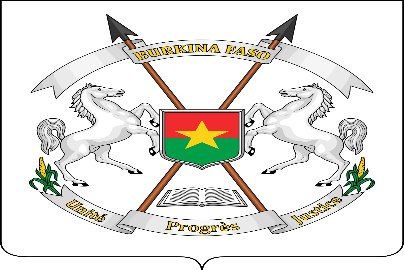
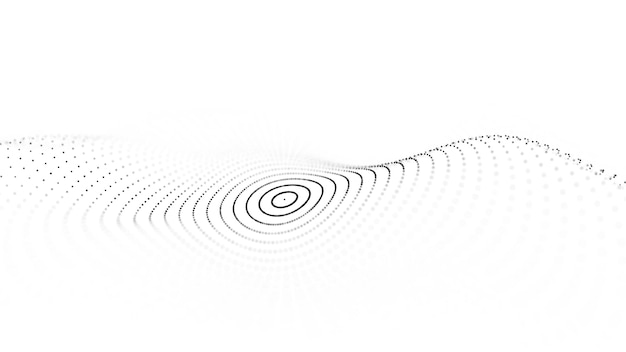
,  

Mémoire présenté en vue de l'obtention partielle du diplôme de Bachelor of Science en Informatique et Entrepreneuriat.



**Directeur de thèse :** **Dr Nicolas pegdwende Sawadogo**

**Co -directeur de thèse :** **Dr Nicolas pegdwende Sawadogo**

Presented and publicly defended by:

**Zida Assomption Désiré Wendlasida**

**Année Académique : 2023-2024**

Dédicaces

Remerciements

Resumé

**TABLE DE MATIERES**

**PARTI I ETUDES THEORIQUES**

**CHAPITRE 1 : Informations Générales sur le projet**

1. **Contexte D’étude**

Le marché des animaux d’élevage connait actuellement une croissante rapide, caractérisée par une demande accrue pour des services diversifiés. Cette demande rassemble non seulement des soins de santé et d’hygiène pour les animaux, mais aussi des produits de vente variés et des conseils professionnels adaptés aux besoins des éleveurs. Ce phénomène est alimenté par une prise de conscience croissante de l’importance du bien-être animal, ainsi que par une exigence accrue de qualité et de traçabilité des produits animaux de la part des consommateurs.

Dans ce contexte, les éleveurs se trouve face à des défis majeurs pour répondre à des obligations tout en gérant convenablement leurs opérations quotidiennes. Ils ont besoin de solutions qui facilitent non seulement la gestion de santé et du bien-être de leur animaux, mais aussi la commercialisation de leurs produits et l’accès à des conseils spécialisés.

Une plateforme intégrée, telle que « AnimalCare » pourrait jouer un rôle important en répondant de manière plus efficace et accessible à ces besoins variés. En misant sur les services de vente d’hygiène de santé et de conseil, cette plateforme permettrait d’optimiser les interactions entre éleveur, client et fournisseur, créant ainsi un écosystème plus conforme et fonctionnel.

1. Problématique

Comment une plateforme intégrée comme « Animal Care » peut-elle amélioré la vente des animaux d’élevage tout en optimisant les services offerts aux éleveurs, tout en tout en optimisant les interactions entre éleveurs, clients et fournisseurs et en facilitant la commercialisation des produits ?

1. **Objectifs**

**III.1. Objectif Général**

Evaluer l’impact d’une plateforme intégrée sur la Santé et le bien-être des animaux d’élevage ainsi que sur l’efficacité des interactions et des transactions entre éleveurs, client, et fournisseurs.

**III.2 Objectifs spécifiques**

* Analyser les avantages commerciaux d’une telle plateforme pour les fournisseurs de services et produits pour animaux.
* Evaluer la satisfaction des éleveurs, des clients et fournisseurs utilisant la plateforme « AnimalCare »
* Identifier les défis et les obstacles à l’implémentation d’une plateforme intégrée.
* Proposer des améliorations basées sur les retours d’expérience des utilisateurs.

1. **Etat de l’art**

**IV.1. Le concept de santé et de Bien-être des animaux d’élevage**

L’élevage moderne vise à renforcer la production animale tout en assurant le bien-être des animaux tout au long de leur vie. La santé et le bien-être des animaux d’élevage sont des concepts dispensables qui influencerons directement la quantité et la productivité des produits animaux. Dans cette section, nous explorerons en détails les notions de santé et de bien-être des animaux d’élevage en mettant l’accent sur l’importance d’une bonne gestion pour assurer la qualité et la productivité des produits animaux.

**IV.2. Sante des animaux d’élevage**

La santé des animaux d’élevage est importante pour prévoir une production optimale. Voici quelques aspects de la gestion de la santé animale :

* + Prévention et Traitement des Maladies : La prévention des maladies par la vaccination, les soins vétérinaires réguliers et les pratiques d’hygiène est fondamentale. Une fois les maladies identifiées, un traitement efficace et rapide essentiel pour éviter leur propagation et minimiser les pertes économiques.
  + Conditions de Vie Adéquates : Les conditions de logement des animaux doivent être propres, sécurisées et confortables. Un bon espace de vie réduit le stress et le risque de maladies.
  + Alimentation Equilibrée : Une bonne nutrition est cruciale pour la croissance, la reproduction et la production dans animaux. Les régimes doivent être équilibres et répondre aux besoins spécifiques des différentes espèces.

**IV.3.**  **Bien-être des animaux d’élevage**

Le bien-être des animaux d’élevage va au-delà de la simple physique et inclut leur état mental et émotionnel. Voici les éléments majeurs :

* Les cinq Libertés : Le bien-être animal est souvent évaluer selon les « cinq libertés »
* Liberté de la faim et de la soif : les animaux doivent avoir accès a l’eau et a une nutrition adéquate.
* Liberté de l’inconfort : il faut fournir un environnement ayant un abri et une zone de repos propre.
* Liberté de la douleur, des blessures et des maladies :il faut une prévention et traitement rapide.
* Liberté d’exprimer des comportements normaux : il faut un espace suffisant et des conditions appropriées.
* Liberté de la peur et de la détresse : Eviter les conditions de stress et assurer des soins appropriés.

**IV.4. Importance D’une bonne Gestion pour assurer la qualité et la productivité**

Une bonne gestion de la santé et du bien-être animal est essentielle pour maintenir des standards élevés de qualité et de productivité. Voici une bonne gestion qui peut avoir un impact positif :

* Amélioration de la Productivité :

Des animaux en bonne santé et bien traité sont plus productifs. Par exemple, des poules pondeuses bien entretenues produisent plus d’œufs, ainsi de même qu’une vache qui produira plus de lait si elle se retrouve dans de bonne condition. Une meilleure croissance des animaux donne un taux élevé de reproductions et une meilleure qualité de la viande.

* Qualité des produits :

Le bien-être des animaux a un impact direct sur la qualité des produit animaux.la viande provenant d’animaux stressés ou malades est souvent de qualité inférieure. De même, le lait et les œufs produit dans de mauvaises conditions peuvent être contaminés et de moindre qualiteuse bonne gestion assure des produits plus sains et de meilleure qualité pour les consommateurs.

* Réduction des couts :

Un bon programme de santé et bien-être animal peut réduire les couts vétérinaires et les pertes dues aux maladies. Moins d’animaux malades signifie moins de dépenses en médicaments et traitements, ainsi qu’une réduction des pertes de production. Cela conduit à une amélioration de l’efficacité économique de l’élevage.

* Acceptation Social et Règlementaire :

Dans de nombreux pays, le bien-être animal est devenu un enjeu sociétal majeur. Les consommateurs sont de plus en plus soucieux des conditions dans lesquelles les animaux sont élevé. Une gestion respectueuse du bien-être animal peut améliorer l’image de marque des produits et répondre aux attentes des consommateurs et des régulateurs. Cela peut également aider à accéder a de nouveau marchés ou les normes de bien-être animal sont strictes.

* Durabilité :

Le bien-être animal contribue également à la durabilité de l’élevage. Des pratiques d’élevages respectueuses de l’animal sont souvent en ligne avec des pratiques respectueuses de l’environnement, réduisant ainsi l’empreinte écologique de l’élevage. Par exemple des pratiques qui minimise le stress animal peuvent également réduire les besoins en ressources et l’impact environnemental.

**IV.5. Revue des études existantes sur les plateformes intégrées pour l’élevage**

L’intégration des plateformes numériques dans l’industrie de l’élevage a connu une montée en popularité, répondant à la demande croissante de solutions innovantes pour améliorer la gestion des élevages, la productivité et le bien-être animal. Cette revue met en lumière les bénéfices observés et les défis rencontrés dans cette transformation numérique, en s’appuyant sur des études et recherches antérieures.

**IV.6 Amélioration de la Gestion et de la Productivité**

1. **Optimisation des Ressources**

Les plateformes intégrées jouent un rôle crucial dans l'optimisation de la gestion des ressources, notamment en ce qui concerne l'alimentation et l'eau. Elles fournissent des données en temps réel qui permettent aux éleveurs de prendre des décisions éclairées. Par exemple, en Australie, l'utilisation de capteurs connectés pour surveiller les besoins nutritionnels des troupeaux a permis de réduire les coûts d'alimentation de 15 % tout en augmentant la productivité. Cette optimisation se traduit par une utilisation plus efficace des ressources et une amélioration des rendements.

1. **Suivi et Traçabilité**

Grâce aux technologies telles que l’identification par radiofréquence (RFID) et les systèmes de gestion de troupeaux, les éleveurs peuvent suivre chaque animal individuellement, facilitant ainsi la traçabilité et améliorant la gestion sanitaire. Une étude menée en Europe a révélé que l'amélioration de la traçabilité a réduit les épidémies de maladies de 30 %. Ces technologies permettent un suivi rigoureux de l’état de santé des animaux et assurent la conformité avec les réglementations sanitaires.

1. **Amélioration de la Santé et du Bien-Être des Animaux**

**V.1. Prévention des Maladies**

L'utilisation de plateformes intégrées, combinant des capteurs biométriques et des algorithmes de machine Learning, joue un rôle crucial dans la prévention des maladies. Ces technologies permettent de surveiller en continu les indicateurs de santé des animaux, identifiant les risques avant l'apparition des symptômes. Une étude réalisée en Amérique du Nord a révélé que l'intégration de ces outils numériques a permis de réduire la mortalité des veaux de 20 %. Cela met en évidence l'importance des technologies avancées pour une gestion proactive et efficace de la santé animale, minimisant ainsi les risques de maladies graves.

**V.2. Réduction du Stress**

La surveillance constante et l’environnement contrôlé offerts par les plateformes intégrées permettent de minimiser les facteurs de stress pour les animaux. Au Japon, une étude a montré que l’utilisation de ces plateformes a réduit le niveau d'hormones de stress chez les bovins de 25 %. Un environnement moins stressant favorise la santé et le bien-être général des animaux, ce qui peut également améliorer la qualité des produits animaux.

**V.3. Efficacité et Gain de Temps pour les Éleveurs**

**Optimisation des Tâches**

Les plateformes intégrées automatisent de nombreuses tâches quotidiennes telles que l’alimentation et le nettoyage, libérant ainsi du temps pour les éleveurs pour se concentrer sur des activités à plus forte valeur ajoutée. En Nouvelle-Zélande, une étude a indiqué que les éleveurs utilisant des plateformes intégrées ont réduit leur temps de travail manuel de 40 %. L'automatisation contribue à une gestion plus efficace de l'élevage et réduit la charge de travail physique des éleveurs.

**V.4. Accès aux Conseils et aux Formations**

Ces plateformes offrent également des modules de formation et des conseils en ligne pour promouvoir les meilleures pratiques d’élevage. En Inde, une étude a révélé que l'accès à ces ressources a considérablement amélioré les compétences et les connaissances des éleveurs, leur permettant d'adopter des pratiques plus durables et rentables.

**V.5. Analyse des Applications Existantes**

Plusieurs applications ont été développées pour répondre aux besoins de l'industrie de l'élevage en termes de gestion et d'efficacité. Voici un aperçu de certaines applications existantes :

1. **FarmLogs** : Cette application aide les agriculteurs à suivre et à analyser les données de leur exploitation agricole pour optimiser la production. Elle offre des fonctionnalités de gestion des stocks, de suivi des cultures et de prévision météorologique.
2. **AgriWebb** : Une application destinée à la gestion des fermes, qui offre des outils pour le suivi des animaux, la gestion des champs, et l'enregistrement des traitements vétérinaires. Elle facilite également la conformité réglementaire.
3. **Moocall** : Un système de surveillance basé sur des capteurs qui alerte les éleveurs sur le début du travail chez les vaches. Cela aide à minimiser les pertes dues à des naissances non supervisées.

**Conclusion**

En comparaison avec les applications existantes, notre application se distingue par son intégration complète des fonctionnalités essentielles à la gestion de l'élevage, incluant :

* **Optimisation des contacts entre éleveurs et clients** : En facilitant les interactions, notre application permet une meilleure mise en relation pour la vente d'animaux et de produits animaux.
* **Approche personnalisée de la gestion des animaux** : Contrairement à certaines applications généralistes, notre solution offre une personnalisation approfondie pour s'adapter aux besoins spécifiques des éleveurs.
* **Interface utilisateur intuitive et moderne** : notre application met l'accent sur une expérience utilisateur fluide et moderne, ce qui est souvent un point faible dans les applications existantes.
* **Fonctionnalités de formation et de suivi en temps réel** : En intégrant des modules de formation directement dans l'application, nous offrons un accès direct à l'éducation continue pour les éleveurs.

En résumé, notre application intègre les meilleures pratiques de l'industrie tout en offrant des fonctionnalités uniques qui répondent aux besoins spécifiques des éleveurs modernes. Elle représente une avancée significative dans l'utilisation de la technologie pour transformer l'industrie de l'élevage.

**Chapitre 2 : Méthodologies et Approches Conceptuelles**

1. **Introduction à UML**
   1. **Définition**

Le Unified Modeling Language (UML) est un outil essentiel pour la modélisation de systèmes logiciels complexes. Il offre une série de notations graphiques qui aident les concepteurs et les développeurs à représenter différents aspects d'un système de manière compréhensible. En utilisant UML, les équipes peuvent créer des représentations visuelles qui facilitent la communication, la documentation et la compréhension des systèmes, ce qui est crucial pour le succès des projets de développement.

UML comprend divers types de diagrammes qui se concentrent sur la structure, le comportement et les interactions du système. Ces diagrammes aident à capturer les exigences, à concevoir les architectures, à planifier les interactions et à définir les flux de travail. Les principaux diagrammes incluent les diagrammes de classes, de séquences, et d'activités, chacun ayant un rôle spécifique dans le processus de modélisation.

* 1. **Conception Orientée Objet**

La conception orientée objet (OO) est une approche qui permet de modéliser des systèmes logiciels en termes d'objets qui interagissent entre eux.

* Modélisation des Entités : Nous avons utilisé des concepts orientés objet pour modéliser les entités principales du système, telles que les éleveurs, les animaux, et les produits. Chaque entité est représentée par une classe qui encapsule les données et les comportements associés.
* Principes SOLID : Les principes SOLID sont un ensemble de bonnes pratiques de conception qui visent à rendre le code plus compréhensible, flexible et maintenable. Ils incluent des concepts tels que la responsabilité unique et l'inversion des dépendances.

1. **Langage de Modélisation UML**

UML offre un ensemble diversifié de diagrammes qui facilitent la visualisation et la spécification des différents aspects du système.

1. **Diagramme de Classe**

Le diagramme de classe est un élément essentiel de l'UML (Unified Modeling Language) qui montre la structure statique d'un système. Il illustre les classes du système, leurs attributs, méthodes et les relations entre elles. Ce diagramme est fondamental pour comprendre les entités du domaine et leurs interactions, ce qui facilite la conception de la base de données et l'implémentation du système.

**Composants d'un Diagramme de Classe**

1. **Classe**
   * **Description** : Représente une entité du système avec ses propriétés et comportements. Chaque classe est décrite par un rectangle divisé en trois sections.
   * **Sections** :
     + **Nom de la Classe** : La partie supérieure du rectangle qui indique le nom de la classe.
     + **Attributs** : La section du milieu qui liste les propriétés ou caractéristiques de la classe, généralement avec le format nom: type (par exemple, nom: String).
     + **Méthodes** : La partie inférieure qui énumère les opérations ou fonctions que la classe peut réaliser, souvent avec le format nom(paramètres): typeRetour (par exemple, getNom(): String).
2. **Association**
   * **Description** : Montre une relation entre deux classes. Elle est représentée par une ligne reliant les deux classes, et peut inclure une flèche pour indiquer la direction de la relation.
   * **Détails** : Les associations peuvent aussi avoir des libellés indiquant la multiplicité (par exemple, 1, 0..1, \*, etc.), indiquant combien d'instances d'une classe sont associées à une instance de l'autre classe.
3. **Agrégation**
   * **Description** : Montre une relation plus faible entre les classes où une classe (la classe contenant) contient une autre classe (la classe composant), mais de manière moins intégrée que dans la composition. Cette relation est représentée par une ligne solide avec un losange vide à l'extrémité de la classe contenant.
   * **Détails** : Contrairement à la composition, les parties peuvent exister indépendamment de l'ensemble. La durée de vie de la partie n'est pas strictement liée à celle du tout.
4. **Dépendance**
   * **Description** : Indique qu'une classe utilise ou dépend d'une autre classe. Cette relation est représentée par une ligne en pointillé avec une flèche pointant vers la classe dont la dépendance est faite.
   * **Détails** : La dépendance est souvent utilisée pour montrer que des modifications dans une classe peuvent affecter une autre classe qui l'utilise.
5. **Diagramme de Séquence**

Le diagramme de classe est un élément central de l'UML (Unified Modeling Language) utilisé pour représenter la structure statique d'un système. Il montre les classes du système, leurs attributs, méthodes, et les relations qui les relient. Ce diagramme est essentiel pour comprendre la structure et les interactions des entités du domaine, ce qui est crucial pour la conception de la base de données et la mise en œuvre du système.

**Composants d'un Diagramme de Classe**

1. **Classe**
   * **Description** : Une classe est une entité fondamentale dans un diagramme de classe, représentant un concept ou un objet dans le système. Elle est décrite par un rectangle divisé en trois parties.
   * **Sections** :
     + **Nom de la Classe** : La partie supérieure du rectangle affiche le nom de la classe en gras.
     + **Attributs** : La section intermédiaire énumère les propriétés de la classe, souvent sous le format nom : type (par exemple, nom : String).
     + **Méthodes** : La partie inférieure du rectangle contient les opérations que la classe peut réaliser, affichées avec le format nom(paramètres) : typeRetour (par exemple, getNom () : String).
2. **Composition**
   * **Description** : Indique une relation forte entre les classes, où une classe (la classe contenant) est composée de parties (la classe composant). Cette relation est montrée par une ligne solide avec un losange noir à l'extrémité de la classe contenant.
   * **Détails** : Dans une composition, la durée de vie de la partie dépend de celle du tout. Si la classe contenante est supprimée, les instances de la classe composant le sont également.
3. **Agrégation**
   * **Description** : Montre une relation plus faible entre les classes, où une classe (la classe contenant) possède une autre classe (la classe composant), mais avec une relation moins intégrée que dans la composition. Représentée par une ligne solide avec un losange vide à l'extrémité de la classe contenant.
   * **Détails** : Contrairement à la composition, les parties peuvent exister indépendamment de l'ensemble. La durée de vie de la partie n'est pas strictement liée à celle du tout.
4. **Dépendance**
   * **Description** : Représente une relation où une classe utilise ou dépend d'une autre classe. Illustrée par une ligne en pointillé avec une flèche pointant vers la classe dont la dépendance est faite.
   * **Détails** : La dépendance indique que les modifications apportées à une classe peuvent influencer une autre classe qui en dépend, souvent utilisée pour les relations temporaires ou moins permanentes.
5. **Diagramme d'Activité**

Les diagrammes d'activité sont des outils clés en UML (Unified Modeling Language) utilisés pour capturer les flux de travail et les processus métiers. Ils illustrent les étapes séquentielles, les choix conditionnels et les parallélismes au sein d'un processus. Ces diagrammes permettent de visualiser de manière claire et détaillée la séquence des activités et les interactions entre elles.

**Composants d'un Diagramme d'Activité**

1. **Activités**
   * **Description** : Les activités représentent des actions ou des tâches individuelles dans le processus. Elles sont représentées par des rectangles arrondis (ou "nœuds d'activité") dans le diagramme.
   * **Détails** : Chaque activité décrit une étape spécifique du processus, comme "Vérifier les informations de l'animal" ou "Enregistrer la vente".
2. **Transitions**
   * **Description** : Les transitions montrent le flux entre les activités. Elles sont représentées par des flèches qui connectent les nœuds d'activité, indiquant l'ordre dans lequel les activités doivent être exécutées.
   * **Détails** : Les transitions peuvent inclure des conditions (comme "si l'information est valide") qui déterminent si le flux doit passer d'une activité à une autre.
3. **Points de Décision**
   * **Description** : Les points de décision représentent des moments où le processus peut diverger en fonction de conditions spécifiques. Ils sont représentés par des losanges.
   * **Détails** : Les points de décision permettent de modéliser des choix ou des branches dans le flux de travail, comme choisir entre "Approuver la demande" ou "Rejeter la demande".
4. **Mélange de Flux**
   * **Description** : Les mélanges de flux montrent des activités qui peuvent se dérouler en parallèle. Ils sont représentés par des barres de synchronisation (ou "nœuds de synchronisation") avec plusieurs flèches entrant et sortant.
   * **Détails** : Les barres de synchronisation permettent de modéliser des activités concurrentes, comme "Préparer les documents" et "Informer le client" qui se déroulent simultanément.
5. **Points de Début et de Fin**
   * **Description** : Les points de début et de fin marquent respectivement le commencement et la conclusion du processus. Ils sont représentés par un cercle plein pour le début et un cercle avec un point noir pour la fin.
   * **Détails** : Ces points permettent de définir clairement où le processus commence et se termine, facilitant ainsi la compréhension du flux global.
6. **Objets et Artefacts**
   * **Description** : Les objets et artefacts représentent les éléments utilisés ou produits au cours du processus. Ils sont généralement illustrés par des rectangles ou des documents attachés aux activités.
   * **Détails** : Ils permettent de suivre les documents, les données ou les produits qui circulent tout au long du processus.

**Application**

Les diagrammes d'activité sont particulièrement utiles pour visualiser des processus complexes, tels que la gestion des animaux ou les transactions commerciales. Ils montrent clairement le déroulement des activités, les choix décisionnels, et les tâches parallèles, facilitant ainsi la compréhension et l'analyse des processus métiers.

1. **Application des Diagrammes UML dans le Développement**

Les diagrammes UML sont des outils puissants qui jouent un rôle crucial dans le développement de logiciels en facilitant la spécification, la visualisation et la documentation des systèmes.

* Spécification et Visualisation : Les diagrammes UML permettent de spécifier clairement les exigences et de visualiser les comportements attendus du système. Ils offrent une représentation graphique qui aide à la communication et à la compréhension entre les développeurs et les parties prenantes.
* Documentation et Communication : En fournissant une documentationdétaillée et standardisée, UML améliore la communication entre les membres de l'équipe de développement et les autres parties prenantes, assurant une compréhension commune des fonctionnalités et de l'architecture du système.

En intégrant UML dans notre méthodologie, nous avons créé une architecture logiciellebien structurée, flexible et prête à évoluer avec les besoins futurs, tout en assurant une communication claire et efficace au sein de l'équipe.

* 1. **Méthodes de Processus et de Développement**

Le développement de notre application de gestion de l'élevage a nécessité l'adoption de méthodologies modernes pour assurer flexibilité et efficacité. Nous avons choisi des méthodes qui favorisent l'adaptabilité aux besoins changeants des utilisateurs tout en garantissant une livraison rapide de solutions fonctionnelles.

* **Méthode Agile**

La méthode agile est une approche de développement logiciel qui privilégie les interactions humaines, les livraisons rapides et une capacité d'adaptation aux changements. Voici comment nous l'avons appliquée :

* Principes Agiles : L'agilité repose sur douze principes, tels que la satisfaction du client par une livraison rapide et continue de logiciels fonctionnels, et l'acceptation du changement, même tard dans le développement. Ces principes ont guidé notre processus, assurant que nous restons alignés avec les attentes des utilisateurs finaux.
* Scrum : Scrum est un cadre agile spécifique qui structure le travail en sprints courts et réguliers, généralement de deux à quatre semaines. Chaque sprint commence par une planification où les priorités sont définies, suivie d'un développement intensif, et se termine par une revue et une rétrospective pour évaluer le travail accompli et identifier les améliorations possibles.

| **Critère** | **Méthode Agile** | **Méthode Traditionnelle** |
| --- | --- | --- |
| **Approche** | Itérative et incrémentale | Linéaire (Cycle en V ou Cascade) |
| **Flexibilité** | Haute, permet des changements même tardifs | Faible, difficile d'incorporer des changements |
| **Livraison** | Livraisons fréquentes (chaque sprint) | Livraison unique en fin de cycle |
| **Collaboration** | Forte collaboration entre les équipes | Collaboration limitée à certaines étapes |
| **Documentation** | Allégée, évolue avec le projet | Lourde, établie en début de projet |

**Comparaison des Méthodes de Développement**

**Méthode Agile vs Méthode Traditionnelle**

**Scrum vs RAD (Rapid Application Development)**

| **Critère** | **Scrum** | **RAD** |
| --- | --- | --- |
| **Durée des cycles** | **Courts sprints de 2 à 4 semaines** | **Phases rapides avec des prototypes** |
| **Focus** | **Livraison d'une version fonctionnelle à chaque sprint** | **Livraison rapide grâce à des prototypes fréquents** |
| **Adaptabilité** | **Haut niveau de flexibilité** | **Flexibilité élevée, mais moins structurée** |
| **Documentation** | **Adaptative, ajustée à chaque sprint** | **Minimaliste, basée sur les prototypes** |
| **Communication** | **Forte communication continue** | **Forte, mais parfois centrée sur l'interface utilisateur** |

1. **Approche Conceptuelle Méthodologique**

Une approche conceptuelle méthodologique bien définie est cruciale pour structurer et organiser le développement de logiciels complexes. Notre approche se concentre sur l'analyse des exigences et la conception orientée objet.

* **Analyse des Exigences**

L'analyse des exigences est la première étape cruciale dans le développement de logiciels, permettant de définir ce que le système doit accomplir.

* Collecte des Exigences : Nous avons mené des ateliers et des entrevues avec les parties prenantes pour recueillir des exigences détaillées. Ces interactions ont permis de comprendre les besoins spécifiques des éleveurs, des clients et des fournisseurs de services.
* Cas d'Utilisation : Les cas d'utilisation sont des descriptions narratives des interactions entre les utilisateurs et le système. Ils ont été utilisés pour documenter comment le système devrait se comporter dans divers scénarios, offrant une base solide pour la conception et le développement.

**Analyse des Exigences : Cas d'Utilisation vs User Stories**

| **Critère** | **Cas d'Utilisation** | **User Stories** |
| --- | --- | --- |
| Format | Narratif détaillé avec diagrammes possibles | Simple phrase : "En tant que... Je veux... Afin de..." |
| Niveau de détail | Très détaillé | Moins détaillé, focalisé sur les besoins utilisateur |
| Documentation | Lourde, nécessaire pour les systèmes complexes | Légère, évolue avec les besoins |
| Objectif | Capturer tous les scénarios possibles | Capturer les besoins utilisateur de manière simple |
| Utilisation | Projets complexes avec beaucoup d'interactions | Projets agiles, centrés sur les utilisateurs |

1. **Conclusion**

Dans ce chapitre dédié à la méthodologie et à l'approche conceptuelle, nous avons exploré les outils essentiels pour la conception et la modélisation de systèmes complexes. Les diagrammes de classe et d'activité jouent un rôle fondamental dans la structuration et la visualisation des processus et des entités du système.

Les diagrammes de classe permettent de définir la structure statique du système en illustrant les classes, leurs attributs, méthodes, et les relations entre elles. Cette représentation est cruciale pour la conception de la base de données et l'organisation des différentes entités du domaine, facilitant ainsi une compréhension claire de la structure et des interactions des classes.

En parallèle, les diagrammes d'activité offrent une vue dynamique des processus métiers, montrant les flux de travail, les choix conditionnels, et les tâches parallèles. Ils sont particulièrement utiles pour modéliser des processus complexes, permettant ainsi une meilleure visualisation des étapes séquentielles et des interactions dans les processus d'affaires.

En intégrant ces outils dans la phase de conception, nous pouvons mieux comprendre les exigences du système, optimiser les processus et garantir une mise en œuvre cohérente et efficace. La combinaison des diagrammes de classe et d'activité assure une approche systématique et complète, offrant une base solide pour la construction de solutions robustes et adaptées aux besoins spécifiques du projet.

**PARTIE II Conception et mise en œuvre**

**Chapitre 1 : Conception, modélisation et approche conceptuelle**

* + 1. **Conception, modélisation**

**1.Diagramme des cas d’utilisation**

Bien avant l’inscription, l’utilisateur n’a que la possibilité de regarder le catalogue de produit présent dans sur la plateforme. Ensuite il pourra s’inscrire pour devenir un client sur notre plateforme mobile.

Après s’être enregistrer il a le droit maintenant de poursuivre les opérations jusqu’à l’achat du produit en ligne sur notre application mobile.

* 1. **Description textuelle du cas d’utilisation de la gestion des animaux**

| **Titre** | **Description Textuelle** |
| --- | --- |
| **Nom du Cas d'Utilisation** | **Gestion des Animaux** |
| **Acteurs** | **Utilisateur, Système de Gestion, Base de Données** |
| **Objectif** | **Permettre à l'utilisateur de gérer les informations de ses animaux** |
| **Préconditions** | **L'utilisateur doit être connecté au système** |
| **Postconditions** | **Les informations de l'animal sont mises à jour et enregistrées dans la base de données** |
| **Flux Principal** | 1. **L'utilisateur accède à la section de gestion des animaux.**  2. **Il sélectionne un animal dans la liste.**  3. **L'utilisateur modifie les informations de l'animal.**  4. **Le système enregistre les modifications.** |
| **Flux Alternatif** | 1. **Si l'utilisateur n'est pas connecté, le système redirige vers la page de connexion.**  2. **Si l'animal sélectionné n'existe pas, le système affiche un message d'erreur.** |
| **Extensions** | **Si la modification inclut un changement d'état de santé, une notification est envoyée au vétérinaire** |
| **Déclencheur** | **L'utilisateur choisit de modifier les informations d'un animal** |
| **Fréquence d'Utilisation** | **Régulière, selon les besoins de l'utilisateur** |
| **Risques** | **Erreur de saisie, perte de connexion lors de l'enregistrement** |

* 1. **Diagramme de séquence**
  2. **Diagramme de classe**

Le diagramme de classes comprend les entités suivantes :

Classe Utilisateur : Contient les attributs nom, prénom, email, motDePasse, et adresse, avec les méthodes seConnecter(), créerCompte(), et mettreAJourInformations().

Classe Animal : Possède les attributs nom, âge, espèce, dateNaissance, photo, et historiqueSoins, avec les méthodes ajouterSoins() et modifierInformations().

Classe Vétérinaire : Comprend les attributs nom, contact, et spécialité, ainsi que les méthodes planifierVisite() et ajouterVaccination().

Classe Vaccination : A les attributs date, typeVaccin, et vétérinaire (référence à un objet Vétérinaire), avec la méthode enregistrerVaccination().

Classe Produit : Contient les attributs nom, description, prix, et photo, avec les méthodes ajouterProduit() et modifierProduit().

Classe Vente : Comprend les attributs animal (référence à un objet Animal), produit (référence à un objet Produit), prix, et dateVente, avec la méthode enregistrerVente().

Classe ComptePremium : A les attributs dateDébut, dateExpiration, et statut, avec la méthode renouvelerAbonnement().

Relations entre les Classes :

* Un Utilisateur peut posséder plusieurs Animaux (1 ; N).
* Un Animal peut recevoir plusieurs Vaccinations, chaque vaccination étant administrée par un Vétérinaire (1 ; N).
* Un Utilisateur peut vendre plusieurs Produits (1 : N)

).

* Un Produit peut être associé à plusieurs Ventes, chaque vente étant spécifique à un produit et un acheteur (1 :1).
* Un Utilisateur peut avoir un ComptePremium (1 :1)

1. **Approche Conceptuelle**
2. **Comparaison de différentes méthodes de développement**

**Tableau Comparatif : Flutter vs. React Native**

| **Critère** | **Flutter** | **React Native** |
| --- | --- | --- |
| **Langage** | Dart | JavaScript |
| **Performance** | Proche du natif grâce à la compilation JIT et AOT | Bonnes performances, mais parfois inférieures au natif |
| **UI et UX** | UI riche avec une large gamme de widgets personnalisables, haute flexibilité | UI personnalisable, mais dépend des composants natifs et des bibliothèques tierces |
| **Hot Reload** | Oui, permet de voir les changements instantanément | Oui, permet de voir les changements instantanément |
| **Écosystème** | Plus jeune, en croissance rapide | Mûr, large communauté et écosystème établi |
| **Plugins et Bibliothèques** | En croissance, certains plugins nécessitent du code natif | Large choix de bibliothèques et plugins, mais qualité variable |
| **Apprentissage** | Langage Dart peut nécessiter un apprentissage supplémentaire | JavaScript est largement connu, plus facile pour les développeurs web |
| **Support des Plateformes** | Support natif pour iOS et Android, ainsi que pour le web et le desktop | Principalement pour iOS et Android, le support web est limité |
| **Intégration Native** | Peut nécessiter du code natif pour certaines fonctionnalités via des plugins | Bonne intégration avec les fonctionnalités natives grâce à des wrappers et des modules natifs |

**Tableau Comparatif : Firestore vs. MySQL**

| **Critère** | **Firestore** | **MySQL** |
| --- | --- | --- |
| **Type de Base de Données** | **NoSQL, base de données orientée documents** | **SQL, base de données relationnelle** |
| **Modèle de Données** | **Documents et collections (non structuré)** | **Tables avec schéma fixe (structuré)** |
| **Scalabilité** | **Scalabilité horizontale élevée, adaptée aux applications évolutives** | **Scalabilité verticale, mais des solutions comme le sharding sont nécessaires pour une scalabilité horizontale** |
| **Temps Réel** | **Support natif des mises à jour en temps réel** | **Pas de support natif, mais peut être réalisé avec des solutions externes** |
| **Transactions** | **Transactions atomiques, mais avec des limitations sur les opérations complexes** | **Transactions ACID robustes et support complet des opérations complexes** |
| **Indexation** | **Indexation automatique, mais personnalisée avec des limitations** | **Indexation flexible et personnalisable, permet une optimisation poussée des requêtes** |
| **Coût** | **Coût basé sur l'utilisation (stockage, lecture/écriture, etc.)** | **Coût basé sur le serveur et le stockage, généralement moins coûteux pour des déploiements sur site** |
| **Flexibilité du Schéma** | **Schéma flexible, idéal pour des données non structurées ou évolutives** | **Schéma fixe, nécessite des migrations pour modifier la structure des données** |
| **Support et Intégration** | **Intégration facile avec Firebase et autres services Google, mais moins de support pour les requêtes complexes** | **Large écosystème, bien supporté par de nombreux outils et frameworks, mais nécessite une gestion plus complexe** |

Pour le développement de notre application nous avons choisi flutter comme méthode de développement et firestore pour la base de données.

**Chapitre2 : Mise en Œuvre et Résultat**

* + 1. **Mise en œuvre**
  1. **Environnement de travail**

1. **Environnements Matériels**

****Processeur :Intel(R) Celeron (R) N4000 CPU @

1.10GHz 1.10 GHz Mémoire RAM : 8,00 Go

Disque dur : 230go

Mémoire RAM : 12go

Disque Dur: 128go

Version android: 13

1. **Environment logiciel**

* **.Flutter**

Flutter est un framework open-source développé par Google qui permet de créer des applications mobiles, web et desktop à partir d'une seule base de code. En utilisant le langage Dart, Flutter offre des capacités de développement d'interfaces utilisateur riches et performantes. Le framework se distingue par sa fonctionnalité de Hot Reload, qui permet aux développeurs de voir instantanément les changements apportés au code sans avoir à redémarrer l'application. Bien que Flutter soit très puissant pour le développement cross-platform, il est encore relativement jeune par rapport à d'autres frameworks comme React Native, et certains plugins peuvent ne pas être aussi bien maintenus ou complets.

* **Draw.io**

Draw.io (ou diagrams.net) est un outil en ligne gratuit conçu pour la création de diagrammes et de maquettes. Il est particulièrement utile pour concevoir des diagrammes de flux, des organigrammes, et des maquettes d'application. Draw.io est accessible directement depuis un navigateur web et permet la collaboration en temps réel avec plusieurs utilisateurs. Il s'intègre facilement avec des services de stockage cloud comme Google Drive et Dropbox. Cependant, bien que Draw.io soit gratuit et accessible, il peut manquer de certaines fonctionnalités avancées présentes dans des outils payants comme Microsoft Visio et peut devenir lent lorsqu'on travaille avec des diagrammes très complexes.

* **Visual Studio Code (VS Code)**

Visual Studio Code est un éditeur de code open-source développé par Microsoft. Il est largement utilisé pour sa légèreté, ses performances rapides et ses capacités de personnalisation. VS Code prend en charge une grande variété de langages de programmation grâce à un large éventail d'extensions, et il offre une bonne intégration avec des systèmes de contrôle de version comme Git, ainsi que des outils de développement comme les débogueurs. Malgré sa popularité, VS Code peut présenter une certaine complexité pour les débutants en raison de ses nombreuses fonctionnalités et options, et peut nécessiter une configuration et une personnalisation spécifiques pour certains langages et frameworks.

* **Firestore**

Firestore est une base de données NoSQL cloud développée par Google, conçue pour stocker et synchroniser des données en temps réel à travers les clients. Elle est particulièrement adaptée aux applications nécessitant une haute scalabilité et des mises à jour en temps réel. Firestore s'intègre facilement avec les autres services Firebase, comme Firebase Authentication et Cloud Functions, et offre un modèle de données flexible qui convient bien aux structures de données non rigides. Cependant, le coût d'utilisation peut devenir élevé avec une utilisation intensive, et les requêtes complexes peuvent être plus difficiles à optimiser par rapport aux bases de données relationnelles comme MySQL.

**Caractéristique des méthodes utilisée**

| **Outil/Technologie** | **Type** | **Langage** | **Description** | **Principales Fonctionnalités** |
| --- | --- | --- | --- | --- |
| **Flutter** | **Framework UI** | **Dart** | **Framework open-source pour le développement d'applications mobiles, web et desktop.** | **Développement cross-platform, Hot Reload, Widgets riches et personnalisables** |
| **Draw.io** | **Outil de Diagramme** | **N/A** | **Outil en ligne pour la création de diagrammes, maquettes et organigrammes.** | **Collaboration en temps réel, Intégration avec le stockage cloud, Gratuit et accessible** |
| **Visual Studio Code (VS Code)** | **Éditeur de Code** | **N/A** | **Éditeur de code open-source avec support pour divers langages de programmation et outils de développement.** | **Extensions variées, Intégration Git, Performances rapides, Personnalisation** |
| **Firestore** | **Base de Données NoSQL** | **N/A** | **Base de données cloud pour le stockage et la synchronisation des données en temps réel.** | **Scalabilité élevée, Intégration avec Firebase, Modèle de données flexible** |

* + 1. **Résultats**