**Rapport Détaillé du Projet Clinique Vétérinaire VetCare360**

**Filière : Ingénierie Logicielle et Cybersécurité (ILCS)**

**Sous la supervision du professeur :** Redouane ESBAI

**Nom : EL OUAHBI et AGHADDAD**

**prénom : FATIMA et SIHAM**

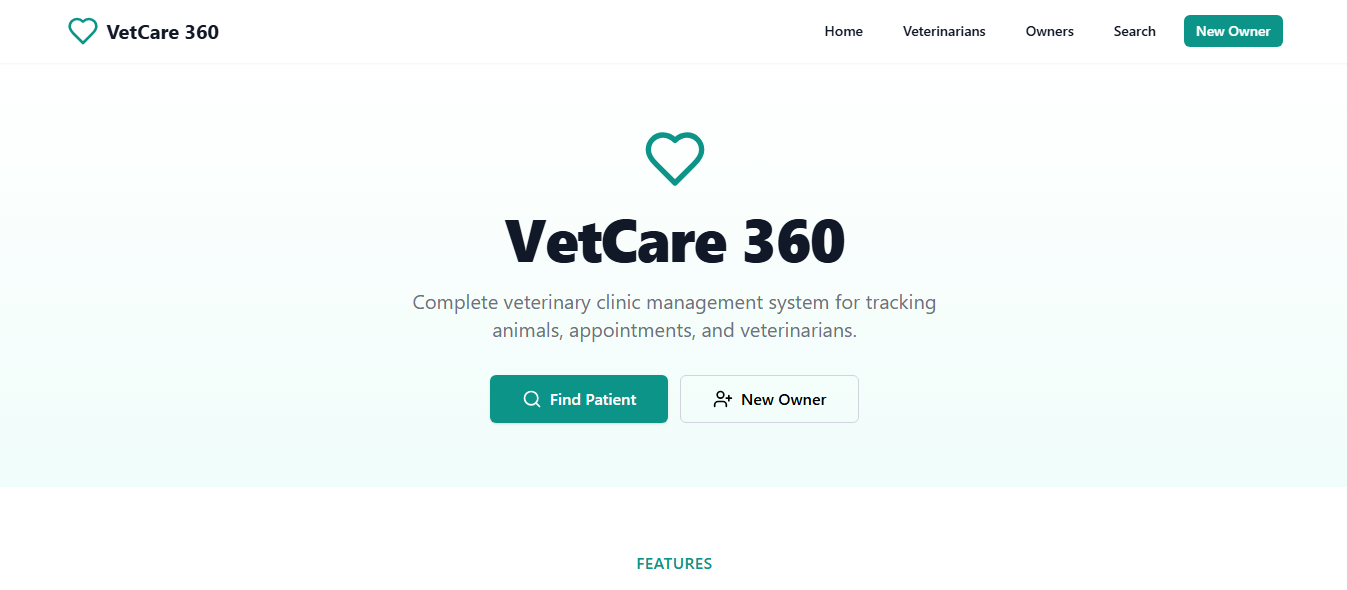
****

Table des matières :

Partie 1 : Introduction et Vue d'ensemble :

1. Introduction au projet

2. Objectifs et portée

3. Public cible

4. Méthodologie de développement

Partie 2 : Analyse et Conception

5. Analyse des besoins

6. Spécifications fonctionnelles

7. Architecture technique

8. Conception de la base de données

Partie 3 : Développement Frontend

9. Configuration de l'environnement React

10. Développement des composants

11. Gestion de l'état et des données

12. Interface utilisateur et UX

Partie 4 : Développement Backend

13. Configuration de la base de données

14. Développement des API

15. Sécurité et validation

16. Tests et débogage

Partie 5 : Déploiement et Maintenance

17. Stratégie de déploiement

18. Documentation technique

19. Plan de maintenance

20. Perspectives d'évolution

Partie 1 : Introduction et Vue d'ensemble

1. Introduction au projet :

1.1 Contexte

*La Clinique Vétérinaire* est une application web moderne et intuitive, développée pour répondre aux besoins croissants des cliniques vétérinaires en matière de gestion numérique efficace. Dans un contexte où la digitalisation des services médicaux s’impose comme une nécessité incontournable, ce projet s’inscrit dans une dynamique d’amélioration continue des processus de gestion au sein des établissements de soins animaliers.

Ce système a été conçu dans le but d’optimiser la gestion quotidienne des différentes activités d’une clinique vétérinaire, en facilitant l’enregistrement, le suivi et la mise à jour des informations relatives aux propriétaires d’animaux, aux animaux eux-mêmes, ainsi qu’aux visites médicales et traitements associés. Grâce à une interface claire et responsive, les utilisateurs peuvent accéder rapidement aux données essentielles et effectuer leurs tâches de manière fluide, ce qui améliore considérablement l’efficacité du personnel soignant.

En s’appuyant sur des technologies modernes comme ReactJS pour le front-end, NodeJS pour le back-end, et MongoDB comme base de données NoSQL, l’application garantit une performance optimale, une évolutivité assurée, et une meilleure expérience utilisateur. Ce projet incarne ainsi une solution concrète et pratique pour accompagner les cliniques vétérinaires dans leur transition numérique, tout en garantissant la fiabilité, la sécurité et la facilité d’accès à l’information.

1.2 Vision du projet

Notre vision est de créer une solution complète et intuitive qui simplifie la gestion des patients, des propriétaires et des visites médicales, tout en améliorant l'efficacité opérationnelle des cliniques vétérinaires.

1.3 Importance stratégique

L'application répond à un besoin crucial de modernisation des pratiques vétérinaires, permettant une transition fluide vers une gestion numérique complète des opérations cliniques.

2. Objectifs et portée du projet

#### 2.1 Objectifs principaux :

Ce projet vise à transformer en profondeur la gestion des cliniques vétérinaires à travers une série d’objectifs fondamentaux, à savoir :

- Digitaliser la gestion des dossiers médicaux : en mettant en place un système numérique sécurisé pour stocker, consulter et mettre à jour les données médicales des animaux.

-Optimiser l'organisation des rendez-vous : via un calendrier interactif permettant de planifier, suivre et gérer les consultations de manière efficace.

-Améliorer le suivi des patients (animaux) : en assurant un historique médical complet, incluant diagnostics, traitements, et recommandations.

-Faciliter la communication avec les propriétaires : grâce à une interface de notification et de rappel concernant les rendez-vous ou l’état de santé de leurs animaux.

-Standardiser les processus cliniques : en instaurant des procédures uniformes pour la saisie des données médicales et la prise en charge des patients.

#### 2.2 Portée du projet :

Le périmètre du projet englobe plusieurs modules essentiels, permettant une gestion intégrée de la clinique vétérinaire :

-Système de gestion des propriétaires : pour l’enregistrement, la consultation et la mise à jour des informations des clients.

-Gestion des dossiers des animaux : comprenant l’identification, l’espèce, la race, la date de naissance, et d'autres caractéristiques.

-Suivi des visites médicales : permettant aux vétérinaires de documenter leurs observations, diagnostics et traitements administrés.

-Interface dédiée aux vétérinaires : conçue pour faciliter l’accès rapide aux données et améliorer la fluidité du travail quotidien.

-Système de recherche et de reporting : pour filtrer les données et générer des rapports utiles à la prise de décision et à l’analyse de performance.

#### 2.3 Livrables attendus :

À l’issue du projet, plusieurs livrables essentiels seront fournis afin d’assurer la pérennité et l’utilisabilité de la solution développée :

-Application web responsive : fonctionnant de manière fluide sur ordinateurs, tablettes et smartphones.

-Base de données sécurisée : garantissant la confidentialité et l’intégrité des données sensibles.

-Documentation technique complète : décrivant l’architecture du système, son installation et sa maintenance.

-Guide utilisateur simplifié : destiné à faciliter la prise en main de l’outil par les utilisateurs finaux.

-Support de formation : pour accompagner le personnel médical et administratif dans l’apprentissage du système.

3. Public cible

3.1 Utilisateurs primaires

- Vétérinaires

- Assistants vétérinaires

- Personnel administratif

- Propriétaires d'animaux

3.2 Besoins spécifiques par utilisateur

\*Vétérinaires :\*

- Accès rapide aux dossiers médicaux

- Saisie efficace des observations

- Suivi des traitements

- Gestion des ordonnances

\*Assistants vétérinaires :\*

- Gestion des rendez-vous

- Enregistrement des nouveaux patients

- Mise à jour des informations

- Suivi des vaccinations

\*Personnel administratif :\*

- Gestion des factures

- Planification des rendez-vous

- Gestion des stocks

- Reporting

\*Propriétaires d'animaux :\*

- Consultation des rendez-vous

- Accès à l'historique médical

- Demande de rendez-vous

- Réception des rappels

4. Méthodologie de développement

4.1 Approche Agile

Le projet a été développé selon une méthodologie Agile, avec des sprints de deux semaines, permettant une adaptation continue aux besoins et aux retours des utilisateurs.

4.2 Phases de développement

1. \*Phase de conception :\*

- Analyse des besoins

- Conception de l'architecture

- Prototypage de l'interface

- Validation des maquettes

2. \*Phase de développement :\*

- Développement frontend

- Développement backend

- Intégration continue

- Tests unitaires et d'intégration

3. \*Phase de test :\*

- Tests fonctionnels

- Tests de performance

- Tests d'acceptation

- Correction des bugs

4. \*Phase de déploiement :\*

- Préparation de l'environnement

- Migration des données

- Formation des utilisateurs

- Mise en production

Partie 2 : Analyse et Conception

5. Analyse des besoins

5.1 Étude préliminaire

Une étude approfondie a été menée auprès de plusieurs cliniques vétérinaires pour identifier les besoins essentiels et les points d'amélioration potentiels.

5.2 Besoins fonctionnels

1. \*Gestion des propriétaires :\*

- Enregistrement des nouveaux propriétaires

- Mise à jour des informations personnelles

- Recherche et filtrage

- Historique des interactions

2. \*Gestion des animaux :\*

- Création de fiches animaux

- Suivi médical

- Historique des traitements

- Gestion des vaccinations

3. \*Gestion des visites :\*

- Planification des rendez-vous

- Enregistrement des consultations

- Suivi des traitements

- Rappels automatiques

5.3 Besoins non fonctionnels

- Performance et rapidité

- Sécurité des données

- Interface intuitive

- Disponibilité 24/7

- Sauvegarde automatique

- Compatibilité multi-plateformes

6. Spécifications fonctionnelles

6.1 Fonctionnalités principales

\*Module Propriétaires :\*

typescript

interface Owner {

id: string;

firstName: string;

lastName: string;

address: string;

city: string;

telephone: string;

email: string;

createdAt: Date;

}

\*Module Animaux :\*

typescript

interface Pet {

id: string;

name: string;

species: string;

breed: string;

birthDate: Date;

weight: number;

ownerId: string;

medicalHistory: MedicalRecord[];

}

\*Module Visites :\*

typescript

interface Visit {

id: string;

date: Date;

petId: string;

veterinarianId: string;

description: string;

diagnosis: string;

treatment: string;

nextVisit?: Date;

}

6.2 Workflows principaux

1. \*Enregistrement d'un nouveau propriétaire :\*

- Saisie des informations personnelles

- Validation des données

- Création du compte

- Envoi d'un email de confirmation

2. \*Ajout d'un nouvel animal :\*

- Sélection du propriétaire

- Saisie des informations de l'animal

- Upload des documents

- Création du dossier médical

3. \*Planification d'une visite :\*

- Sélection de la date et heure

- Choix du vétérinaire

- Motif de la consultation

- Confirmation et rappel

7. Architecture technique

7.1 Stack technologique

\*Frontend :\*

- React 18.3.1

- TypeScript

- Tailwind CSS avec bootstrap

- React Router DOM

- Lucide React pour les icônes

\*Backend :\*

- Base de données MongoDB

- API RESTful

- Gestion des sessions

- Validation des données

7.2 Architecture des composants

typescript

// Structure des composants React

src/

├── components/

│ ├── layout/

│ │ ├── Footer.tsx

| | |\_\_\_PageHeader.tsx

│ │ ├── Header.tsx

│ │ └── layout.tsx

│ ├── ownwrs/

│ │ ├── OwnerForm.tsx

│ │

│ └── pets/

│ ├── PetForm.tsx

│

│ |── visits/

| | |\_\_\_\_VisitForm.tsx

| |\_\_ui/

| | |\_\_\_\_\_\_\_Botton.tsx

| | |\_\_\_\_\_\_Card.tsx  
 | | |\_\_\_\_\_\_input.tsx.tsx

| | |\_\_\_\_\_\_Select.tsx.tsx

| |\_\_context/

| | |\_\_\_\_AppContext.tsx

| |\_\_Server/

| | |\_\_\_\_db.ts

| | |\_\_\_index.ts

| |\_\_Services/

| | |\_\_\_dataService.ts

├── pages/

│ ├── Home.tsx

│ ├── Owners/

│ ├── Pets/

│ └── Visits/

…………..

8. Conception de la base de données

8.1 Schéma de la base de données

mongoDB

// ===== Owner Schema =====

const ownerSchema = new mongoose.Schema({

firstName: { type: String, required: true },

lastName: { type: String, required: true },

address: { type: String, required: true },

city: { type: String, required: true },

telephone: { type: String, required: true },

email: { type: String, unique: true, sparse: true },

createdAt: { type: Date, default: Date.now }

});

export const Owner = mongoose.model('Owner', ownerSchema);

// ===== Pet Schema =====

const petSchema = new mongoose.Schema({

name: { type: String, required: true },

birthDate: { type: Date, required: true },

type: { type: String, required: true },

breed: { type: String },

weight: { type: Number },

ownerId: { type: mongoose.Schema.Types.ObjectId, ref: 'Owner', required: true },

createdAt: { type: Date, default: Date.now }

});

export const Pet = mongoose.model('Pet', petSchema);

// ===== Visit Schema =====

const visitSchema = new mongoose.Schema({

date: { type: Date, required: true },

description: { type: String, required: true },

diagnosis: { type: String },

treatment: { type: String },

petId: { type: mongoose.Schema.Types.ObjectId, ref: 'Pet', required: true },

createdAt: { type: Date, default: Date.now }

});

Partie 3 : Développement Frontend

9. Configuration de l'environnement React

9.1 Installation et configuration

bash

# Installation des dépendances

npm install react@18.3.1 react-dom@18.3.1

npm install -D typescript @types/react @types/react-dom

npm install tailwindcss postcss autoprefixer

npm install react-router-dom lucide-react

9.2 Configuration de TypeScript

json

{

"compilerOptions": {

"target": "ES2020",

"useDefineForClassFields": true,

"lib": ["ES2020", "DOM", "DOM.Iterable"],

"module": "ESNext",

"skipLibCheck": true,

"moduleResolution": "bundler",

"allowImportingTsExtensions": true,

"resolveJsonModule": true,

"isolatedModules": true,

"noEmit": true,

"jsx": "react-jsx",

"strict": true,

"noUnusedLocals": true,

"noUnusedParameters": true,

"noFallthroughCasesInSwitch": true

},

"include": ["src"],

"references": [{ "path": "./tsconfig.node.json" }]

}

10. Développement des composants

10.1 Composants réutilisables

typescript

// Button component

interface ButtonProps {

variant: 'primary' | 'secondary' | 'danger';

size: 'sm' | 'md' | 'lg';

children: React.ReactNode;

onClick?: () => void;

disabled?: boolean;

}

const Button: React.FC<ButtonProps> = ({

variant,

size,

children,

onClick,

disabled

}) => {

const baseStyles = "rounded-md font-medium focus:outline-none";

const variantStyles = {

primary: "bg-green-600 text-white hover:bg-green-700",

secondary: "bg-gray-600 text-white hover:bg-gray-700",

danger: "bg-red-600 text-white hover:bg-red-700"

};

const sizeStyles = {

sm: "px-2 py-1 text-sm",

md: "px-4 py-2",

lg: "px-6 py-3 text-lg"

};

return (

<button

className={`${baseStyles} ${variantStyles[variant]} ${sizeStyles[size]}`}

onClick={onClick}

disabled={disabled}

>

{children}

</button>

);

};

### 11. Gestion de l'état et des données

#### 11.1 Hooks personnalisés

typescript

// Hook de gestion des formulaires

function useForm<T>(initialState: T) {

const [values, setValues] = useState<T>(initialState);

const [errors, setErrors] = useState<Partial<Record<keyof T, string>>>({});

const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {

const { name, value } = e.target;

setValues(prev => ({

...prev,

[name]: value

}));

};

const validate = (validationRules: Record<keyof T, (value: any) => string | null>) => {

const newErrors: Partial<Record<keyof T, string>> = {};

let isValid = true;

Object.keys(validationRules).forEach(key => {

const error = validationRules[key as keyof T](values[key as keyof T]);

if (error) {

newErrors[key as keyof T] = error;

isValid = false;

}

});

setErrors(newErrors);

return isValid;

};

return { values, errors, handleChange, validate };

}

### 12. Interface utilisateur et UX

#### 12.1 Design System

typescript

// Système de couleurs

const colors = {

primary: {

50: '#f0fdf4',

100: '#dcfce7',

500: '#22c55e',

600: '#16a34a',

700: '#15803d'

},

gray: {

50: '#f9fafb',

100: '#f3f4f6',

700: '#374151',

800: '#1f2937',

900: '#111827'

}

};

// Typographie

const typography = {

fontFamily: {

sans: ['Inter', 'system-ui', 'sans-serif'],

serif: ['Merriweather', 'serif']

},

fontSize: {

xs: '0.75rem',

sm: '0.875rem',

base: '1rem',

lg: '1.125rem',

xl: '1.25rem',

'2xl': '1.5rem'

}

};

Partie 4 : Développement Backend

13. Configuration de la base de données

13.1 Initialisation de la base de données

typescript

import { createClient } from '@libsql/client';

const db = createClient({

url: 'file:local.db',

});

export async function initializeDatabase() {

// Création des tables

await db.execute(`

CREATE TABLE IF NOT EXISTS owners (

id TEXT PRIMARY KEY,

firstName TEXT NOT NULL,

lastName TEXT NOT NULL,

address TEXT NOT NULL,

city TEXT NOT NULL,

telephone TEXT NOT NULL,

createdAt DATETIME DEFAULT CURRENT\_TIMESTAMP

)

`);

// ... autres tables

}

14. Développement des API

14.1 Services de données

typescript

// Service des propriétaires

export class OwnerService {

async create(owner: Omit<Owner, 'id' | 'createdAt'>) {

const id = randomUUID();

await db.execute({

sql: `

INSERT INTO owners (id, firstName, lastName, address, city, telephone)

VALUES (?, ?, ?, ?, ?, ?)

`,

args: [id, owner.firstName, owner.lastName, owner.address, owner.city, owner.telephone]

});

return id;

}

async findById(id: string) {

const result = await db.execute({

sql: 'SELECT \* FROM owners WHERE id = ?',

args: [id]

});

return result.rows[0];

}

// ... autres méthodes

}

15. Sécurité et validation

15.1 Validation des données

typescript

// Validateurs

const validators = {

required: (value: any) => {

return value ? null : 'Ce champ est requis';

},

phone: (value: string) => {

const phoneRegex = /^\d{10}$/;

return phoneRegex.test(value) ? null : 'Numéro de téléphone invalide';

},

email: (value: string) => {

const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

return emailRegex.test(value) ? null : 'Email invalide';

}

};

16. Tests et débogage

16.1 Tests unitaires

typescript

import { describe, it, expect } from 'vitest';

import { OwnerService } from '../services/OwnerService';

describe('OwnerService', () => {

const service = new OwnerService();

it('should create a new owner', async () => {

const ownerData = {

firstName: 'John',

lastName: 'Doe',

address: '123 Main St',

city: 'Paris',

telephone: '0123456789'

};

const id = await service.create(ownerData);

expect(id).toBeDefined();

const owner = await service.findById(id);

expect(owner).toMatchObject(ownerData);

});

});

Partie 5 : Déploiement et Maintenance

17. Stratégie de déploiement

17.1 Configuration de l'environnement de production

typescript

// Configuration de l'environnement

const config = {

development: {

database: {

url: 'file:local.db'

},

server: {

port: 3000

}

},

production: {

database: {

url: process.env.DATABASE\_URL

},

server: {

port: process.env.PORT || 8080

}

}

};

18. Documentation technique

18.1 API Documentation

typescript

/\*\*

\* @api {post} /api/owners Créer un nouveau propriétaire

\* @apiName CreateOwner

\* @apiGroup Owners

\*

\* @apiParam {String} firstName Prénom du propriétaire

\* @apiParam {String} lastName Nom du propriétaire

\* @apiParam {String} address Adresse

\* @apiParam {String} city Ville

\* @apiParam {String} telephone Numéro de téléphone

\*

\* @apiSuccess {String} id ID du propriétaire créé

\* @apiSuccess {Object} owner Données du propriétaire

\*/

19. Plan de maintenance

19.1 Monitoring et logging

typescript

// Service de logging

class Logger {

static info(message: string, context?: object) {

console.log(JSON.stringify({

level: 'info',

timestamp: new Date().toISOString(),

message,

context

}));

}

static error(error: Error, context?: object) {

console.error(JSON.stringify({

level: 'error',

timestamp: new Date().toISOString(),

message: error.message,

stack: error.stack,

context

}));

}

}

20. Perspectives d'évolution

20.1 Roadmap technique

1. \*Court terme (3-6 mois) :\*

- Amélioration des performances

- Optimisation des requêtes

- Ajout de nouvelles fonctionnalités

2. \*Moyen terme (6-12 mois) :\*

- Intégration de l'authentification

- Module de facturation

- API mobile

3. \*Long terme (12+ mois) :\*

- Intelligence artificielle pour le diagnostic

- Intégration IoT pour le suivi des animaux

- Plateforme de téléconsultation

**Conclusion :**

Au terme de ce projet, nous avons pu concevoir et développer un système d'information complet dédié à la gestion des cliniques vétérinaires, dans le but d'améliorer la qualité des services vétérinaires grâce à un outil numérique efficace et facile à utiliser. Ce système repose sur un ensemble de technologies modernes largement utilisées dans le marché du travail, notamment : ReactJS pour la création d’interfaces utilisateur interactives, NodeJS pour le développement du back-end, et MongoDB comme base de données flexible et puissante de type NoSQL.

Ce système permet d’enregistrer et de suivre toutes les informations relatives aux propriétaires d’animaux, aux animaux de compagnie eux-mêmes, ainsi qu’aux différentes visites et consultations médicales, de manière organisée et fluide. Il offre des fonctionnalités de recherche, d’ajout, de mise à jour et de suppression, ce qui facilite considérablement le travail quotidien au sein de la clinique, tout en réduisant les erreurs liées à la gestion manuelle traditionnelle.

D’un point de vue académique et technique, ce projet nous a permis d’appliquer les concepts théoriques acquis durant notre formation, tout en développant nos compétences pratiques dans la création d’applications web complètes. Il nous a également permis d’approfondir notre compréhension des bases de données NoSQL, de maîtriser les communications entre le front-end et le back-end via des API, et d’améliorer notre capacité à travailler en équipe, à gérer notre temps, et à résoudre des problèmes de manière méthodique.

Ainsi, ce projet ne constitue pas seulement une réalisation technique, mais représente une véritable expérience professionnelle qui nous prépare concrètement à intégrer le monde du développement logiciel, en renforçant notre confiance et notre aptitude à relever de futurs défis dans ce domaine.