المدرسة العليا للتكنولوجيا الناظور
École Supérieure de Technologie de Nador
ⵜⵉⵎⵙⵙⵓⵔⵉⵜ ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵜⵓⵜⵜⵔⵉⵙⵜⵉⵜ ⵅⵍⵓⵎ
جامعة محمد الأول بوجدة
UNIVERSITÉ MOHAMMED PREMIER OUJDA
ⵜⴰⵙⴷⴰⵡⵉⵜ ⵏ ⵎⵓⵃⴰⵎⵎⴷ ⴰⵎⴰⵣⵡⴰⵔ ⵏ ⵡⵊⴷⴰ

Projet de Fin de semestre

Diplôme Universitaire de Technologie

Filière : Ingénierie Logicielle et Cybersecurity

(ILCS)

Intitulé du projet :

Une application web pour la gestion d'une clinique vétérinaire

Vetecare 360 « Mongodb »

Réalisée par:

Alae Majjati

&

Oumaima Marzak

Professeur :

Mr. Esbai Redouane

Année universitaire : 2024/2025

Table des matières

Introduction

VetCare 360 est une application web de gestion pour clinique vétérinaire basée sur l'architecture MERN (MongoDB, Express, React, Node.js). Ce système permettra de gérer efficacement les dossiers des propriétaires d'animaux, les informations sur les animaux de compagnie, les visites médicales et la liste des vétérinaires.

Ce compte rendu présente la méthodologie détaillée pour développer cette application en respectant les fonctionnalités demandées et en suivant les meilleures pratiques de développement web.

Vision

Notre vision est de devenir la solution de référence pour la gestion vétérinaire, en offrant une application évolutive capable de s’adapter aux besoins spécifiques de chaque clinique, tout en intégrant des fonctionnalités avancées comme l’analyse de données ou l’intelligence artificielle pour les diagnostics.

Contexte du Projet

Le secteur vétérinaire connaît une transformation digitale importante. Les cabinets vétérinaires doivent s'adapter aux nouvelles technologies pour améliorer leur

efficacité et la qualité de service. Ce projet s'inscrit dans cette dynamique de modernisation.

Enjeux et Défis

- Digitalisation des processus manuels

- Sécurisation des données sensibles

- Amélioration de la traçabilité des soins

- Optimisation du temps de travail

- Réduction des erreurs médicales

Objectifs du projet

Le projet **VetCareMongo** a été conçu avec plusieurs objectifs clés :

1. **Centralisation des données** : Fournir une base de données unifiée pour stocker les informations des animaux, des propriétaires, des rendez-vous, et des dossiers médicaux.
2. **Automatisation des processus** : Réduire les tâches manuelles grâce à des fonctionnalités comme la planification automatique des rendez-vous et les rappels par e-mail.
3. **Expérience utilisateur optimale** : Offrir une interface intuitive et responsive, accessible sur différents appareils.
4. **Sécurité des données** : Protéger les informations sensibles (données médicales, informations personnelles) avec des mécanismes d’authentification et de cryptage.

5. **Évolutivité** : Concevoir une architecture flexible permettant l'ajout de nouvelles fonctionnalités, comme l'intégration de rappels SMS ou de tableaux de bord analytiques.
6. **Accessibilité financière** : Proposer une solution open-source abordable pour les petites cliniques vétérinaires.

Objectifs Spécifiques

- Réduire le temps de gestion administrative de 40%

- Améliorer la satisfaction client de 30%

- Réduire les erreurs de prescription de 25%

- Augmenter la productivité du personnel de 35%

Analyse des besoins

Fonctionnalités principales

1. Gestion des propriétaires d'animaux
    - Ajouter de nouveaux propriétaires
    - Modifier les informations des propriétaires existants
    - Supprimer des propriétaires
    - Rechercher des propriétaires par nom de famille
2. Gestion des animaux de compagnie
    - Ajouter des animaux et les associer à leurs propriétaires
    - Modifier les informations des animaux

    - Consulter les détails des animaux
3. Gestion des visites médicales
    - Enregistrer les nouvelles visites
    - Consulter l'historique des visites par animal
    - Afficher les détails des visites
4. Liste des vétérinaires
    - Afficher la liste des vétérinaires disponibles

Analyse comparative des bases relationnelles vs MongoDB

Le choix entre SQL (relationnel) et MongoDB repose sur des différences fondamentales dans la manière dont les données sont structurées et manipulées. SQL, avec sa structure rigide, repose sur une normalisation forte des données, permettant ainsi une gestion précise des relations entre les entités à l'aide de requêtes complexes incluant des jointures.

Cette approche est idéale pour les systèmes nécessitant une intégrité de données stricte et des relations entre les tables bien définies. En revanche, MongoDB offre une flexibilité accrue en permettant de gérer des structures de données moins rigides, particulièrement adaptées aux modèles hiérarchiques. Par exemple, dans un cas comme celui de la gestion des visites d'animaux, MongoDB permet d'imbriquer directement les informations relatives aux animaux et à leurs visites sans avoir besoin d'un schéma strict, ce qui simplifie et accélère le processus.

Ce modèle de données flexible est particulièrement pertinent pour des systèmes comme Vetcare 360, où les entités, telles que les animaux et leurs visites, sont

étroitement liées et bénéficient d'une structure hiérarchique sans les contraintes d'un schéma relationnel rigide.

vantages de MongoDB pour ce Projet

- Flexibilité du schéma pour les dossiers médicaux

- Performance pour les requêtes complexes

- Facilité de mise à l'échelle

- Support natif pour les données JSON

- Intégration facile avec Node.js

Conception de la base de données

Structure des collections MongoDB

Collection Proprietaires

json

```
{
  "_id": "ObjectId(‘’)",
  " firstName ": "String",
 " lastName ": "String",
 " email ": "String",
 " phone ": "String",
 " address ": "String",
 " createdAt ": "Date",
```

" updatedAt ": "Date",

" __v ": 0

}

Collection Animaux

json

{

"_id": "ObjectId()",

" name ": "String",

" species ": "String",

" breed ": "String",

" birthDate ": "Date",

" gender ": "String",

" owner ": "ObjectId ()",

" medicalHistory ": "String",

"createdAt": "Date",

"updatedAt": "Date",

" __v ": 0

}

Collection Visites

json

{

"_id": "ObjectId()",

" pet ": "ObjectId (ref: Animaux)",

" veterinarian ": "ObjectId (ref: Veterinaires)",

"date": "Date",

```
" reason ": "String",
" diagnosis ": "String",
" treatment ": "String",
 "notes": "String",
 "createdAt": "Date",
 "updatedAt": "Date",
 " __v " : 0
}
```


Collection Veterinaires

json


```
{
  "_id": "ObjectId()",
 " firstName ": "String",
 " lastName ": "String",
 " specialization ": "String",
 " email ": "String",
 " phone ": "String",
 " licenseNumber ": "String",
  "createdAt": "Date",
  "updatedAt": "Date",
  " __v " : 0
}
```


Architecture technique

`Stack MERN`	• **MongoDB(MongoDB Atlas)** : Base de données NoSQL orientée documents • **Express.js** : Framework backend pour Node.js • **React** : Bibliothèque JavaScript pour l'interface utilisateur • **Node.js** : Environnement d'exécution JavaScript côté serveur
`Outils supplémentaires`	• **Bootstrap** : Framework CSS pour un design responsive • **Mongoose** : ODM (Object Data Modeling) pour MongoDB • **Git/GitHub** : Gestion de versions et collaboration

Choix technologiques et alternatives

Les technologies choisies pour **VetCareMongo** ont été sélectionnées pour leur performance et leur popularité. Voici une analyse des choix et des alternatives possibles :

MongoDB vs. PostgreSQL

- **MongoDB** :
  - Avantages : Flexibilité pour les données non structurées, scalabilité horizontale.
  - Inconvénients : Moins adapté aux relations complexes.
- **PostgreSQL** (alternative) :
  - Avantages : Support des relations, SQL standard.
  - Inconvénients : Moins flexible pour les données dynamiques.

Node.js/Express vs. Django

- **Node.js/Express** :
  - Avantages : Rapide, non bloquant, écosystème JavaScript.
  - Inconvénients : Moins de conventions que Django.
- **Django** (alternative) :
  - Avantages : ORM puissant, sécurité intégrée.
  - Inconvénients : Plus lourd, nécessite Python.

React vs. Vue.js

- **React** :
  - Avantages : Large communauté, écosystème riche.
  - Inconvénients : Courbe d'apprentissage pour les débutants.
- **Vue.js** (alternative) :
  - Avantages : Simplicité, documentation claire.
  - Inconvénients : Moins de popularité que React.
  -

Justification des Choix

- React pour sa performance et son écosystème

- Node.js pour son asynchronicité et sa compatibilité avec MongoDB

- MongoDB pour sa flexibilité et ses performances

- Docker pour la portabilité et la scalabilité

Scénario utilisateur : enregistrement d'une visite vétérinaire

Dans le cadre d'un fonctionnement interne à la clinique, un membre du personnel – par exemple un réceptionniste ou un vétérinaire – utilise l'interface de Vetcare Mongo pour **ajouter une nouvelle visite médicale**. Depuis la page d'accueil, il commence par enregistrer les informations du **propriétaire de l'animal** s'il n'existe pas déjà dans la base de données : nom, adresse, numéro de téléphone, etc. Ensuite, il crée une fiche pour l'animal associé, en renseignant des détails comme le nom, l'espèce, la race, l'âge et les éventuels antécédents. Une fois le propriétaire et l'animal enregistrés, l'utilisateur accède à la section « Visites », où il peut créer une nouvelle entrée de visite : il précise la **date**, les **symptômes**, les **observations du vétérinaire**, ainsi que les **médicaments prescrits**. Le vétérinaire responsable est sélectionné dans une liste, s'il a déjà été enregistré dans la section correspondante. Ces opérations créent automatiquement des documents dans MongoDB via des appels API, reliant entre eux propriétaires, animaux, visites et vétérinaires. Ce scénario montre comment l'application facilite la saisie centralisée des informations, même en l'absence d'un système d'authentification ou de gestion des rôles.

Installation et configuration

Prérequis

- Node.js (version 16.x ou supérieure)
- MongoDB (version 5.x ou supérieure)
- npm (installé avec Node.js)
- Git

Configuration initiale

Création de la structure du projet

```
mkdir vetcare360

cd vetcare360

mkdir backend frontend
```

Configuration du backend

```
cd backend
npm install

// Crier un fichier .env dans le repertoire backend avec se
contenue :
PORT=5000
MONGODB_URI=mongodb+srv://(nom d'utilisateur):(mot de passe)
@mernapp.3pdrd.mongodb.net/vetcare?retryWrites=true&w=majority&appNa
me=MERNapp
```

Configuration du frontend

```
cd ../frontend
npm install
```

Développement du backend et du frontend

Structure des dossiers

```
vetcare360/
├── backend/
│   ├── src/
│   │   ├── config/
│   │   ├── controllers/
│   │   ├── models/
│   │   └── routes/
│   ├── .env
│   ├── package.json
│   └── server.js
│
├── frontend/
│   ├── public/
│   ├── src/
│   │   ├── components/
│   │   ├── pages/
│   │   ├── services/
│   └── package.json
│
├── start.js
└── README.md
```

∅ Détails de chaque dossier backend :

a) config/ (Configuration)

· Contient les paramètres de configuration

· Gestion de la connexion MongoDB

· Configuration des variables d'environnement

· Middleware de sécurité

b) controllers/ (Contrôleurs)

· Gestion de la logique métier

· Fonctions pour :

- Gestion des propriétaires d'animaux
- Gestion des animaux

- Gestion des visites
- Gestion des vétérinaires

c) models/ (Modèles)

Schémas MongoDB

d) routes/ (Routes API)

Points d'entrée API :

```
/api/owners # Gestion des propriétaires
/api/pets # Gestion des animaux
/api/visits # Gestion des visites
/api/vets # Gestion des vétérinaires
```

∅ Détails de chaque dossier frontend :

a) components/ (Composants)

Composants réutilisables :

· Header.js : Barre de navigation

· Footer.js : Pied de page

· Forms/ :

- OwnerForm.js
- PetForm.js
- VisitForm.js

· Tables/ :

- OwnersTable.js

- PetsTable.js
- VisitsTable.js

· Cards/ :

- PetCard.js
- OwnerCard.js
- VetCard.js

b) pages/ (Pages)

Pages principales :

· Dashboard.js : Vue d'ensemble

· Owners/ :

- OwnersList.js
- OwnerDetails.js
- AddOwner.js

· Pets/ :

- PetsList.js
- PetDetails.js
- AddPet.js

· Visits/ :

- VisitsList.js
- ScheduleVisit.js
- VisitDetails.js

c) services/ (Services)

Services API :

```
- ownerService.js    // CRUD propriétaires
- petService.js      // CRUD animaux
- visitService.js    // CRUD visites
- vetService.js      // CRUD vétérinaires
```

Flux de Données

1. L'utilisateur interagit avec l'interface (frontend)
2. Le frontend envoie une requête API
3. Le backend traite la requête via les routes
4. Les contrôleurs gèrent la logique métier
5. Les modèles interagissent avec MongoDB
6. La réponse remonte jusqu'au frontend
7. L'interface se met à jour

Détails du Flux

1. Requête client → Frontend

- Validation des entrées
- Gestion des états
- Optimisation des requêtes

2. Frontend → API Backend

- Authentification
- Sérialisation des données

- Gestion des erreurs

3. Backend → Base de données

- Validation des données
- Optimisation des requêtes
- Gestion des transactions

4. Base de données → Backend

- Traitement des résultats
- Mise en cache
- Formatage des données

5. Backend → Frontend

- Sérialisation des réponses
- Gestion des erreurs
- Mise à jour du cache

6. Frontend → Client

- Mise à jour de l'interface
- Gestion des états
- Feedback utilisateur

Démarrage de l'application

Nous avons deux options pour démarrer cette application :

Option1 : -Démarrer le Backend et le Frontend séparément :

```
cd backend
npm start
// Le backend va se démarrer sur http://localhost:5000

cd frontend
npm start
// Le frontend va se démarrer sur http://localhost:3000
```

Option2 : -Utiliser la combinaison de Start à travers le fichier start.js :

```
node start.js
```

Perspectives d'amélioration

Bien que Vetcare Mongo offre déjà une base fonctionnelle pour la gestion des animaux, des propriétaires, des visites et des vétérinaires, plusieurs améliorations peuvent être envisagées pour enrichir l'application et la rendre plus complète et plus professionnelle. L'ajout d'un **système d'authentification sécurisé** avec gestion des rôles (administrateur, vétérinaire, réceptionniste) permettrait de mieux contrôler l'accès aux données sensibles. Il serait également pertinent d'intégrer un **système de planification de rendez-vous**, avec gestion des disponibilités des vétérinaires et affichage d'un calendrier interactif. Une **interface plus évoluée** pourrait offrir un tableau de bord avec des statistiques (nombre de visites par mois, suivi des vaccins,

etc.). Par ailleurs, des fonctionnalités comme l'**envoi automatique d'e-mails ou de SMS** pour rappeler les rendez-vous ou les traitements pourraient considérablement améliorer l'expérience utilisateur. Enfin, le **déploiement de l'application sur le cloud** (via des services comme Render, Vercel ou Heroku) rendrait la solution accessible à distance, favorisant son adoption dans des environnements professionnels réels.

Résolution des problèmes courants

- **Erreur de connexion MongoDB** :
  - Vérifiez que MongoDB est en cours d'exécution.
  - Assurez-vous que MONGODB_URI est correct dans .env.
  - Mettre à jour l'adresse IP dans Mongodb Atlas
- **Erreur 500 sur l'API** :
  - Consultez les logs dans la console ou dans logs/error.log.
  - Vérifiez les dépendances avec npm install.
- **Problèmes de CORS** :
  - Ajoutez votre domaine à la liste des origines autorisées dans backend/config/cors.js.

Perspectives d'Évolution

- Intégration d'IA pour le diagnostic

- Application mobile

- Interface propriétaire

- Analytics avancés

- Intégration avec d'autres systèmes

Conclusion :

VetCare 360 représente une solution complète et innovante pour la gestion des cliniques vétérinaires, développée avec la stack MERN (MongoDB, Express, React, Node.js). Cette application combine une interface utilisateur moderne et intuitive avec une architecture backend robuste, offrant ainsi une expérience utilisateur optimale pour les vétérinaires et leur personnel. Grâce à ses fonctionnalités complètes de gestion des propriétaires d'animaux, de suivi médical, et de planification des rendez-vous, l'application répond efficacement aux besoins quotidiens d'une clinique vétérinaire. L'architecture modulaire et évolutive du projet permet non seulement une maintenance facile mais aussi une expansion future des fonctionnalités. La sécurité des données, la validation des entrées, et la gestion des erreurs ont été particulièrement soignées pour garantir la fiabilité du système.

Si vous voulez découvrir notre application voici le liens vers le compte GitHub pour une meilleure expérience :

https://github.com/Alae06/vetcareMongo

Merci.