

Ecole Supérieure de Technologie

Département d’informatique

RAPPORT DE PROJET

VetCare 360

Application web de gestion d’une clinique vétérinaire – Architecture MERN

Réalisé par : Souhayla Ouchen

Encadré par : M.Redouane Esbai

Année universitaire : 2024/2025

TABLE DES MATIÈRES

1. Introduction générale ........................................................................... 3
2. Problématique et contexte ................................................................... 4
3. Étude théorique des technologies MERN ............................................ 4
4. 3.1 Définition du stack MERN ........................................................................ 4
5. 3.2 MongoDB : définition et rôle dans le projet .............................................. 5
6. 3.3 Express.js : définition et utilisation ........................................................... 6
7. 3.4 React.js : définition et rôle dans l’interface .............................................. 7
8. 3.5 Node.js : définition et rôle côté serveur ................................................... 8
9. Comparaison entre les outils MERN .................................................... 8
10. Modélisation de la base de données ................................................. 9
11. Conception des interfaces ................................................................. 11
12. Architecture du projet ........................................................................ 13
13. Développement Frontend .................................................................. 14
14. Développement Backend ........................................................ 16
15. Connexion à MongoDB ..................................................................... 17
16. Gestion de version avec GitHub ........................................................ 19
17. Tests et validation ............................................................................. 20
18. Problèmes et solutions ...................................................................... 21
19. Conclusion et perspectives ............................................................... 22
20. Annexes ............................................................................................. 23

## Introduction générale

### **1. Présentation du projet**

Le projet *VetCare 360* est une application web destinée à faciliter la gestion quotidienne d’une clinique vétérinaire. Cette application permet de centraliser les informations concernant les propriétaires d’animaux, les animaux eux-mêmes, les visites médicales, ainsi que les vétérinaires.  
 Elle a été développée en utilisant le **stack MERN** (MongoDB, Express.js, React.js, Node.js), une architecture moderne et complète permettant de construire des applications web performantes en JavaScript, du serveur jusqu’au navigateur.

### **2. Objectifs du projet**

Les objectifs principaux de ce projet sont :

* Offrir une solution numérique moderne pour les cliniques vétérinaires.
* Permettre une gestion structurée des dossiers médicaux des animaux.
* Améliorer l’organisation des rendez-vous et des visites.
* Simplifier la consultation des informations via une interface conviviale.
* Mettre en pratique les connaissances en développement web acquises pendant la formation.

### **3. Contexte pédagogique**

Ce travail s’inscrit dans le cadre d’un **mini-projet académique** réalisé en première année universitaire. Il a pour but de :

* Introduire les étudiants aux concepts de développement web complet
* .Favoriser l’autonomie et le travail en équipe.
* Apprendre à structurer et documenter un projet informatique du début à la fin.

## **Problématique et contexte**

### **1. Problématique**

Dans les cliniques vétérinaires traditionnelles, la gestion des dossiers médicaux des animaux est souvent réalisée de manière manuelle ou via des outils non spécialisés (comme Excel ou papier).  
 Cela engendre plusieurs **problèmes** :

* Perte ou duplication des informations.
* Difficulté d’accès rapide à l’historique médical d’un animal.
* Organisation inefficace des rendez-vous.
* Manque d’interface adaptée aux besoins des vétérinaires.

### **2. Contexte d’utilisation**

VetCare 360 a été conçu pour répondre à ces besoins dans le **contexte d’une petite ou moyenne clinique vétérinaire** :

* Plusieurs vétérinaires travaillent en alternance.
* Les visites peuvent être urgentes ou programmées.
* Il est essentiel de retrouver facilement les informations liées aux animaux, à leurs maîtres, et aux soins reçus.

### **3. Enjeux**

Offrir une solution numérique moderne, accessible, simple à utiliser et qui améliore le **flux de travail** dans la clinique.

## **Étude théorique des technologies MERN**

### **1. Définition du stack MERN**

Le stack **MERN** est un ensemble de technologies JavaScript utilisées pour créer des applications web complètes .  
 Il est composé de :

* **MongoDB** : Base de données NoSQL.
* **Express.js** : Cadre (framework) côté serveur.
* **React.js** : Bibliothèque pour créer l’interface utilisateur.
* **Node.js** : Environnement d’exécution JavaScript côté serveur.

MERN permet d’utiliser **JavaScript dans tout le projet**, du client au serveur.

### **2. Pourquoi MERN ?**

* Open-source, gratuit et largement documenté.
* Basé sur une seule langue de programmation (JS).
* Convient aux petites comme aux grandes applications.
* Flexibilité et rapidité de développement.

## **MongoDB – Définition et rôle dans le projet**

### **1. Définition**

**MongoDB** est une base de données **NoSQL** qui stocke les données sous forme de **documents JSON**.  
 Contrairement aux bases relationnelles, elle ne nécessite pas de schéma fixe.

### **2. Avantages**

* Structure souple (flexible schema).
* Adaptée aux projets web modernes.
* Haute performance en lecture et écriture.
* Compatible avec Node.js via Mongoose.

### **3. Rôle dans VetCare 360**

Dans ce projet, MongoDB est utilisée pour stocker toutes les données de la clinique, notamment :

* Les **propriétaires** (nom, contact, adresse...),
* Les **animaux** (espèce, nom, âge...),
* Les **visites médicales** (date, diagnostic...),
* Les **vétérinaires** (nom, spécialité...).

Chaque entité est modélisée comme un **document** indépendant, ce qui permet une grande souplesse dans les ajouts et modifications.

## **Express.js – Définition et rôle dans le projet**

### **1. Définition**

**Express.js** est un *framework* minimaliste pour **Node.js** qui facilite le développement d'applications web et d'API. Il permet de gérer les routes, les requêtes HTTP, les réponses, la sécurité, etc.

### 2. Avantages

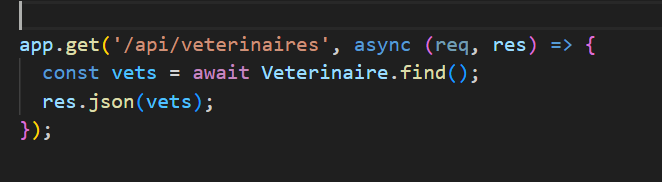
|  |  |
| --- | --- |
| **Fonctionnalité** | **Description** |
| Légèreté | Ne surcharge pas le projet avec des dépendances inutiles |
| Middleware | Permet d’ajouter facilement des fonctionnalités |
| Routing flexible | Gestion simple des URL et des méthodes HTTP |
| Intégration avec MongoDB | Fonctionne très bien avec Mongoose |

### **3. Rôle dans VetCare 360**

Dans VetCare 360, **Express.js** est utilisé pour :

* Créer des **routes REST API** telles que :
  + GET /api/animaux
  + POST /api/proprietaires
  + PUT /api/visites/:id
* Gérer les requêtes HTTP envoyées depuis le frontend (React).
* Sécuriser l'accès à certaines données (via middleware).

### 4. Exemple de code



## **React.js – Définition et rôle dans le projet**

### **1. Définition**

**React.js** est une bibliothèque JavaScript développée par Facebook, utilisée pour construire des **interfaces utilisateur dynamiques**. Elle repose sur le concept de **composants réutilisables** et la gestion d’un **état (state)** local ou global.

### 2. Avantages

|  |  |
| --- | --- |
| **Atout** | **Explication** |
| Composants réutilisables | Chaque partie de l'interface peut être isolée et réutilisée |
| Virtual DOM | Meilleure performance dans le rendu |
| Réactivité | Interface mise à jour automatiquement selon les données |
| Écosystème riche | Compatible avec Redux, Bootstrap, React Router... |

### **3. Rôle dans VetCare 360**

React est utilisé pour :

* Afficher les pages dynamiques :
  + Liste des vétérinaires (Page 2),
  + Formulaire de recherche des propriétaires (Page 3),
  + Formulaire d’ajout d’animal ou de visite (Pages 7 et 11).
* Gérer les **états des composants** (ex. propriétaire sélectionné).
* Naviguer entre les pages avec **React Router**.

## **Node.js – Définition et rôle dans le projet**

### **1. Définition**

**Node.js** est un environnement d'exécution pour JavaScript, qui permet d’exécuter du code JS côté **serveur**. Il est basé sur le moteur V8 de Google Chrome.

### 2. Avantages

|  |  |
| --- | --- |
| **Avantage** | **Description** |
| Un langage unique | Utilisation de JavaScript côté client et serveur |
| Haute performance | Architecture non-bloquante, idéale pour les API |
| Écosystème riche | Accès à des milliers de modules via npm |
| Compatible avec Mongo | Intégration naturelle avec Mongoose/MongoDB |

### **3. Rôle dans VetCare 360**

Node.js sert à :

* **Exécuter le backend** de l’application.
* Héberger l’API Express.
* Gérer les appels à la base MongoDB.
* Servir les fichiers statiques si nécessaire (optionnel en dev).

### **4. Schéma récapitulatif du rôle de chaque technologie (Stack MERN)**

React (Frontend) ←→ Express + Node (API Backend) ←→ MongoDB (Base de données)

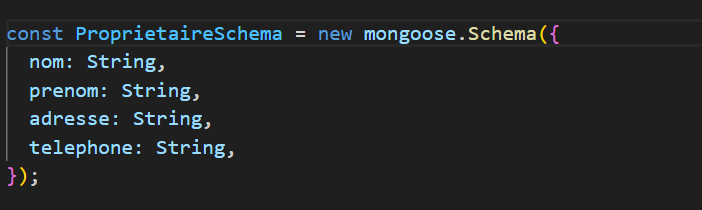
## **Modélisation de la base de données**

### **1. Approche NoSQL avec MongoDB**

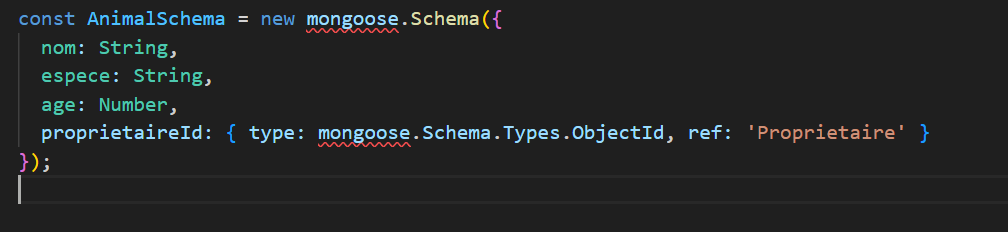
MongoDB utilise une structure en **documents JSON** (plutôt que des tables comme en SQL). Chaque document représente une entité : *propriétaire, animal, vétérinaire, visite*.

### **2. Modèles de données (Schemas)**

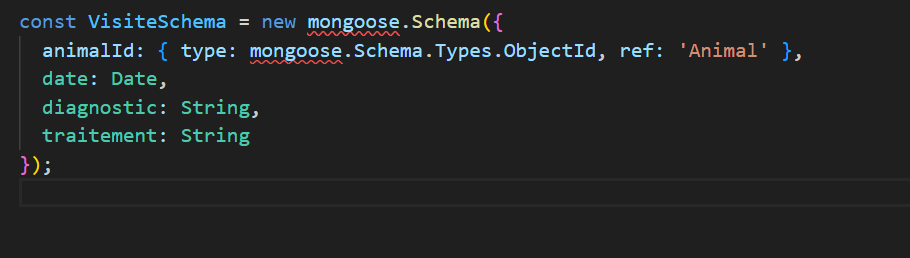
#### **➤ Exemple de schéma Mongoose pour un propriétaire :**



#### ➤ Exemple pour un animal :



#### ➤ Exemple de visite médicale :



### 3. Relations entre les entités

|  |  |
| --- | --- |
| **Entité** | **Relation** |
| Propriétaire | Possède plusieurs animaux |
| Animal | Appartient à un seul propriétaire |
| Animal | Peut avoir plusieurs visites |
| Visite | Rattachée à un animal |

### **4. Diagramme conceptuel simplifié (ERD)**

[Propriétaire] 1───∞ [Animal] 1───∞ [Visite]

## **Conception des interfaces utilisateur (UI/UX)**

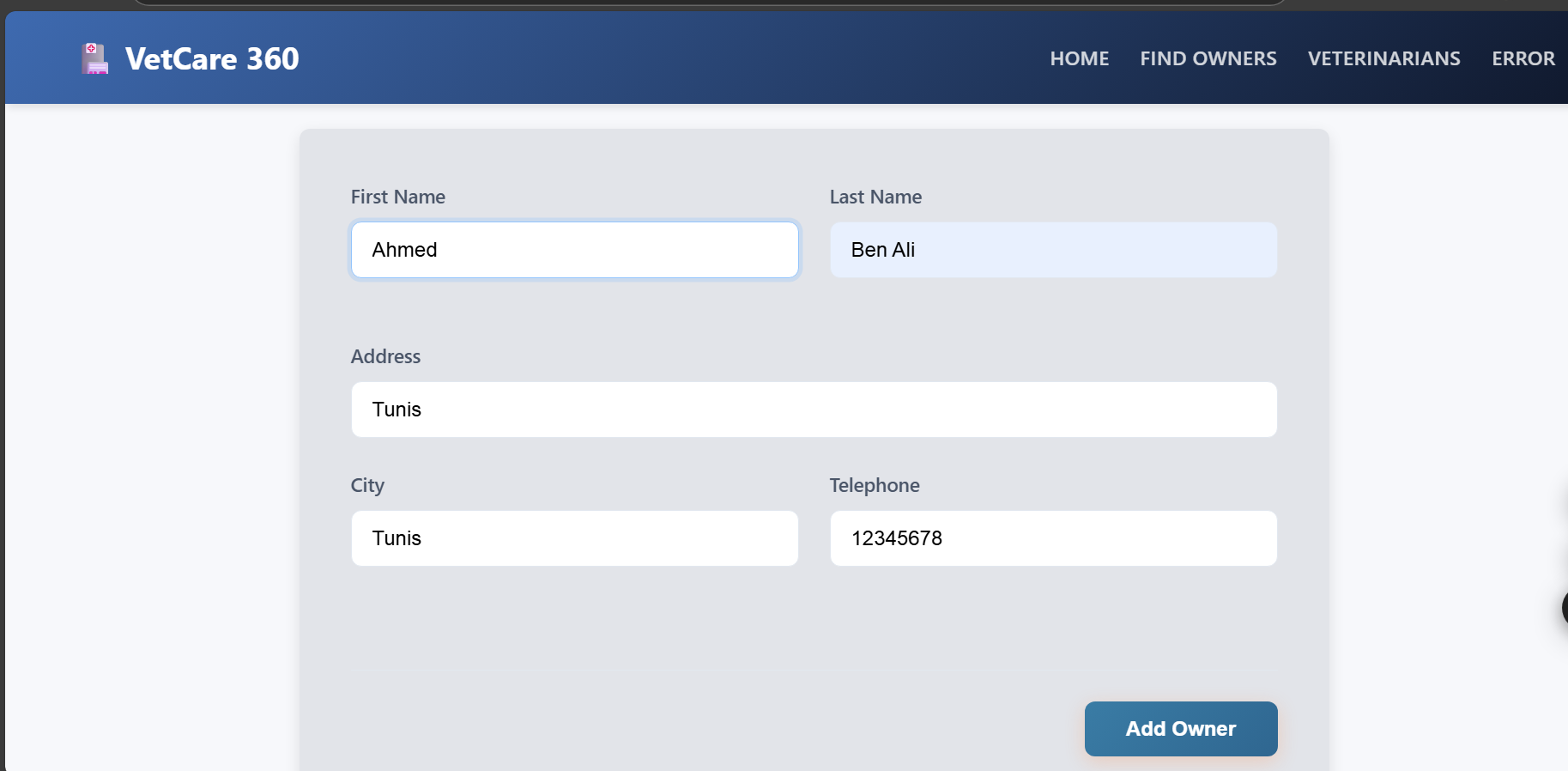
### **1. Pages principales (répartition)**

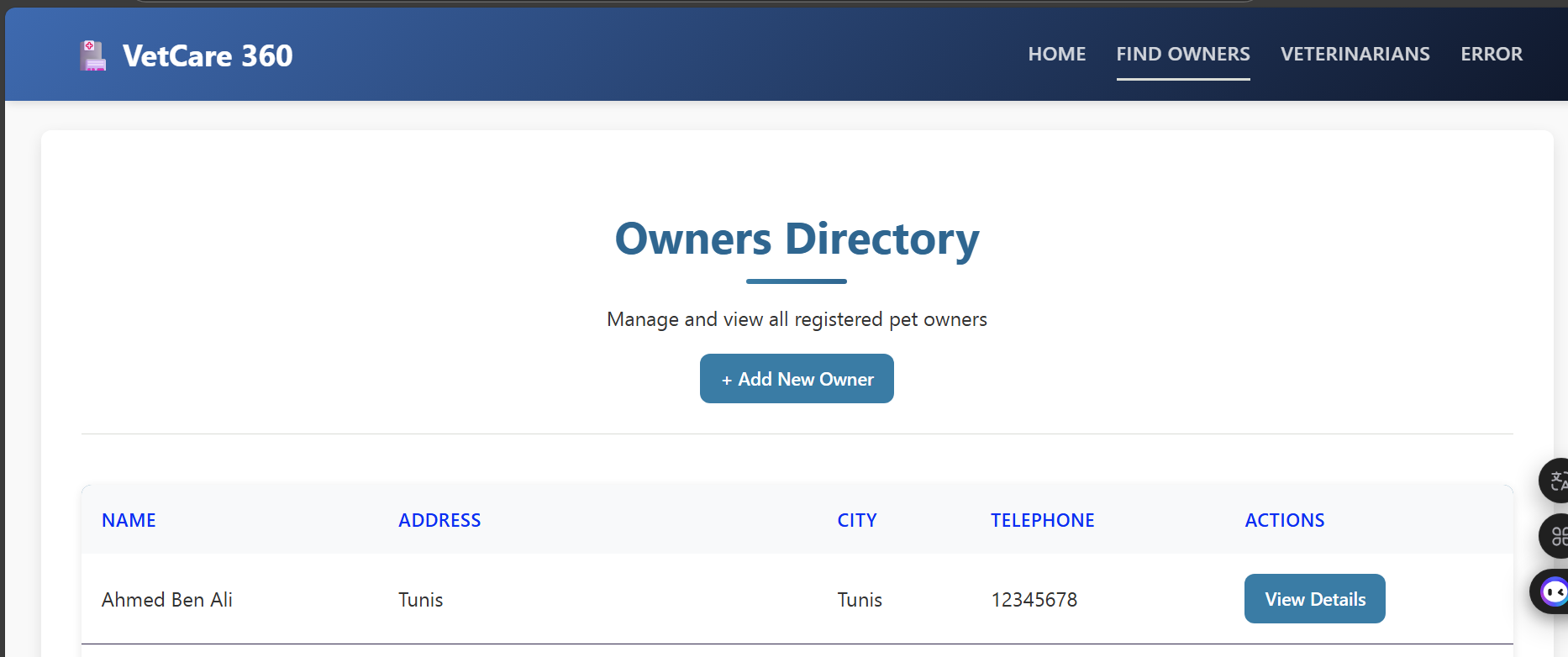
|  |  |
| --- | --- |
| **Page** | **Fonction** |
| 1 | Accueil |
| 2 | Liste des vétérinaires (lecture depuis MongoDB) |
| 3 | Recherche de propriétaire (par nom) |
| 4 | Formulaire : nouveau propriétaire |
| 5 | Résultat + options : modifier / ajouter animal |
| 6 | Modifier les infos du propriétaire |
| 7 | Ajouter un animal |
| 8 | Liste des propriétaires trouvés |
| 9 | Détail propriétaire + animaux + visites |
| 10 | Modifier animal |
| 11 | Ajouter une visite |
| 12 | Synthèse après visite |

### **2. Principes de design UX**

* Utilisation de **Bootstrap** pour un style réactif et clair.
* Navigation fluide entre les pages avec **React Router**.
* Champs de formulaire bien identifiés, avec validation simple.
* Boutons clairs : "Ajouter", "Modifier", "Retour", etc.

### **3. Exemple visuel simplifié**





## **Architecture du projet**

### **1. Organisation des dossiers**

#### **▶ Côté Frontend (React)**

frontend

├── node\_modules

├── public

├── src

│ ├── components → Composants réutilisables (Navbar, Form...)

│ ├── pages → Pages spécifiques (Accueil, Liste, Formulaires...)

│ │── services → Appels API (axios)

│ ├── App.css

│ ├── App.js → Routeur principal

│ ├── App.test.js

│ ├── index.css

│ ├── index.js

│ ├── logo.svg

│ ├── reportWebVitals.js

│ └── setupTests.js

│

├── env

├── .gitignore

├── package-lock.json

├── package.json

└── README.md

#### ***▶ Côté Backend (Node.js + Express)***

backend

├── config

├── controllers → Logique métier

├── models → Schémas Mongoose

├── node\_modules

├── routes → Fichiers de routes (proprietaireRoutes.js, animalRoutes.js...)

├── .env

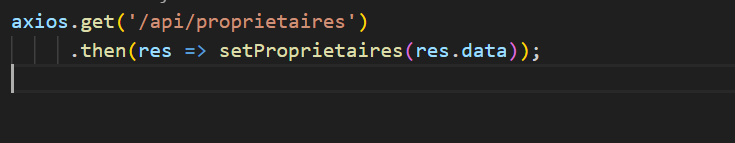
├── serveur.js → Point d’entrée principal

└── package-lock.json

### **2. Communication entre frontend et backend**

Via des appels **HTTP** utilisant **Axios** dans React.

Exemple :



### ***3. Outils complémentaires***

|  |  |
| --- | --- |
| **Outil** | **Rôle** |
| Git/GitHub | Suivi de version du code |
| Postman | Test des API manuellement |
| Mongoose | Modélisation des données |
| React Router | Navigation entre les pages |

## **Développement Frontend (React.js)**

### **1. Technologies utilisées**

* **React.js** : Bibliothèque principale.
* **React Router** : Pour la navigation entre les pages.
* **Axios** : Pour les appels API.
* **Bootstrap** : Pour le style responsive.
* **CSS personnalisé** : dans /styles et dans chaque composant.

## **2. Organisation des composants**

|  |  |
| --- | --- |
| **Dossier/Composant** | **Rôle** |
| /components/Navbar/Navbar.js | Barre de navigation commune |
| /components/OwnerForm/ | Formulaire pour créer ou modifier un propriétaire |
| /components/PetForm/ | Formulaire pour créer ou modifier un animal |

### **3. Structure des pages dans /pages**

|  |  |
| --- | --- |
| **Page** | **Description** |
| HomePage | Page d’accueil |
| Vets | Liste des vétérinaires |
| OwnerSearch | Recherche d’un propriétaire par nom |
| OwnerForm, EditOwner | Ajout et modification d’un propriétaire |
| OwnersList | Liste des résultats de recherche |
| OwnerDetails | Détail du propriétaire et de ses animaux et visites |
| AddPet, EditPet | Gestion des animaux de compagnie |
| AddVisit | Ajout d’une nouvelle visite médicale |

### **4. Appels API via Axios**

Tous les appels sont centralisés dans /services :

# 

# ***Développement Backend (Express.js + Node.js)***

### ***1. Structure du Backend***

|  |  |
| --- | --- |
| **Dossier/Fichier** | **Description** |
| /routes | Déclare les routes (CRUD) |
| /controllers | Contient la logique de traitement des requêtes |
| /models | Définition des schémas Mongoose |
| App.js | Point d’entrée de l’application Express |
| /config/db.js | Gère la connexion à MongoDB |
| .env | Variables d’environnement (URI Mongo, port, etc.) |

### ***2. Exemple de route et contrôleur***

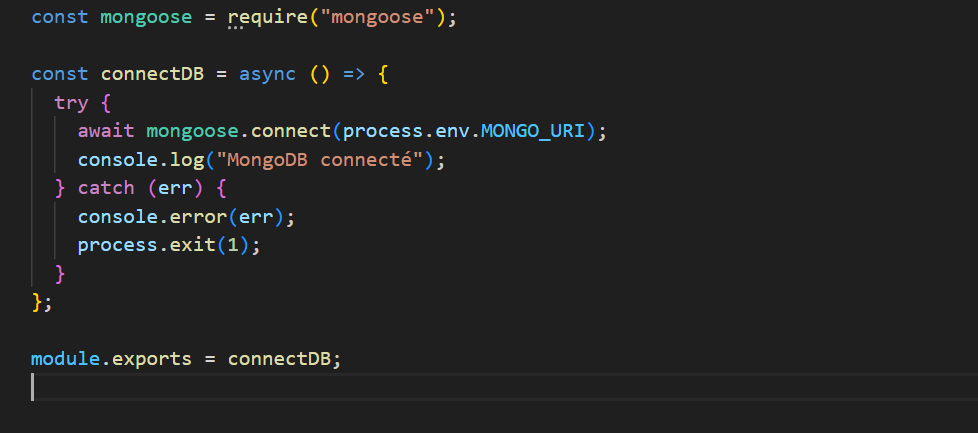


### ***3. Contrôleurs disponibles***

|  |  |
| --- | --- |
| **Contrôleur** | **Fonction** |
| OwnerController.js | CRUD propriétaires |
| VetController.js | Lecture de la liste des vétérinaires |
| VisitController.js | Ajout et affichage des visites |
| GetController.js | Données combinées / requêtes avancées |

## **Connexion à la base de données (MongoDB)**

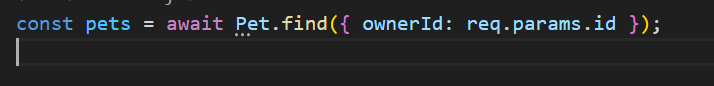
### **1. Connexion via /config/db.js**



### ***2. Schémas dans /models***

|  |  |
| --- | --- |
| **Fichier** | **Contenu** |
| Owner.js | Nom, prénom, adresse, téléphone |
| Pet.js | Nom, espèce, âge, référence vers un propriétaire (ownerId) |
| Visit.js | Date, diagnostic, traitement, référence vers un animal (petId) |
| Vet.js | Nom, spécialité |

### ***3. Exemple d’opérations MongoDB avec Mongoose***



* Les documents sont reliés entre eux via les **ObjectId** (ref).
* Relations modélisées sans jointures lourdes grâce au NoSQL.

## Gestion du projet avec Git et GitHub

### 1. Pourquoi utiliser Git/GitHub ?

|  |  |
| --- | --- |
| Objectif | Avantage |
| Historique de version | Suivi de toutes les modifications |
| Travail organisé | Par branches : main, dev, feature/... |
| Collaboration | Possibilité de partage avec d'autres développeurs |
| Sécurité et sauvegarde | Code sauvegardé sur un dépôt distant |

### 2. Structure du dépôt

VET\_CARE360/

├── backend/

├── frontend/

└── README.md

* Le fichier README.md contient des instructions d’installation pour dev.

### 3. Commandes essentielles

* git init
* git add .
* git commit -m "Initial commit"
* git branch -M main
* git remote add origin https://github.com/utilisateur/VET\_CARE360.git
* git push -u origin main

### 4. Bonnes pratiques adoptées

* Commits fréquents avec messages clairs.
* Noms de branches explicites : feature/ajout-pet, bugfix/api-error.
* Dépôt bien structuré pour faciliter la lecture.

## Tests et validation

### 1. Types de tests effectués

|  |  |
| --- | --- |
| Type de test | Objectif |
| Tests fonctionnels | Vérifier que chaque fonctionnalité attendue marche |
| Tests de navigation | Assurer la fluidité entre les pages React |
| Tests d’intégration | Vérifier les connexions entre frontend ↔ backend |
| Tests manuels | Utilisation réelle de l’application par l’étudiante |

### 2. Méthodologie de test

* Utilisation de Postman pour tester les routes Express/Mongo.
* Vérification des retours JSON (statuts HTTP, données correctes).
* Simulation de scénarios utilisateurs (ajout d’un propriétaire, d’un animal, d’une visite).

### 3. Exemples de cas testés

|  |  |  |
| --- | --- | --- |
| Cas de test | Résultat attendu | Statut |
| Ajouter un nouveau propriétaire | Le formulaire s’envoie avec succès | ✅ |
| Modifier les infos d’un animal | Les données sont mises à jour | ✅ |
| Supprimer une visite | La visite disparaît de la fiche | ✅ |
| Recherche propriétaire inexistant | Message : "Aucun résultat trouvé" | ✅ |

### 4. Retours utilisateur (UX)

Interface jugée claire, facile à comprendre.  
 Navigation intuitive grâce à une structure cohérente.

## Problèmes rencontrés et solutions

### 1. Difficultés techniques

|  |  |
| --- | --- |
| Problème rencontré | Solution mise en œuvre |
| Connexion instable à MongoDB | Utilisation correcte de .env et configuration mongoose |
| Affichage vide lors de la recherche | Correction dans l’URL axios et gestion du useEffect() |
| Problèmes de CORS entre frontend et backend | Ajout du middleware cors() côté backend |
| Répétition de code dans les formulaires | Création de composants réutilisables |

### 2. Défis personnels

* Compréhension du fonctionnement des hooks React.
* Organisation des composants et logique métier.
* Utilisation de Git et résolution des conflits.

### 3. Solutions et apprentissages

Grâce à la documentation, à l’expérimentation et à l’aide du superviseur, chaque difficulté a été surmontée progressivement

## Conclusion et perspectives

### 1. Bilan du projet

Le projet VetCare 360 a permis de développer une application fonctionnelle pour la gestion d’une clinique vétérinaire. Il a permis de :

* Mettre en pratique les notions de développement web (MERN).
* Travailler sur un vrai cas d’usage utile.
* Développer des compétences en programmation full-stack, modélisation, gestion de projet.

### 2. Objectifs atteints

✔ Création des principales fonctionnalités :  
 ✔ Interface utilisateur dynamique (React).  
 ✔ API REST opérationnelle.  
 ✔ Stockage des données avec MongoDB.  
 ✔ Structure de projet propre et versionnée (GitHub).

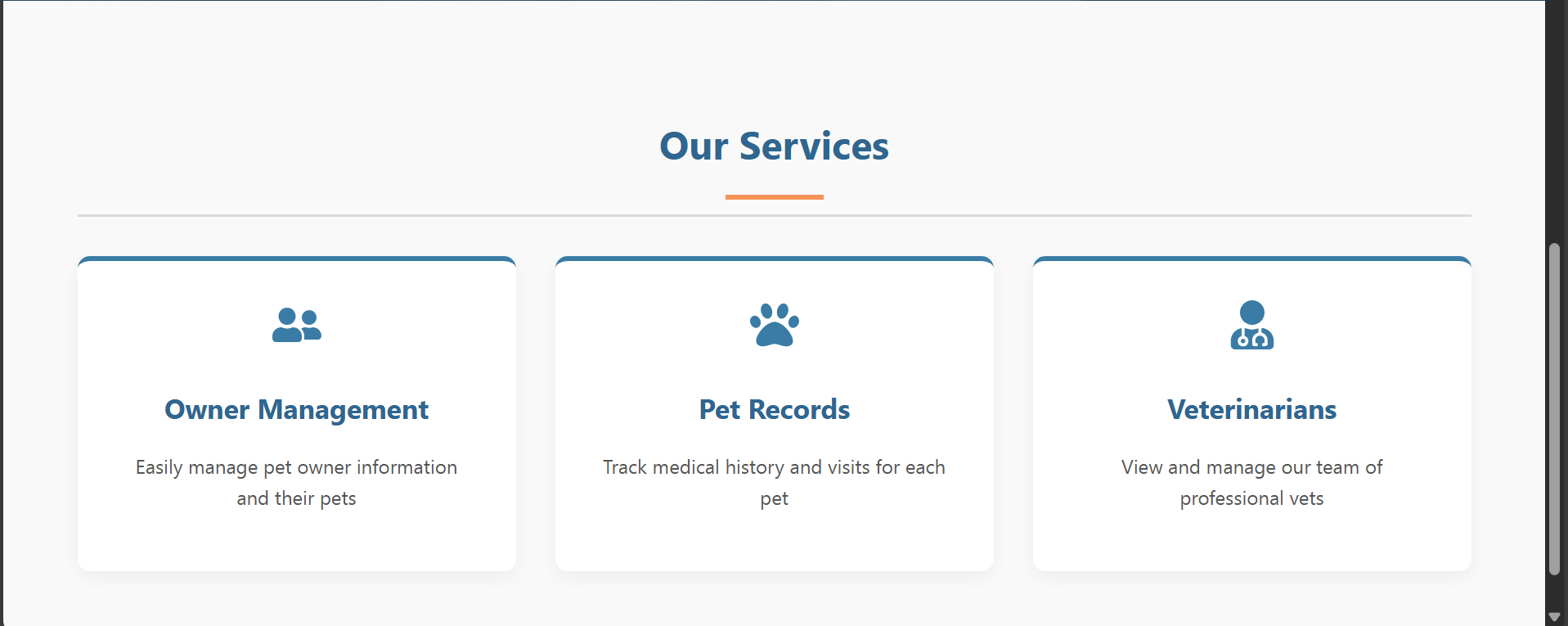
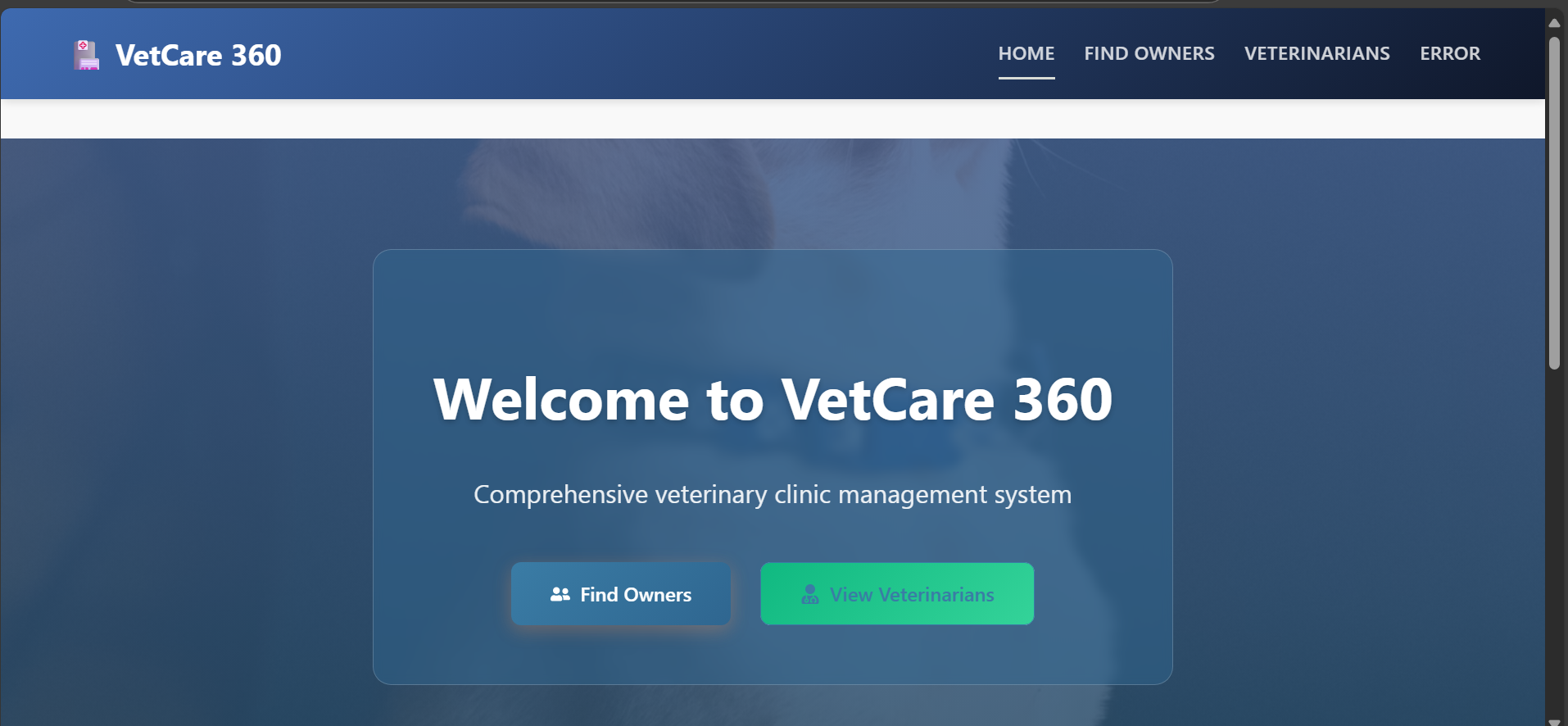
### 3. Perspectives d’amélioration

|  |  |
| --- | --- |
| Amélioration possible | Description |
| Authentification des utilisateurs | Ajouter un système de connexion (admin / vétérinaire) |
| Tableau de bord | Visualisation statistique (nombre de visites par mois) |
| Export PDF | Génération de rapports imprimables |
| Design avancé | Utilisation de Material UI ou Tailwind |

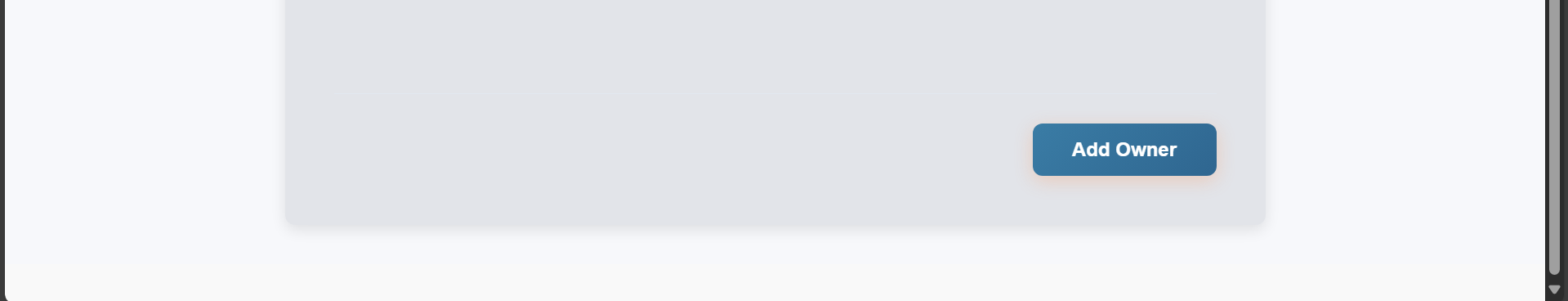
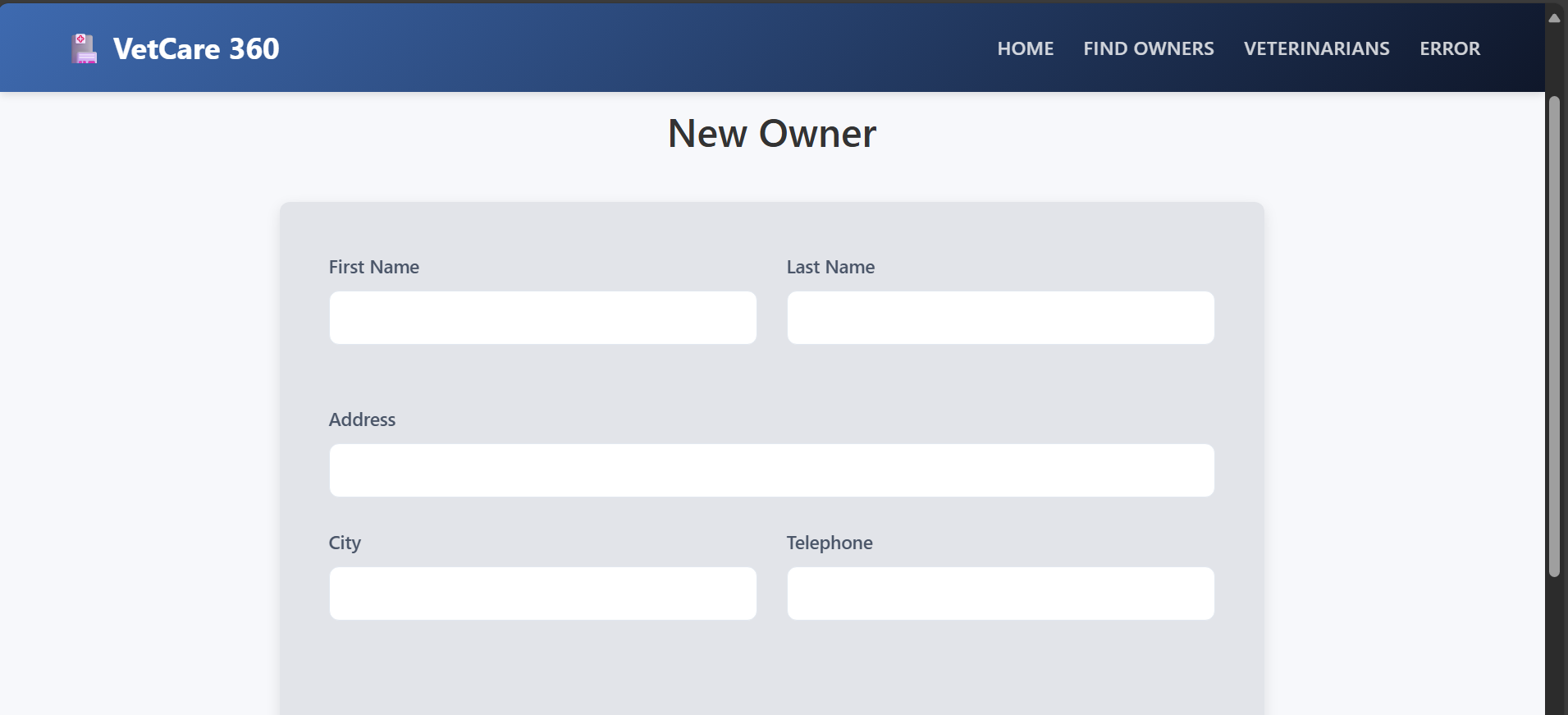
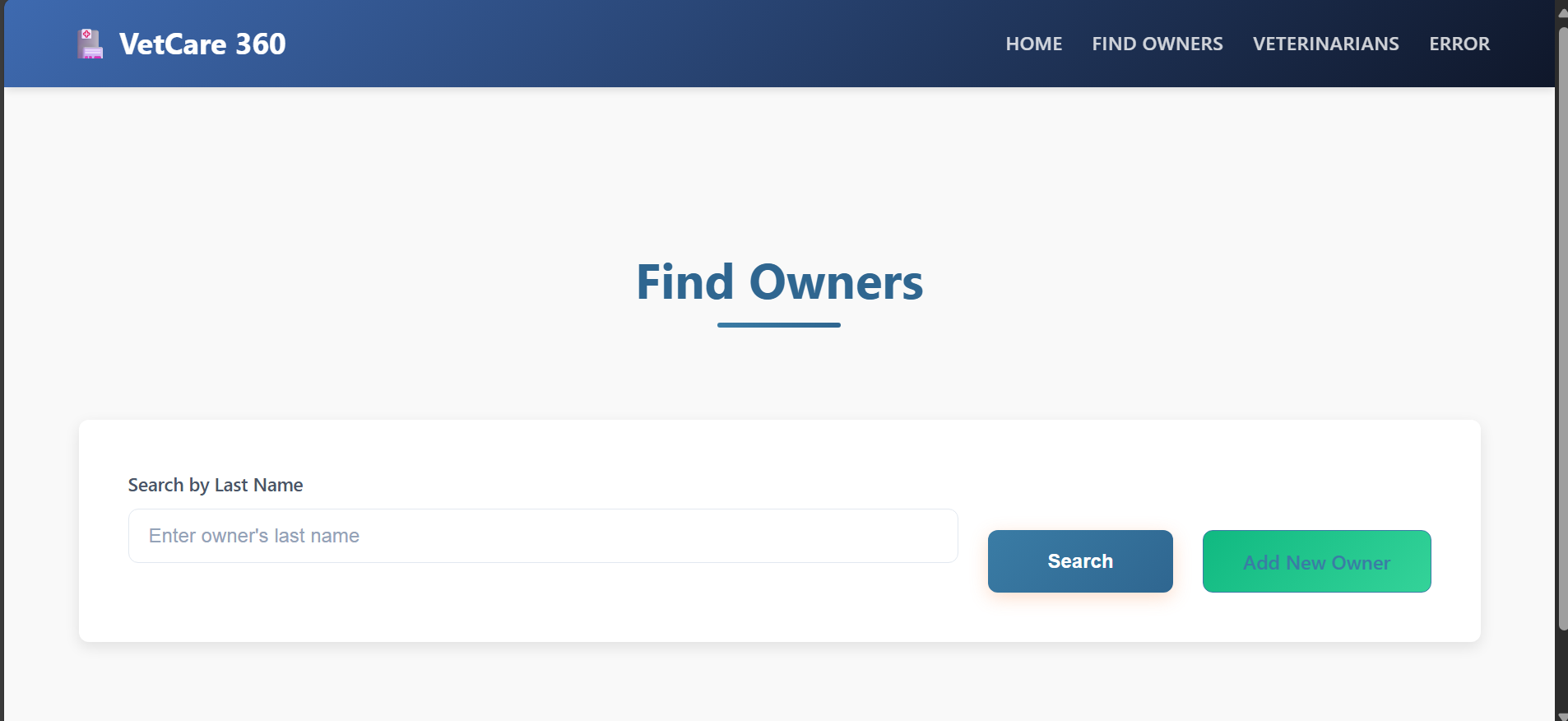
## **Annexes**

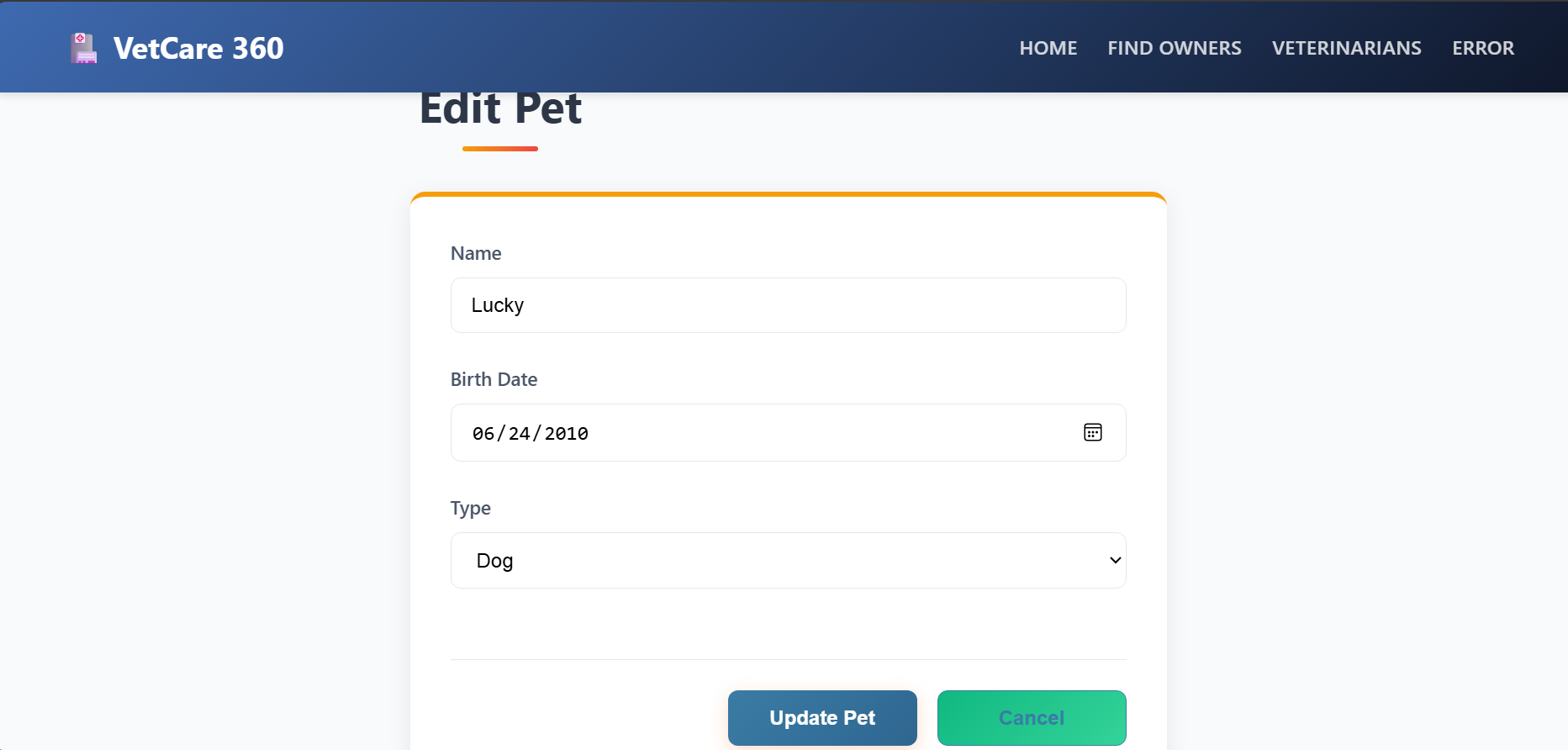
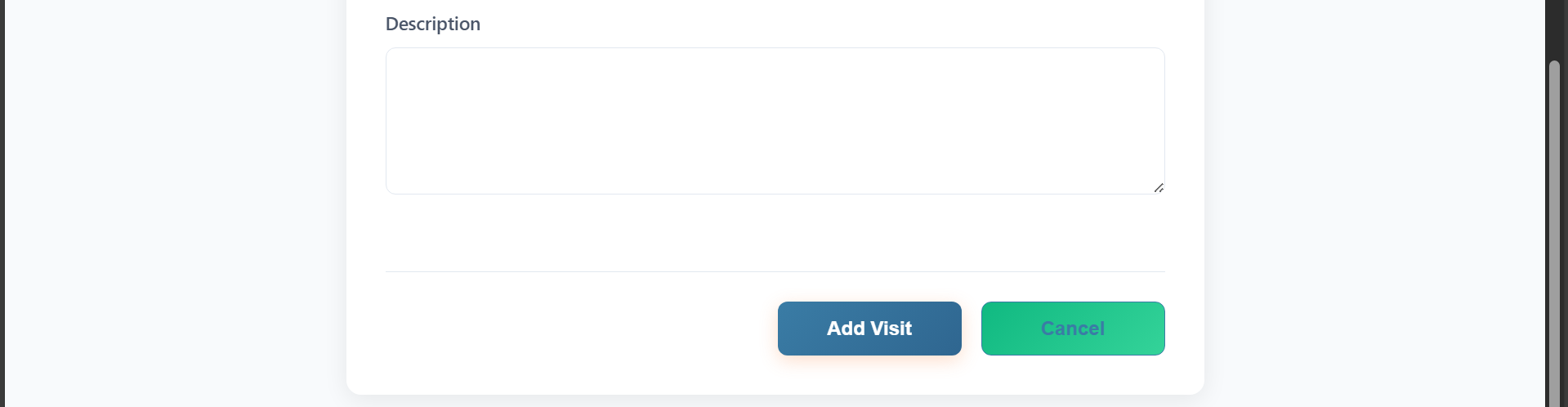
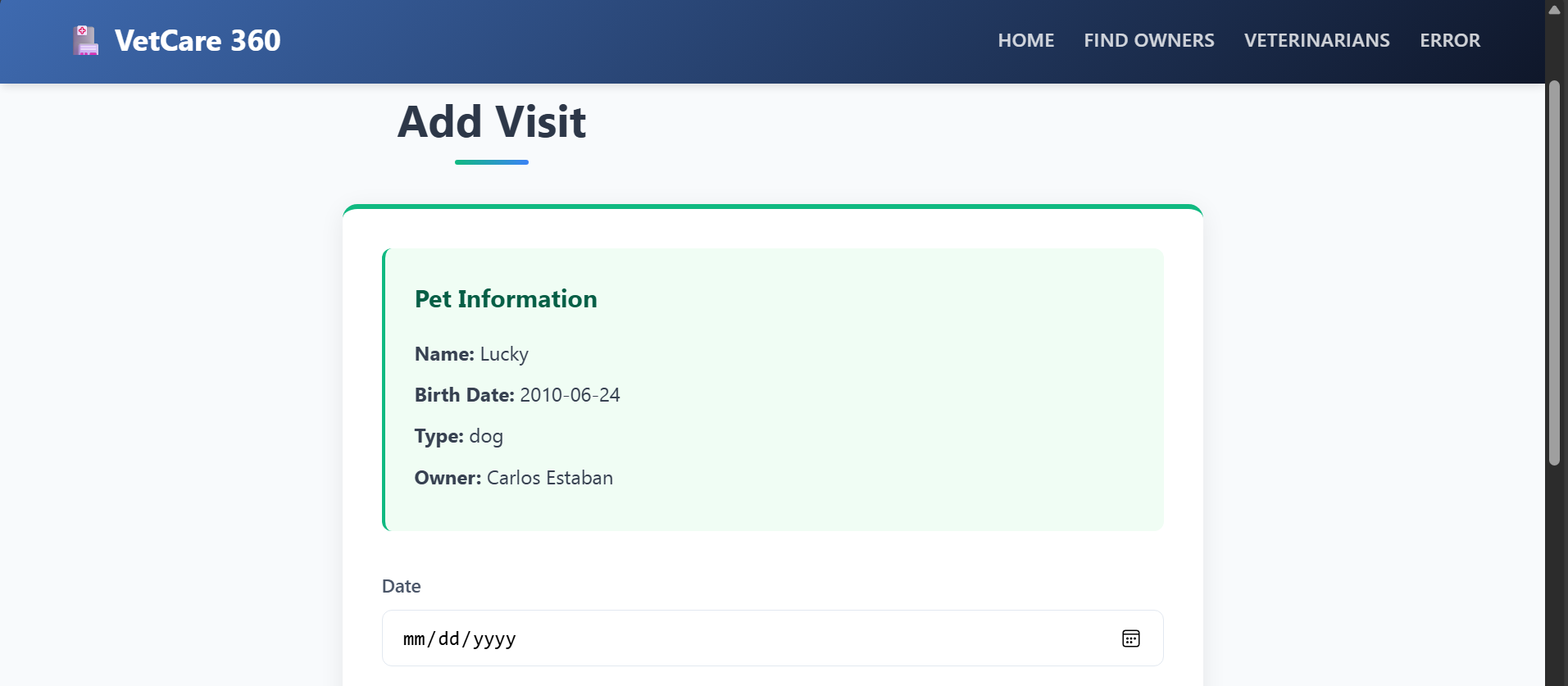
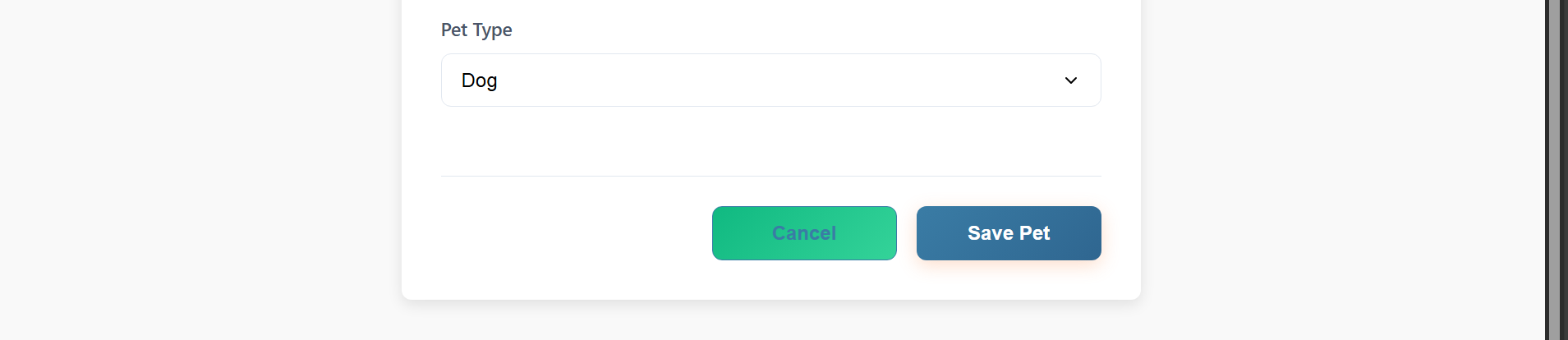
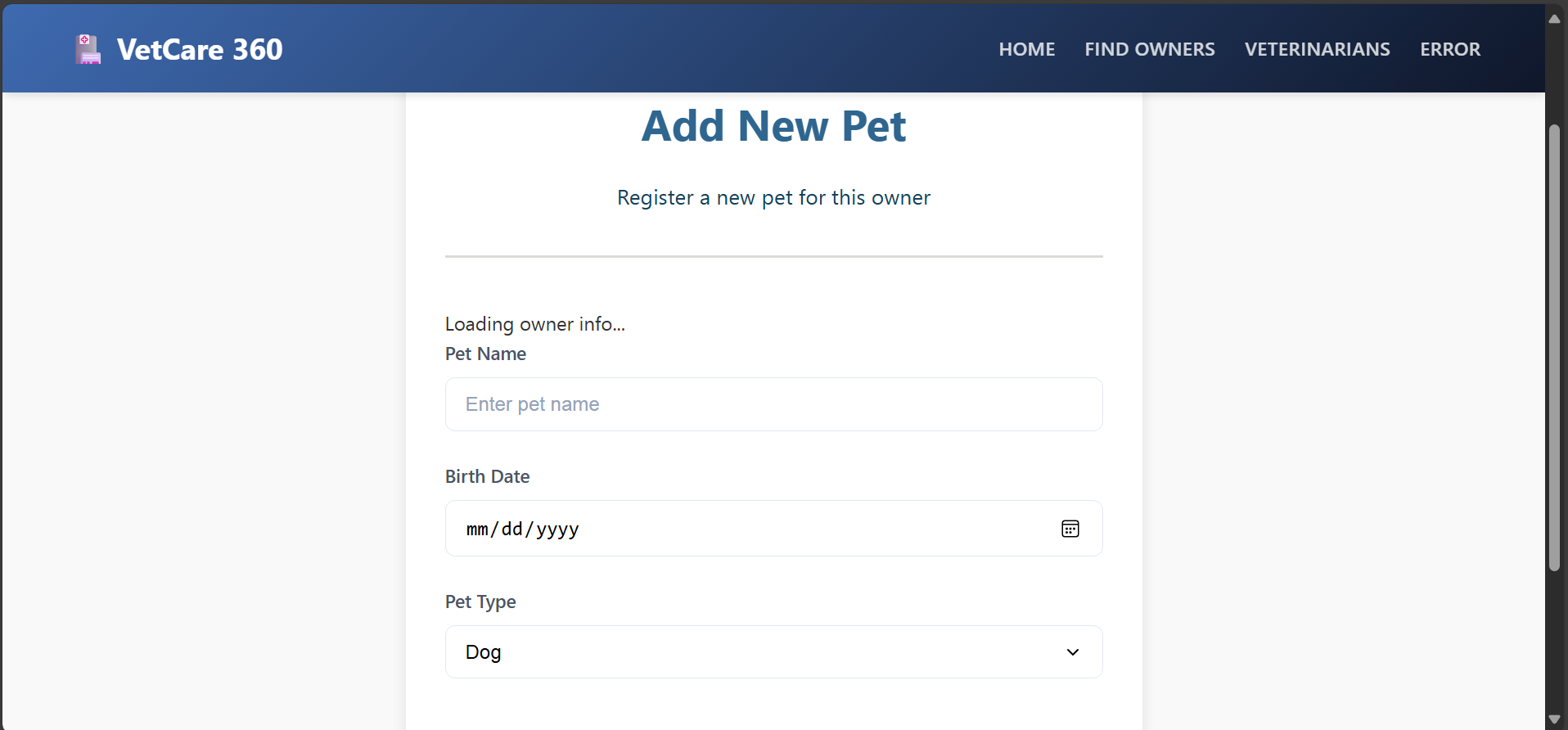
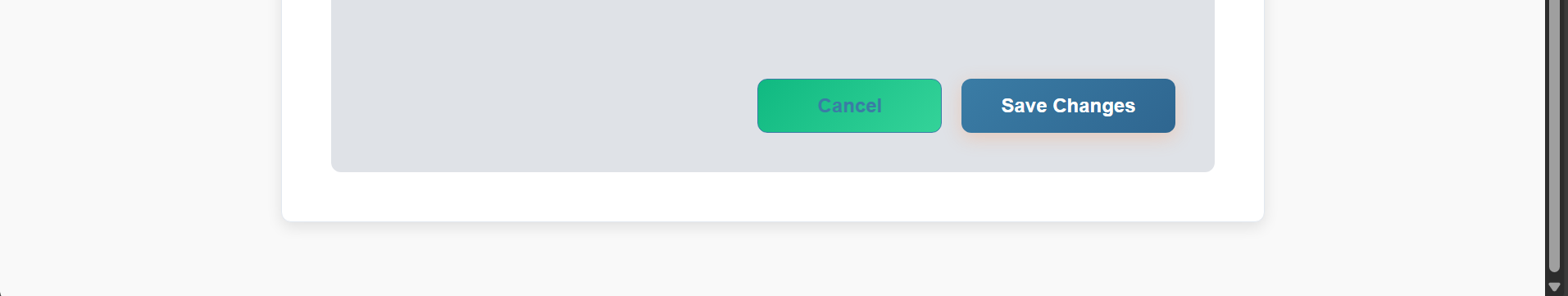
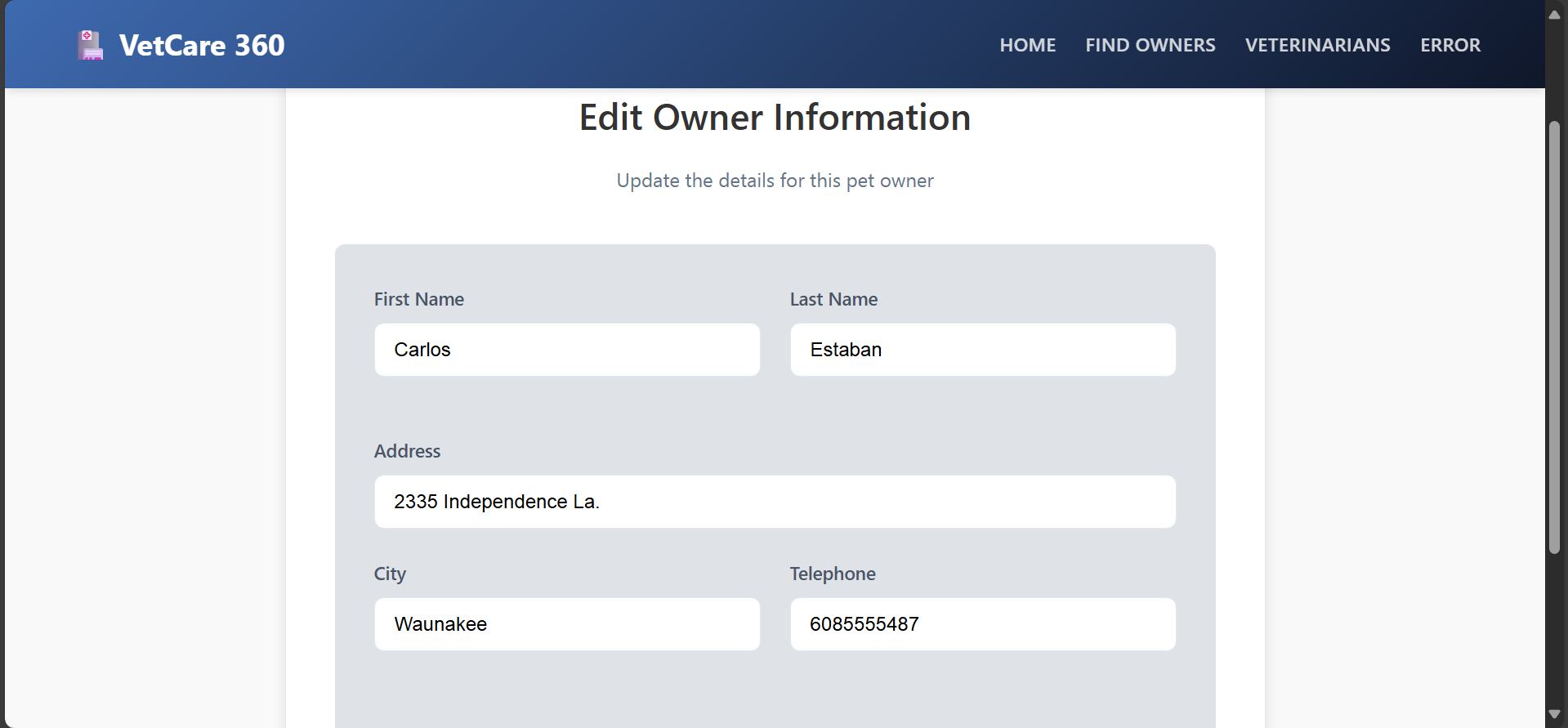
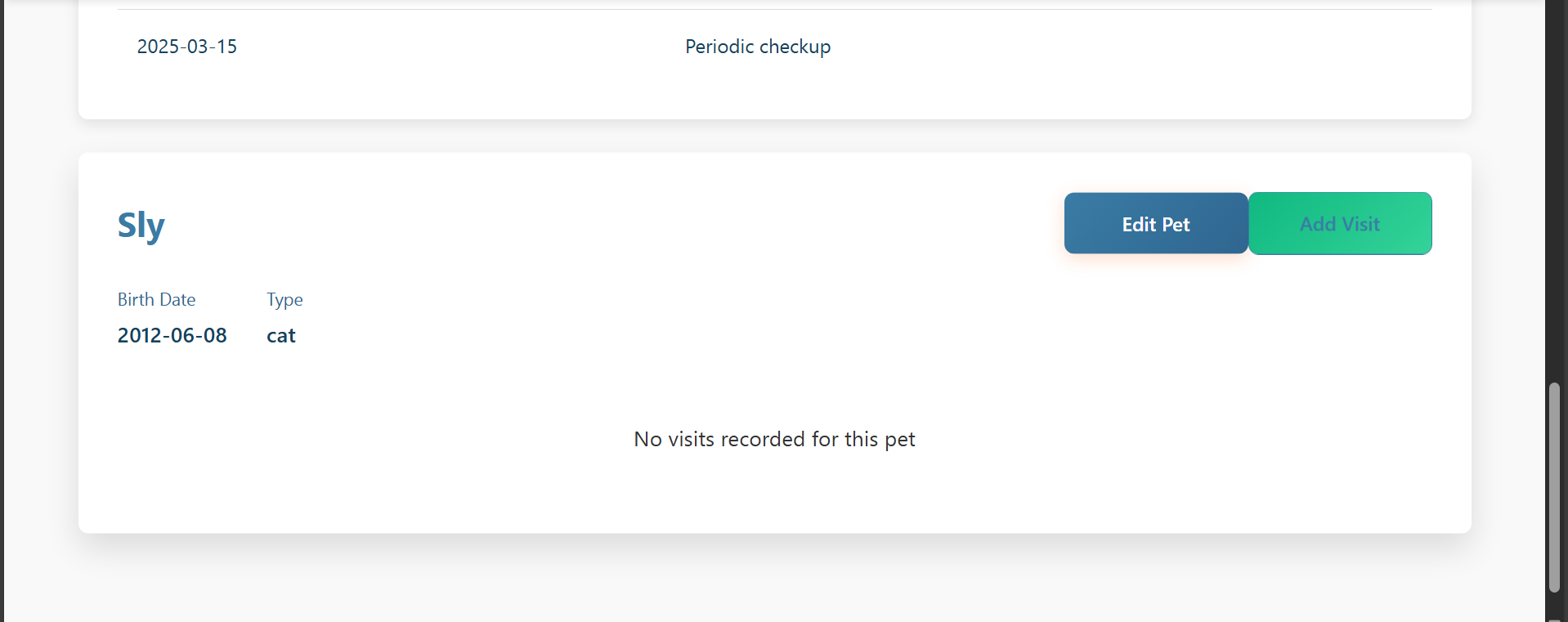
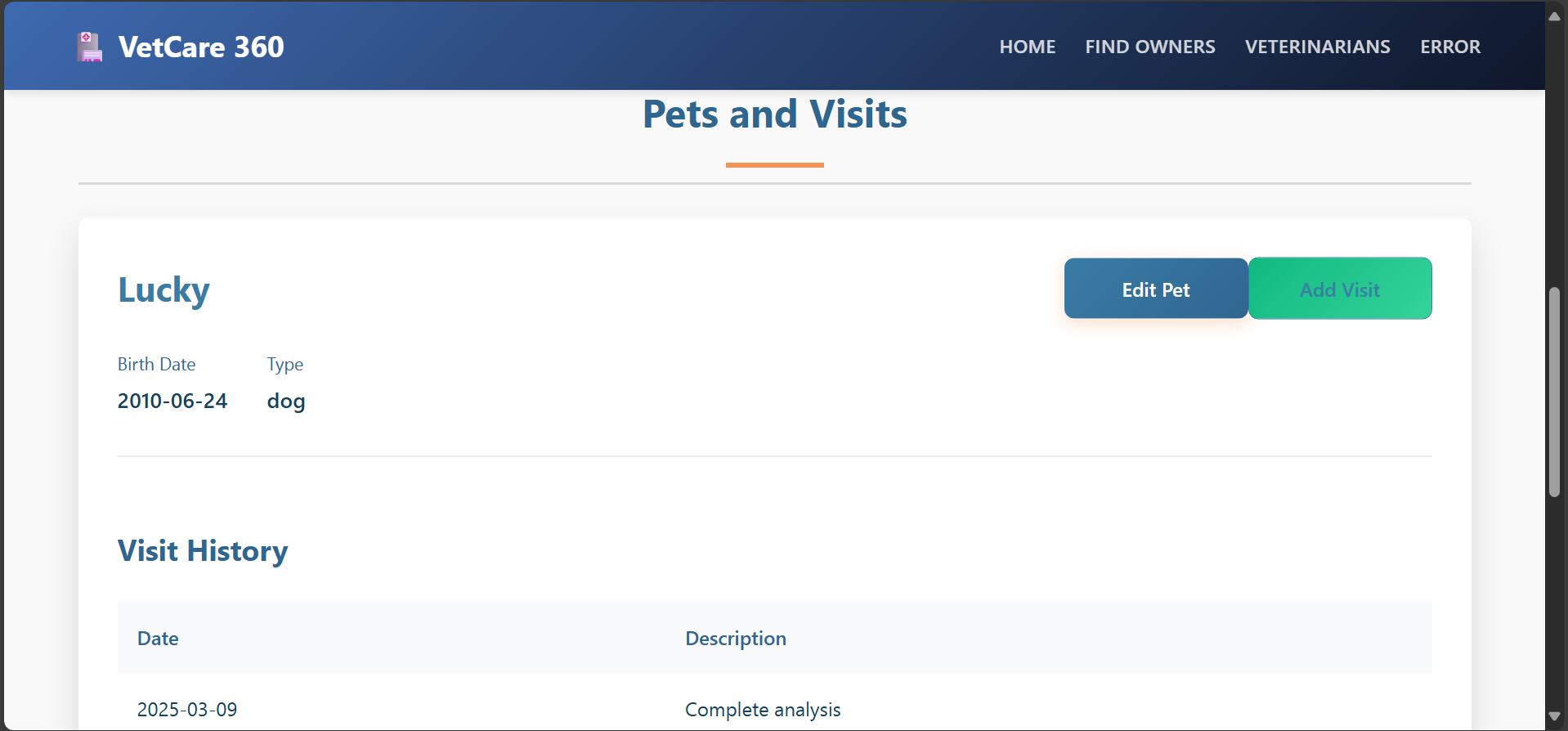
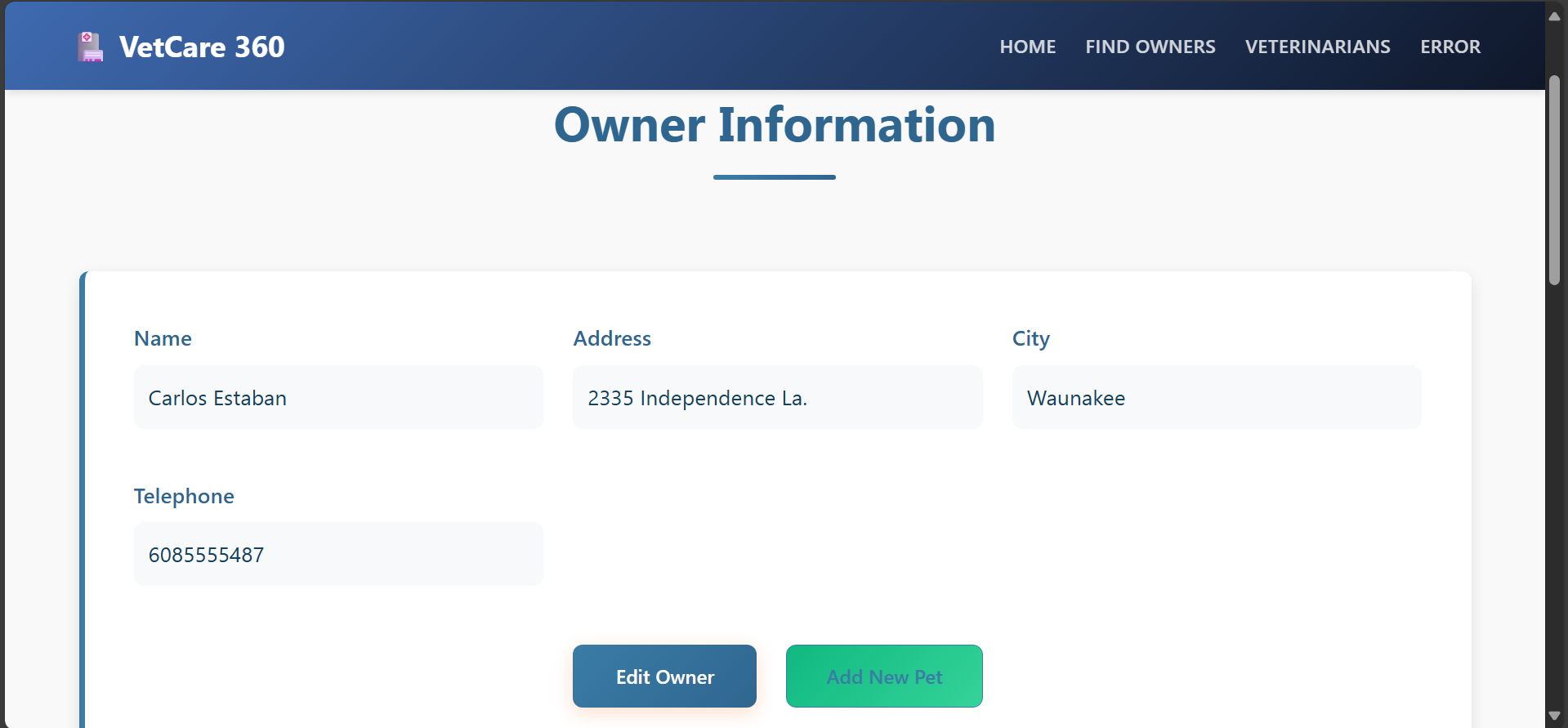
### **1. Captures d’écran**

* Interface d’accueil

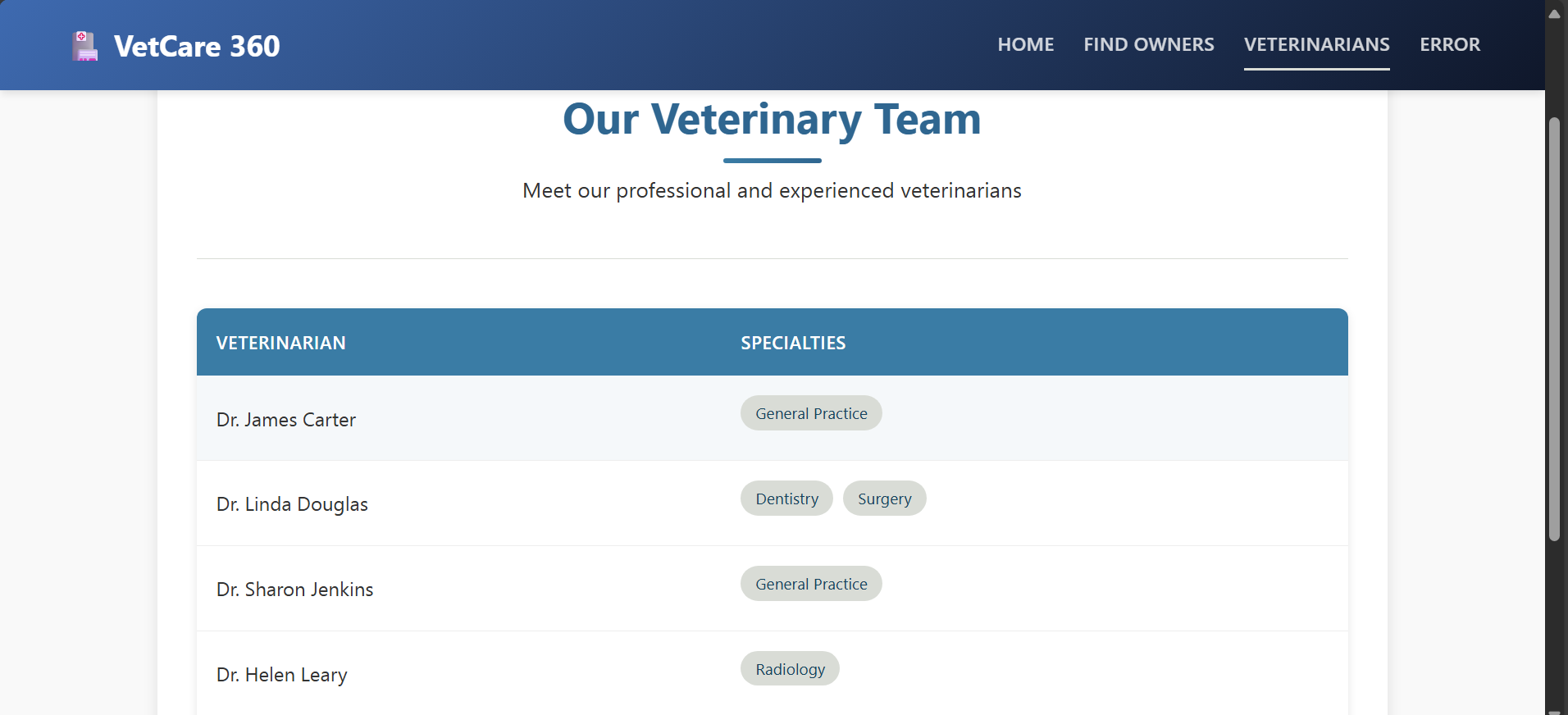


* Formulaire d’ajout de propriétaire
* Fiche de détail d’un propriétaire et ses animaux
* Historique des visites

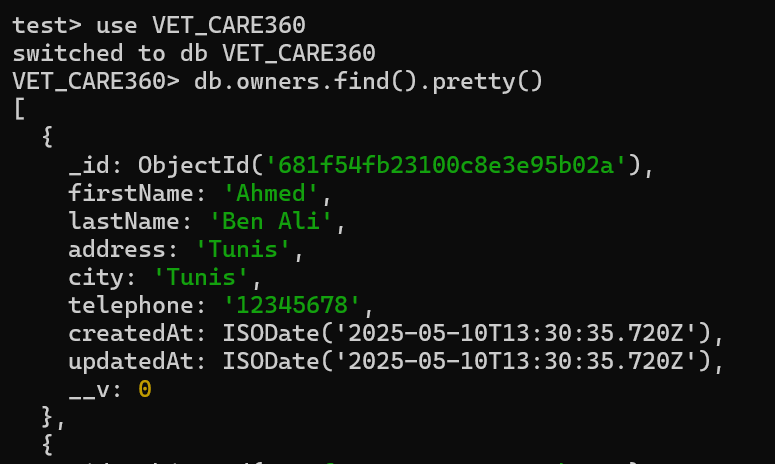




* Liste des vétérinaires



* Mongodb







### **2. Extraits de code**

|  |  |
| --- | --- |
| **Fonctionnalité** | **Fichier concerné** |
| Route POST visite | /routes/visitRoutes.js |
| Composant PetForm | /components/PetForm.js |
| Connexion MongoDB | /config/db.js |