).
* **Les autres systèmes** : ce sont les systèmes extérieurs avec lesquels notre application peut communiquer (comme un logiciel de gestion de pharmacie vétérinaire).

**Cas d’utilisation**  
Un cas d’utilisation décrit une action ou une fonctionnalité que le système fournit pour répondre à un besoin d’un acteur. Chaque cas d’utilisation produit un résultat observable et utile pour l’acteur concerné, comme "Ajouter un dossier animal", "Planifier un rendez-vous" ou "Consulter les antécédents médicaux".

**Relation**  
Les relations expriment les liens entre les acteurs et les cas d’utilisation. Elles permettent aussi de montrer comment différents cas d’utilisation peuvent être reliés entre eux. Il existe trois types principaux de relations :

* **La relation de généralisation** : utilisée pour montrer qu’un acteur ou un cas d’utilisation hérite des caractéristiques d’un autre.
* **La relation d’extension (extend)** : elle permet d’ajouter un comportement optionnel à un cas d’utilisation principal.
* **La relation d’inclusion (include)** : elle permet de réutiliser un comportement commun dans plusieurs cas d’utilisation.

**2.3.1.3 Diagramme de cas d’utilisation de la plateforme**

Le diagramme de cas d’utilisation de notre plateforme est la suivante :

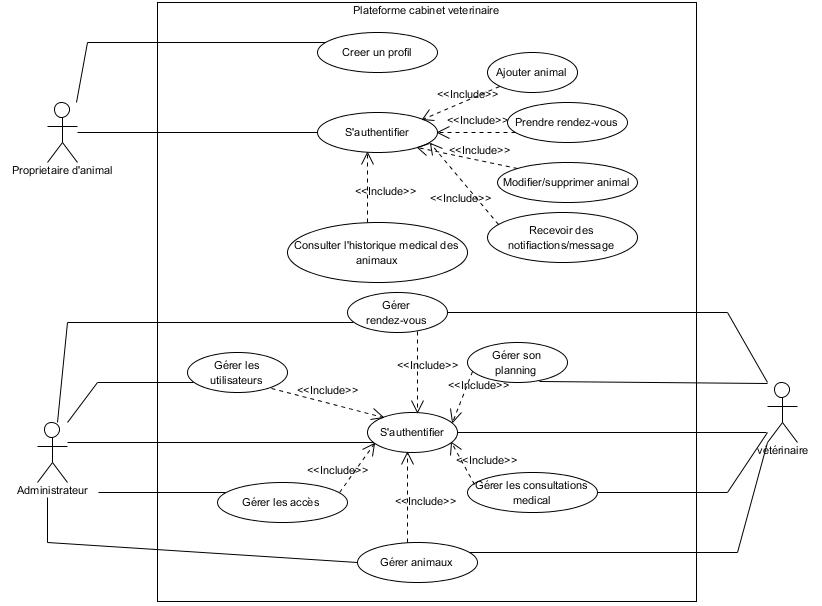
****

Figure 2.02.1: Diagramme de cas d'utilisation de Vetika

**2.3.2 Description textuelle**

**2.3.2.1 Définition**

La description textuelle constitue un complément indispensable aux diagrammes de cas d'utilisation. Elle permet de détailler les interactions entre les acteurs et le système, en exposant le déroulement des scénarios, les objectifs visés ainsi que les différentes variantes possibles. Chaque cas d’utilisation doit ainsi faire l’objet d’une description précise et rigoureuse.

Les éléments clés à décrire pour chaque cas d'utilisation sont les suivants :

**Scénario nominale**

Également appelé scénario principal ou scénario de base, il représente la séquence d'étapes standard, attendue et idéale d’un cas d’utilisation. Il décrit comment le système ou l’application doit réagir dans des conditions normales, sans perturbation, pour atteindre l’objectif souhaité par l’acteur principal.

**Scénarios alternatifs**

Les scénarios alternatifs correspondent à des variations du scénario nominal. Ils sont déclenchés par des situations spécifiques qui entraînent un déroulement différent, sans pour autant constituer une erreur. Ces scénarios permettent d’explorer des comportements alternatifs du système ou de l’utilisateur, tout en restant dans un cadre de fonctionnement valide.

**Scénario d’exception**

Les scénarios d’exception décrivent les situations anormales ou imprévues pouvant interrompre ou modifier la séquence d'actions du scénario nominal. Ils mettent en évidence les réponses du système face à des erreurs, à des entrées invalides ou à des événements imprévus, afin d’assurer la robustesse du fonctionnement global.

**Préconditions**

Les préconditions précisent les conditions initiales requises pour que le cas d’utilisation puisse débuter. Elles répondent notamment à la question :

"Quand et comment le cas d’utilisation commence-t-il ?"

Cela inclut les états du système, les droits de l’acteur et les éventuels éléments déjà présents dans l’environnement.

**2.3.2.2 Description textuelle des cas d’utilisation**

* **Cas d’utilisation n°1 : « Prendre rendez-vous »**

**Généralité**

Cas d’utilisation : « Prendre rendez-vous »

Acteur : Propriétaire d’animal

Précondition : Le propriétaire d’animal est connecté à la plateforme, dispose d’un compte valide et a enregistré au moins un animal dans son profil.

**Scénario nominale**

**1**. Le propriétaire d’animal accède à la section « Prendre rendez-vous » depuis le tableau de bord.

**2**. Il sélectionne l’animal concerné parmi ceux qu’il a enregistrés.

**3**. Il choisit un vétérinaire disponible et un créneau horaire proposé.

**4**. Il confirme le rendez-vous.

**5**. Le système enregistre le rendez-vous et affiche un message de confirmation.

**Scénario alternatif**

**2.**

**2.1** Le système affiche un message l’invitant à enregistrer un animal.  
**2.2** Le propriétaire d’animal remplit le formulaire d’ajout.  
**2.3** Enregistrement de l’animal, retour à l’étape 2 du scénario nominal.

* **Cas d’utilisation n°2 : « Gérer planning »**

**Généralité**

Cas d’utilisation : « Gérer planning »

Acteur : Vétérinaire

Précondition : Le vétérinaire est connecté à la plateforme et a accès à son espace personnel. Il possède les droits nécessaires pour visualiser, modifier et valider les rendez-vous qui lui sont assignés.

**Scénario nominale**

**1**. Le vétérinaire accède à la section « Mon planning » via le tableau de bord.

**2**. Le système affiche une vue calendrier avec les rendez-vous programmés.

**3**. Le vétérinaire consulte les détails des rendez-vous à venir.

**4**. Il valide les rendez-vous confirmés avec les propriétaires d’animaux.

**5**. Il peut modifier ou reprogrammer un créneau si besoin.

**6**. Il peut ajouter une période d’indisponibilité (ex. : congé, réunion).

**7**. Le système met automatiquement à jour le planning et notifie les personnes concernées (propriétaires d’animaux).

**Scénario alternatif**

**5.**

**5.1** Il sélectionne le rendez-vous à modifier.  
**5.2** Il change la date ou l’heure selon ses disponibilités.  
**5.3** Le système envoie une notification au propriétaire pour validation.  
**5.4** Le rendez-vous passe à l’état « En attente de confirmation ».  
**5.5** Une fois confirmé, il est intégré dans le planning.

* **Cas d’utilisation n°3 : « Consulter l’historique médical de son animal »**

**Généralité**

Cas d’utilisation : Consulter l’historique médical de son animal

Acteur : Propriétaire d’animal

Précondition : Le propriétaire d’animal est authentifié sur la plateforme et dispose d’au moins un animal enregistré dans son compte. Des consultations antérieures doivent être enregistrées dans le système pour l’animal concerné.

**Scénario nominal**

**1.** Le propriétaire d’animal accède à la section « Mes animaux » depuis son tableau de bord.

**2.** Il sélectionne un animal dans la liste.

**3.** Il clique sur le bouton « Historique médical ».

**4.** Le système affiche la liste chronologique des consultations vétérinaires de l’animal sélectionné.

Pour chaque consultation, le propriétaire peut consulter :

* + La date de la visite
  + Le vétérinaire consulté
  + Le motif de consultation
  + Le diagnostic
  + Le traitement ou ordonnance associée (le cas échéant)

**5.** Le propriétaire peut télécharger un compte rendu en PDF s’il est disponible.

**6.** Il quitte la section ou revient à la liste de ses animaux.

**Scénario alternatif**

**3.**

**3.1** Le système affiche un message : « Aucun historique médical disponible pour cet animal. »

**3.2** Le propriétaire peut retourner à la liste de ses animaux.

* **Cas d’utilisation n°4 : « Gérer l’accès »**

**Généralité**

Cas d’utilisation : « Gérer l’accès »

Acteur : Administrateur

Précondition : L’administrateur est connecté à la plateforme avec un compte disposant des droits complets. Il accède à la section de gestion des comptes et des rôles

**Scénario nominal**

**1.** L’administrateur accède à la section « Gestion des utilisateurs ».

**2.** Il consulte la liste des comptes existants (propriétaires d’animaux, vétérinaires, autres admins).

**3.** Si un nouveau compte vétérinaire est créé ou si un utilisateur demande à devenir vétérinaire :

* La demande d’accès vétérinaire apparaît dans une liste d’approbation.
* L’administrateur examine les informations et justificatifs fournis.
* Il peut :
* Approuver la demande (le compte vétérinaire est activé, l’utilisateur reçoit une notification d’acceptation)
* Rejeter la demande (l’utilisateur reçoit une notification de refus avec éventuellement un motif)

**4.** Lors d’une modification ou suppression, une confirmation est demandée.

**5.** Le système enregistre les changements et affiche un message de succès.

**6.** L’utilisateur concerné reçoit une notification (si applicable).

**Scénario alternatif**

**2.**

**2.1** Il utilise les filtres disponibles (ex : rôle, statut actif/inactif, demandes en attente d’approbation)

**2.2** Le système affiche uniquement les utilisateurs correspondant aux critères sélectionnés

**2.3** Reprise à l’étape 3 du scénario nominal

**2.3.3 Diagramme de séquence**

**2.3.3.1 Définition**

Le **diagramme de séquence** est un diagramme d’interaction UML qui représente les échanges de messages entre les objets d’un système, en mettant l’accent sur la dimension **temporelle** de ces interactions. Ce diagramme illustre les **réactions du système aux actions des utilisateurs** en modélisant la **création, l’appel de méthodes, l’envoi de messages et la destruction éventuelle des objets** impliqués.

**2.3.3.2 Elements constitutifs du diagramme de séquence**

Les principaux éléments composant un diagramme de séquence sont les suivants :

**Ligne de vie (Lifeline)**

Elle représente un objet ou un acteur présent dans le scénario.  
Visuellement, il s’agit d’une **ligne verticale en pointillé** partant du haut vers le bas, indiquant la durée pendant laquelle l’objet est actif.  
Chaque ligne de vie est nommée selon l’objet ou l’acteur qu’elle représente.

**Messages**

Généralement un appel ou un signal ou une réponse. Représenté par des flèches horizontales

reliant la ligne de vie de l’objet émetteur à la ligne de vie de l’objet récepteur.

* Message de création

Un message spécifique qui donne lieu au début de la ligne de vie d’un nouvel objet

* Message de destruction

Un message envoyé à un objet existant et qui donne lieu à la fin de sa ligne de vie

* Message réflexif

Un objet peut envoyer un message à lui-même

* Message récursif

Elle se présente par un dédoublement de la barre d’activation

**Activation**

Une activation représente la **période pendant laquelle un objet exécute une opération** ou reste actif en attente de réponse.  
Elle est représentée par un **rectangle vertical fin** posé sur la ligne de vie, indiquant qu’un objet est en cours de traitement d’une opération (appelant ou recevant).

**Interactions**

Les interactions sont les **échanges de messages entre objets**, essentiels pour illustrer le déroulement du scénario. Elles reflètent la **logique temporelle** du processus métier ou technique modélisé.

**Fragment combiné (optionnel)**

Regroupement logique représenté par un rectangle et contenant les structures conditionnelles qui affecte le flux de message. Il décrit les diagrammes de séquence de manière compacte. Il est défini par un opérateur et des opérandes. Les opérateurs conditionnent la signification du fragment combiné.

* + - 1. **Diagramme de séquence de la plateforme**
* **Cas d’utilisation n°1 : « Prendre rendez-vous »**

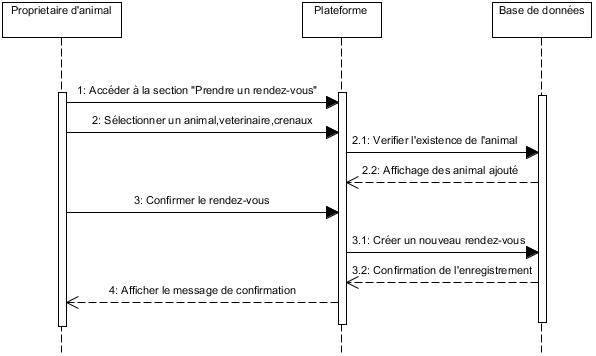
****La figure 2.02 représente le diagramme de séquence de cas d’utilisation « Prendre rendez-vous ».

Figure 2.02.2: Diagramme de sequence de cas d'utilisation n°1

* **Cas d’utilisation n°2 « Gérer planning »**

La figure 2.03 représente le diagramme de séquence de cas d’utilisation « Gérer planning ».

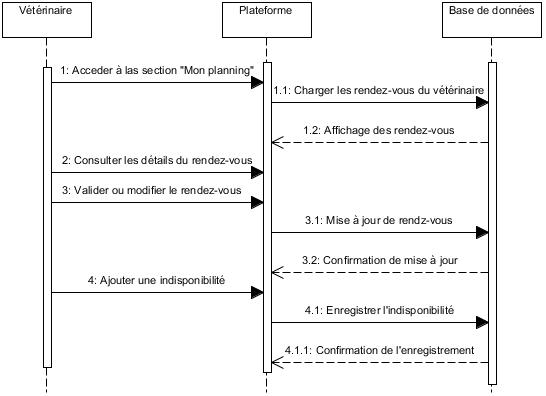


Figure 2.02.3: Diagramme de séquence de cas d'utilisation n°2

* **Cas d’utilisation n°3 « Consulter l’historique médical de son animal »**

La figure 2.04 représente le diagramme de séquence de cas d’utilisation « Consulter l’historique médical de son animal ».

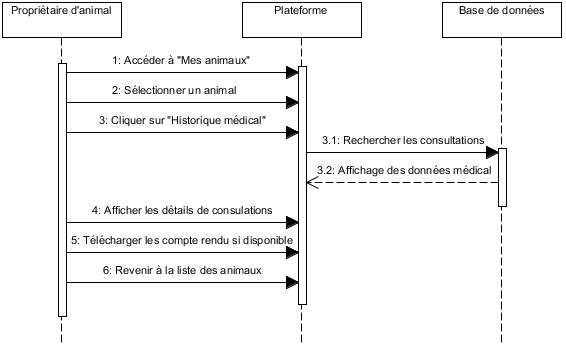
* **Cas d’utilisation n°4 « Gérer l’accès »**

Figure 2.02.4: Diagramme de séquence de cas d'utilisation n°3

La figure 2.05 représente le diagramme de séquence de cas d’utilisation « Consulter l’historique médical de son animal ».

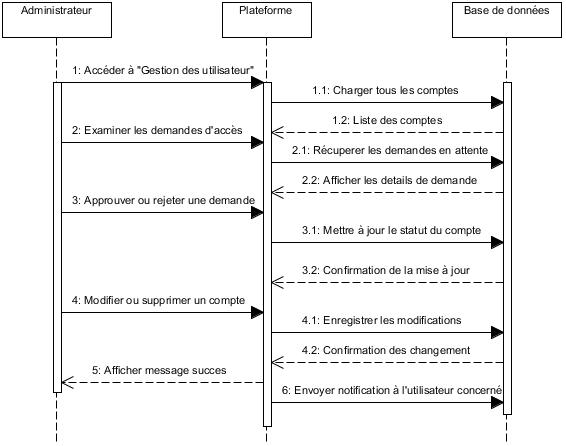


Figure 2.02.5: Diagramme de séquence de cas d'utilisation n°4

**2.3.4 Diagramme d’activité**

**2.3.4.1 Définition**

Les **diagrammes d’activité,** dans le cadre de la modélisation UML (Unified Modeling Language), constituent un outil fondamental pour décrire les **comportements dynamiques** d’un système. Ils permettent de représenter de manière claire les processus internes ou externes, en mettant en évidence la logique des enchaînements et les différentes étapes impliquées. Ce type de diagramme facilite la compréhension des flux de traitement, notamment dans les situations complexes où interviennent des décisions ou des parcours parallèles. Il s’avère ainsi particulièrement utile lors de l’analyse fonctionnelle, en amont du développement, pour formaliser et structurer les processus métier [2.03]

**2.3.4.2 Elément de diagramme d’activités**

Le diagramme d’activité en UML repose sur plusieurs éléments fondamentaux, notamment **: les activités, les transitions, les nœuds de contrôle** et **le nœud initial**.

**Activités**

Une activité représente un comportement global composé d’un ensemble structuré d’unités d’exécution appelées **actions**. Elle décrit donc une séquence logique d’actions réalisées dans le cadre d’un processus donné.

**Transitions**

Les transitions permettent de passer d’une activité à une autre. Elles sont représentées graphiquement par des flèches pleines reliant les différentes activités entre elles, illustrant ainsi le sens du déroulement du processus.

**Nœuds de contrôle**

Les nœuds de contrôle sont des éléments abstraits utilisés pour organiser et synchroniser les flux entre les différentes parties d’une activité. On distingue **sept types de nœuds de contrôle**, dont les plus courants sont :

* **Nœud initial**

Il marque le début du flux d’exécution d’une activité lorsqu’elle est déclenchée. Ce nœud est représenté graphiquement par un petit cercle noir plein.

* **Nœud de fin d’activité**

Lorsqu’une transition active ce nœud, l’exécution de l’ensemble de l’activité prend fin. Tous les flux et nœuds en cours dans cette activité sont alors interrompus. Ce nœud est représenté par un cercle vide contenant un petit cercle plein.

* **Nœud de bifurcation (ou fourche)**

Ce nœud divise un flux unique en plusieurs flux parallèles qui peuvent s’exécuter simultanément. Il est représenté par un trait horizontal ou vertical plein.

* **Nœud de décision**

Ce nœud permet de choisir entre plusieurs flux sortants selon les **conditions de garde** associées à chacun. Il possède un seul flux entrant et plusieurs flux sortants, l’un correspondant à la condition satisfaite, l’autre pouvant représenter un cas alternatif (par exemple "sinon").

**2.3.4.3 Diagramme d’activité de l’application**

* **Cas d’utilisation n°1 « Prendre rendez-vous »**

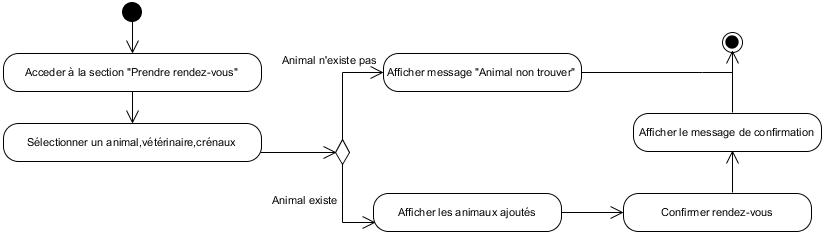
La figure 2.06 représente le diagramme d’activité de cas d’utilisation « Prendre rendez-vous ».

Figure 2.6: Diagramme d'activité de cas d'utilisation n°1

* **Cas d’utilisation n°2 « Gérer planning »**

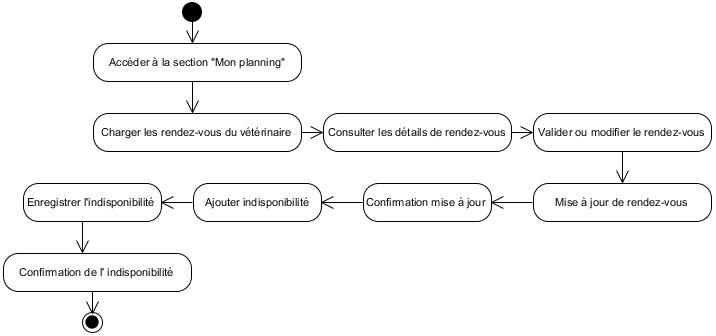
La figure 2.07 représente le diagramme d’activité de cas d’utilisation « Gérer planning ».

Figure 2.7: Diagramme d'activité de cas d'utilisation n°2

* **Cas d’utilisation n°3 « Consulter l’historique médical de son animal »**

La figure 2.08 représente le diagramme d’activité de cas d’utilisation « Consulter l’historique médical de son animal ».

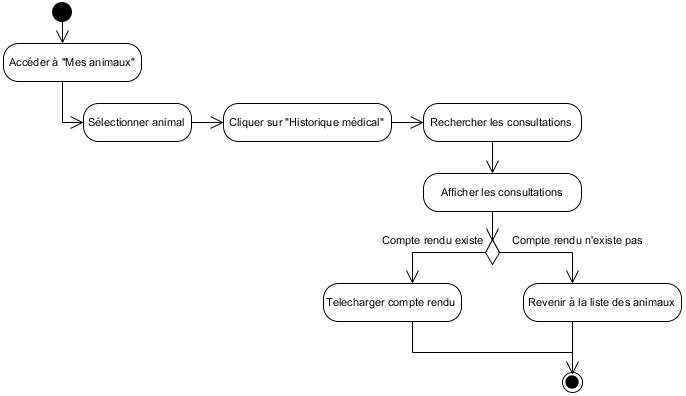
****

Figure 2.8: Diagramme d'activité de cas d'utilisation n°3

* **Cas d’utilisation n°4 « Gérer l’accès »**

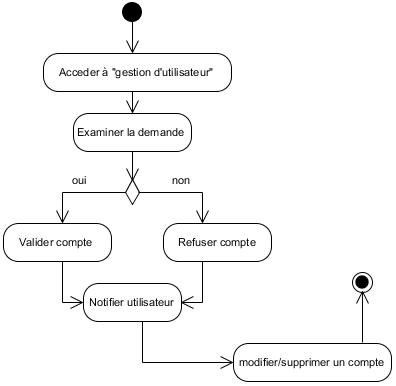
****La figure 2.09 représente le diagramme d’activité de cas d’utilisation « Gérer l’accès »

Figure 2.9: Diagramme d'activité de cas d'utilisation n°4

**2.3.5 Règle de gestion**

La règle de gestion est une contrainte métier qui définit les conditions, obligations ou restrictions qu’un système doit respecter. Elle exprime une logique fonctionnelle propre au domaine métier, indépendamment de l’implémentation technique. Elle peut concerner les données, les processus, ou les interactions entre les utilisateurs et le système.

Les règles de gestion applicables à notre plateformesont les suivantes :

* Un propriétaire d’animal peut enregistrer zéro ou plusieurs animaux.
* Un propriétaire d’animal peut prendre zéro ou plusieurs rendez-vous pour ses animaux.
* Un vétérinaire peut enregistrer zéro ou plusieurs consultations médicales par animal.
* Un **vétérinaire** peut gérer **zéro ou plusieurs rendez-vous.**
* Un animal peut avoir zéro ou plusieurs consultations médicales dans son historique.
* Un **administrateur** peut gérer **zéro ou plusieurs utilisateurs.**
* Un **utilisateur** peut avoir **un et un seul rôle** (propriétaire, vétérinaire ou administrateur).

**2.3.6 Dictionnaire de données**

Un dictionnaire des données est un document qui regroupe toutes les données utilisées dans le projet. Ces données sont stockées dans la base de données et sont représentées dans les modèles de diagrammes, notamment les diagrammes de classes, les modèles relationnels et les diagrammes de séquence.

Il sert de référence pour comprendre :

* La structure des entités manipulées par le système,
* Les types de données utilisés,
* Les relations entre les différentes entités.

Le tableau 2.01 présente le dictionnaire de données de notre plateforme

|  |  |  |  |
| --- | --- | --- | --- |
| **Nom de la classe** | **Attributs** | **Description** | **Types et tailles** |
| Utilisateur | id | Identifiant unique | Int(6) |
| email | Adresse email de l’utilisateur | Varchar(50) |
| password | Mot de passe de l’utilisateur | Varchar(100) |
| roles | Roles de l’utilisateur | Varchar(50) |
| **Nom de la classe** | **Attributs** | **Description** | **Types et tailles** |
| Propietaire | id | Identifiant unique | Int(6) |
| Nom | Nom du proprietaire | Varchar(50) |
| prenom | Prénom du proprietaire | Varchar |
| Contact | Contact du proprietaire | Varchar (20) |
| Adresse | Adresse du proprietaire | Varchar(20) |
| Vétérinaire | Id | Identifiant unique | Int(6) |
| Nom | Nom du vétérinaire | Varchar(50) |
| Prenom | Prenom du veterinaire | Varchar(50) |
| Contact | Contact du vétérinaire | Varchar(20) |
| Specialite | Specialité du vétérinaire | Varchar(20) |
| Administrateur | Id | Identifiant unique | Int(6) |
| Nom | Nom de l’administrateur | Varchar(50) |
| Prenom | Prenom de l’administrateur | Varchar(50) |
| Animal | Id | Identifiant unique | Int(6) |
| Nom | Nom de l’animal | Varchar(50) |
| Espece | Espèce de l’animal | Varchar(30) |
| Age | Age de l’animal | Int(2) |
| Sexe | Sexe de l’animal | Varchar(10) |
| Proprietaire\_id | Reference au proprietaire | Int(6) |
| RendezVous | id | Identifiant unique | Int(6) |
| Date\_rdv | Date et heures du rendez-vous | datetime |
| Motif | Motif du rendez-vous | Varchar(100) |
| Statut | Statut de rendez-vous | Varchar(30) |
| Animal\_id | Référence de l’animal | Int(6) |
| Veterinaire\_id | Reference de veterinaire | Int(6) |
| Consultation | id | Identifiant unique | Int(6) |
| Date\_consultation | Date de la consultation | Datetime |
| **Attributs** | **Description** | **Type et taille** |
| Consultation | Diagnostic | Diagnotsic | Longtext |
| Traitement | Traittement prescrit | Longtext |
| Animal\_id | Reference de l’animal | Int(6) |
| Veterinaire\_id | Reference de veterinaire | Int(6) |
| Planning | Id | Identifiant du planning | Int(6) |
| Veterinaire\_id | Référence au vétérinaire | Int(6) |
| Date\_debut | Date de début | datetime |
| Date\_fin | Date de fin | datetime |
| type | Type(dispo/indispo) | Varchar(30) |
| Notification | Id | Identifiant de la notification | Int(6) |
| Utilisateur\_id | Reference vers l’utilisateur | Int(6) |
| Date\_envoi | Date d’envoi | datetime |
| statut | Lu/non lu | Varchar(10) |

**Nom de la classe**

**2.3.7 Diagramme de classe**

**2.3.7.1 Définition**

Le diagramme de classes est un diagramme structurel statique utilisé pour représenter l’architecture interne d’un système orienté objet. Il permet de visualiser :

* Les classes, avec leurs attributs et méthodes (opérations),
* Les associations ou relations entre ces classes.

Ce type de diagramme est l’un des plus fondamentaux en UML. Il est indispensable dans tout processus de modélisation objet, car il constitue la base de la conception du système.

Une classe représente une entité abstraite regroupant des objets partageant les mêmes caractéristiques. Elle définit la structure (attributs) et le comportement (méthodes) des objets qui en seront issus. Elle joue ainsi le rôle de modèle ou de plan pour la création des objets dans l’application.

**2.3.7.2 Représentation des classes**

En UML, une **classe** est représentée sous forme d’un **rectangle**, également appelé **classeur**, divisé en **trois compartiments distincts** :

▪ Premier compartiment : le nom de la classe

* Il indique le **type d’objet** que la classe permet d’instancier.
* Il commence toujours par une **lettre majuscule**.
* Il est **centré** dans la partie supérieure du rectangle.
* Il est écrit en **gras** pour le distinguer.
* Si la classe est **abstraite** (c’est-à-dire qu’on ne peut pas en créer directement d’objets), le nom est affiché en **italique**.

▪ Deuxième compartiment : les attributs

* Les **attributs** représentent les **propriétés** ou **caractéristiques** associées à la classe.
* Ce sont des données élémentaires qui permettent de **décrire l’état** ou les **informations propres** à chaque instance de la classe.

▪ Troisième compartiment : les méthodes

* Les **méthodes** (ou opérations) définissent les **comportements** ou **fonctions** que la classe peut exécuter.
* Elles expriment la **logique métier** et décrivent **comment la classe interagit avec ses données** ou avec d’autres classes.

La figure 2.12 illustre la représentation d’une classe.

****

Figure 2.10: Representation du classe

**2.3.7.3 Relation entre classes**

Dans un diagramme de classes UML, plusieurs types de relations peuvent exister entre les classes :

**Association**

Lien entre deux classes qui montre qu’elles sont reliées entre elles. Elle indique combien d’objets participent à la relation (cardinalité).

**Agrégation**

Relation qui signifie qu’un objet peut **contenir** un ou plusieurs autres objets, mais ces objets peuvent exister indépendamment.

**Composition**

Relation plus forte que l’agrégation. Les objets liés **dépendent totalement** de l’objet principal et ne peuvent exister seuls.

**Généralisation**

Relation d’héritage entre une classe parent et une ou plusieurs classes enfants. Les classes enfants reprennent les attributs et méthodes de la classe parent.

**Dépendance**

Relation temporaire où une classe **utilise** une autre classe pour effectuer une tâche ou une action.

**2.3.7.4 Multiplicité**

**La cardinalité** indique combien d’objets d’une classe peuvent être associés à un objet d’une autre classe.  
Si aucune précision n’est donnée, la cardinalité par défaut est **1**.

Le tableau 2.02 illustre les multiplicités possibles

|  |  |
| --- | --- |
| **Cardinalités** | **Significations** |
| 1 | Un et un seul |
| 0…1 | Zéro ou un |
| N…N | De N à N(entiers naturels) |
| \* | De zéro à plusieurs |
| 0…\* |
| 1…\* | D’un à plusieurs |
| N | Exactement N(entier naturel) |

Le diagramme de classe se présente par l’ensemble des différentes classes relier entre eux.

La figure 2.13 représente le diagramme de classe pour la réalisation de notre plateforme

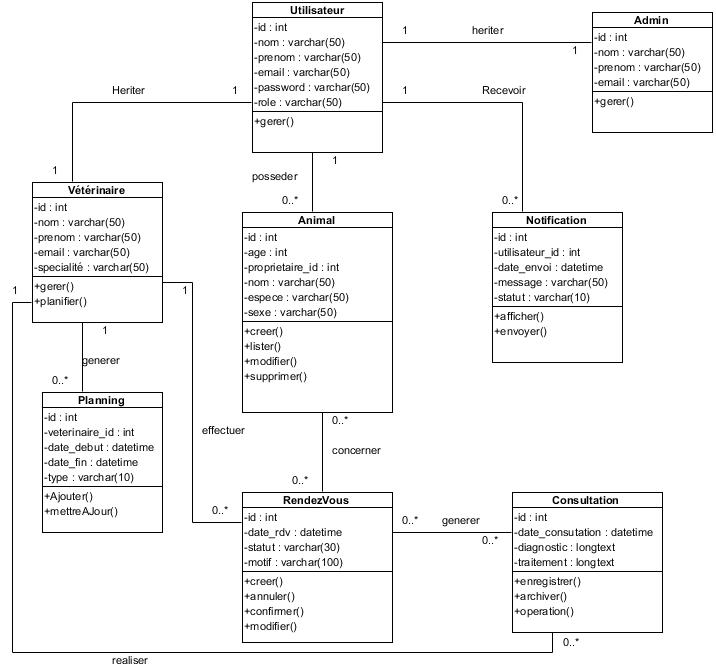
****

Figure 2.11: Diagramme de classe de l'application

**2.4 Conclusion**

Dans ce chapitre, nous avons présenté une analyse détaillée de notre plateforme, accompagnée d’une conception du système rigoureuse. Cette modélisation a été réalisée à l’aide de Visual Paradigm et du langage UML, illustré par différents diagrammes : cas d’utilisation, activités, classes et séquences. Le chapitre suivant abordera la phase de réalisation concrète de la plateforme, ainsi que les outils de développement retenus pour sa mise en œuvre.

# CHAPITRE 3 : REALISATION DU PROJET

**3.1 Introduction**

Dans ce chapitre, nous présentons la réalisation de notre plateforme de gestion vétérinaire, développée avec le stack MERN : MongoDB, Express.js, React.js et Node.js. Nous expliquerons comment les différentes fonctionnalités (gestion des animaux, rendez-vous, consultations, utilisateurs) ont été mises en place.

Chaque partie sera accompagnée d'exemples de code pour montrer concrètement la mise en œuvre. Nous verrons aussi comment les composants du frontend et du backend ont été intégrés pour assurer une bonne communication.

Enfin, nous évoquerons les tests effectués, les difficultés rencontrées, et les améliorations apportées pour garantir une application fluide, fonctionnelle et facile à utiliser.

**3.2 Langages utilisés**

**3.2.1 HTML**

HTML est le langage de balisage standard utilisé pour créer la structure des pages web. Il définit les éléments de contenu tels que les titres, les paragraphes, les images, les liens et les formulaires. HTML est un langage déclaratif simple à apprendre, accessible même aux débutants. Il sert de fondation au développement web, en s’intégrant étroitement avec CSS (pour la mise en forme) et JavaScript (pour l’interactivité). Le langage HTML se décrit comme un ensemble de balises ouvrantes et fermantes qui contiennent le contenu (texte, image, animation) et qui sont interprétées par le navigateur client (Internet Explorer, Netscape, Mozilla, Firefox, …). L’interprétation de certaines balises peut varier d’un navigateur à l’autre. (Ex: taille des textes).[3.01]

La figure 3.01 suivante montre le logo de HTML.

Figure 3.03.1: Logo du HTML

**Avantages**  
Les avantages clés de l'utilisation de HTML pour le développement web sont les suivants :

**Standard universel du web**  
HTML est le langage de balisage standard utilisé pour créer la structure des pages web. Il est compris par tous les navigateurs et constitue la base de tout site web.

**Simplicité d’apprentissage**  
HTML est facile à apprendre et à comprendre, même pour les débutants. Sa syntaxe est claire, ce qui en fait une excellente porte d’entrée dans le développement web.

**Compatibilité multiplateforme**  
Les pages HTML s’affichent correctement sur toutes les plateformes et tous les appareils, à condition que le navigateur soit à jour.

**Support large des navigateurs**  
Tous les navigateurs modernes prennent en charge HTML5, ce qui garantit une bonne compatibilité pour les applications web.

**Gratuit et open source**  
HTML ne nécessite aucun coût de licence ou outil propriétaire pour être utilisé. Il est totalement libre d’accès.

**Évolutivité avec HTML5**  
La dernière version, HTML5, introduit des éléments sémantiques, du multimédia natif (audio, vidéo), des formulaires avancés, etc., facilitant le développement moderne.

**Accessible pour les outils d’assistance**  
HTML permet de créer des sites accessibles, utiles pour les personnes en situation de handicap avec des lecteurs d’écran.

**3.2.2 JavaScript**

JavaScript est un langage de programmation qui permet d'implémenter des mécanismes complexes sur une page web. À chaque fois qu'une page web fait plus que simplement afficher du contenu statique Javascript affiche du contenu mis à jour à des temps déterminés, des cartes interactives, des animations, des menus défilants, ou autre, JavaScript a de bonnes chances d'être impliqué dans notre application web.

JavaScript est un langage de programmation du web et est aujourd’hui utilisé par plus de développeurs que tout autre langage de programmation.[3.02]

****La figure 3.02 suivante montre le logo de JavaScript.

Figure 3.2: Logo de JavaScript

**Avantages**

Les avantages de l'utilisation de JavaScript pour le développement web sont :

**Portabilité et compatibilité multi-navigateurs**

JavaScript est pris en charge par tous les navigateurs web modernes, ce qui permet d'assurer la compatibilité des applications web sur différents environnements.

**Programmation événementielle**

JavaScript permet une programmation événementielle, ce qui facilite la gestion des interactions utilisateur et la création d'applications web dynamiques.

**Productivité accrue**

Grâce à l'automatisation des tâches de build, de test et de déploiement, JavaScript permet d'améliorer la productivité des développeurs.

**Valeur ajoutée sur le marché de l'emploi**

La maîtrise de JavaScript est très recherchée par les entreprises, ce qui en fait un atout important pour les développeurs sur le marché de l'emploi.

**Interactivité côté client**

JavaScript permet de rendre les pages web interactives en contrôlant le comportement des éléments HTML et en réagissant aux actions de l'utilisateur, comme les clics, les survols de souris, les saisies de formulaire.

**Développement full-stack**

Avec l'émergence de Node.js, JavaScript peut désormais être utilisé non seulement côté client, mais aussi côté serveur, permettant ainsi un développement full-stack avec un seul langage.

**Écosystème riche et évolutif**

JavaScript bénéficie d'un écosystème très large et en constante évolution, avec de nombreuses bibliothèques, outils (React, Angular, Vue.js, jQuery, etc.) qui facilitent le développement.

**3.3 Technologies utilisés**

**3.3.1 CSS**

Le **CSS** *(*Cascading Style Sheets*)* est un langage de feuille de style qui permet de définir l’apparence visuelle des documents rédigés en **HTML** ou **XML**. Il sert à contrôler la mise en page, les couleurs, les polices, les marges et bien d’autres aspects esthétiques des éléments d’une page web. En séparant le contenu (HTML) de la présentation (CSS), il facilite la maintenance, la mise à jour et garantit une meilleure cohérence visuelle à travers l’ensemble du site.

La figure 3.03 suivante montre le logo de CSS



Figure 3.03.3: Logo du CSS

**Avantages**

L’utilisation du **CSS** offre de nombreux bénéfices, parmi lesquels :

**Séparation du contenu et de la présentation**  
CSS permet de dissocier l’aspect visuel du contenu HTML, ce qui facilite la gestion, la maintenance et la réutilisation des styles sur plusieurs pages.

**Allègement du code HTML**  
En évitant la répétition des styles dans le fichier HTML, CSS rend le code plus lisible, plus propre et plus léger.

**Uniformité de la présentation**  
Il garantit une apparence cohérente sur l’ensemble du site, avec la possibilité de modifier l’esthétique globale à partir d’un seul fichier.

**Meilleure accessibilité et adaptabilité**  
En séparant structure et design, CSS favorise l’accessibilité et permet d’adapter facilement l’interface aux différents écrans (responsive design).

**Contrôle avancé de la mise en page**  
Des outils comme Flexbox ou Grid offrent un contrôle précis sur la disposition des éléments, avec la possibilité d’ajouter des animations et transitions dynamiques.

**Amélioration des performances**  
En réduisant la taille des fichiers HTML et en chargeant les styles de façon optimisée, CSS contribue à accélérer le chargement des pages et à améliorer l’expérience utilisateur.

**3.3.2 Node.js**

Node.js est un outil gratuit et open source qui permet d'exécuter du JavaScript en dehors du navigateur web. [3.04] Il permet aux développeurs d'utiliser JavaScript pour écrire des applications côté backend, ce qui unifie le développement front-end et back-end avec un seul langage. Node.js est reconnu pour sa rapidité, son modèle événementiel non bloquant (asynchrone) et sa capacité à gérer un grand nombre de connexions simultanées. Grâce à son gestionnaire de paquets NPM, il offre un accès facile à des milliers de bibliothèques pour accélérer le développement. Node.js est particulièrement adapté aux applications en temps réel, comme les messageries instantanées ou les tableaux de bord interactifs. Il est largement utilisé dans les architectures modernes comme les stacks MERN ou MEAN.

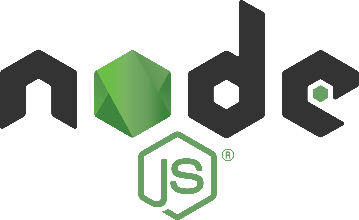
****La figure 3.03 suivante montre le logo de NodeJS.

Figure 3.03.4: Logo de Node.JS:

**Avantages**  
Les avantages clés de l'utilisation de Node**.**js pour le développement web sont les suivants :

**Simplicité et rapidité d’exécution**  
Node.js utilise le moteur V8 de Google, ce qui permet d’exécuter du JavaScript côté serveur de manière très rapide. Il est idéal pour les applications en temps réel.

**Architecture orientée événements**  
Grâce à son modèle non bloquant , Node.js permet de gérer un grand nombre de connexions simultanées sans ralentir, ce qui le rend performant pour les applications réseau.

**Langage unifié (JavaScript)**  
Node.js permet aux développeurs d’utiliser un seul langage, le JavaScript, pour le frontend et le backend. Cela simplifie le développement et améliore la productivité des équipes.

**Gratuit et open-source**  
Node.js est libre d’utilisation et possède un code source ouvert, avec une communauté active qui contribue régulièrement à son amélioration.

**Écosystème riche (npm)**  
Node.js dispose de npm (Node Package Manager), l’un des plus grands écosystèmes de bibliothèques open-source, facilitant la réutilisation et l’intégration de modules tiers.

**Haute scalabilité**  
Node.js est conçu pour développer des applications évolutives, notamment grâce à sa gestion asynchrone des E/S et son architecture légère.

**Multiplateforme**  
Node.js fonctionne sur différents systèmes d’exploitation comme Windows, Linux et macOS, ce qui le rend très flexible pour le déploiement.

**Support des API et microservices**  
Il est très adapté à la création d’API RESTful et d’architectures en microservices grâce à sa légèreté et sa rapidité.

**3.4 Framework**

**3.4.1 Express.js**

Express**.**js est un framework web open-source basé sur Node**.**js, utilisé pour développer des applications web et des API rapides et efficaces. Il offre une structure légère et minimaliste qui simplifie la gestion des routes, des requêtes HTTP et des middlewares.

Grâce à sa flexibilité, Express.js permet aux développeurs de personnaliser leur architecture et d’ajouter uniquement les composants nécessaires. Il est compatible avec tous les systèmes prenant en charge Node.js (Windows, macOS, Linux), ce qui en fait un outil multiplateforme.

Express.js est particulièrement adapté au développement d’applications web modernes, d’APIRESTful et de services connectés au cloud, tout en garantissant de bonnes performances et une mise en œuvre rapide. Express.js est un framework qui simplifie l’ecriture du coté serveur. Le framework Express permet de definir des routes et des specifications sur l’action à entreprendre lorsqu’une requete HTTP correspondant à un certain modèle arrive. [3.04]

La figure 3.04 suivante montre le logo de Express.js

Figure 3.03.5: Logo de Express.js

**Avantages**

L'utilisation d’Express**.**js présente plusieurs avantages pour le développement web :

**Simplicité et légèreté**  
Express.js offre une structure minimaliste qui permet de créer rapidement des applications web et des API sans complexité excessive.

**Flexibilité**  
Il ne dicte pas une architecture stricte, ce qui laisse aux développeurs la liberté d’organiser leur code selon leurs besoins.

**Gestion facile des routes**  
Express simplifie la définition des routes HTTP, ce qui facilite la création d’interfaces RESTful claires et efficaces.

**Support des middlewares**  
Il permet d’utiliser une série de middlewares pour gérer les requêtes, les réponses, l’authentification, les erreurs, etc.

**Idéal pour les API REST**  
Express est particulièrement adapté à la création d’API modernes, utilisées dans les architectures orientées services ou les applications SPA (Single Page Applications).

**3.4.2 React avec Vite**

React avec Vite constitue une solution moderne et performante pour le développement d’interfaces utilisateur dynamiques. Vite est un outil de build rapide qui optimise le processus de développement avec React, en offrant un temps de démarrage quasi instantané et une actualisation à chaud ultra-rapide.

Grâce à son architecture légère et modulaire, Vite permet aux développeurs de profiter d’un environnement flexible, tout en simplifiant la configuration du projet React. Il est compatible avec tous les systèmes prenant en charge Node.js, ce qui en fait un outil multiplateforme.

Cette combinaison est particulièrement adaptée à la création d’applications web interactives et réactives, avec un chargement rapide, une productivité accrue et une excellente expérience développeur.

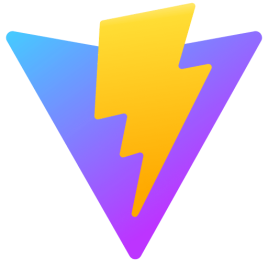
La figure 3.05 suivante montre le logo de React avec Vite

Figure 3.6: Logo de React avec Vite

**Avantages**

L’utilisation de React avec Vite offre de nombreux atouts pour le développement d’applications web modernes :

**Démarrage ultra-rapide**  
Vite utilise un serveur de développement basé sur ES Modules, ce qui permet un temps de démarrage quasi instantané, même pour de gros projets.

**Configuration simplifiée**  
Vite fournit une configuration de projet prête à l’emploi pour React, réduisant le besoin de réglages complexes comme avec Webpack.

**Modularité et extensibilité**  
Vite prend en charge les plugins et permet une personnalisation facile selon les besoins du projet.

**Expérience développeur améliorée**  
Le combo React + Vite offre un environnement fluide, réactif et moderne qui facilite le développement rapide d'interfaces.

**Multiplateforme**  
Fonctionne sur tous les systèmes compatibles avec Node.js (Windows, macOS, Linux).

**3.4.3 Tailwind CSS**

Tailwind CSS est un framework CSS qui simplifie le développement web grâce à des classes utilitaires prédéfinies pour assurer la mise en forme des éléments HTML. Il permet une personnalisation complète qui peut se marier très bien avec tous les plateformes web. Il présente l’avantage d’être moins encombrant que les autres frameworks CSS, de ce fait, il n’alourdit pas le code à l’exécution. TailwindCSS s’intègre très bien avec les frameworks JavaScript populaires tout en accélérant le temps de développement pour créer des interfaces utilisateurs web élégantes et réactives. [3.05]

La figure 3.06 suivante montre le logo de TailwindCSS

Figure 3.07: Logo de TailwindCSS

**Avantages**

L'utilisation de **Tailwind CSS** offre de nombreux bénéfices pour le développement front-end :

**Approche utility-first**  
Tailwind propose des classes utilitaires pour appliquer directement des styles dans le HTML, ce qui accélère le développement et évite la création excessive de fichiers CSS personnalisés.

**Personnalisation facile**  
Il est entièrement configurable : on peut modifier les couleurs, tailles, espacements, polices, etc., via un fichier de configuration (tailwind.config.js).

**Productivité accrue**  
Les développeurs peuvent construire des interfaces rapidement sans quitter le fichier HTML ou JSX, ce qui améliore le flux de travail.

**Design cohérent**  
Grâce à son système de design basé sur des classes prédéfinies, Tailwind garantit une interface homogène et facilement maintenable.

**Communauté active et écosystème riche**  
Tailwind bénéficie d’une large communauté, de plugins officiels, de bibliothèques de composants (comme Tailwind UI) et d’une documentation claire.

**Compatibilité avec les frameworks modernes**  
Il s’intègre facilement dans des environnements comme **React, Vue, Next.js, Vite**, etc., facilitant la construction d’interfaces modernes.

**3.5 Logiciels utilisés**

**3.5.1 Visual Studio code**

Nous avons utilisé Visual Studio Code comme éditeur de code, car il s’agit d’un outil open-source développé par Microsoft, reconnu pour sa légèreté, sa rapidité et sa facilité de prise en main. Il prend en charge une grande variété de langages grâce à un vaste écosystème d’extensions.

Visual Studio Code offre des fonctionnalités avancées telles que la coloration syntaxique, l’auto**-**complétion, le débogageintégré et la prise en charge du terminal, tout en étant multiplateforme (Windows, macOS, Linux).

La figure 3.07 suivante présente le logo officiel de Visual Studio Code.

Figure 3.08: Logo de VSCode

**Avantages**

**Gratuit et open-source**  
VS Code est entièrement gratuit, développé par Microsoft, et bénéficie d’une communauté active qui contribue à son amélioration continue.

**Multiplateforme**  
Il fonctionne sur Windows, macOS et Linux, ce qui permet une utilisation cohérente sur différents systèmes.

**Large prise en charge des langages**  
Grâce aux extensions, VS Code supporte un grand nombre de langages (JavaScript, Python, Java, C++, HTML, etc.).

**Extensions et personnalisation**  
Un marketplace riche permet d’ajouter des thèmes, des outils de développement, des linters, des snippets et bien plus.

**Fonctionnalités intelligentes**  
Il propose l’auto-complétion, la coloration syntaxique, l’intelliSense, et la navigation rapide dans le code.

**Terminal intégré**  
Il permet d’exécuter des commandes directement dans l’éditeur via un terminal embarqué, sans avoir à basculer entre fenêtres.

**Intégration Git native**  
L’éditeur intègre des outils Git pour le suivi de version, la gestion des branches, des commits et des conflits.

**3.5.2 Postman**

Postman est une application permettant de tester des API. Postman simplifie chaque étape du cycle de vie des API et rationalise la collaboration afin que nous puissions créer des API plus rapidement. Il sert à exécuter des appels HTTP directement depuis une interface graphique. [3.06]

Postman offre des fonctionnalités avancées telles que la création de collections de requêtes, le test automatisé, la gestion des environnements, ainsi que l’historique des requêtes, ce qui facilite grandement le développement et le débogage des API REST.

La figure 3.08 suivante présente le logo officiel de Postman.

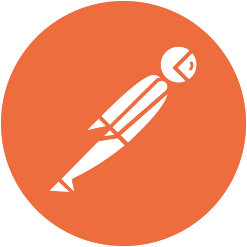
**Avantages**

Figure 3.09: Logo officiel de Postman

**Interface intuitive et conviviale**  
Postman propose une interface graphique simple d’utilisation, permettant aux développeurs de créer, envoyer et tester des requêtes HTTP sans écrire de code complexe.

**Support complet des requêtes HTTP**  
Il prend en charge les méthodes courantes telles que GET, POST, PUT, DELETE, PATCH, ainsi que l’ajout d’en-têtes, de paramètres, de corps de requête, etc.

**Gestion des environnements**  
Postman permet de définir des environnements (dev, test, prod) avec des variables pour automatiser et adapter facilement les tests à différents contextes.

**Création et exécution de tests automatisés**  
Des scripts peuvent être ajoutés pour tester automatiquement les réponses (code de statut, corps JSON, en-têtes, etc.).

**Organisation en collections**  
Les requêtes peuvent être regroupées en collections pour une meilleure organisation, partage ou documentation d’un ensemble d’API.

**Documentation automatique des API**  
Postman peut générer automatiquement une documentation claire et interactive à partir des collections.

**Support du travail collaboratif**  
Grâce au cloudPostman, plusieurs développeurs peuvent travailler sur les mêmes collections en temps réel, avec un contrôle de version.

**Multiplateforme**  
Disponible sur Windows, macOS, Linux et via une version web, ce qui le rend accessible depuis presque n’importe quel environnement.

**3.5.3 MongoDB Compass**

Nous avons utilisé **MongoDB Compass** comme interface graphique pour interagir avec le **SGBD NoSQL MongoDB**, en raison de sa simplicité d’utilisation, de ses fonctionnalités avancées et de sa compatibilité multiplateforme. MongoDB est la base de données utilisée dans le MERN Stack. Il s’agit d’une base de données NoSQL orientée document, dotée d’un schéma flexible et d’un langage de requête JSON. De nombreuses entreprises modernes (dont Facebook et Google) utilisent MongoDB en production, mais des entreprises plus anciennes, telles que SAP et Royal Bank of Scotland, ont également adopté MongoDB. [3.07]

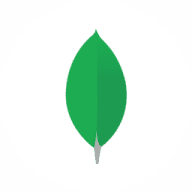
****La **figure 3.08 suivante** illustre le **logo officiel de MongoDB Compass.**

Figure 3.10: Logo de MongoDB Compass

**Avantages**

**Interface graphique intuitive**  
MongoDB Compass permet d’explorer les bases de données MongoDB via une interface visuelle conviviale, sans avoir à écrire de commandes.

**Visualisation des données en temps réel**  
Il offre une visualisation claire des documents, collections et structures JSON, ce qui facilite la compréhension des données stockées.

**Création de requêtes sans code**  
Les requêtes peuvent être construites visuellement à l’aide de filtres, ce qui est utile même pour les utilisateurs non expérimentés.

**Analyse de la performance**  
Compass propose des outils pour inspecter les performances des requêtes, surveiller les index et optimiser la base de données.

**Gestion des index simplifiée**  
Il permet de créer, modifier ou supprimer des index facilement, ce qui améliore l’efficacité des recherches dans les collections.

**Support des opérations CRUD**  
L’utilisateur peut effectuer directement les opérations de lecture, insertion, mise à jour et suppression sur les documents.

**Multiplateforme**  
L’outil fonctionne sur Windows, macOS et Linux, ce qui le rend accessible à tous les développeurs.

**3.5.4 Visual Paradigm**

Nous avons utilisé Visual Paradigm comme outil de modélisation UML pour concevoir et documenter les différentes parties de notre système. Il s’agit d’un logiciel puissant, multiplateforme et largement utilisé dans le domaine de l’ingénierie logicielle pour la création de diagrammes et la modélisation des processus métiers.

Visual Paradigm permet de générer facilement des diagrammes UML tels que les casd’utilisation, diagrammes de classes, de séquence, d’activités, et bien d’autres encore. Il offre également des fonctionnalités avancées telles que la gestion de projet, la documentation automatisée, ainsi que la collaboration en ligne via des espaces de travail partagés.

La figure 3.09 suivante présente le logo officiel de Visual Paradigm.

Figure 3.11: Logo de Visual Paradigm

**Avantages**

**Outil de modélisation complet**  
Visual Paradigm prend en charge une large gamme de diagrammes UML (cas d'utilisation, classes, séquence, activités, etc

**Interface intuitive et ergonomique**  
Son interface est conviviale, avec des outils de glisser-déposer et une personnalisation facile des éléments graphiques.

**Intégration avec les environnements de développement**  
Visual Paradigm peut s'intégrer avec des IDE comme Eclipse, NetBeans, IntelliJ et des outils de gestion comme JIRA.

**Support multiplateforme**  
Fonctionne sous Windows, macOS et Linux, ce qui le rend accessible à tous les développeurs.

**Versions communautaire et professionnelle**  
Propose une version gratuite pour l’apprentissage et des versions payantes avec des fonctionnalités avancées pour les entreprises.

**3.5.5 Brave Browser**

**Brave Browser** est un navigateur moderne, rapide et sécurisé, qui s’est imposé comme une alternative efficace pour le développement et la navigation web. Reposant sur Chromium (comme Google Chrome), il est compatible avec la majorité des technologies web actuelles, tout en offrant de meilleures performances en matière de confidentialité.

Pour lancer et tester les interfaces de notre projet, **Brave** s’est révélé parfaitement adapté grâce à sa compatibilité avec les outils de développement Chrome, son efficacité, et son support des serveurs locaux.

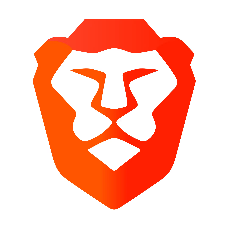
****La **figure 3.12 suivante** présente le **logo officiel de Brave Browser**.

Figure 3.12: Logo de Brave

**Avantages**

Parmi les principaux avantages de Brave, on peut citer :

**Performances élevées**  
Navigation fluide avec un moteur rapide basé sur Chromium.

**Respect de la vie privée**  
Blocage natif des publicités, des trackers et des scripts nuisibles.

**Compatibilité avec les outils développeur**  
Accès à la console, l’inspecteur d’éléments et aux autres outils DevTools comme sur Chrome.

**Consommation optimisée des ressources**  
Utilise moins de mémoire et de CPU dans certains cas que les navigateurs concurrents.

**Multiplateforme**  
Disponible sur Windows, macOS, Linux, Android et iOS.

**Support des extensions Chrome**  
Compatible avec presque toutes les extensions du Chrome Web Store.

**3.6 Présentation des interfaces graphiques**

**3.6.1 Planning de disponibilités**

L’interface de planning de disponibilités est une page dédiée à la gestion des créneaux horaires des vétérinaires. Elle permet à chaque vétérinaire d’ajouter, modifier ou supprimer ses plages de disponibilité afin de permettre la prise de rendez-vous cohérente avec son emploi du temps. L’interface assure aussi que seuls les vétérinaires authentifiés puissent accéder et manipuler leurs propres créneaux.

Cette page joue un rôle crucial dans la régulation des rendez-vous au sein de la plateforme, en empêchant les conflits d’horaire et en facilitant la planification automatique. En outre, elle offre une visualisation claire des créneaux existants à travers un calendrier interactif.

La figure 3.13 suivante illustre l’interface de gestion du planning de disponibilités.

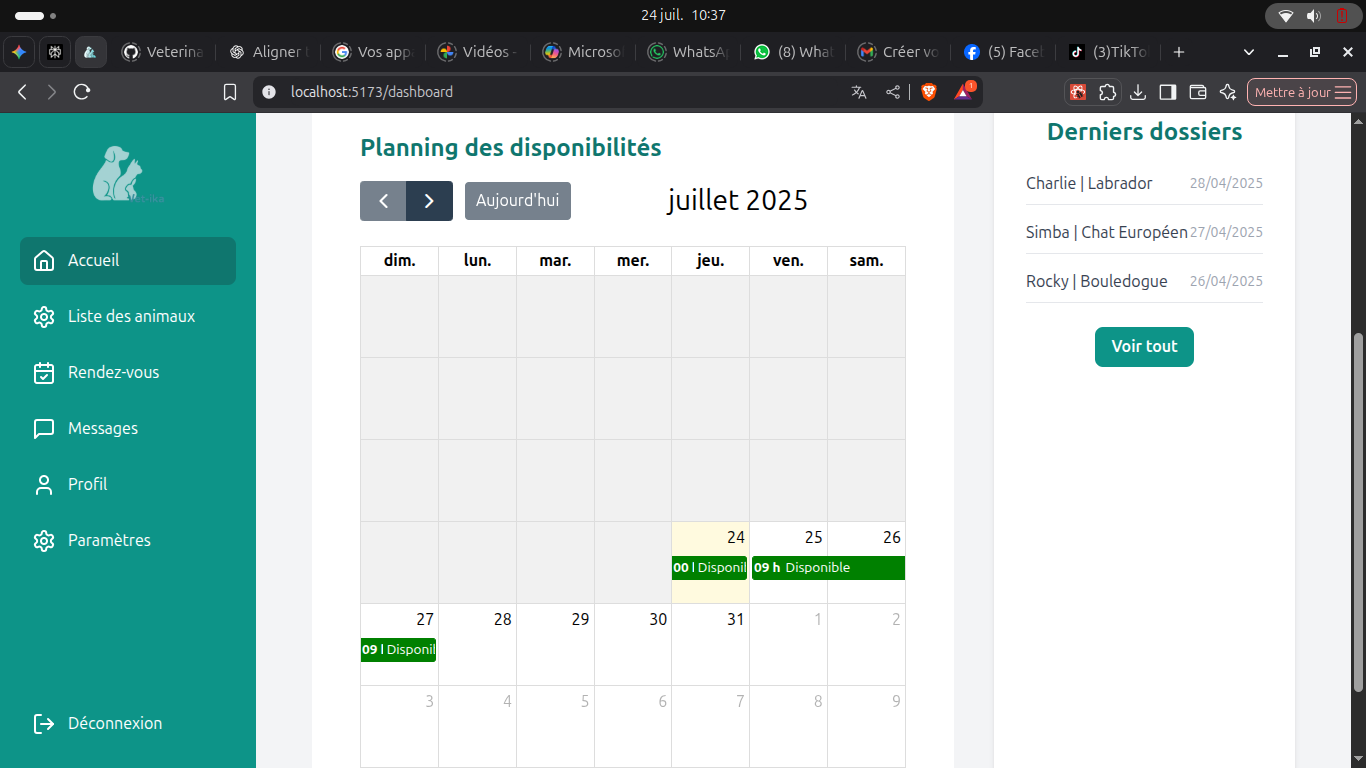


Figure 3.13: Planning de disponibilités

**3.6.2 Profile d’utilisateur**

L’interface du profil utilisateur permet à chaque utilisateur de consulter et de modifier ses informations personnelles. En fonction du rôle (administrateur, vétérinaire ou propriétaire d’animal), cette page affiche des données spécifiques telles que :

* Les informations d’identité (nom, prénom, email, rôle),
* La liste des animaux (pour les propriétaires),
* Les rendez-vous programmés (pour tous les rôles, avec affichage adapté),

Chaque utilisateur peut modifier ses informations personnelles, telles que le nom d’utilisateur ou la photo de profil, et consulter ses interactions passées avec la plateforme.

Cette interface centralise les éléments liés à l'utilisateur connecté et constitue un point d'accès rapide à ses activités et responsabilités au sein du système.

La figure 3.14 suivante illustre l’interface du profil utilisateur.

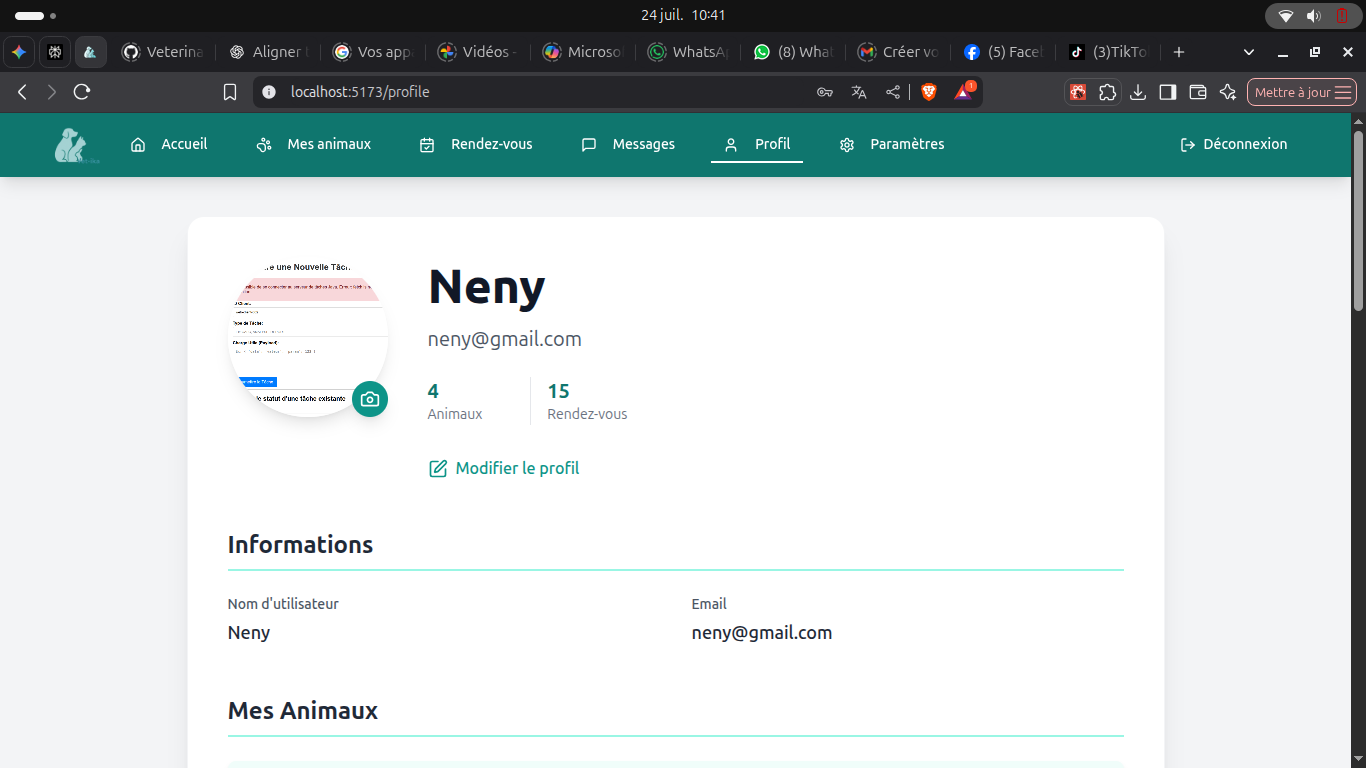


Figure 3.14: Profile d'utilisateur

**CONCLUSION ET PERSPECTIVE**

La mise en place de notre plateforme de gestion vétérinaire constitue une solution efficace pour améliorer et faciliter la prise en charge des animaux au sein d’un cabinet vétérinaire. Les méthodes traditionnelles de gestion des dossiers, de planification des consultations et des rendez-vous présentent des limites telles que la perte d’informations, le manque de centralisation et la complexité de suivi des traitements.

Nous avons d’abord réalisé une étude approfondie des besoins réels dans les cabinets vétérinaires, en analysant les contraintes de gestion des rendez-vous, de suivi médical des animaux et d'interaction avec les propriétaires. Ensuite, nous avons conçu un modèle conceptuel basé sur l’UML (Unified Modeling Language), avec une analyse fonctionnelle complète représentée par des cas d’utilisation, des diagrammes d’activités, de séquence, et de classes.

L’environnement technique choisi pour la réalisation repose sur la stack MERN (MongoDB, Express.js, React.js, Node.js), développée sous Visual Studio Code avec un serveur local pour les tests. Nous avons illustré notre projet à travers une arborescence claire et des interfaces intuitives pour les vétérinaires, les administrateurs et les clients.

Notre ambition à travers cette application est de fournir une plateforme complète pour la gestion des animaux, des rendez-vous, des consultations et du suivi médical. En perspective, nous prévoyons d’intégrer un module de vente de médicaments directement depuis la plateforme, accompagné d’un système de gestion de stock, afin de faciliter la prescription et le suivi des traitements. Ces évolutions permettront aux cabinets vétérinaires de mieux gérer leurs ressources et d’offrir un service plus complet et digitalisé à leurs clients.

**ANNEXES**

**Annexes 1 Quelques interfaces graphiques**

* **Message entre vétérinaire et propriétaire d’animal**

Après avoir pris un rendez-vous, le propriétaire de l’animal peut envoyer un message au vétérinaire via une interface de messagerie intégrée. Cette fonctionnalité permet d’assurer une meilleure communication entre les deux parties, notamment pour poser des questions, envoyer des documents ou recevoir des conseils.

Le vétérinaire peut également répondre directement depuis son espace personnel, facilitant ainsi le suivi des patients et des échanges liés aux consultations.

La figure A.29 suivante illustre la page de messagerie entre le propriétaire et le vétérinaire.

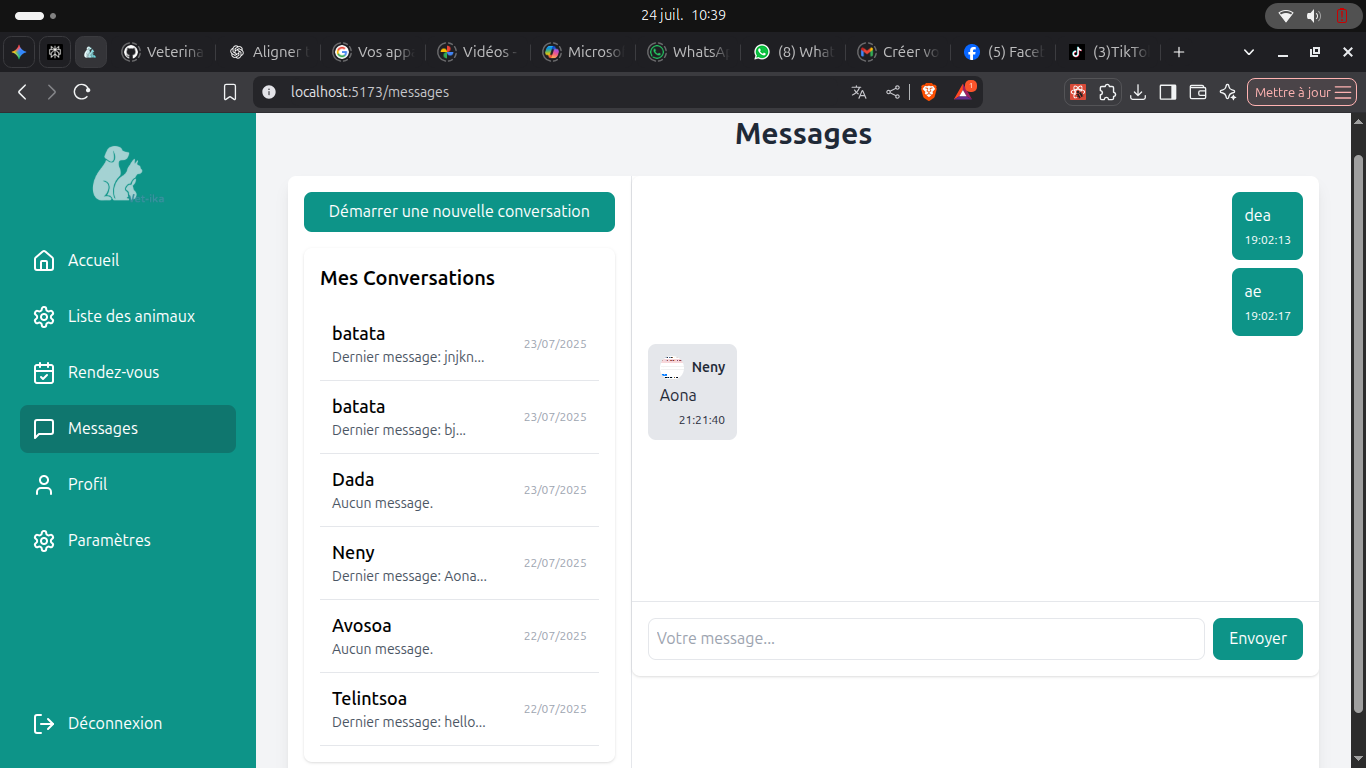


Figure A:3.15: Message

**REFERENCES**

**FICHE DE RENSEIGNEMENTS**