



Projet de Fin de semestre

Diplôme Universitaire de Technologie

Filière : Ingénierie Logicielle et Cybersecurity

*Une application web pour
la gestion d'une clinique vétérinaire
vetcare 360*

Réalisé par :

-Mohamed Aymane Jlida
-Norhane Ramzi

Encadré par :

Mr. Esbai Redouane

Table des matières

1. Introduction
2. Présentation du Projet
3. Objectifs du Projet
4. Fonctionnalités Principales
 - o Gestion des propriétaires
 - o Gestion des animaux
 - o Gestion des visites
 - o Affichage des vétérinaires
5. Architecture
 - o Architecture Technique Détailée
 - o Environnement de Développement
 - o Structure du Code
 - o Explication des fichiers du projet
6. Comparaison entre SQL et MongoDB pour VetCare 360
 - o Choix de MongoDB face aux bases relationnelles
 - o Avantages de MongoDB dans le contexte de VetCare 360
7. Scénario Utilisateur
 - o Enregistrement d'une Visite Médicale
8. Flux de Données
9. Détails du Flux
10. Démarrage de l'application
 - o Étapes de démarrage
11. Perspectives d'Amélioration
12. Résolution des Problèmes Courants
13. Conclusion

1. Introduction

Dans le cadre de notre formation, nous avons réalisé VetCare 360, une application web dédiée à la gestion des cliniques vétérinaires. Ce projet répond à un besoin réel : centraliser les données des propriétaires, des animaux, des vétérinaires, ainsi que l'historique complet des visites médicales. Pensée pour une utilisation professionnelle, l'application permet une gestion fluide et efficace grâce à une interface moderne développée avec React.js, et un backend robuste propulsé par Node.js et Express.js. Les données sont stockées dans une base MongoDB, garantissant souplesse et rapidité d'accès. Ce projet met en pratique les fondements du développement web moderne tout en répondant aux exigences concrètes d'un environnement métier.

2. Présentation du Projet

Le projet **VetCare 360** est une application web complète conçue pour la gestion quotidienne d'une clinique vétérinaire. Développée avec la stack MERN (MongoDB, Express.js, React.js, Node.js), elle répond aux besoins concrets des professionnels du domaine en centralisant toutes les données essentielles au même endroit.

Cette solution permet de :

- Gérer les **propriétaires d'animaux** : informations personnelles, fiches clients.
- Administrer les **animaux** : dossiers médicaux, races, âges.
- Organiser les **vétérinaires** : spécialisations, disponibilité.
- Suivre les **visites médicales** : motifs, diagnostics, traitements.

Grâce à son interface intuitive construite avec **React**, et sa base de données flexible en **MongoDB**, VetCare 360 facilite le travail des cliniques vétérinaires en offrant une gestion fluide, moderne et centralisée.

3. Objectifs du Projet

- Gérer les propriétaires et leurs informations.
- Associer les animaux à leurs propriétaires.
- Planifier et suivre les visites médicales des animaux.
- Lister les vétérinaires et leurs interventions.
- Proposer une interface moderne et fonctionnelle avec React.

4. Fonctionnalités Principales

Gestion des propriétaires :

- Ajouter un nouveau propriétaire.
- Modifier les informations d'un propriétaire.
- Supprimer un propriétaire.
- Rechercher un propriétaire par son nom.

Gestion des animaux :

- Associer des animaux à un propriétaire.
- Modifier les informations d'un animal.
- Supprimer un animal.
- Voir l'historique des visites d'un animal.

Gestion des visites :

- Ajouter une visite médicale pour un animal.
- Visualiser l'historique des visites.
- Supprimer une visite médicale.

Affichage des vétérinaires :

- liste récupérée depuis la base de données.

5. Architecture

Architecture Technique Détailée

Le projet **VetCare 360** repose sur une architecture MERN (MongoDB, Express.js, React.js, Node.js) moderne, respectant les principes du développement web full-stack.

Environnement de Développement :

- **React.js** : bibliothèque JavaScript pour construire une interface utilisateur dynamique.
- **Node.js + Express.js** : backend léger, rapide, conçu pour créer des API RESTful.
- **MongoDB** : base de données NoSQL orientée documents.
- **Mongoose** : ODM pour modéliser les schémas et interagir avec MongoDB.
- **Axios** : client HTTP pour les appels API côté React.
- **Git & GitHub** : pour le versionnement et la collaboration.

Structure du Code :

Frontend – React

```
vetcare/
├── public/
└── src/
  ├── components/          # Composants réutilisables
  │   ├── Footer.js        # Pied de page
  │   └── Navbar.js         # Barre de navigation principale
  ├── pages/               # Pages principales de l' application
  │   ├── AddOwner.jsx      # Formulaire d'ajout d'un propriétaire
  │   └── EditOwner.jsx     # Modification d'un propriétaire
```

```
|   |   ├── OwnerDetails.jsx    # Fiche détaillée d'un propriétaire  
|   |   ├── OwnersList.jsx    # Liste des propriétaires  
|   |   ├── AddPet.jsx        # Ajout d' un animal de compagnie  
|   |   ├── EditPet.jsx       # Modification d' un animal  
|   |   ├── AddVisit.jsx      # Ajout d'une visite médicale  
|   |   ├── VetsList.jsx      # Liste des vétérinaires  
|   |   └── HomePage.jsx      # Page d' accueil  
|   └── App.tsx              # Composant principal React  
|   └── index.tsx            # Point d'entrée de l'application  
|   └── index.css            # Styles globaux  
└── reportWebVitals.ts      # Outil de mesure de performance
```

Backend – Node.js / Express

```
vet-backend/  
├── models/                  # Modèles Mongoose (MongoDB)  
|   ├── Owner.js              # Modèle des propriétaires  
|   ├── Pet.js                # Modèle des animaux  
|   ├── Visit.js              # Modèle des visites médicales  
|   └── Veterinarian.js       # Modèle des vétérinaires  
├── routes/                  # Routes Express pour l' API REST  
|   ├── ownerRoutes.js         # Routes liées aux propriétaires  
|   ├── petRoutes.js          # Routes liées aux animaux  
|   ├── visitRoutes.js         # Routes liées aux visites  
|   └── veterinarianRoutes.js # Routes liées aux vétérinaires  
└── middleware/
```

```
|   └── errorHandler.js      # Gestion centralisée des erreurs  
|  
├── seedOwners.js          # Script de peuplement de la base  
|  
├── seedPets.js  
|  
├── seedVisits.js  
|  
└── seedVeterinarians.js  
  
├── .env                   # Variables d' environnement (ex : URI MongoDB)  
|  
├── server.js              # Point d' entrée du serveur Express  
|  
└── package.json            # Dépendances et scripts de lancement
```

Explication des fichiers du projet VetCare 360 (MERN)

Frontend – React

- **src/components/Navbar.js et Footer.js**

Contiennent des éléments d'interface réutilisables comme la barre de navigation et le pied de page visibles sur toutes les pages.

- **src/pages/**

Dossier central du frontend, il regroupe toutes les pages du site :

- AddOwner.jsx / EditOwner.jsx : ajout et modification d'un propriétaire.
- OwnerDetails.jsx : page de détails affichant un propriétaire et ses animaux.
- AddPet.jsx / EditPet.jsx : ajout et modification des animaux de compagnie.
- AddVisit.jsx : ajout d'une visite médicale pour un animal.
- VetsList.jsx : liste des vétérinaires enregistrés.
- OwnersList.jsx : liste globale des propriétaires.

- HomePage.jsx : page d'accueil de l'application.
 - **App.tsx**

Composant principal de l'application React ; il gère le routage entre les différentes pages.
 - **index.tsx**

Point d'entrée de l'application React. C'est ici que l'application est rendue dans le DOM.
 - **index.css / OwnerList.css, etc.**

Fichiers CSS spécifiques aux composants ou pages pour un design personnalisé.
-
- ## Backend – Node.js / Express
- **models/**
 - Owner.js, Pet.js, Visit.js, Veterinarian.js : fichiers définissant les schémas de données MongoDB avec Mongoose pour chaque entité (propriétaires, animaux, visites, vétérinaires).
 - **routes/**
 - ownerRoutes.js, petRoutes.js, visitRoutes.js, veterinarianRoutes.js : fichiers définissant les routes Express pour accéder et manipuler les données de chaque entité via des API REST (GET, POST, PUT, DELETE).
 - **middleware/errorHandler.js**

Gère les erreurs côté serveur et envoie des réponses claires aux clients.
 - **seedOwners.js, seedPets.js, seedVisits.js, seedVeterinarians.js**

Scripts pour remplir automatiquement la base de données avec des données de test.

- **.env**

Contient les variables sensibles comme l'URL de connexion MongoDB ou le port utilisé par le serveur.

- **server.js**

Point d'entrée du backend : il initialise le serveur Express, connecte MongoDB, applique les middlewares, et active les routes.

- **package.json**

Fichier de configuration contenant les dépendances Node.js, les scripts de démarrage (npm start) et d'autres métadonnées du projet.

6. Comparaison entre SQL et MongoDB pour VetCare 360

Choix de MongoDB face aux bases relationnelles

Le choix entre une base de données relationnelle (SQL) et une base de données orientée documents comme **MongoDB** repose sur des différences fondamentales dans la manière de structurer et manipuler les données.

Les bases **SQL** se caractérisent par une structure rigide, fondée sur des schémas fixes et la normalisation des données. Elles sont parfaitement adaptées aux systèmes nécessitant une forte **intégrité des données**, avec des **relations complexes** entre entités, gérées via des **jointures**.

Cette approche est idéale dans des environnements où les dépendances entre les tables sont critiques et doivent être strictement contrôlées.

À l'inverse, **MongoDB** propose une structure **plus souple**, permettant de manipuler des données semi-structurées grâce à des documents JSON imbriqués. Cela le rend particulièrement adapté aux **modèles hiérarchiques**, comme dans le cas de **VetCare 360**, où chaque animal peut avoir plusieurs visites médicales associées. Au lieu de répartir ces données sur plusieurs tables comme en SQL, MongoDB permet de les regrouper directement dans un même document, simplifiant ainsi la lecture et l'écriture des données.

Ce modèle de données flexible est un véritable atout pour VetCare 360, où les entités (animaux, visites, propriétaires) sont **étroitement liées**. Il permet une meilleure performance sans les contraintes imposées par un schéma relationnel strict.

Avantages de MongoDB dans le contexte de VetCare 360 :

- **Flexibilité du schéma** : idéal pour les dossiers médicaux évolutifs.
- **Performance accrue** : lecture/écriture rapide sans jointures complexes.
- **Évolutivité** : mise à l'échelle horizontale facilitée.
- **Format JSON natif** : parfaitement compatible avec les APIs REST.
- **Intégration fluide avec Node.js** : via Mongoose, simplifiant la manipulation des données côté backend.

7. Scénario Utilisateur

Enregistrement d'une Visite Médicale

Dans le cadre du fonctionnement quotidien d'une clinique vétérinaire, un membre du personnel — tel qu'un réceptionniste ou un vétérinaire — utilise l'interface de l'application VetCare 360 pour enregistrer une nouvelle consultation médicale.

Depuis la page OwnersList, l'utilisateur commence par vérifier si le **propriétaire de l'animal** est déjà enregistré. Si ce n'est pas le cas, il saisit les informations nécessaires : nom, prénom, adresse, ville et numéro de téléphone. Une fois le propriétaire ajouté, il procède à la création de la **fiche de l'animal**, en renseignant son nom, sa date de naissance, et son type (ex. : chat, lézard).

Ensuite, l'utilisateur accède à la section “**Visites**” pour créer une nouvelle consultation. Il remplit un formulaire incluant la **date de la visite**, une **description des symptômes ou observations**. Cette étape permet également d'ajouter des notes sur le **diagnostic** ou les **traitements prescrits**.

Toutes les informations saisies sont automatiquement reliées entre elles via des **appels API** vers le backend Express, ce qui génère des documents interconnectés dans la base de données **MongoDB** (propriétaires → animaux → visites → vétérinaires). Ce scénario illustre la **simplicité d'utilisation** et la **centralisation des données** dans l'application, même sans système d'authentification ou de rôles utilisateurs.

8. Flux de Données

- 1 L'utilisateur interagit avec l'interface utilisateur développée en React.

- 2 Le frontend envoie une requête HTTP à l'API backend via axios.
- 3 Le backend, construit avec Express.js, traite la requête à travers les routes définies.
- 4 Les contrôleurs associés appliquent la logique métier : validation, vérifications, décisions.
- 5 Les modèles Mongoose se chargent de communiquer avec la base de données MongoDB.
- 6 Une fois la requête traitée, la réponse remonte au frontend.
- 7 L'interface utilisateur se met à jour dynamiquement en fonction des données reçues.

9. Details du Flux

- **Frontend → API Backend :**

Avant l'envoi de chaque requête, le frontend valide les champs saisis, gère les états locaux avec useState, et optimise les appels pour éviter les surcharges inutiles.

- **Communication vers l'API :**

Les données sont ensuite sérialisées (format JSON) et envoyées. Si une authentification était requise, un jeton serait ajouté dans les headers.

- **Traitement côté Backend :**

Le backend vérifie les données reçues, exécute les opérations demandées (recherche, ajout, mise à jour, suppression), et optimise les accès à MongoDB via des méthodes adaptées.

- **Interaction avec MongoDB :**

La base de données traite les requêtes. Les données peuvent être filtrées, enrichies (populate) ou formatées avant d'être retournées.

- **Retour des données :**

Le backend renvoie une réponse claire, gère les éventuelles erreurs (statuts HTTP) et formate les résultats pour le frontend.

- **Affichage côté client :**

Dès réception, React met à jour automatiquement l'interface (DOM virtuel), ajuste les composants affichés et fournit un retour utilisateur (notification, message de succès, etc.).

10. Démarrage de l'application

L'application **VetCare 360** se lance en deux étapes distinctes : le démarrage du serveur backend, puis celui du frontend React. Chaque partie fonctionne indépendamment, mais communique via une API REST.

Étapes de démarrage :

- **1. Lancer le backend** (dans le dossier `vet-backend/`) :
 - `npm install`
 - `npm start`

```
PS C:\Users\Lenovo\Desktop\vetcare360\vetcare360\vet-backend> npm start
> vetcare-backend@1.0.0 start
> node server.js

Connecté à MongoDB
Serveur prêt sur http://localhost:5000
```

Ce processus initialise le serveur Express et établit la connexion avec la base de données MongoDB. Par défaut, le backend est accessible à l'adresse :

`http://localhost:5000`

- **2. Lancer le frontend** (dans le dossier `vetcare/`) :

- npm install
- npm start

```
PS C:\Users\Lenovo\Desktop\vetcare360v\vetcare360v\vetcare> npm start

> vetcare@0.1.0 start
> react-scripts start

(node:27616) [DEP_WEBPACK_DEV_SERVER_ON_AFTER_SETUP_MIDDLEWARE] DeprecationWarning: 'onAfterSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
(Use `node --trace-deprecation ...` to show where the warning was created)
(node:27616) [DEP_WEBPACK_DEV_SERVER_ON_BEFORE_SETUP_MIDDLEWARE] DeprecationWarning: 'onBeforeSetupMiddleware' option is deprecated. Please use the 'setupMiddlewares' option.
Starting the development server...
Compiled successfully!

You can now view vetcare in the browser.

Local:          http://localhost:3000
On Your Network:  http://192.168.56.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Cette commande lance l'interface utilisateur développée avec React, généralement accessible via :

<http://localhost:3000>

La communication entre les deux serveurs est assurée par un proxy configuré dans le frontend (package.json), redirigeant automatiquement les appels API vers le backend.

11. Perspectives d'Amélioration

Bien que **VetCare 360** constitue déjà une solution fonctionnelle pour la gestion des animaux, des propriétaires, des visites médicales et des vétérinaires, plusieurs axes d'évolution peuvent être envisagés afin d'enrichir l'application et de la rendre plus robuste et adaptée à un usage professionnel réel.

Parmi les améliorations envisageables :

A. Ajout d'un système d'authentification :

intégrer une authentification sécurisée avec gestion des rôles (administrateur, vétérinaire, réceptionniste) permettrait de restreindre l'accès aux données sensibles en fonction du profil utilisateur.

B. Gestion des rendez-vous avec calendrier interactif :

l'implémentation d'un module de planification permettrait aux vétérinaires de visualiser leurs créneaux disponibles et aux utilisateurs de réserver directement en ligne.

C. Tableau de bord analytique :

un espace d'administration pourrait afficher des statistiques clés telles que le nombre de visites par période, le suivi vaccinal des animaux, ou encore les vétérinaires les plus sollicités.

D. Notifications automatiques :

l'envoi d'e-mails ou de SMS pour rappeler les rendez-vous ou les traitements en attente améliorerait considérablement l'expérience utilisateur et la relation client.

E. Déploiement cloud :

héberger l'application sur une plateforme cloud (comme Vercel, Render, Railway ou Heroku) permettrait un accès distant à la solution, rendant son utilisation possible dans un cadre professionnel multi-utilisateurs.

Ces évolutions renforceraient la valeur ajoutée de VetCare 360 et en feraient une plateforme complète, moderne et adaptée aux besoins des cliniques vétérinaires de demain.

12. Résolution des Problèmes Courants

Durant le développement de **VetCare 360**, plusieurs problèmes techniques ont été rencontrés. Voici un résumé des erreurs les plus fréquentes et des solutions mises en œuvre :

◊ Problème 1 : Erreurs CORS lors des appels API

Symptôme : Le navigateur bloque les requêtes entre le frontend (port 3000) et le backend (port 5000).

Solution : Utilisation du middleware `cors` dans le backend Express pour autoriser les requêtes croisées :

```
const cors = require("cors");
app.use(cors());
```

◊ Problème 2 : Connexion échouée à MongoDB

Symptôme : Le backend ne parvient pas à se connecter à la base de données.

Solution : Vérification de la variable `MONGO_URI` dans le fichier `.env` et bon démarrage de MongoDB local.

◊ Problème 3 : Mauvaise structuration ou duplication du code

Symptôme : Difficulté à maintenir ou réutiliser certains composants.

Solution : Réorganisation du code en composants modulaires (`pages/`, `components/`) et utilisation des props pour les rendre génériques.

◊ Problème 4 : API ne répond pas ou crash à une requête

Symptôme : Erreur 500 ou crash serveur.

Solution : Mise en place d'un middleware `errorHandler` pour capturer et afficher proprement les erreurs sans arrêter le serveur.

◊ Problème 5 : Problèmes de synchronisation entre frontend et backend

Symptôme : Conflits entre les formats de données échangés.

Solution : Validation et sérialisation des données envoyées/retournées (format JSON bien structuré, respect des noms de champs).

Cette gestion proactive des problèmes a permis d'améliorer la stabilité, la maintenabilité et l'expérience utilisateur de l'application.

13. Conclusion

VetCare 360 constitue une solution complète, moderne et efficace pour la gestion des cliniques vétérinaires, s'appuyant sur la stack technologique **MERN** (MongoDB, Express.js, React.js, Node.js). L'application allie une interface utilisateur intuitive et responsive à un backend structuré, assurant ainsi une expérience fluide et performante pour les vétérinaires, les assistants et les personnels d'accueil.

Grâce à ses fonctionnalités couvrant la gestion des propriétaires, des animaux, des visites médicales et la planification des rendez-vous, VetCare 360 répond aux principaux besoins d'une clinique vétérinaire dans sa gestion quotidienne. L'architecture modulaire adoptée facilite la maintenance du code et prépare le terrain pour de futures évolutions (authentification, notifications, tableau de bord, etc.).

Par ailleurs, une attention particulière a été portée à la **sécurité des données**, à la **validation des entrées**, et à la **gestion des erreurs**, garantissant la fiabilité et la robustesse de l'application dans un environnement professionnel.

VetCare 360 représente ainsi une base solide et évolutive pour bâtir une application métier pleinement opérationnelle et adaptée aux besoins concrets des professionnels du secteur vétérinaire.

**Pour Consulter le code source complet de l'application, Vous Pouvez
Accéder au dépôt GitHub via le lien suivant :**

<https://github.com/M-Aymane-404/vetcare360VWeb>

N'hésitez pas à explorer le projet pour une meilleure compréhension de l'architecture, des fonctionnalités et des choix techniques mis en œuvre.

Merci pour votre attention.