VETZOONE

INGENIERÍA EN SOFTWARE

Equipo:

223201 - Avendaño Argueta José Alberto

223215 - Ramírez Mendoza Karla

223264 - Marroquín Ocaña Carlos

Universidad Politécnica de Chiapas

MSTG-ARCH-1

> **Todos los componentes se encuentran identificados y asegurar que son necesarios**

Implementación

Componentes Identificados

Microservicios Core

Servicio	Descripción	Prioridad
Auth Service	Gestión de identidad y autorización	CRÍTICO
Validation Service	Validación de identidad y cédula profesional del veterinario	CRÍTICO
Pet Service	Gestión de información de animales	ALTO
Medical Records Service	Expedientes clínicos digitales (Diagnóstico, vacunas y tratamientos)	CRÍTICO
Appointment Service	Sistema de citas	ALTO
Notification Service	Comunicaciones automáticas via email	MEDIO
Analytics Service	Procesamiento de registros de auditoría utilizados para el entrenamiento de un modelo de ML y NLP.	CRÍTICO
Prediction Service	Gestión de los modelos entrados para predicción de cancelación de citas y búsqueda inteligente de veterinarios mediante descripción de los síntomas de la mascota.	CRÍTICO

Infraestructura

Componente	Descripción
API Gateway	AWS API Gateway para enrutamiento centralizado
Microservices	EC2 para alojar las instancias de los microservicios
Database	PostgreSQL con RDS para persistencia individual por microservicio

Servicios Externos

Servicio Externo	Descripción
Cloudinary	Almacenamiento y procesamiento de imágenes
Resend	Para envíos de correos

Justificación de Necesidad

Cada componente cumple funciones específicas basadas en los hallazgos de la encuesta realizada para validar el proyecto:

- Expediente clínico digital (58.8% prioridad)
- Gestión de citas (47.1% prioridad)
- Recordatorios automáticos (41.2% prioridad)

Componentes No Considerados para esta versión del Software

Debido a la baja demanda en la encuesta se excluyen las siguientes funciones del MVP del proyecto:

- Módulo de facturación
- Módulo de análisis financieros

MSTG-ARCH-2

> **Los controles de seguridad nunca se aplican solo en el cliente, sino que también en los respectivos servidores**

Implementación

La implementación de este punto se explica en el documento ***Validación de entradas de datos*** anexado en la entrega.

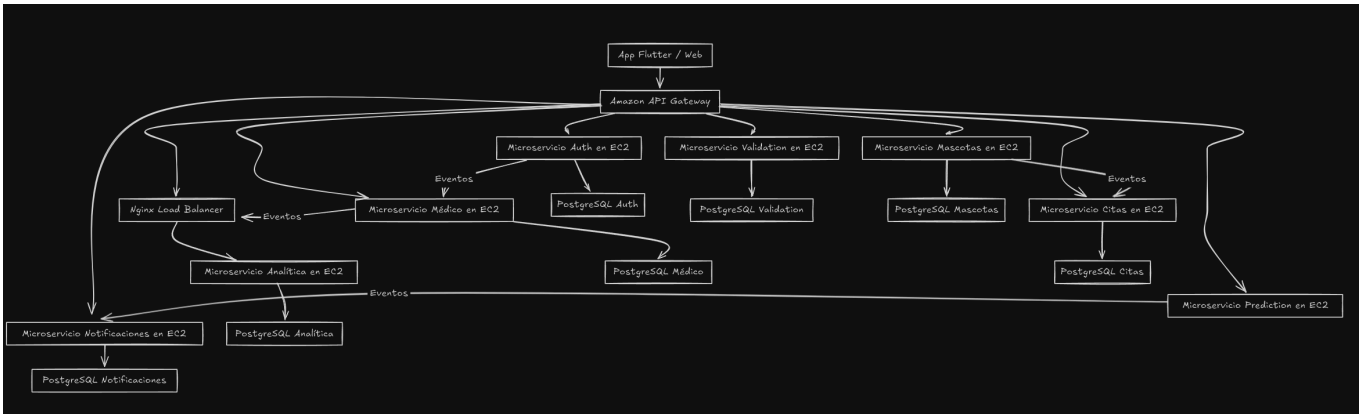
MSTG-ARCH-3

> **Se definió una arquitectura de alto nivel para la aplicación móvil y los servicios y se incluyeron controles de seguridad en la misma**

Implementación

Arquitectura de Alto Nivel

Diagrama de Arquitectura del Sistema

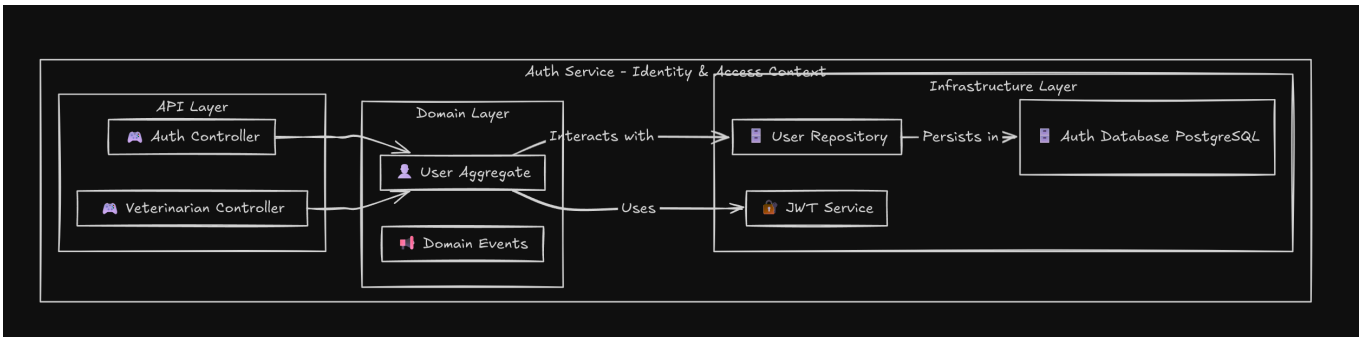


El flujo de la arquitectura inicia en una aplicación cliente desarrollada en Flutter, la cual se comunica exclusivamente con un Amazon API Gateway. Este gateway actúa como una puerta de enlace única y segura, encargándose de enrutar las solicitudes entrantes a los microservicios correspondientes. Cada microservicio está desplegado de forma independiente en una instancia de AWS EC2. Para garantizar el desacoplamiento y la autonomía, cada microservicio tiene su propia base de datos PostgreSQL aislada, alojada en una instancia AWS EC2. Este diseño promueve la escalabilidad, la resiliencia y la mantenibilidad del sistema.

Diagrama de componentes

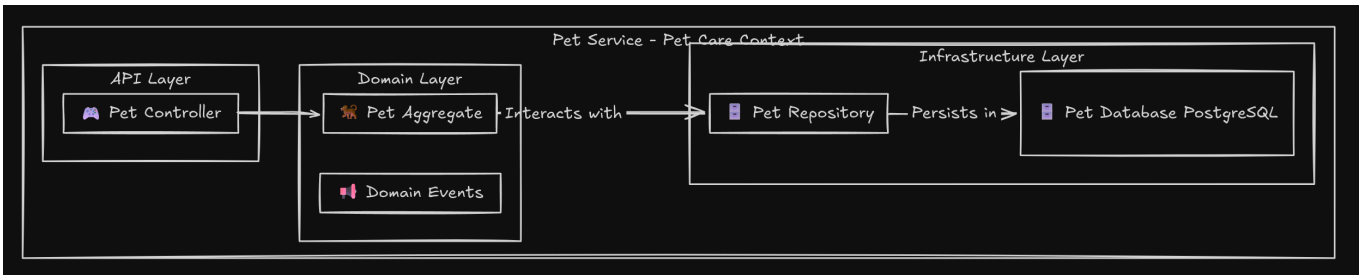
1. Auth

El Servicio de Autenticación funciona como el núcleo de la identidad y el control de acceso del sistema. Es responsable de gestionar el ciclo de vida completo de los usuarios, incluyendo el registro, inicio de sesión (autenticación) y la validación de permisos (autorización) tanto para los dueños de mascotas como para los veterinarios. Utiliza controladores para exponer los puntos de entrada de la API, un agregado de dominio para encapsular la lógica de negocio del usuario y servicios de infraestructura como JWT para generar tokens de seguridad y un repositorio para persistir los datos del usuario en su base de datos PostgreSQL dedicada.



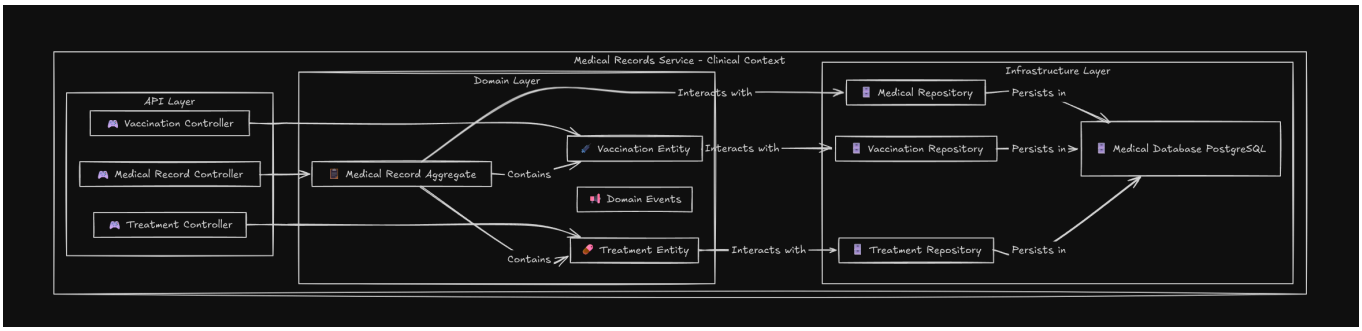
2. Pet

El Servicio de Mascotas está enfocado en la gestión integral de la información de las mascotas. Su principal responsabilidad es manejar los perfiles de los animales, lo que incluye datos como nombre, raza, edad, y su historial. A través de su `Pet Controller`, expone funcionalidades para crear, leer, actualizar y eliminar registros de mascotas. La lógica de negocio se encuentra encapsulada en el `Pet Aggregate`, que asegura la consistencia de los datos. Toda la información se persiste en una base de datos PostgreSQL dedicada a través del `Pet Repository`, manteniendo la información de las mascotas aislada y bien estructurada.



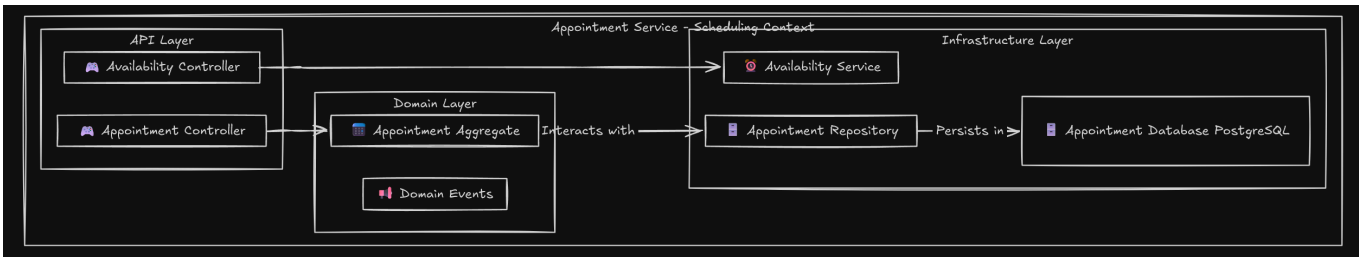
3. Medical

El Servicio de Registros Médicos es el encargado de gestionar toda la información clínica de las mascotas. Este microservicio centraliza el historial médico, los registros de vacunación y los tratamientos aplicados. Su capa de API cuenta con controladores especializados para cada tipo de dato clínico, lo que permite una gestión granular. En la capa de dominio, el `Medical Record Aggregate` actúa como la raíz que agrupa las entidades de `Vaccination` y `Treatment`, manteniendo la integridad del historial clínico. La capa de infraestructura utiliza repositorios específicos para cada entidad, los cuales persisten los datos en una única base de datos PostgreSQL para este contexto clínico.



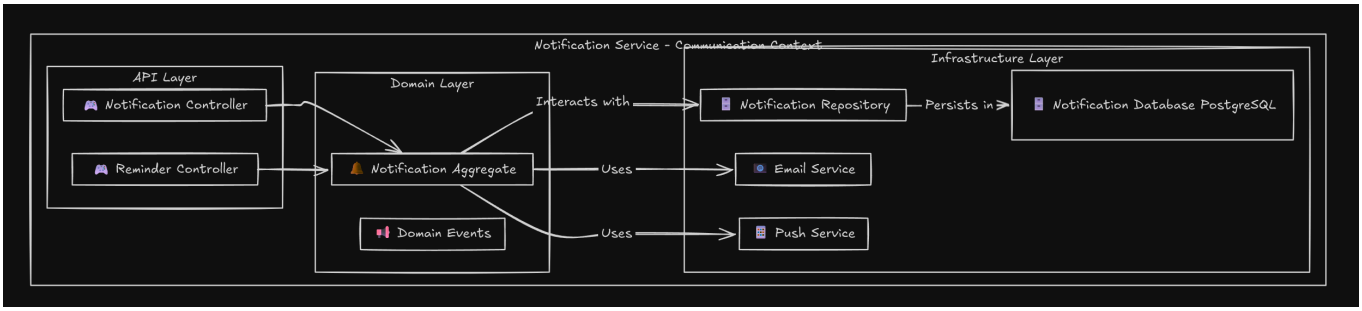
4. Appointment

El Servicio de Citas es el responsable de toda la lógica de programación y gestión de horarios. Permite a los usuarios agendar, consultar y cancelar citas con los veterinarios. A través de sus controladores, expone funcionalidades para manejar las citas (`Appointment Controller`) y para consultar la disponibilidad de los profesionales (`Availability Controller`). La lógica de negocio, como la validación de horarios y la creación de la cita, se encapsula en el `Appointment Aggregate`. La capa de infraestructura se apoya en un repositorio para la persistencia en su base de datos PostgreSQL y en un servicio de disponibilidad para gestionar los horarios de los veterinarios.



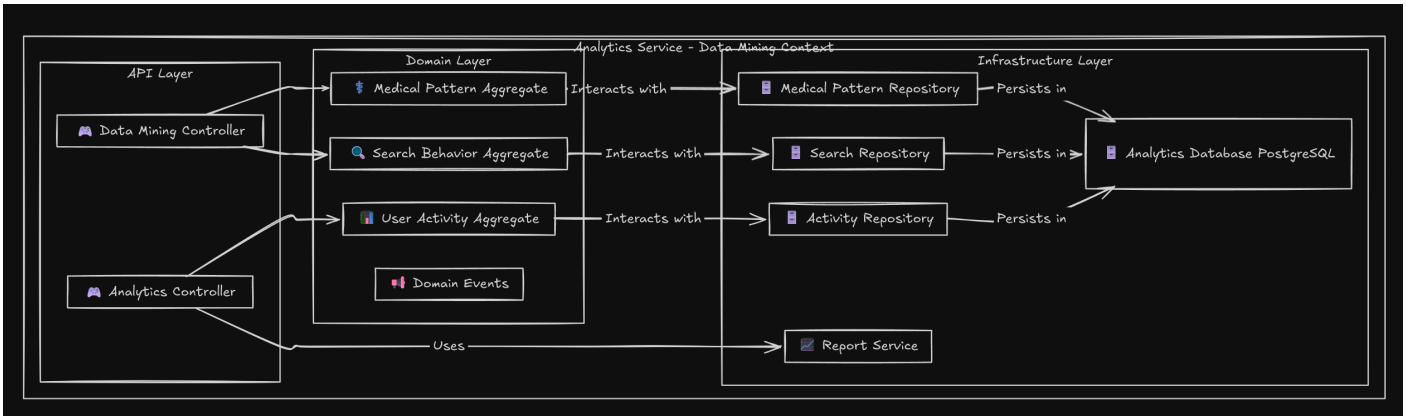
5. Notification

El Servicio de Notificaciones es el canal de comunicación proactiva con los usuarios. Su función es enviar comunicaciones relevantes como recordatorios de citas, confirmaciones de registro o actualizaciones importantes. La capa de API gestiona las solicitudes de notificaciones y recordatorios. La lógica de negocio, encapsulada en el `Notification Aggregate`, define el contenido y el momento del envío. Finalmente, la capa de infraestructura utiliza servicios externos (`EmailService`, `PushService`) para la entrega efectiva de los mensajes a través de diferentes canales, mientras que un repositorio se encarga de registrar el historial de notificaciones en su base de datos PostgreSQL.



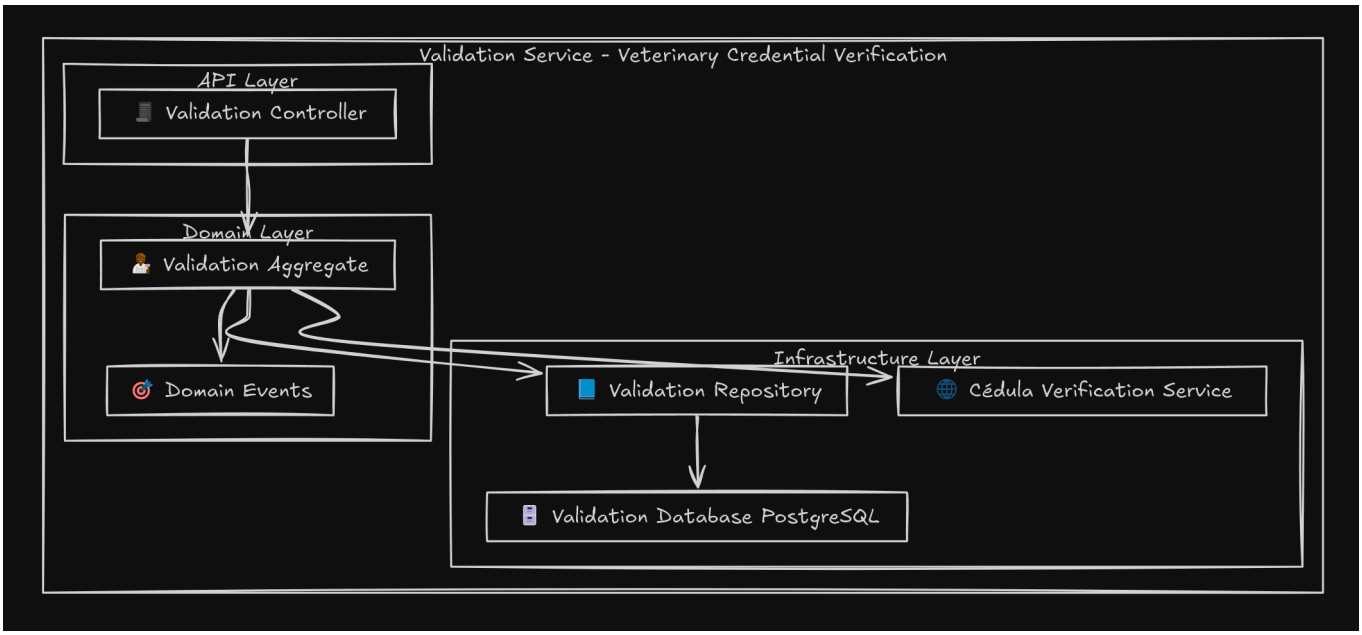
6. Analytics

El Servicio de Analítica está diseñado para recopilar y procesar datos de toda la plataforma con el fin de generar información de valor. Es responsable de rastrear la actividad del usuario, analizar patrones de búsqueda y identificar tendencias en los datos clínicos. Su capa de API permite tanto la consulta de reportes como la ingesta de datos para minería. El dominio se estructura en agregados específicos para cada tipo de dato analítico (actividad, búsqueda, patrones médicos), lo que facilita el procesamiento. La capa de infraestructura utiliza repositorios para persistir estos logs en una base de datos PostgreSQL optimizada para consultas y un servicio de reportes para generar las visualizaciones.



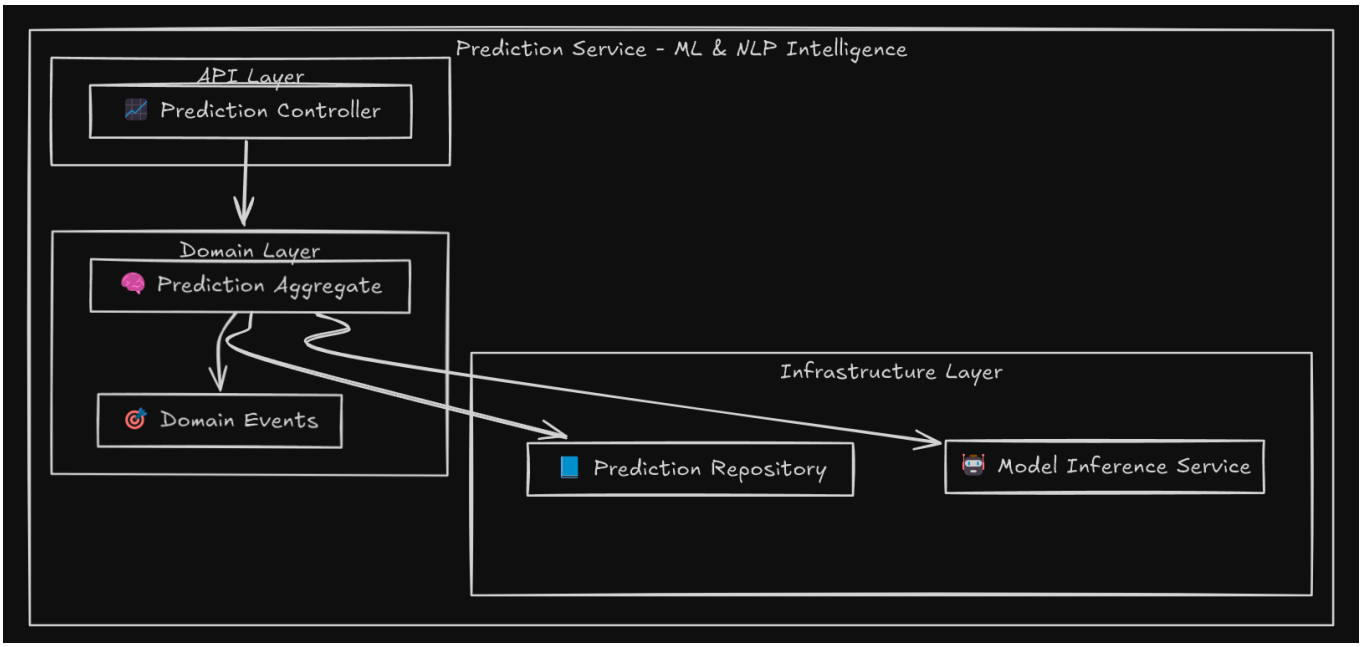
7. Validation

El Servicio de Validación es responsable de verificar la autenticidad de los profesionales veterinarios. Su objetivo principal es asegurar que quienes se registran como veterinarios en la plataforma cuenten con cédulas profesionales válidas. El `Validation Controller` expone los endpoints necesarios para cargar y validar esta información. En la capa de dominio, el `Validation Aggregate` administra la lógica de validación, incluyendo reglas sobre formatos, fechas de emisión y coincidencias con registros oficiales. La infraestructura incluye un `Validation Repository` para almacenar los resultados de las verificaciones y un `Cedula Verification Service`, que se conecta con fuentes externas del gobierno mexicano para contrastar la información ingresada.



8. Prediction

El Servicio de Predicción proporciona capacidades de machine learning e inteligencia artificial dentro del sistema. Su principal responsabilidad es inferir comportamientos futuros, como la probabilidad de que una cita sea cancelada o sugerencias inteligentes de veterinarios según los síntomas ingresados. En su capa de API, el `Prediction Controller` gestiona las solicitudes de predicción. El `Prediction Aggregate` en la capa de dominio maneja la lógica del modelo, encapsulando los criterios de entrada y salida. En la infraestructura, se conecta con un `ML Inference Service` que ejecuta los modelos previamente entrenados, y almacena resultados o logs relevantes en su base de datos PostgreSQL.



Controles de Seguridad por Capa

Capa de Presentación (App Móvil)

Componente	Descripción
Ofuscación de código	Protección contra ingeniería inversa
Secure Storage	Keychain/Keystore para tokens

Capa de Red

Componente	Descripción
WAF	Filtrado de ataques comunes
Rate Limiting	Por IP y usuario en API Gateway
TLS y SSL	Encriptación en tránsito

Capa de Aplicación (API Gateway)

- **JWT Validation**: Verificación centralizada de tokens
- **CORS Configuration**: Restricción de orígenes permitidos

Capa de Datos

- **Database Firewall**: Restricción de conexiones
- **Data Masking**: Ofuscación en entornos no productivos

Segmentación de Red

- **VPC Privada**: Servicios internos sin acceso directo
- **Security Groups**: Reglas específicas por microservicio y base de datos

---

MSTG-ARCH-4

> **Se identificó claramente la información considerada sensible en el contexto de la aplicación móvil**

Implementación

Clasificación de Información Sensible

Datos Críticos (Máxima Protección)

- **Credenciales de Veterinarios**: Cédula profesional
- **Passwords y Tokens**: Autenticación y autorización

Datos Sensibles (Alta Protección)

- **Información Personal**: Nombres, teléfonos, direcciones de clientes

Datos Internos (Protección Media)

- **Información de Mascotas**: Edad, peso, raza (sin contexto médico)

Controles por Tipo de Dato

Para Datos Críticos

Ejemplo de la encriptación de datos sensibles en la lógica de negocio:

```
// Encriptación AES-256 + campo específico
@Column({
  type: 'text',
  transformer: {
    to: (value: string) => encrypt(value),
    from: (value: string) => decrypt(value)
  }
})
medicalDiagnosis: string;
```

Se repite el mismo patrón para toda la información considerada en este nivel de criticidad.

Para Datos Sensibles

- Encriptación en tránsito (TLS 1.3)
- Hashing irreversible para contraseñas utilizando bcrypt y su método salt.

Para Datos Internos

- Logs sin información identificable
- Anonimización en minería de datos

Ubicación y Flujo de Datos Sensibles

Almacenamiento

- **Base de Datos**: Conexión PostgreSQL utilizando TSL

Procesamiento

- **APIs**: Sanitización en serialización JSON

MSTG-ARCH-5

> **Todos los componentes de la aplicación están definidos en términos de la lógica de negocio o las funciones de seguridad que proveen**

Implementación

Definición por Lógica de Negocio

Auth Service - Gestión de Identidad

Función de Negocio: Verificación y control de acceso profesional

- Registro de veterinarios con validación de licencia
- Autenticación de dueños de mascotas
- Control de roles y permisos específicos

Funciones de Seguridad:

- Verificación de credenciales profesionales
- Gestión de sesiones JWT
- Control de acceso basado en roles (RBAC)
- Prevención de cuentas duplicadas

Medical Records Service - Expediente Clínico Digital

Función de Negocio: Centralización del historial médico veterinario

- Creación de expedientes clínicos (58.8% prioridad en encuesta)
- Gestión de tratamientos y prescripciones
- Histórico médico completo por mascota

Funciones de Seguridad:

- Encriptación de datos médicos sensibles
- Control de acceso por veterinario autorizado
- Integridad de registros médicos

Appointment Service - Gestión de Citas

Función de Negocio: Optimización de agenda veterinaria

- Programación de citas (47.1% prioridad)
- Gestión de disponibilidad de veterinarios
- Coordinación de consultas

Funciones de Seguridad:

- Validación de disponibilidad real
- Prevención de doble reserva
- Control de acceso a agenda privada

Notification Service - Comunicación Automatizada

Función de Negocio: Recordatorios y comunicación efectiva

- Recordatorios automáticos (41.2% prioridad)
- Notificaciones de citas y tratamientos

Funciones de Seguridad:

- Validación de destinatarios autorizados
- Audit de comunicaciones críticas

Pet Service - Gestión de Mascotas

Función de Negocio: Administración de información básica de mascotas

- CRUD de información de mascotas
- Organización por propietario
- Gestión de perfiles animales

Funciones de Seguridad:

- Verificación de ownership
- Control de acceso a información privada
- Validación de datos de entrada

Analytics Service - Minería de Datos

Función de Negocio: Procesamiento de registros de auditoría para ML y PLN.

- Registro de logs de acciones para alimentar el modelo de ML (búsquedas de veterinario y agendamiento/cancelación de citas).
- Búsquedas de veterinario mediante análisis de síntomas detectados en la mascota
- Generación de probabilidad de cancelación de cita mediante un modelo de ML.

Funciones de Seguridad:

- Anonimización automática de datos personales
- Agregación estadística segura
- Control de acceso a datos sensibles

MSTG-ARCH-6

> **Se realizó un modelado de amenazas para la aplicación móvil y los servicios en el que se definieron las mismas y sus contramedidas**

Implementación

Metodología de Modelado de Amenazas

Utilizando el framework **STRIDE** aplicado a la arquitectura de microservicios veterinarios:

Spoofing (Suplantación de Identidad)

Amenazas Identificadas:

- Suplantación de veterinarios sin licencia válida
- Creación de cuentas falsas para acceder a información médica
- Ataques de phishing dirigidos a veterinarios

Contramedidas:

```
// Verificación obligatoria de licencia veterinaria
@Entity()
export class VeterinarianProfile {
  @Column({ unique: true })
  licenseNumber: string;
```

```
  @Column({ default: false })
  isVerified: boolean; // Verificación externa requerida
}
```

- Verificación manual de licencias veterinarias
- Validación cruzada con entidad gubernamental que valide cédulas profesionales

Tampering (Alteración de Datos)

Amenazas Identificadas:

- Modificación no autorizada de expedientes médicos
- Alteración de fechas de tratamientos o vacunaciones
- Cambio de diagnósticos por personal no autorizado

Contramedidas:

```
// Immutable medical records con audit trail
@Entity()
export class MedicalRecord {
  @Column({ type: 'enum', enum: ['DRAFT', 'FINAL'] })
  status: RecordStatus;

  @Column({ type: 'jsonb' })
  auditTrail: AuditEntry[]; // Histórico de cambios

  // Solo veterinarios pueden finalizar registros
  @ManyToOne(() => VeterinarianProfile)
  veterinarian: VeterinarianProfile;
}
```

- Registros médicos inmutables una vez finalizados
- Audit trail completo de modificaciones
- Firmas digitales para documentos críticos
- Control de versiones en expedientes

Repudiation (Repudio)

Amenazas Identificadas:

- Negación de creación de prescripciones médicas
- Rechazo de responsabilidad en diagnósticos

- Disputas sobre tratamientos recomendados

Contramedidas:

```
// Logging exhaustivo de acciones críticas
@Injectable()
export class AuditLogger {
  async logMedicalAction(
    veterinarianId: string,
    action: string,
    petId: string,
    details: any
  ) {
    await this.auditRepository.save({
      veterinarianId,
      action,
      petId,
      timestamp: new Date(),
      ipAddress: this.request.ip,
      userAgent: this.request.headers['user-agent'],
      details
    });
  }
}
```

- Logs inmutables de todas las acciones médicas
- Timestamps criptográficamente seguros
- Geolocalización de acciones críticas
- Respaldos distribuidos de audit trails

Information Disclosure (Divulgación de Información)

Amenazas Identificadas:

- Acceso no autorizado a historiales médicos
- Filtración de datos personales de clientes
- Exposición de información veterinaria confidencial

Contramedidas:

```
// Encriptación a nivel campo para datos sensibles
@Column({
  transformer: {
    to: (value: string) => this.cryptoService.encrypt(value),
```

```
    from: (value: string) => this.cryptoService.decrypt(value)
  }
})
diagnosis: string;

// Control granular de acceso
@UseGuards(OwnershipGuard)
async getMedicalRecord(@Param('id') id: string) {
  // Solo propietario o veterinario tratante puede acceder
}
```

- Encriptación AES-256 para datos médicos
- Control de acceso basado en roles granular
- Principio de menor privilegio
- Enmascaramiento de datos en logs

Denial of Service (Denegación de Servicio)

Amenazas Identificadas:

- Ataques DDoS contra servicios críticos
- Sobrecarga de sistema de citas
- Spam en sistema de notificaciones

Contramedidas:

```
// Rate limiting por usuario y IP
@RateLimit({ windowMs: 15 * 60 * 1000, max: 100 })
@Controller('appointments')
export class AppointmentController {
  // Límites específicos para operaciones críticas
}
```

- Rate limiting escalonado por tipo de usuario
- CDN con protección DDoS

Elevation of Privilege (Elevación de Privilegios)

Amenazas Identificadas:

- Escalación de permisos de cliente a veterinario
- Acceso no autorizado a funciones administrativas
- Bypass de controles de authorization

Contramedidas:

```
// Validación estricta de roles en cada endpoint
@UseGuards(JwtAuthGuard, RoleGuard)
@Roles('VETERINARIAN')
async createMedicalRecord() {
  // Verificación adicional de licencia activa
  const vet = await this.verifyActiveLicense(user.id);
  if (!vet.isActive) throw new UnauthorizedException();
}
```

- Verificación de roles en cada request
- Tokens JWT con claims específicos
- Validación de licencias
- Separación estricta de responsabilidades

Matriz de Riesgos vs Contramedidas

Amenaza	Probabilidad	Impacto	Riesgo	Contramedida Principal
Suplantación Veterinario	Media	Alto	Alto	Verificación manual licencias
Alteración Expedientes	Baja	Crítico	Alto	Registros inmutables + audit
Filtración Datos Médicos	Media	Crítico	Crítico	Encriptación multicapa
DDoS Servicios	Alta	Medio	Alto	Rate limiting + auto-scaling
Escalación Privilegios	Baja	Alto	Medio	Validación rol por request

---

MSTG-ARCH-10

> **La implementación de medidas de seguridad es una parte esencial durante todo el ciclo de vida del desarrollo de software de la aplicación**

Implementación

Desarrollo de cada Microservicio en NestJS

Validación y Seguridad en Código

```
// main.ts - Configuración global de seguridad
app.useGlobalPipes(new ValidationPipe({
  whitelist: true,
  forbidNonWhitelisted: true,
  transform: true
}));
app.use(helmet());
app.enableCors({ origin: process.env.ALLOWED_ORIGINS });
```

DTOs con Validación

```
export class CreateVetDTO {
  @IsString()
  name: string;

  @IsNumber()
  license: number;

  @IsString()
  description: string;

  @IsUUID()
  user_id: string;
}
```

CI/CD - GitHub Actions

Pipeline de Seguridad

```
name: Deploy with Security
on:
  push:
    branches: [main]

jobs:
  security:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Security Audit
        run: |
          npm audit --audit-level moderate
          npm run test:security
```

```
      - name: Build & Deploy
        run: |
          npm run build
          rsync -avz dist/ ${{ secrets.EC2_USER }}@${{ secrets.EC2_HOST
}}:/var/www/vetzoone/
```

Infraestructura EC2 + Nginx + PM2

Nginx Security Config

```
server {
    listen 443 ssl http2;

    # Security headers
    add_header X-Content-Type-Options nosniff;
    add_header X-Frame-Options DENY;
    add_header X-XSS-Protection "1; mode=block";

    # Rate limiting
    limit_req zone=api burst=20 nodelay;

    location /api {
        proxy_pass http://localhost:3000;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

PM2 Security Process

```
{
  "apps": [{
    "name": "{{nombre_del_microservicio}}",
    "script": "dist/main.js",
    "env": {
      "NODE_ENV": "production"
    },
    "error_file": "/var/log/{{nombre_del_microservicio}}/error.log",
    "out_file": "/var/log/{{nombre_del_microservicio}}/out.log",
    "max_memory_restart": "1G"
  }]
}
```

Monitoring y Mantenimiento

Security Tests en NestJS

```
describe('Security Tests', () => {
  it('should reject malicious SQL injection', async () => {
    const maliciousInput = "'; DROP TABLE users; --";
    await expect(
      service.findUser(maliciousInput)
    ).rejects.toThrow();
  });
});
```

Environment Security

```
# .env privado y agregado al .gitignore
JWT_SECRET=random_256_bit_key
DB_PASSWORD=encrypted_password
API_RATE_LIMIT=100
```

Resumen de Métricas

Checkpoint	Acción
Pre-commit	`npm audit`
CI/CD	Security tests + dependency scan
Deploy	SSL cert validation
Runtime	Error logging + rate limiting

MSTG-ARCH-12

> **La aplicación debería de cumplir con las leyes y regulaciones de privacidad**

Implementación

Para garantizar el cumplimiento de la **Ley Federal de Protección de Datos Personales en Posesión de los Particulares (LFPDPPP)** y otras regulaciones aplicables, VetZoone implementa un Aviso de Privacidad claro y accesible. Este documento informa a los usuarios sobre el tratamiento de su información, otorgándoles control sobre sus datos y cumpliendo con

los principios de licitud, consentimiento, información, calidad, finalidad, lealtad, proporcionalidad y responsabilidad.

Aviso de Privacidad

Aviso de Privacidad Simplificado de VetZoone

1. Identidad y Domicilio del Responsable VetZoone (en adelante "la Plataforma"), con domicilio ficticio en Av. Tecnologías 123, Ciudad de México, es el responsable del uso y protección de sus datos personales.

2. Datos Personales Recabados Para cumplir con las finalidades descritas, recabamos las siguientes categorías de datos:

- **Datos de Identificación y Contacto:** Nombre completo, correo electrónico, número de teléfono, contraseña de acceso.
- **Datos de Veterinarios (Críticos y Sensibles):** Adicionalmente a los anteriores, se recaba la cédula profesional para su validación.
- **Datos de Mascotas:** Nombre, edad, raza, peso y fotografías.
- **Datos de Salud (Sensibles):** Expedientes clínicos digitales, incluyendo diagnósticos, vacunas, tratamientos y notas médicas relacionadas con las mascotas.
- **Datos de Uso de la Plataforma:** Registros de actividad, interacciones, búsquedas realizadas y logs de auditoría.

3. Finalidades del Tratamiento de Datos

Sus datos personales son utilizados para las siguientes finalidades:

A. Finalidades Primarias (Necesarias para el servicio):

- Gestionar su registro y autenticación en la plataforma.
- **Validar la identidad y cédula profesional** de los veterinarios para garantizar la seguridad y confianza.
- Crear y administrar los perfiles de las mascotas.
- Gestionar el **expediente clínico digital** de las mascotas.
- Agendar, modificar y cancelar citas con los veterinarios.
- Enviar **notificaciones y recordatorios** sobre citas y tratamientos vía correo electrónico.

B. Finalidades Secundarias (No necesarias para el servicio, pero que nos permiten mejorar la experiencia):

- Analizar el comportamiento del usuario de forma anonimizada para **entrenar modelos de Machine Learning y NLP**, con el objetivo de predecir la cancelación de citas y mejorar la

búsqueda inteligente de veterinarios.
- Generar estadísticas sobre el uso de la plataforma para la mejora continua del servicio.

Usted puede oponerse al tratamiento de sus datos para las finalidades secundarias en cualquier momento a través del mecanismo descrito en la sección de Derechos ARCO.

4. Transferencia de Datos Personales Para operar, VetZoone comparte información de manera segura y limitada con los siguientes proveedores, que son esenciales para la arquitectura del servicio:

- **Amazon Web Services (AWS):** Para el alojamiento de la infraestructura (API Gateway, EC2, RDS).
- **Resend:** Para el envío de correos electrónicos transaccionales y notificaciones. No se comparte información médica sensible.
- **Cloudinary:** Para el almacenamiento y procesamiento de las imágenes de perfil de usuarios y mascotas. No se almacenan imágenes de diagnóstico médico.

Fuera de estos casos, VetZoone no transferirá su información personal a terceros sin su consentimiento previo, salvo las excepciones previstas por la ley.

5. Ejercicio de los Derechos ARCO y Revocación del Consentimiento Usted tiene derecho a **Acceder, Rectificar, Cancelar** sus datos personales, así como a **Oponerse** a su tratamiento (Derechos ARCO) o **revocar su consentimiento**. Para ello, puede enviar una solicitud al correo electrónico support@zifra.mx, detallando su petición y acreditando su identidad.

6. Cambios al Aviso de Privacidad Cualquier modificación a este aviso de privacidad le será notificada a través de la aplicación móvil o por correo electrónico.

7. Link del aviso: Aviso de privacidad

Fecha de última actualización: 21 de julio de 2025.

MSTG-STORAGE-1

> **Las funcionalidades de almacenamiento de credenciales del sistema deben de ser utilizadas para almacenar información sensible, tal como información personal, credenciales de usuario o claves criptográficas**

Implementación

App Móvil - Flutter

Flutter Secure Storage

```dart
// secure_storage.service.dart
class SecureStorageService {
  static const _storage = FlutterSecureStorage(
    aOptions: AndroidOptions(
      encryptedSharedPreferences: true,
    ),
    iOptions: IOSOptions(
      accessibility: IOSAccessibility.first_unlock_this_device,
    ),
  );

  Future<void> storeToken(String token) async {
    await _storage.write(key: 'jwt_token', value: token);
  }

  Future<void> storeBiometricKey(String key) async {
    await _storage.write(key: 'biometric_key', value: key);
  }
}
```

Datos NO almacenados localmente

```dart
// ❌ Nunca se almacena en SharedPreferences normales:
// - Contraseñas
// - Datos médicos
// - Información personal sensible

// ✅ Solo se almacenan en Secure Storage:
// - JWT tokens
// - Role
```

Backend - NestJS

Variables de Entorno Seguras

```typescript
// config.service.ts
import "dotenv/config";
import * as joi from "joi";

interface EnvVars {
  STAGE: string;
  PORT: number;
```

```
  DB_HOST: string;
  DB_PORT: number;
  DB_USER: string;
  DB_PASSWORD: string;
  DB_NAME: string;
}

const envsSchema = joi
  .object({
    STAGE: joi.string().valid("dev", "prod").default("dev"),
    PORT: joi.number().required(),
    DB_HOST: joi.string().required(),
    DB_PORT: joi.number().required(),
    DB_USER: joi.string().required(),
    DB_PASSWORD: joi.string().required(),
    DB_NAME: joi.string().required(),
  })
  .unknown(true);

const { error, value } = envsSchema.validate(process.env);

if (error) throw new Error(`Config validation error: ${error.message}`);

const envVars: EnvVars = value;

export const envs = {
  stage: envVars.STAGE,
  port: envVars.PORT,
  dbHost: envVars.DB_HOST,
  dbPort: envVars.DB_PORT,
  dbUser: envVars.DB_USER,
  dbPassword: envVars.DB_PASSWORD,
  dbName: envVars.DB_NAME,
};
```

Infraestructura

EC2 Security

```
# ~/.aws/credentials (encriptado)
# /etc/ssl/private/{{nombre_del_microservicio}}.key (600 permissions)
# PM2 ecosystem con variables encriptadas
```

Controles Implementados

Tipo de Dato	Almacenamiento	Método
JWT Tokens	Flutter Secure Storage	Keychain/Keystore
Passwords	PostgreSQL	bcrypt hash
Datos Médicos	PostgreSQL	AES-256 encryption
SSL Certs	EC2 protected dirs	600 permissions

Nunca se ve en código/logs

- Contraseñas plaintext
- Claves de encriptación
- Tokens de acceso
- Información médica

MSTG-STORAGE-2

> **No se debe almacenar información sensible fuera del contenedor de la aplicación o del almacenamiento de credenciales del sistema**

Implementación

App Móvil - Flutter

Almacenamiento Seguro Únicamente

```
// ✅ Correcto - Flutter Secure Storage
await secureStorage.write(key: 'jwt_token', value: token);

// ❌ Prohibido - Almacenamiento externo
// - SharedPreferences para datos sensibles
// - Archivos en directorio público
// - Cache no encriptado
// - SD card/almacenamiento externo
```

Validaciones Implementadas

```
class DataValidator {
  static bool isSensitiveData(String data) {
    return data.contains(RegExp(r'password|token|medical|diagnosis'));
```

```
  }

  static Future<void> store(String key, String value) async {
    if (isSensitiveData(value)) {
      await FlutterSecureStorage().write(key: key, value: value);
    } else {
      await SharedPreferences.getInstance()
        .then((prefs) => prefs.setString(key, value));
    }
  }
}
```

Backend - NestJS

Contenedores Seguros

```
// Variables de entorno únicamente
process.env.JWT_SECRET; // AWS Secrets Manager
process.env.DB_PASSWORD; // Nunca en archivos

// ❌ Nunca se utiliza:
// - Archivos de configuración con secrets
// - Logs con información sensible
// - Temp files con datos médicos
```

Infraestructura

EC2 Configuración

```
# Secrets en memoria/variables encriptadas
export JWT_SECRET=$(aws secretsmanager get-secret-value --secret-id
vetzoone/jwt)

# ❌ No se almacenan en:
# - /tmp/ con datos sensibles
# - Logs sin sanitización
# - Backups no encriptados
```

Controles de Cumplimiento

Ubicación	Permitido	Prohibido
App Container	Secure Storage	SharedPreferences

Ubicación	Permitido	Prohibido
Backend Container	Environment vars	Config files
Database	Encrypted fields	Plain text

MSTG-STORAGE-3

> **No se escribe información sensible en los registros (logs) de la aplicación**

Implementación

Sanitización de Logs en el Backend - NestJS

Logger Personalizado

```
// secure-logger.service.ts
@Injectable()
export class SecureLogger {
  private sensitiveFields = ['password', 'token', 'diagnosis', 'email',
'phone'];

  sanitize(data: any): any {
    const sanitized = { ...data };
    this.sensitiveFields.forEach(field => {
      if (sanitized[field]) {
        sanitized[field] = '***REDACTED***';
      }
    });
    return sanitized;
  }

  log(message: string, data?: any) {
    const cleanData = data ? this.sanitize(data) : undefined;
    console.log(message, cleanData);
  }
}
```

Interceptor Global

```
// logging.interceptor.ts
@Injectable()
```

```
export class LoggingInterceptor implements NestInterceptor {
  intercept(context: ExecutionContext, next: CallHandler) {
    const request = context.switchToHttp().getRequest();

    // Log solo headers seguros
    this.logger.log('Request', {
      method: request.method,
      url: request.url,
      userAgent: request.headers['user-agent'],
      ip: request.ip
      // NO logear: Authorization, body con passwords
    });

    return next.handle();
  }
}
```

Configuración de Logs

PM2 Log Config

```
{
  "apps": [{
    "name": "{{nombre_del_microservicio}}",
    "log_file": "/var/log/{{nombre_del_microservicio}}/combined.log",
    "error_file": "/var/log/{{nombre_del_microservicio}}/error.log",
    "merge_logs": true,
    "log_date_format": "YYYY-MM-DD HH:mm:ss Z"
  }]
}
```

Nginx Log Sanitization

```
# Solo IPs y métodos, NO query params sensibles
log_format secure '$remote_addr - $request_method $uri $status';
access_log /var/log/nginx/access.log secure;
```

Datos Prohibidos en Logs

Nunca Logear	Permitido Logear
Passwords	Request methods
JWT tokens	HTTP status codes

Nunca Logear	Permitido Logear
Diagnósticos médicos	IP addresses
Datos personales	Timestamps
API keys	Error types (sanitized)

MSTG-STORAGE-4

> **No se comparte información sensible con servicios externos salvo que sea una necesidad de la arquitectura**

Implementación

Servicios Externos Autorizados

Necesarios para Arquitectura

```
// Únicos servicios externos con datos sensibles
const AUTHORIZED_EXTERNAL_SERVICES = {
  cloudinary: 'Imágenes de mascotas (no médicas)',
  resend: 'Notificaciones por email (sin diagnósticos)',
  aws_rds: 'Base de datos (encriptada)'
};
```

Controles de Intercambio

Email Service (Resend)

```
// notification.service.ts
async sendAppointmentReminder(appointment: Appointment) {
  // ✅ Datos no sensibles son utilizados para los emails
  const emailData = {
    to: appointment.ownerEmail,
    subject: 'Recordatorio de cita',
    petName: appointment.petName,
    date: appointment.scheduledAt
  };

  // ❌ Nunca se envian datos sensibles como: diagnósticos, tratamientos,
datos médicos
```

```
  await this.resend.emails.send(emailData);
}
```

Image Storage (Cloudinary)

```
// Solo se suben fotos de perfil o de mascotas, NO imágenes diagnósticas
async uploadPetPhoto(file: Express.Multer.File) {
  return await cloudinary.uploader.upload(file.path, {
    folder: 'pets/profiles',
    resource_type: 'image'
  });
}
```

Datos Restringidos

Servicio	Permitido	Prohibido
Resend	Recordatorios básicos	Diagnósticos médicos
Cloudinary	Fotos de perfil / mascotas	Imágenes diagnósticas
Analytics	Datos anonimizados	IDs reales

Anonimización para Analytics

```
// Interfaz representativa de los logs
// Solo datos agregados sin identificadores
const analyticsData = {
  petSpecies: 'DOG',
  ageRange: '1-3',
  region: 'NORTH', // No ciudad específica
  appointmentType: 'CHECKUP'
  // NO enviar: nombres, IDs, diagnósticos
};
```

MSTG-STORAGE-5

> **Se desactiva la caché del teclado en los campos de texto que contienen información sensible**

Implementación

App Móvil - Flutter

Campos Sensibles sin Caché

```
// Passwords y datos médicos
TextField(
  autocorrect: false,
  enableSuggestions: false,
  keyboardType: TextInputType.visiblePassword,
  decoration: InputDecoration(labelText: 'Contraseña'),
)

// Diagnósticos médicos
TextField(
  autocorrect: false,
  enableSuggestions: false,
  decoration: InputDecoration(labelText: 'Diagnóstico'),
)

// Información personal
TextField(
  autocorrect: false,
  enableSuggestions: false,
  keyboardType: TextInputType.emailAddress,
  decoration: InputDecoration(labelText: 'Email'),
)
```

Campos Normales con Caché

```
// Nombres de mascotas (no sensible)
TextField(
  autocorrect: true,
  enableSuggestions: true,
  decoration: InputDecoration(labelText: 'Nombre de mascota'),
)
```

Campos Afectados

Campo	Caché Desactivado	Motivo
Password	✅	Credenciales
Email	✅	Datos personales
Teléfono	✅	Información personal

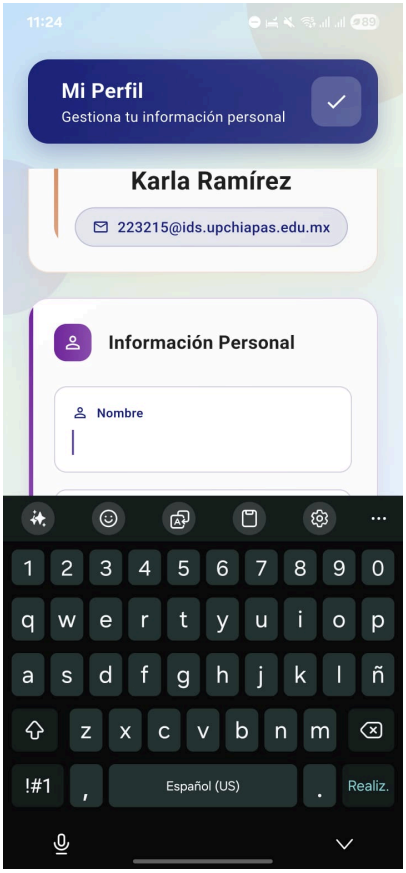
Campo	Caché Desactivado	Motivo
Diagnóstico	✅	Datos médicos
Tratamiento	✅	Información clínica
Nombre mascota	❌	No sensible
Raza	❌	No sensible

Validación de Implementación

```
class SensitiveFieldWidget extends StatelessWidget {
  final String label;
  final bool isSensitive;

  Widget build(BuildContext context) {
    return TextField(
      autocorrect: !isSensitive,
      enableSuggestions: !isSensitive,
      decoration: InputDecoration(labelText: label),
    );
  }
}
```

Esto previene que información médica o personal quede almacenada en el caché del teclado del dispositivo.



MSTG-STORAGE-7

> **No se expone información sensible como contraseñas y números de tarjetas de crédito a través de la interfaz o capturas de pantalla**

Implementación

App Móvil - Flutter

Campos con Ofuscación

```
// Contraseñas
TextField(
  obscureText: true,
  enableSuggestions: false,
  decoration: InputDecoration(labelText: 'Contraseña'),
)
```

Datos Médicos Protegidos

```
// Diagnósticos parcialmente ocultos
Text(isDoctor ? fullDiagnosis : '${diagnosis.substring(0, 20)}...')

// Background blur en app switcher
@override
void didChangeAppLifecycleState(AppLifecycleState state) {
  if (state == AppLifecycleState.paused) {
    SystemChrome.setApplicationSwitcherDescription(
      ApplicationSwitcherDescription(label: 'VetZoone', primaryColor:
0xFF000000)
    );
  }
}
```

Controles por Pantalla

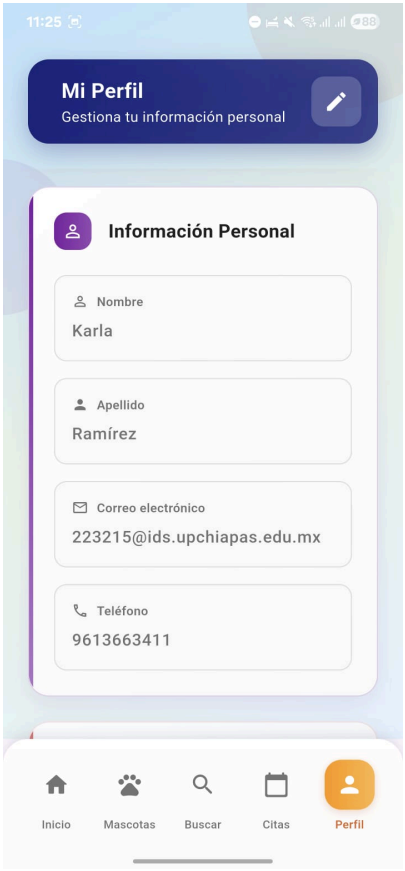
Pantalla	Control Implementado
Login	Password obscureText + FLAG_SECURE
Expediente Médico	Blur en background + datos truncados
Perfil Veterinario	Solo nombre visible, licencia parcial

Backend - Serialización Segura

```
// Nunca retornar passwords en APIs
@Exclude()
password: string;

// Datos médicos solo para veterinarios autorizados
@Transform(({ obj, key }) => obj.role === 'VET' ? obj[key] : '***')
diagnosis: string;
```

Nota: No manejamos tarjetas de crédito (fuera del alcance del proyecto integrador).



MSTG-STORAGE-11

> **La aplicación obliga a que exista una política mínima de seguridad en el dispositivo, como que el usuario deba configurar un código de acceso**

Implementación

App Móvil - Flutter

Validación de Seguridad del Dispositivo

```
// device_security.service.dart
class DeviceSecurityService {
  Future<bool> checkDeviceSecurity() async {
    final localAuth = LocalAuthentication();

    // Verificar biométricos o PIN configurado
    final isAvailable = await localAuth.canCheckBiometrics;
    final deviceSupported = await localAuth.isDeviceSupported();

    return isAvailable && deviceSupported;
```

```
  }
}
```

Bloqueo de Acceso

```
// Pantalla inicial con validación
class SecurityCheckScreen extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return FutureBuilder<bool>(
      future: DeviceSecurityService().checkDeviceSecurity(),
      builder: (context, snapshot) {
        if (snapshot.data == false) {
          return SecurityWarningScreen(); // Bloquea acceso
        }
        return LoginScreen();
      },
    );
  }
}
```

Políticas Mínimas Requeridas

Requisito	Implementación
PIN/Password	Verificación con `local_auth`
Bloqueo automático	Timeout de sesión
Almacenamiento y Cámara	Imágenes para el perfil del usuario o fotos de la mascota

---

MSTG-STORAGE-13

> **No se guarda ningún tipo de información sensible de forma local en el dispositivo móvil. En su lugar, esa información debería ser obtenida desde un sistema remoto sólo cuando es necesario y únicamente residir en memoria**

Implementación

App Móvil - Flutter

Solo en Memoria

```
// Variables en memoria únicamente
class MedicalDataController {
  List<MedicalRecord> _medicalRecords = []; // Solo en RAM
  String? _currentDiagnosis; // Temporal

  @override
  void dispose() {
    _medicalRecords.clear();
    _currentDiagnosis = null;
    super.dispose();
  }
}
```

Fetch On-Demand

```
Future<List<MedicalRecord>> getMedicalRecords(String petId) async {
  // Siempre desde API, nunca cache local
  final response = await http.get('/api/medical-records/$petId');
  return MedicalRecord.fromJsonList(response.data);
}
```

Datos NO Almacenados Localmente

Tipo	Almacenamiento	Duración
Expedientes médicos	Solo memoria	Por sesión
Diagnósticos	API fetch	Temporal
Datos personales	Memoria	Hasta logout
Tokens	Secure Storage	Persistente*

*Solo JWT tokens para autenticación

Limpieza Automática

```
@override
void didChangeAppLifecycleState(AppLifecycleState state) {
  if (state == AppLifecycleState.paused) {
    clearSensitiveMemoryData();
```

```
  }
}
```

Toda información médica y personal se obtiene en tiempo real desde la API y se mantiene únicamente en memoria durante el uso activo.

MSTG-STORAGE-14

> **En caso de ser necesario guardar información sensible de forma local, ésta debe de ser cifrada usando una clave derivada del hardware de almacenamiento seguro, el cual debe requerir autenticación previa**

Implementación

App Móvil - Flutter

Hardware-Based Encryption

```
// Derivación de clave desde hardware seguro
class HardwareSecureStorage {
  static const _storage = FlutterSecureStorage(
    aOptions: AndroidOptions(
      encryptedSharedPreferences: true,
      requireAuthentication: true, // Biométricos requeridos
    ),
    iOptions: IOSOptions(
      accessibility: IOSAccessibility.when_unlocked_this_device,
      biometricOnly: true, // Solo con Touch/Face ID
    ),
  );

  Future<void> storeEncrypted(String key, String value) async {
    await _storage.write(key: key, value: value);
  }
}
```

Autenticación Previa

```
Future<String?> getSecureData(String key) async {
  final localAuth = LocalAuthentication();
```

```
  final authenticated = await localAuth.authenticate(
    localizedReason: 'Acceso a datos médicos',
    options: AuthenticationOptions(
      biometricOnly: true,
      stickyAuth: true,
    ),
  );

  if (authenticated) {
    return await HardwareSecureStorage._storage.read(key: key);
  }
  return null;
}
```

Datos Cifrados Localmente

Dato	Cifrado	Autenticación
JWT Token	Keychain/Keystore	Usuario - Password

MSTG-CRYPTO-1

> **La aplicación no depende únicamente de criptografía simétrica cuyas claves se encuentran directamente en el código fuente de la misma**

Implementación

Gestión de Claves Externas

AWS Secrets Manager

```
// config.service.ts
@Injectable()
export class ConfigService {
  async getEncryptionKey(): Promise<string> {
    return await this.secretsManager.getSecretValue({
      SecretId: 'vetzoone/encryption-key'
    }).promise();
  }
}
```

Hardware-Derived Keys (Mobile)

```
// Claves derivadas del hardware del dispositivo
final key = await FlutterSecureStorage().read(key: 'derived_key');
// Generada automáticamente por Keychain/Keystore
```

Criptografía Híbrida

```
// medical-data.service.ts
async encryptMedicalData(data: string): Promise<string> {
  const symmetricKey = await this.generateRandomKey(); // AES-256
  const publicKey = await this.getPublicKey(); // RSA desde AWS

  const encryptedData = this.aesEncrypt(data, symmetricKey);
  const encryptedKey = this.rsaEncrypt(symmetricKey, publicKey);

  return { encryptedData, encryptedKey };
}
```

Controles Implementados

Componente	Método	Ubicación de Clave
Datos médicos	AES-256 + RSA	AWS Secrets Manager
JWT	HMAC-SHA256	Environment variable
Mobile storage	Hardware-derived	Keychain/Keystore

Nunca en código fuente: Claves de encriptación hardcodeadas.

MSTG-CRYPTO-2

> **La aplicación utiliza implementaciones de criptografía probadas**

Implementación

Backend - NestJS

Librerías Estándar

```
// bcrypt para passwords (FIPS-approved)
import * as bcrypt from 'bcrypt';
const hash = await bcrypt.hash(password, 12);

// crypto nativo Node.js para AES-256
import { createCipher, randomBytes } from 'crypto';
const key = randomBytes(32); // 256-bit
const cipher = createCipher('aes-256-gcm', key);

// jsonwebtoken para JWT
import * as jwt from 'jsonwebtoken';
const token = jwt.sign(payload, secret, { algorithm: 'HS256' });
```

App Móvil - Flutter

Implementaciones Probadas

```
// crypto package oficial
import 'package:crypto/crypto.dart';
final digest = sha256.convert(utf8.encode(data));

// Flutter Secure Storage (usa Keychain/Keystore)
import 'package:flutter_secure_storage/flutter_secure_storage.dart';
```

Algoritmos Utilizados

Uso	Algoritmo	Librería
Passwords	bcrypt	bcrypt (FIPS)
Datos médicos	AES-256-GCM	Node.js crypto
JWT	HMAC-SHA256	jsonwebtoken
Mobile storage	AES-256	Keychain/Keystore

No implementaciones custom: Solo librerías estándar y probadas en producción.

MSTG-CRYPTO-4

> **La aplicación no utiliza protocolos o algoritmos criptográficos ampliamente considerados obsoletos para su uso en seguridad**

Implementación

Algoritmos Modernos Utilizados

Backend - NestJS

```
// ✅ Algoritmos seguros
- AES-256-GCM (encriptación)
- bcrypt rounds=12 (passwords)
- HMAC-SHA256 (JWT)
- RSA-2048+ (asimétrico)
- TLS 1.3 (transporte)
```

App Móvil - Flutter

```
// ✅ Implementaciones actuales
- Keychain/Keystore nativo
- AES-256 hardware
- SHA-256 (hashing)
```

Algoritmos Prohibidos

Obsoleto	Reemplazo
❌ MD5	✅ SHA-256
❌ SHA-1	✅ SHA-256
❌ DES/3DES	✅ AES-256
❌ RC4	✅ AES-GCM
❌ TLS 1.0/1.1	✅ TLS 1.3

Configuración

```
# Nginx - Solo TLS modernos
ssl_protocols TLSv1.3;
ssl_ciphers ECDHE-RSA-AES256-GCM-SHA384;
```

Todos los algoritmos cumplen estándares actuales de seguridad.

MSTG-CRYPTO-5

> **La aplicación no reutiliza una misma clave criptográfica para varios propósitos**

Implementación

Claves Específicas por Propósito

Backend - NestJS

```
// Claves separadas por función
export class CryptoService {
  private readonly jwtSecret = process.env.JWT_SECRET;
  private readonly dbEncryptionKey = process.env.DB_ENCRYPTION_KEY;
  private readonly medicalDataKey = process.env.MEDICAL_DATA_KEY;
  private readonly auditLogKey = process.env.AUDIT_LOG_KEY;
}
```

App Móvil - Flutter

```
// Claves derivadas específicas
const keyPurposes = {
  'user_session': 'session_key',
  'biometric_data': 'bio_key',
  'cache_encryption': 'cache_key'
};
```

Segregación de Claves

Propósito	Clave	Algoritmo
JWT tokens	JWT_SECRET	HMAC-SHA256
Datos médicos	MEDICAL_KEY	AES-256-GCM
Base de datos	DB_KEY	AES-256
Audit logs	AUDIT_KEY	AES-256
Mobile storage	Device-derived	Keychain/Keystore

MSTG-AUTH-1

> **Si la aplicación provee acceso a un servicio remoto, un mecanismo aceptable de autenticación como usuario y contraseña es realizado en el servidor remoto**

Implementación

Autenticación en Backend - NestJS

NestJS Authentication

```
@Post('login')
async login(@Body() loginDto: LoginDto) {
  const user = await this.authService.validateUser(
    loginDto.email,
    loginDto.password
  );

  if (!user) {
    throw new UnauthorizedException('Credenciales inválidas');
  }

  return this.authService.login(user);
}
```

Validación de Credenciales

```
async validateUser(email: string, password: string) {
  const user = await this.userRepository.findByEmail(email);
  const isValid = await bcrypt.compare(password, user.passwordHash);

  return isValid ? user : null;
}
```

App Móvil - Flutter

Solo Envío de Credenciales

```
Future<AuthResponse> login(String email, String password) async {
  final response = await http.post('/api/auth/login', body: {
    'email': email,
    'password': password
  });

  // Nunca almacenar password localmente
```

```
  return AuthResponse.fromJson(response.data);
}
```

Flujo de Autenticación

1. **Cliente**: Envía email/password
2. **Servidor**: Valida con bcrypt
3. **Respuesta**: JWT token
4. **Almacenamiento**: Solo token en Secure Storage

No se almacenan passwords en cliente ni se validan localmente.

MSTG-AUTH-3

> **Si se utiliza la autenticación basada en tokens sin estado, el servidor proporciona un token que se ha firmado utilizando un algoritmo seguro**

Implementación

JWT con Algoritmos Seguros

Token Generation (NestJS)

```
@Injectable()
export class AuthService {
  generateToken(user: User): string {
    const payload = {
      sub: user.id,
      email: user.email,
      role: user.role
    };

    return jwt.sign(payload, process.env.JWT_SECRET, {
      algorithm: 'HS256', // HMAC-SHA256
      expiresIn: '1h'
    });
  }
}
```

Token Validation

```
@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy) {
  constructor() {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      secretOrKey: process.env.JWT_SECRET,
      algorithms: ['HS256'] // Solo algoritmos seguros
    });
  }
}
```

Algoritmos Utilizados

Componente	Algoritmo	Fortaleza
JWT Signature	HMAC-SHA256	256-bit
Secret Key	Random 256-bit	Rotación semanal

Mobile Implementation

```
// Headers con token firmado
final headers = {
  'Authorization': 'Bearer $jwtToken',
  'Content-Type': 'application/json'
};
```

Tokens stateless firmados con HMAC-SHA256, validados en cada request.

MSTG-AUTH-4

> **Cuando el usuario cierra sesión se termina la sesión también en el servidor**

Implementación

Justificación de No Aplicabilidad

Arquitectura JWT Stateless Este control no aplica completamente a nuestra arquitectura debido a:

1. **Tokens JWT sin estado**: No mantenemos sesiones server-side
2. **Expiración automática**: Tokens expiran en 1 hora automáticamente
3. **Complejidad vs beneficio**: Blacklist requiere Redis adicional para MVP

Implementación Actual

Mobile Logout

```
Future<void> logout() async {
  await secureStorage.delete(key: 'jwt_token');
  // Token eliminado del dispositivo
}
```

Server Validation

```
// Tokens válidos hasta expiración natural (1h)
@UseGuards(JwtAuthGuard)
async protectedRoute() {
  // Verificación de firma y expiración solamente
}
```

Mitigaciones Implementadas

Control	Implementación
Expiración corta	1 hora máximo
Eliminación local	Secure Storage cleared
Re-autenticación	Requerida cada hora

Para futuras versiones se considerará blacklisting con Redis para cumplimiento completo.

MSTG-AUTH-5

> **Existe una política de contraseñas y es aplicada en el servidor**

Implementación

Validación en Backend - NestJS

DTO con Política Estricta

```
export class RegisterUserDto {
  @IsString()
  @MinLength(9)
  @MaxLength(20)
  @IsStrongPassword({
    minLowercase: 1,
    minUppercase: 1,
    minNumbers: 1,
    minSymbols: 1,
  })
  password: string;
}
```

Política de Contraseñas

Requisito	Validación
Longitud	9-20 caracteres
Minúsculas	Mínimo 1
Mayúsculas	Mínimo 1
Números	Mínimo 1
Símbolos	Mínimo 1

Aplicación Server-Side

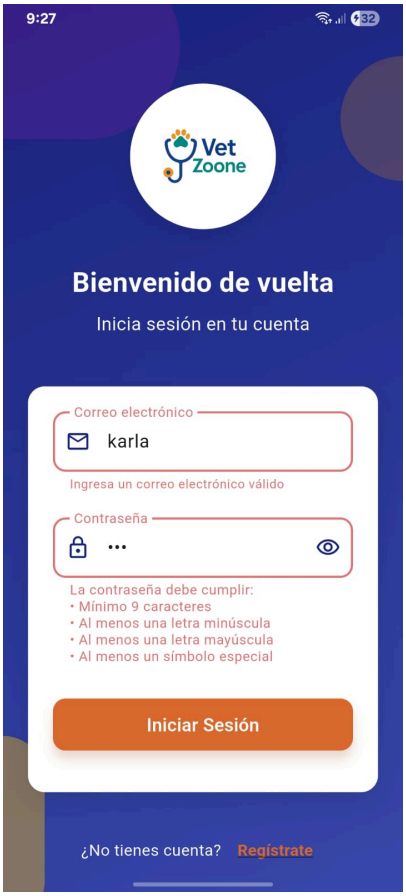
ValidationPipe Global

```
// main.ts
app.useGlobalPipes(new ValidationPipe({
  whitelist: true,
  forbidNonWhitelisted: true
}));
```

Hashing Seguro

```
async hashPassword(password: string): Promise<string> {
  return bcrypt.hash(password, 12); // Salt rounds = 12
}
```

La validación se ejecuta obligatoriamente en el servidor antes del registro, rechazando contraseñas débiles.



MSTG-NETWORK-1

> **La información es enviada cifrada utilizando TLS. El canal seguro es usado consistentemente en la aplicación**

Implementación

Configuración TLS

Nginx SSL

```
server {
    listen 443 ssl http2;
    ssl_certificate /etc/ssl/certs/vetzoone.crt;
    ssl_private_key /etc/ssl/private/vetzoone.key;
    ssl_protocols TLSv1.3;
```

```
    ssl_ciphers ECDHE-RSA-AES256-GCM-SHA384;
}
```

NestJS HTTPS Enforcement

```
// main.ts
app.use(helmet({
  hsts: { maxAge: 31536000, includeSubDomains: true }
}));
```

App Móvil - Flutter

HTTPS Only

```
// HTTP client configuration
final dio = Dio();
dio.options.baseUrl = 'https://api.vetzoone.com';

// Certificate pinning
(dio.httpClientAdapter as DefaultHttpClientAdapter).onHttpClientCreate =
(client) {
  client.badCertificateCallback = (cert, host, port) => false;
};
```

Controles Implementados

Componente	TLS Version	Validación
API Gateway	TLS 1.3	Certificate pinning
Database	TLS 1.2+	SSL required
Mobile	TLS 1.3	HTTPS only

Todo el tráfico usa HTTPS/TLS sin excepciones.

MSTG-NETWORK-3

> **La aplicación verifica el certificado X.509 del sistema remoto al establecer el canal seguro y sólo se aceptan certificados firmados por una CA de confianza**

Implementación

Mobile Certificate Validation

Flutter HTTPS Client

```
(dio.httpClientAdapter as DefaultHttpClientAdapter).onHttpClientCreate =
(client) {
  client.badCertificateCallback = (cert, host, port) {
    // Solo acepta certificados válidos de CA confiables
    return false; // Rechaza certificados inválidos
  };
};
```

Backend SSL Verification

NestJS Client Requests

```
// Para llamadas externas (validación automática)
const httpsAgent = new https.Agent({
  rejectUnauthorized: true // Rechaza certificados no confiables
});
```

Infraestructura

EC2 SSL Certificate

```
# Let's Encrypt (CA confiable)
ssl_certificate /etc/letsencrypt/live/api.vetzoone.com/fullchain.pem;
ssl_certificate_key /etc/letsencrypt/live/api.vetzoone.com/privkey.pem;
```

Controles

Conexión	Validación CA	Acción
Mobile → API	Automática	Rechaza inválidos
API → Externos	`rejectUnauthorized: true`	Falla conexión
Nginx	Let's Encrypt	CA confiable

Todas las conexiones validan certificados X.509 de CAs confiables automáticamente.

MSTG-PLATFORM-1

> **La aplicación requiere la cantidad de permisos mínimamente necesaria**

Implementación

Android Permissions

android/app/src/main/AndroidManifest.xml

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.CAMERA" />
```

Web Permissions

web/index.html

```
<!-- Solo permisos necesarios via JavaScript APIs -->
<script>
// Camera para fotos de mascotas
navigator.mediaDevices.getUserMedia({ video: true });

// Notificaciones para recordatorios
Notification.requestPermission();
</script>
```

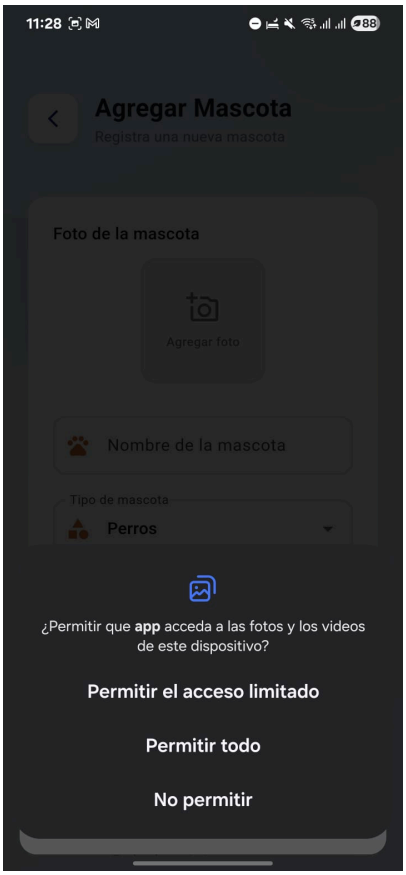
Permisos Justificados

Permiso	Uso	Justificación
INTERNET	API calls	Funcionalidad core
CAMERA	Fotos mascotas	Personalización
STORAGE	Fotos	Fotos de perfil / Mascota

Permisos NO Solicitados

- Contactos
- Micrófono

Solo permisos esenciales para funcionalidades veterinarias básicas.



MSTG-CODE-1

> **La aplicación es firmada y provista con un certificado válido, cuya clave privada está debidamente protegida**

Implementación

Android App Signing

GitHub Actions

```
- name: Sign APK
  run: |
    jarsigner -verbose -sigalg SHA256withRSA -digestalg SHA-256 \
    -keystore ${{ secrets.KEYSTORE_FILE }} \
    -storepass ${{ secrets.KEYSTORE_PASSWORD }} \
    app-release.apk vetzoone
```

Keystore Protection

```
# Almacenado en GitHub Secrets (encriptado)
KEYSTORE_PASSWORD: ${{ secrets.KEYSTORE_PASSWORD }}
KEY_ALIAS: vetzoone
KEY_PASSWORD: ${{ secrets.KEY_PASSWORD }}
```

Web App Signing

SSL Certificate

```
# Let's Encrypt para dominio
certbot --nginx -d vetzoone.zifra.mx
```

Protección de Claves

Componente	Almacenamiento	Acceso
Android Keystore	GitHub Secrets	CI/CD únicamente
SSL Private Key	EC2 `/etc/ssl/private/`	600 permissions

Certificados válidos con claves protegidas en almacenamiento seguro.