**Rapport de Projet**

**Sous thème**

**« VITECARE »**

Filière : ILCS

Nom et prénom :

Reda Elbikri,

Yahya Es-salhi

Encadre par :

Prof. Redouane Esbaie .

1. ***Remerciment :***

Nous tenons à remercier chaleureusement notre encadrant, M. Redouan Esbai, pour son soutien, ses conseils et sa disponibilité tout au long de ce projet. Nous lui sommes reconnaissants pour ses orientations qui ont grandement contribué à la réalisation de notre travail.

1. ***Introduction :***

Ce projet consiste en la réalisation d'une application web dédiée à la gestion des vétérinaires et de leurs activités, dans le but de faciliter leur travail au quotidien. L’application permet notamment la gestion des visites médicales pour les animaux, chaque visite étant associée à un vétérinaire ainsi qu’au propriétaire de l’animal concerné.

L’objectif principal est de proposer une solution intuitive et efficace qui optimise l'organisation des consultations vétérinaires, tout en garantissant un accès rapide et sécurisé aux informations de santé des animaux. Par ailleurs, l’application met à disposition une liste de vétérinaires disponibles 24h/24, renforçant ainsi la réactivité du service.

Pour mener à bien ce projet, nous avons utilisé plusieurs technologies modernes du développement web (React (avec le framework Next.js) pour le développement de l’interface utilisateur côté client, Bootstrap pour une mise en page responsive , Node.js et Express.js pour la partie serveur et la gestion des API. ,et mongodb comme base de données )

La réalisation de cette application nous a permis de consolider nos compétences techniques en développement web .

***III)REALISATION TECHNIQUE :***

***1)BASE DE DONNEE :***

Dans l’application web VITECARE ,il est nécessaire d’utiliser une base de donnée pour stocker les informations des utilisateurs ,que ca soit une base de donnée SQL ou même NOSQL ,

Et Dans ce cas la , on a choisit une base de données NoSQL plutôt qu’une base SQL a cause de flexibilité de NOSQL dans la structure des données, de performance sur de gros volumes non structurés, ou de scalabilité horizontale. Contrairement aux bases SQL qui imposent un schéma fixe et sont idéales pour gérer des relations complexes entre les données, les bases NoSQL permettent de stocker des informations sous forme de documents ou d’objets sans schéma rigide, ce qui facilite l’évolution de l’application . Ainsi que NoSQL est particulièrement utile pour les projets qui manipulent des données variées, volumineuses ou qui changent fréquemment.

NoSQL contient en effet plusieurs types de bases de données, notamment orientées document, graphes, colonnes et clé-valeur. Parmi celles-ci, nous avons choisi une base orientée document avec MongoDB car elle offre un excellent équilibre entre flexibilité, performance et simplicité d’utilisation. MongoDB permet de stocker les données sous forme de documents JSON (ou BSON), ce qui rend la structure des données très souple et facilement modifiable sans migration complexe du schéma. Cela est particulièrement utile lorsque les données ont des structures variables ou évolutives, comme dans le cas d’applications web ou mobiles modernes. De plus, MongoDB facilite la hiérarchisation des données, réduit le besoin de jointures complexes et permet une bonne scalabilité horizontale, ce qui en fait un excellent choix pour des projets dynamiques avec une croissance rapide.

Nous avons utilisé aussi MongoDB Compass qu’est un outil graphique essentiel pour interagir facilement avec une base de données MongoDB pour simplifier l'exploration, la gestion et l'optimisation de notre base de données. Grâce à Compass, nous avons pu visualiser les données de manière intuitive, effectuer des requêtes complexes sans écrire de code, et analyser la structure de nos collections. Cela nous a permis de mieux comprendre nos données, d'identifier les indices nécessaires et d'optimiser les performances de la base de données. En outre, Compass a facilité la création et la modification de documents, tout en offrant une vue d’ensemble de la validité du schéma et des types de données, ce qui a amélioré notre efficacité dans le développement et la maintenance de l’application.

***1)BACK-END:***

Nous avons utilisé Node.js, qui est un environnement d’exécution JavaScript côté serveur permettant d’exécuter du code JavaScript en dehors du navigateur. Grâce à Node.js, nous avons pu développer la partie backend de notre application en JavaScript, en créant des routes, en gérant les requêtes HTTP, et en interagissant avec notre base de données MongoDB. Node.js est basé sur le moteur V8 de Google Chrome, ce qui lui permet d’exécuter le code rapidement et efficacement. Il adopte un modèle asynchrone et non bloquant, idéal pour les applications web modernes qui doivent gérer un grand nombre de connexions simultanées.

Nous avons aussi utilisé le framework Express.js au lieu de développer un serveur HTTP natif avec Node.js, car Express simplifie énormément la création d’API et la gestion des routes. Alors que Node.js seul nécessite beaucoup de code pour gérer les requêtes, les réponses et les chemins d’URL, Express fournit une structure légère et rapide qui permet de définir facilement des routes, des middlewares, et de gérer les erreurs. Il offre également une meilleure organisation du code, ce qui rend le développement plus rapide, plus lisible et plus maintenable. Grâce à Express, nous avons pu construire un backend plus propre, modulaire et adapté aux bonnes pratiques du développement web.

GESTION DE CODE DANS VSCODE :

Pour structurer notre backend, nous avons adopté une architecture claire et organisée autour de **trois dossiers principaux** et **deux fichiers essentiels**, ce qui facilite la lisibilité, la maintenance et l’évolutivité du projet. Tout d’abord, nous avons un **fichier de configuration**, chargé d’établir la connexion à la base de données MongoDB à l’aide des paramètres requis, tels que l’URL de connexion et les options de sécurité. Ensuite, le **dossier controllers** regroupe l’ensemble des fonctions qui gèrent les différentes requêtes NoSQL : ajout, modification, suppression ou récupération de données. Ces fonctions centralisent la logique métier de notre application. Le **dossier models**, quant à lui, contient les schémas de données définissant la structure des collections MongoDB, garantissant ainsi la cohérence des documents stockés dans la base. Le **fichier routes** joue un rôle d’intermédiaire : il mappe les différentes URL de l’API vers les fonctions appropriées des contrôleurs, permettant ainsi une gestion claire et centralisée des endpoints. Enfin, le **fichier serveur** (souvent nommé server.js ou app.js) constitue le point d’entrée de l’application. Il se charge de démarrer le serveur HTTP, d’établir la connexion avec MongoDB, et d’appliquer les routes de l’API. Cette organisation modulaire permet un développement plus propre, plus rapide, et facilite les évolutions futures du projet.

***1)Test de Fonctionnement du BACK-END:***

Les tests de backend sont essentiels pour garantir que les routes de l'API, les interactions avec la base de données et la logique métier fonctionnent correctement. Pour tester efficacement un backend, plusieurs outils peuvent être utilisés, chacun ayant ses avantages spécifiques. Des outils comme Postman, Insomnia, et cURL permettent d'envoyer des requêtes HTTP, de simuler des appels clients et de vérifier les réponses du serveur. On peut aussi utiliser des solutions comme Swagger UI pour générer des interfaces de test interactives, ou des frameworks comme Jest et Supertest pour effectuer des tests automatisés au niveau du code. De plus, pour tester la performance et la scalabilité, des outils comme K6 sont utilisés pour simuler des charges élevées sur l'API. Ces tests aident non seulement à vérifier le bon fonctionnement du backend, mais aussi à détecter les erreurs, les bogues et à optimiser les performances. Dans notre cas, nous avons choisi Postman, car il est le plus simple à utiliser (c’est à dire que nous pouvons l’utiliser sans aucune expérience ) il offre une interface graphique intuitive qui nous a permis de tester rapidement et efficacement les différentes routes de notre API, de vérifier les réponses et de simuler des scénarios réels sans avoir besoin d’écrire du code supplémentaire.

***1)FRONT -END:***

Dans le frontend, nous avons utilisé React, une bibliothèque JavaScript populaire pour la création d'interfaces utilisateur modernes et dynamiques. React se distingue par sa capacité à rendre les applications réactives, en permettant de créer des interfaces interactives et performantes, tout en optimisant l’expérience utilisateur. Grâce à React, nous avons pu organiser notre code de manière modulaire en divisant l'interface en composants réutilisables. Chaque composant gère son propre état, ce qui nous permet de construire des éléments tels que la navbar, le footer, les formulaires, et d’autres sections, de manière indépendante et autonome, facilitant ainsi l'organisation et la maintenance du projet.

React repose également sur un système de hooks, qui offre des fonctionnalités avancées pour gérer l’état et les effets secondaires dans les composants fonctionnels. Parmi ces hooks, useState est utilisé pour déclarer et manipuler l'état local dans un composant. Par exemple, il permet de suivre des variables comme des entrées de formulaire ou des états de validation. D'autre part, le hook useEffect est utilisé pour exécuter des effets secondaires, tels que les requêtes API, l’abonnement à des événements, ou la mise à jour de l'interface en réponse à des changements d’état ou de props. useEffect permet ainsi de gérer le cycle de vie du composant de manière déclarative, ce qui simplifie le code et le rend plus lisible.

Un autre avantage clé de React est son moteur de mise à jour virtuel (Virtual DOM), qui optimise les performances en minimisant le nombre de modifications effectuées dans le DOM réel. Lorsqu'une modification se produit dans l'état d’un composant, React ne met pas immédiatement à jour le DOM, mais crée une version virtuelle du DOM. Il compare ensuite cette version virtuelle avec le DOM actuel, et effectue uniquement les changements nécessaires. Cela permet d'améliorer considérablement la réactivité et la performance de l'application, notamment pour les applications à grande échelle.

De plus, React fonctionne selon le principe d’application à une seule page (SPA - Single Page Application), ce qui signifie que le navigateur n'a pas besoin de recharger la page entière pour chaque action. Au lieu de cela, seule une partie de la page est mise à jour, ce qui offre une expérience fluide et rapide. Cela permet d'éviter les rechargements de page longs et d'assurer des transitions rapides entre les différentes vues de l'application. En combinant ces avantages avec des interactions en temps réel avec le backend, React nous a permis de créer une application moderne, réactive et optimale pour l'utilisateur, offrant ainsi une expérience utilisateur fluide et sans interruptions.

En somme, l’utilisation de React nous a permis de développer une interface à la fois performante, réactive et facile à maintenir, tout en garantissant une expérience utilisateur fluide et dynamique grâce à sa gestion des états, des effets secondaires et son modèle de mise à jour efficace.

**LANGUAGE POUR DEVELOPER AVEC REACT :**

Le langage utilisé pour développer avec React est principalement JavaScript. React est une bibliothèque JavaScript, et le code écrit pour créer des composants et des interfaces utilisateur avec React est principalement du JavaScript, souvent en utilisant des fonctionnalités modernes du langage telles que les ES6+ features (comme les arrow functions, async/await, destructuring, etc.).

Cependant, on a utilisé pour améliorer l'expérience de développement avec React, un autre langage appelé TypeScript. TypeScript est un sur-ensemble de JavaScript qui ajoute des fonctionnalités de typage statique et des outils de vérification de type à JavaScript. Cela permet d'améliorer la sécurité du code, de faciliter la maintenance et de détecter les erreurs à la compilation plutôt qu'à l'exécution.

**FRAMEWORK AVEC REACT :**

Il existe plusieurs frameworks populaires que l’on peut utiliser avec React pour optimiser le développement d’applications web modernes et performantes. Parmi ces frameworks, Next.js et Vite sont les plus couramment utilisés. Next.js est un framework basé sur React qui permet de créer des applications web avec des fonctionnalités comme le rendu côté serveur (SSR), la génération de pages statiques (SSG) et le routage automatique. Grâce à ces fonctionnalités, Next.js permet d'améliorer les performances et le SEO des applications. Il propose également une approche hybride qui permet de mélanger le rendu côté serveur et le rendu statique, ce qui rend le développement plus flexible et performant. D'autre part, Vite est un outil de build moderne qui offre des temps de démarrage ultra-rapides et un hot-reloading instantané, ce qui est idéal pour le développement rapide.

Cependant, nous avons choisi Next.js pour notre projet, car il offre des fonctionnalités avancées de rendu côté serveur et de génération de pages statiques, ce qui nous permet d'optimiser le temps de chargement des pages et d'améliorer le référencement naturel (SEO) de l'application. En particulier ,la chose qui nous a poussé à utiliser next.js est le routage automatique de Next.js simplifie considérablement la gestion des routes. Contrairement à d'autres frameworks, où il faut définir manuellement les routes dans un fichier de configuration, Next.js crée automatiquement des routes en fonction des fichiers présents dans le dossier pages. Chaque fichier ajouté à ce dossier devient une route accessible dans l'application, avec des sous-dossiers permettant de gérer des routes imbriquées. Cela permet de travailler de manière très intuitive, sans avoir à se soucier de la configuration complexe des routes. De plus, Next.js offre une prise en charge native des routes dynamiques grâce à un système de fichiers qui permet de créer des paramètres dynamiques dans les URL (par exemple, pour des pages de profils ou des articles). Ces fonctionnalités rendent le développement beaucoup plus rapide et fluide, tout en garantissant une structure claire et cohérente pour l’ensemble de l’application. Son système de routage automatique, combiné à ses API routes intégrées, simplifie également la gestion des endpoints côté serveur, rendant le processus de développement globalement plus efficace.

**FRAMEWORK CSS :**

Il existe plusieurs frameworks CSS populaires utilisés avec React pour faciliter le développement d'interfaces utilisateur modernes et réactives. Parmi eux, Bootstrap offre un large éventail de composants prêts à l'emploi et un système de grille flexible, tandis que Tailwind CSS adopte une approche utility-first, permettant une personnalisation fine des styles grâce à des classes utilitaires. Material-UI fournit des composants basés sur Material Design, parfaits pour des applications riches et interactives. D'autres options comme Bulma, Foundation, et Chakra UI offrent des solutions similaires, chacune avec ses spécificités, comme Flexbox pour Bulma et une approche accessible pour Chakra UI. Enfin, des outils comme SASS/SCSS et Styled Components permettent de gérer les styles de manière plus dynamique et modulaire, notamment avec le CSS-in-JS. Pour notre projet, nous avons choisi [le framework utilisé], car il répondait à nos besoins de [raison spécifique], nous offrant une solution flexible, performante et facile à intégrer avec React.

Pour les projets Next.js, Tailwind CSS est souvent recommandé en raison de sa flexibilité et de son approche utility-first, qui permet une personnalisation détaillée des styles. Cependant, dans notre projet, nous avons choisi d'utiliser Bootstrap car nous avions déjà une expérience préalable avec ce framework. Bootstrap nous a permis de bénéficier rapidement de ses composants préconçus et de son système de grille flexible, ce qui a facilité le développement tout en garantissant une interface soignée et réactive. De plus, avec l'usage de React-Bootstrap, une bibliothèque qui permet d'intégrer facilement les composants Bootstrap dans React, nous avons pu bénéficier de l'optimisation du framework pour React, avec des composants entièrement réactifs, modifiables et compatibles avec les hooks, ce qui a renforcé l'interactivité et la fluidité de notre interface.

**ARCHITECTURE DE FRONTEND :**

Dans le dossier frontend de notre projet, nous avons organisé les fichiers en plusieurs composants afin de structurer l’application de manière claire et modulaire. Chaque fichier est dédié à une fonctionnalité spécifique, ce qui facilite la gestion et la maintenance du code. Par exemple, les fichiers ajoutProprietaire et ajoutPet permettent respectivement d'ajouter de nouveaux propriétaires et animaux via des formulaires interactifs. Les fichiers afficherProprietaire et afficherProprietaires sont responsables de l'affichage des informations relatives aux propriétaires, avec le second affichant une liste complète. Le fichier ajoutVisite gère l’ajout de visites médicales pour les animaux, tandis que voirVeterinaires affiche la liste des vétérinaires disponibles. Enfin, le fichier main sert de point d’entrée principal à l’application, où la logique de routage et la structure globale sont définies. Cette organisation claire et fonctionnelle permet une gestion efficace du code et une évolution facile du projet à mesure qu'il grandit.

**LE STACK MERN :**

Tout cela est inclus dans le stack MERN, un ensemble complet de technologies qui permet de construire des applications web modernes, robustes et performantes, en utilisant JavaScript à tous les niveaux du développement, du backend au frontend. Ce stack est constitué de quatre technologies principales : MongoDB, Express.js, React, et Node.js, qui travaillent ensemble de manière fluide et cohérente.

* MongoDB, la base de données NoSQL, permet de gérer facilement de grandes quantités de données non structurées ou semi-structurées, avec un schéma flexible qui évolue facilement en fonction des besoins de l'application. Son modèle orienté document est particulièrement adapté aux applications qui nécessitent une haute scalabilité et une gestion dynamique des données.
* Express.js, un framework léger et minimaliste pour Node.js, simplifie la création de serveurs web et d'API RESTful. Il gère efficacement les requêtes HTTP, les routes, et permet de configurer des middlewares pour des fonctionnalités supplémentaires comme la gestion des erreurs, la validation des données, ou la sécurisation des routes.
* React, la bibliothèque JavaScript développée par Facebook, permet de créer des interfaces utilisateurs interactives et dynamiques grâce à son approche basée sur des composants réutilisables. Son système de gestion de l'état et son moteur de rendu virtuel (Virtual DOM) assurent une expérience utilisateur fluide, avec un affichage rapide des données en temps réel.
* Node.js, un environnement d’exécution JavaScript côté serveur, offre une exécution rapide et non bloquante du code, idéale pour gérer un grand nombre de connexions simultanées. Il est particulièrement adapté aux applications en temps réel qui nécessitent une haute performance, comme les applications de chat, les réseaux sociaux, ou les plateformes de streaming.

L'un des principaux avantages du stack MERN est sa cohérence, puisqu'il repose entièrement sur JavaScript, simplifiant ainsi le processus de développement en permettant aux développeurs de travailler avec un seul langage de programmation sur toute l'application. Cela facilite la collaboration entre les équipes frontend et backend et réduit la courbe d'apprentissage.

De plus, grâce à cette intégration étroite, le MERN stack offre une grande flexibilité et scalabilité. MongoDB et Node.js permettent de gérer des données et des connexions utilisateurs à grande échelle, tandis qu'Express.js et React offrent des outils pour une architecture modulaire et un développement rapide. Le stack MERN est idéal pour créer des applications web complexes, performantes et interconnectées, telles que des plateformes de e-commerce, des applications de gestion, des réseaux sociaux, ou des solutions SaaS.

En résumé, le MERN stack est une solution puissante et complète qui permet de créer des applications web modernes, dynamiques, et évolutives, tout en facilitant le développement grâce à l'utilisation d'un seul langage à chaque étape du processus.

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

1. Amélioration de l’interface utilisateur :

* Optimisation de la navigation mobile : Bien que l'application soit déjà responsive, une attention particulière pourrait être portée à l’amélioration de l’expérience utilisateur sur mobile en affinant certaines interactions, comme les menus ou les formulaires.
* Interface plus intuitive : Simplifier davantage la navigation en permettant à l’utilisateur de trouver rapidement un vétérinaire ou de prendre un rendez-vous via des filtres de recherche plus détaillés (par exemple, filtrer par spécialité, localisation, ou disponibilité en temps réel).
* Ajout d'une fonctionnalité de chat en direct : Intégrer une fonctionnalité de chat en direct permettant aux utilisateurs de discuter avec un vétérinaire en ligne pour des consultations rapides.

2. Fonctionnalités avancées

* Système de notation et d’avis : Permettre aux propriétaires d'animaux de laisser des avis et des notes sur les vétérinaires qu'ils ont consultés, afin de créer une communauté de confiance et d’aider d'autres utilisateurs à choisir un professionnel en fonction de ces retours.
* Rappels de rendez-vous et notifications : Intégrer un système de notifications push ou par email pour rappeler aux propriétaires les prochains rendez-vous ou les rappels de visites médicales régulières pour leurs animaux.
* Système de paiement intégré : Ajouter une fonctionnalité permettant aux utilisateurs de payer en ligne pour les visites médicales ou d’autres services vétérinaires directement via l’application, avec la possibilité de générer une facture numérique.

Sécurisation avancée :

* Mettre en place une authentification multifactorielle pour renforcer la sécurité des comptes utilisateurs, notamment pour les vétérinaires et les administrateurs.
* Ajouter une journalisation des actions pour suivre les actions importantes dans l’application, en particulier pour les données sensibles (historique des visites, informations médicales).

8. Analyse des données et rapports

* Statistiques et rapports pour les vétérinaires : Permettre aux vétérinaires de consulter des rapports statistiques sur leurs consultations, leur taux de satisfaction client, et d’autres métriques utiles pour améliorer leur pratique.
* Outils d’analyse des données utilisateurs : Ajouter des outils d’analyse qui permettent d’observer les tendances des visites (par exemple, quels types de visites sont les plus fréquentes, les animaux les plus vus) pour adapter les services proposés.

***IIIII) Conclusion :***

Le projet ViteCare représente une solution moderne et complète pour la gestion des vétérinaires et des visites médicales des animaux. En utilisant un ensemble de technologies robustes, telles que MongoDB, Node.js, Express.js, React et Next.js, l'application offre une plateforme performante, réactive et facile à maintenir. Grâce à son architecture modulaire et sa base de données NoSQL, elle est parfaitement adaptée à la gestion des informations de manière flexible et scalable, tout en permettant une évolution future avec des ajouts et des améliorations continues.

L'interface utilisateur, bien que déjà optimisée pour une expérience fluide, pourrait encore bénéficier d'améliorations sur le plan de la navigation mobile et de l'intuitivité. Des fonctionnalités supplémentaires, telles que le chat en direct, le système de notation des vétérinaires, les rappels de rendez-vous, et l'intégration d'un système de paiement en ligne, contribueraient à enrichir l'expérience utilisateur et à renforcer la valeur ajoutée de l'application.

Les perspectives de sécurisation avancée, telles que l'authentification multifactorielle et la journalisation des actions, permettent d’assurer la protection des données sensibles et de garantir un environnement sécurisé pour les utilisateurs. Enfin, les outils d'analyse des données et de création de rapports offriront aux vétérinaires des moyens efficaces pour optimiser leurs pratiques et répondre aux besoins de leurs clients, tout en ajustant les services selon les tendances observées.

En somme, ViteCare a le potentiel de devenir une plateforme incontournable pour la gestion des vétérinaires et des consultations vétérinaires, avec une forte capacité d'adaptation aux besoins futurs, tout en restant simple, accessible et sécurisée. Les prochaines étapes d'amélioration permettront de renforcer davantage l’expérience utilisateur, tout en ajoutant des fonctionnalités innovantes pour répondre aux exigences du marché.