



المدرسة العليا للتكنولوجيا الناصور
École Supérieure de Technologie de Nador
ⵜⴰⵎⴰⵔⵜ ⵜⴰⵏⵓⵔⴰⵢⵜ ⵜⴰⵖⵓⵔⴰⵏⵜ ⵜⴰⵏⵓⵔⴰⵢⵜ ⵜⴰⵖⵓⵔⴰⵏⵜ

Rapport sur l'application web VetCare360

MERN



Réalisé par : LETRACH IBTIHAL

OUKHSSANE IHSSANE

Encadrant : Professeur M.ESBAI REDOUANE

MAI 2025

Résumé

VetCare est une application web complète, conçue afin de répondre à l'ensemble du pilotage d'une clinique vétérinaire au quotidien. Développée sur la stack MERN (MongoDB, Express, React, Node.js), elle offre une interface moderne et dynamique, enrichie par Bootstrap et du CSS personnalisé pour un parcours utilisable fluide et intuitif.

Le projet répond à trois types de profils d'utilisateurs — administrateurs, vétérinaires, propriétaires d'animaux — chacun ayant des droits et une vue dédiés. Les administrateurs ont un tableau de bord central et permettent la supervision de tout le fonctionnement, la gestion des comptes utilisateurs et l'accès aux statistiques de fréquentation et de performance. Les vétérinaires peuvent ainsi programmer et enregistrer les consultations, suivre l'historique médical de leurs animaux et générer des rapports de suivi. Quant aux propriétaires, ils ont juste accès aux informations relatives à leurs animaux (profil, rendez-vous, factures) pour, en quelques clics, prendre un rendez-vous.

L'application web VetCare s'inscrit dans un projet complet de gestion quotidienne d'une clinique vétérinaire. Développée sur la stack MERN (MongoDB, Express, React, Node.js), elle est dotée d'une interface moderne et réactive, enrichie par du bootstrap et du CSS personnalisé, pour un parcours utilisateur fluide et intuitif.

Du point de vue technique, l'API back-end soutenue par Express et Node.js gère les données de manière sécurisée grâce à JWT pour l'authentification et Mongoose pour les schémas MongoDB. Le front-end React est modélisé en composants réutilisables pour communiquer, grâce à Axios, avec l'API pour faire en sorte que les données soient mises à jour en temps réel. Des tests unitaires et d'intégration ont été effectués pour assurer la fiabilité et la maintenabilité du code.

L'architecture modulaire et évolutive de VetCare facilite la montée en charge et l'implémentation future de nouvelles fonctionnalités telles que la télémedecine, les notifications SMS ou un module de facturation avancée.



En centralisant l'information et les processus, l'application permet d'améliorer l'efficacité opérationnelle de la clinique, mais également d'augmenter la satisfaction des clients, tout en fournissant un nouvel outil efficace pour accompagner son développement.



Table des matières

1. Résumé.....	2
2. Table des matières	4
3. Introduction.....	5
4. Présentation générale de VetCare	6
5. Analyse des besoins et cas d'utilisation	8
6. Conception du backend.....	10
7. Conception du frontend.....	12
8. Les outils utilisés	14
9. Les outils utilisés suite	15
10. Les outils utilisés suite.....	16
11. Les outils utilisés suite.....	17
12. L'installation des outils	18
13. Conclusions et recommandations.....	20
14. Bibliographie	23
15. Remerciements.....	25

Présentation générale de VetCare360

VetCare est une suite web destinée à moderniser la gestion quotidienne au sein des cliniques vétérinaires. Développée sur la stack MERN (MongoDB, Express, React et Node.js), l'application se distingue par un design minimaliste, responsive, enrichi grâce à Bootstrap et un CSS spécifique.

Au cœur de VetCare, se trouvent trois types d'utilisateurs, chacun disposant de divers besoins :

L'administrateur dont le tableau de bord lui permet d'effectuer toutes les gestions : les comptes utilisateurs, consulter les statistiques (taux de fréquentation, partage des rôles, indicateurs de performance), configurer les droits d'accès.

Le vétérinaire qui accède à un espace dédié à la planification de rendez-vous, à l'historique médical des animaux et qui peut ajouter des notes de suivi ou émettre des rapports.

Le propriétaire d'un animal qui, grâce à une interface ergonomique, peut prendre un rendez-vous, suivre la santé de son animal, factures et documents à télécharger.

C'est ici une solution qui, sur le fond, est modulaire, extensible et sécurisée, et qui, sur la forme, permet d'optimiser le fonctionnement de la clinique pour garantir un meilleur suivi des patients et une plus grande satisfaction des propriétaires d'animaux.

Sur le plan technique, l'API back-end repose sur Express et Node.js, avec un mécanisme d'authentification basé sur des tokens JWT et une modélisation des données (notamment validation des données...) via Mongoose sur MongoDB. Le front-end, construit en React, est basé sur des composants réutilisables et se connecte à l'API via un client HTTP Axios permettant des mises à jour en temps réel du front sans rechargement de page.



Sur le plan du système de développement, l'ensemble réparti du logiciel répond à une stratégie de mise en œuvre en CI/CD. Des tests unitaires et d'intégration lui assurent la robustesse du code, tandis qu'un plan de maintenance évolutif permet d'ajouter facilement a priori de nouvelles fonctionnalités



Analyse des besoins et cas d'utilisation

Le processus de cartographie des activités de VetCare représente les principales articulations du fonctionnement de la clinique vétérinaire. Son exploitation assure la cohérence des échanges entre utilisateurs et entre usagers et système, à partir du centralisateur de données – garant de l'automatisation des tâches récurrentes –.

1. La gestion des utilisateurs regroupe l'instance de création, de modification, de suppression des comptes et leur paramétrage conditionnel (administrateurs, vétérinaires, propriétaires, ...) en tant que configuration granulaire des droits affectés aux utilisateurs à partir d'un module d'administration détaché.
2. La gestion des dossiers animaux aborde tous les enregistrements des informations démographiques et des antécédents médicaux et des traitements permettant de faire le suivi longitudinal de chaque patient animal.
3. La planification et le suivi des rendez-vous permettent, tout d'abord aux propriétaires de solliciter un créneau, aux vétérinaires de valider ou d'ajuster la programmation et de déclencher automatiquement les notifications et de facturer ensuite.

Spécification des cas d'utilisation principaux

Les cas d'utilisation caractérisent chacune des fonctionnalités de VetCare en objets opérationnels essentiels qui viennent représenter chacun des besoins métiers. Ils cerneront les différentes interactions entre les différents acteurs (administrateur, vétérinaire et propriétaire) et le système, pour garantir le service à chaque étape en toute sécurité. Le tableau ci-après synthétise tous les cas d'utilisation – à valeur ajoutée - représentant non seulement les actions essentielles que sont l'authentification, la gestion des rendez-vous et le suivi médical, mais également des actions à fort impact stratégique comme la gestion des droits d'accès des comptes, l'édition de rapports et l'export des



factures. Cette vue d'ensemble permet de prioriser le développement, de contrôler la cohérence fonctionnelle et de suivre les évolutions futures de la plateforme :

Cas d'utilisation	Acteur	Description	Priorité
Authentification et sécurité	Tous	Connexion chiffrée via JWT et gestion sécurisée des sessions.	Critique
Administration des comptes	Administrateur	CRUD des utilisateurs et configuration des droits.	Élevée
Gestion des dossiers animaux	Vétérinaire, Propriétaire	Ajout, modification et consultation de l'historique médical.	Élevée
Prise de rendez-vous	Propriétaire, Vétérinaire	Planification, annulation et confirmation automatique des créneaux.	Critique
Production de rapports	Administrateur	Export de rapports statistiques et factures clients.	Moyenne



Conception du back-end

```
Backend/  
├── config/  
│   └── db.js  
├── models/  
│   ├── Owner.js  
│   ├── Pet.js  
│   ├── Vet.js  
│   └── Visit.js  
├── routes/  
│   ├── ownerRoutes.js  
│   ├── petRoutes.js  
│   ├── vetRoutes.js  
│   └── visitRoutes.js  
├── server.js  
├── package.json  
├── package-lock.json  
└── .env
```

La structure du répertoire backend a été pensée pour garantir la clarté, la modularité et la facilité de maintenance. Chaque dossier contient des fichiers dédiés à une responsabilité spécifique.

1. **backend/** — Racine du back-end, point d'entrée de l'API
2. **config/** — Configuration globale (connexion à la base, variables d'environnement, middlewares)
3. **config/db.js** — Initialisation et connexion à MongoDB via Mongoose
4. **models/** — Définition des schémas de données Mongoose

5. **models/Owner.js** — Modèle des propriétaires d'animaux
6. **models/Pet.js** — Modèle des animaux gérés par la clinique
7. **models/Vet.js** — Modèle des vétérinaires autorisés
8. **/Visit.js** — Modèle des visites médicales et consultations
9. **routes/** — Définition des routes Express pour chaque ressource
10. **routes/ownerRoutes.js** — Endpoints CRUD pour les propriétaires
11. **routes/petRoutes.js** — Endpoints CRUD pour les animaux
12. **routes/vetRoutes.js** — Endpoints CRUD pour les vétérinaires
13. **routes/visitRoutes.js** — Endpoints CRUD pour les visites
14. **server.js** — Point d'entrée de l'application : chargement des modules, middlewares globaux et lancement du serveur
15. **package.json** — Liste des dépendances et scripts NPM
16. **package-lock.json** — Versionning précis des modules installés
17. **.env** — Variables d'environnement : URI de MongoDB, port, clés secrètes

Cette structure compartimentée en dossiers et fichiers distincts permet d'isoler clairement les responsabilités: la configuration n'est pas mise au même niveau que les modèles de données, même indépendants des routes qui pilotent la logique métier. Au sens où le fichier `server.js` orchestre l'application, en ne chargeant que ce qui est strictement nécessaire. Un découpage qui facilite les tests unitaires, l'ajout de nouvelles ressources et la montée en charge future.



Conception du frontend

L'organisation du répertoire front-end de VetCare360 reflète la séparation claire entre les composants, les pages, les ressources statiques et la configuration de l'application React.

1. **frontend/** — Racine du projet front-end
2. | **node_modules/** — Dépendances npm installées
3. | **public/** — Fichiers statiques (index.html, favicon, manifest, assets)
4. | **src/** — Code source de l'application
5. | | **api/** — Services d'appel à l'API (Axios)
6. | | **assets/** — Images et ressources statiques personnalisées
7. | | | **3a.jpg** — Image d'illustration
8. | | | **vet.jpg** — Logo ou visuel de la clinique
9. | | **components/** — Composants réutilisables
10. | | | **Layout.js** — Disposition générale (Header, Footer)
11. | | | **Sidebar.js** — Menu de navigation latéral
12. | | **pages/** — Vues et pages principales de l'application
13. | | | **Home.js** — Page d'accueil et tableau de bord
14. | | | **Owner.js** — Liste et gestion des propriétaires
15. | | | **OwnerDetails.js** — Détails et édition d'un propriétaire
16. | | | **OwnerSearch.js** — Recherche avancée de propriétaires
17. | | | **AddOwner.js** — Formulaire d'ajout d'un propriétaire
18. | | | **EditOwner.js** — Formulaire de modification d'un propriétaire
19. | | | **PetList.js** — Liste des animaux
20. | | | **PetDetails.js** — Détails et édition d'un animal
21. | | | **AddPet.js** — Formulaire d'ajout d'un animal
22. | | | **EditPet.js** — Formulaire de modification d'un animal
23. | | | **VetList.js** — Liste des vétérinaires
24. | | | **VetDetails.js** — Détails et édition d'un vétérinaire
25. | | | **AddVet.js** — Formulaire d'ajout d'un vétérinaire



- 26. | | └─ **EditVet.js** — Formulaire de modification d'un vétérinaire
- 27. | └─ **VisitList.js** — Liste des visites/rendez-vous
- 28. | | | └─ **AddVisit.js** — Formulaire d'ajout de visite
- 29. | | | └─ **EditVisit.js** — Formulaire de modification d'une visite
- 30. | | └─ **App.js** — Point d'entrée React et configuration des routes
- 31. | | └─ **index.js** — Montage de l'application dans le DOM
- 32. | | └─ **App.css** — Styles globaux pour l'application
- 33. | | └─ **index.css** — Réinitialisation CSS et styles de base
- 34. | | └─ **setupTests.js** — Configuration des tests unitaires
- 35. | └─ **package.json** — Dépendances et scripts npm
- 36. | └─ **package-lock.json** — Verrouillage des versions des modules
- 37. | └─ **.gitignore** — Fichiers et répertoires exclus du versionnage
- 38. | └─ **README.md** — Documentation et instructions d'installation

La structure de la partie front-end de VetCare360 se compose d'une racine contenant public/, pour les fichiers statiques (HTML, favicon, manifest et assets), et src/, pour tout le code source : les services d'API dans api/ (c'est-à-dire les fonctions Axios pour communiquer avec le back-end), les images et médias dans assets/, les composants réutilisables (en l'occurrence barre de navigation, layout) dans components/, et les pages principales (listes, détails, formulaires d'ajout et de modification pour propriétaires, animaux, vétérinaires, visites) dans pages/. App.js initialise React Router pour assurer la navigation sans rechargement, index.js monte l'application dans le DOM ; App.css et index.css proposent des styles globaux. Fichiers de configuration (package.json, package-lock.json, .gitignore, README.md) et setupTests.js pour les tests apparaissent dans cette architecture modulaire, avantageuse pour la maintenabilité, la réutilisation des composants et l'évolution de l'application.



Les outils utilisés

VetCare360 est une application full-stack, notamment elle est capable de gérer toutes les couches (frontend, backend et base de données), elle se compose de deux parties distinctes et complémentaires.

Premièrement on trouve le backend qui expose un ensemble de points d'accès standards CRUD (Create, Read, Update, Delete) qui désigne les quatres opérations de base pour la persistance des données, en particulier le stockage d'informations en base de données) pour quatre entités clés: propriétaires, vétérinaires, animaux et visites. Chaque requêtes http (Get, Post, Put, Delete) est traitée par un contrôleur Express, qui consulte ou modifie les données dans une base MongoDB via Mongoose les outils utilisés auparavant:

Node.js qui permet d'exécuter du JavaScript en dehors de naviguer et de créer un serveur http e de gérer l'asynchrone.

Express.js:(Framework web léger pour Node.js) il gère la structure l'API

Rest: definition des routes (Get, Pot, Put, delete), gestion des middleware body-parser, cors, erreurs)

MongoDb:(Base de données NoSQL orientée document) qui stocke les données sous forme de document) qui stocke les données sous forme de documents JSON (propriétaires, animaux, visites, vétérinaires)

Mongoose ODM (Object-Document Mapper pour MongoDB) définit des schémas, effectue la validation, transforme les objets JS en documents Mongo

Dotenv:(Gestion des variables d'environnement) charge automatiquement les variables (connexion DB, port du serveur) depuis un fichier .env

Nodemon:(Outil de développement) qui surveille les changements de code et relance automatiquement le serveur, pour un développement plus fluide.

Cors c'est un Middleware Express pour CORS qui configure les en-têtes Cross-Origin pour autoriser le frontend (sur localhost:3000) a appeler l'API (sur localhost:5000)

NPM: est le gestionnaire de paquets (package manager) par défaut pour l'écosystème Node.js
il permet:

- ❖ D'installer des bibliothèques et outils JavaScript tiers (ex: Express, React, Mongoose...)
- ❖ De gérer les versions de ces dépendances deux on projet
- ❖ De publier les propres module sur le registre npm pour un partage commun

Une API est un ensemble de règles et de points d'accès (endpoint) qui permet à une application (le client) de communiquer avec une autre (le serveur)

Deuxièmement on trouve le frontend qui structure l'interface en modules navigables (liste, détail, ajout, modification) grâce à React Router et qui utilise Axios pour émettre les requêtes vers l'API (par exemple: récupérer la liste des propriétaires ou créer une nouvelle visite) et mettre à jour son état local (via les hooks, useState et useEffect) dès que les données arrivent, entraînant un nouveau rendu React du composant concernant les outils utilisés:





L'installation des outils

Premièrement on a téléchargé et installer depuis <https://nodejs.org>

(version LTS recommandée) puis on vérifie avec **node --version** et **npm --version**.

Ensuite on a installé MongoDB et initialiser le projet par la création du dossier VetCare360 puis on a créé ses dossiers backend et frontend puis on a utilisé **npm init-y** pour créer très vite un package.json avec les valeurs par défaut.

Par ailleurs on a installé les dépendances par les commandes suivantes:

❖ **npm install express mongoose cors dotenv**

❖ **npm install --save -dev nodemon**

Puis on a configuré les scripts dans backend/package.json ensuite on a créé un fichier de configuration .env à la racine de backend/ puis on a remplie server.js, charger dotenv, connecter MongoDB, monter les routes Express... puis on a démarré le serveur en développement avec

❖ **npm run dev**

Ou

❖ **npm start**

En outre dans le côté frontend on a créé l'app React par la commande

❖ **npx create-react-app frontend**

Puis on a installé les dépendances UI et routing par la commande: **-npm install react-router-dom axios bootstrap react-bootstrap react-icons**



puis on a configuré le proxy ,on a ajouté en frontend/package.json:

❖ **"proxy":"http://localhost:5000"**

Puis on a importé Bootstrap et on a créé l'arborescence qui se situe dans la racine qui se compose des assets/ qui est constitué des images (logo, background...) et des components/ qui se compose des pages de l'application considérons comme exemple Home.js, OwnerSearch.js, PetList.js... et de App.js qui contient les routes React Router et de App.css qui contient des styles globaux puis on l'a démarré par

❖ **npm start**

Conclusion et recommandations

VetCare360 s'affirme comme la solution performante et évolutive au service d'une gestion centralisée des dossiers patients, des agendas médicaux, des comptes rendus tout en améliorant la productivité des équipes et la satisfaction des propriétaires. Portée par une architecture MERN cohérente, l'application restitue une expérience utilisateur fluide et ergonomique des tableaux de bord administrateur jusqu'aux formulaires de consultation, en recourant à des tokens JWT, des middlewares élaborés et des composants React modulables. L'intérêt de l'application gagnerait en attractivité et efficacité en élargissant l'offre à un module de télémedecine sécurisé, permettant aux vétérinaires de réaliser des consultations et de transmettre en temps réel des documents.

D'autre part, il serait opportun de doter le logiciel de notifications automatiques pour un suivi proactif des rendez-vous et des relances de soins. De plus, l'ajout d'indicateurs analytiques avancés dans le tableau de bord faciliterait les décisions stratégiques pour la planification des ressources. Enfin, le passage à une architecture multi-clinique et la prise en charge de plusieurs langues permettraient à VetCare360 de s'ouvrir à l'international, répondant ainsi aux besoins croissants des établissements vétérinaires quelle que soit leur taille – et de leurs clients.

Les technologies sélectionnées pour la construction de VetCare360 sont un ensemble cohérent et éprouvé, garantissant la robustesse, la flexibilité et l'évolutivité du système. Côté serveur, Node.js est recommandé pour assurer la gestion efficace et performante de l'I/O asynchrone, mais également pour l'exécution de JavaScript côté back-end, garantissant ainsi la cohésion de langage avec le front-end, tout en offrant des performances élevées. Associé à Express, micro-framework léger et extensible, il permet la structuration claire et évolutive d'une API REST, la définition facile des routes, l'intégration de middlewares pour la gestion des erreurs, du CORS et de l'analyse des requêtes et l'évolutivité rapide de l'architecture pour répondre à des besoins métier exigeants. La persistance des données fait appel à MongoDB, base NoSQL orientée document particulièrement adaptée à la structure dynamique d'une base de données

de dossier médical vétérinaire, car elle permet le stockage libre et adapté des profils utilisateurs, des historiques d'animaux et des visites, tout en assurant les performances de montée en charge. Mongoose, en tant qu'ODM, met à disposition de l'API des modèles orientés documents qui permettent de manipuler les objets contenus dans la base MongoDB. Par ailleurs, le stockage des fichiers multimédia, notamment les images des animaux, passe, documents pour la dématérialisation des actes vétérinaires, peut être assuré grâce à l'objet Bucket de MongoDB. , vient enrichir cette base avec des schémas prédéfinis, des validations automatiques et des hooks personnalisables, ce qui garantit la qualité des données tout en facilitant le développement des modèles et des requêtes. Enfin pour la gestion des variables d'environnement, dotenv assure une approche à la fois simple et sécurisée grâce à un isolement de la configuration (URI de connexion, ports, clés secrètes) en dehors du code source, alors que Nodemon améliore la fluidité du cycle de développement par un rechargement automatique du serveur à chaque modification.

Côté client, la combinaison React et Create React App offre une fondation solide pour élaborer une application monopage (SPA) moderne : React facilite la création de composants réutilisables, la gestion de l'état avec les hooks `useState` et `useEffect`, et la composition de l'interface en un ensemble cohérent, tandis que Create React App prend en charge la configuration de Webpack, Babel et le hot-reload sans effort, permettant aux développeurs de se concentrer sur les fonctionnalités plutôt que sur l'outillage. La communication avec l'API est assurée par Axios, qui simplifie l'envoi de requêtes HTTP asynchrones (GET, POST, PUT, DELETE) et le traitement des réponses, garantissant un rafraîchissement partiel performant sans rechargement de page. Pour le design, Bootstrap et React Bootstrap confèrent une grille responsive, des composants esthétiques et des thèmes personnalisables, tandis que le CSS sur mesure permet d'affiner la charte graphique selon l'identité visuelle de la clinique. Le routage interne s'appuie sur React Router DOM, garantissant une navigation fluide et sécurisée, y compris pour les routes protégées nécessitant une authentification. La sécurisation des échanges est complétée par l'utilisation de JSON Web Tokens (JWT) et de cookies `HttpOnly`, empêchant les failles XSS et assurant une gestion de session fiable. Enfin, l'écosystème npm



accompagne l'ensemble du projet, facilitant l'installation, la mise à jour et la publication des dépendances, avec un contrôle précis des versions grâce au package-lock.json. Cet ensemble d'outils, soigneusement choisi et configuré, forme une chaîne de développement unifiée qui maximise la productivité, la maintenabilité et la qualité du code, tout en préparant VetCare360 à évoluer vers des fonctionnalités toujours plus avancées



Bibliographie

- Docker, Inc. (2025). Docker Documentation. Retrieved May 11, 2025, from <https://docs.docker.com/>
- Express.js (2025). Express: Fast, unopinionated, minimalist web framework for Node.js. Retrieved May 11, 2025, from <https://expressjs.com/>
- JSON Web Tokens (n.d.). Introduction to JSON Web Tokens. Retrieved May 11, 2025, from <https://jwt.io/introduction/>
- Mongoose (2025). Mongoose: MongoDB object modeling tool designed to work in an asynchronous environment. Retrieved May 11, 2025, from <https://mongoosejs.com/>
- MongoDB, Inc. (2025). MongoDB Documentation. Retrieved May 11, 2025, from <https://docs.mongodb.com/>
- Node.js Foundation (2025). Node.js. Retrieved May 11, 2025, from <https://nodejs.org/>
- React (2025). React – A JavaScript library for building user interfaces. Retrieved May 11, 2025, from <https://reactjs.org/>



- React Bootstrap (2025). React Bootstrap documentation. Retrieved May 11, 2025, from <https://react-bootstrap.github.io/>
- Bootstrap (2025). Bootstrap: The most popular HTML, CSS, and JS library in the world. Retrieved May 11, 2025, from <https://getbootstrap.com/>
- Axios (2025). Axios – Promise based HTTP client for the browser and node.js. Retrieved May 11, 2025, from <https://axios-http.com/>
- Oracle university from <https://www.oracle.com/education/>
- Github from <https://github.com/>
- Chatgpt from <https://chatgpt.com/>



Glossaire des termes techniques

- **MERN** : Stack JavaScript composée de MongoDB, Express, React et Node.js
- **API REST** : Interface de programmation basée sur le protocole HTTP pour la manipulation de ressources
- **JWT** : JSON Web Token, standard pour transmettre des données authentifiées entre client et serveur
- **CRUD** : Opérations fondamentales Create, Read, Update, Delete pour la gestion des données

Remerciements

On tient à remercier chaleureusement notre encadrant de projet **Professeur M.ESBAI REDOUANE**, et tous qui nous ont aidé à réaliser cette application et partager leur expertise et leurs retours précieux tout au long du développement de VetCare360.