Ingeniería del Software II

Proyecto de Desarrollo de Software

Sistema de recuperación de mascotas perdidas en Oro Verde

Inderkumer, Priscila

Murzi, Ana Sol

04 de Noviembre 2025

Introducción

Necesidad que el sistema aborda

Ante la falta de un método eficaz para encontrar mascotas perdidas se propone como solución un **“Sistema de recuperación de mascotas perdidas en Oro Verde”**

Usuarios principales

Identificación de los usuarios principales

Vecinos de Oro Verde

Dueños de mascotas

Municipalidad de Oro Verde

Objetivos del software

Facilitar la comunicación inmediata entre usuarios ante una pérdida

Centralizar y mantener actualizada la información de mascotas registradas

Utilizar la geolocalización para optimizar la búsqueda de mascotas

Integrar notificaciones automáticas y visualización en mapas

Proveer un sistema escalable y de fácil acceso desde la web o dispositivos móviles

Análisis y diseño

Requerimientos funcionales

1. Registrar usuarios
2. Registrar y actualizar datos de mascotas
3. Reportar una mascota perdida o un avistamiento
4. Consultar reportes activos y sus ubicaciones
5. Generar notificaciones automáticas a usuarios cercanos cuando una mascota sea reportada como perdida
6. Integrar un servicio externo de geolocalización
7. Visualizar estadísticas y coincidencias entre reportes

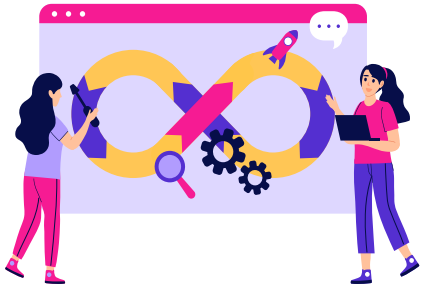
Requerimientos no funcionales

- Usabilidad
- Disponibilidad
- Seguridad
- Escalabilidad
- Mantenibilidad
- Interoperabilidad
- Performance
- Recuperación
- Confiabilidad
- Administrabilidad

Casos de Uso

UC22

Generar alerta de mascota perdida



UC23

Consultar ubicación a sistema de geolocalización

Arquitectura y despliegue

Diagrama de contexto

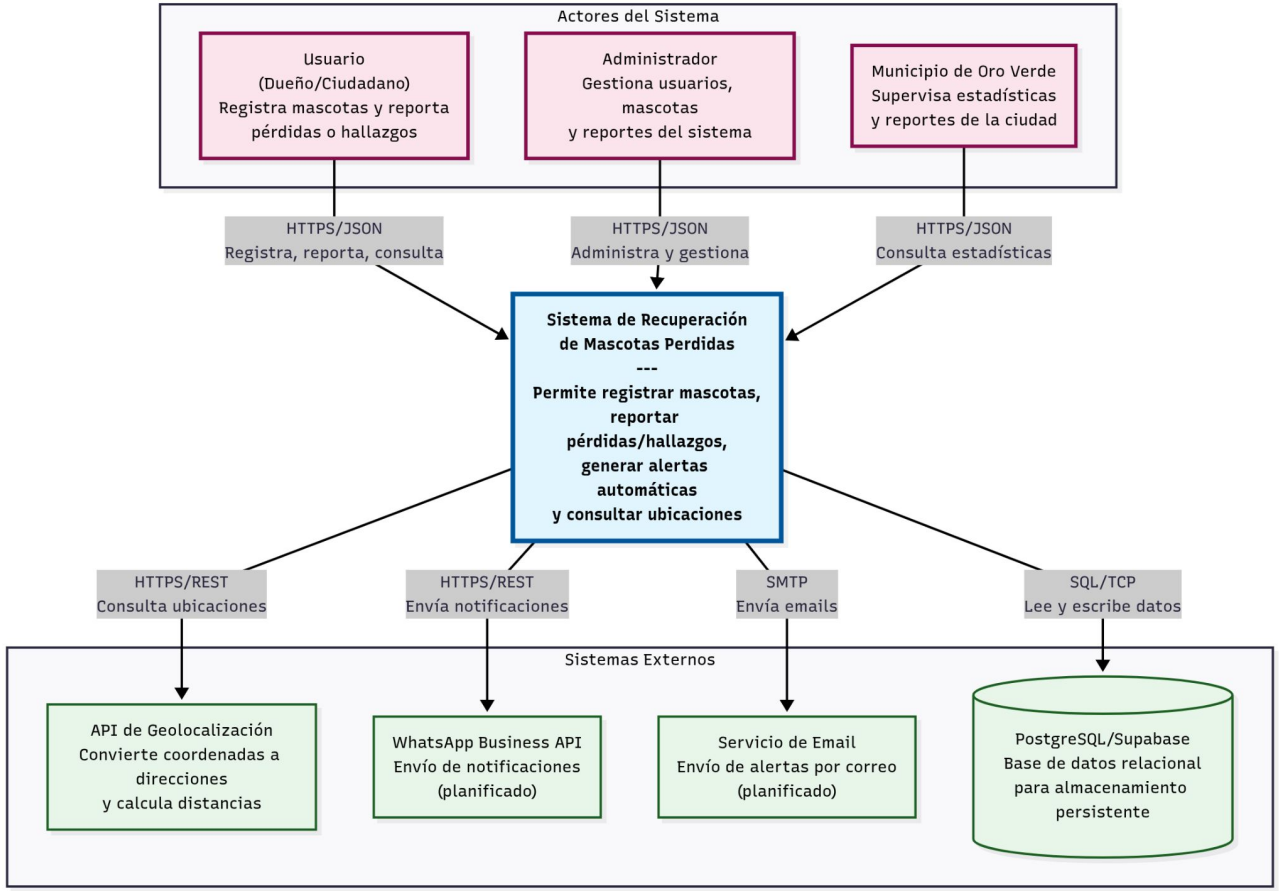


Diagrama de contenedores

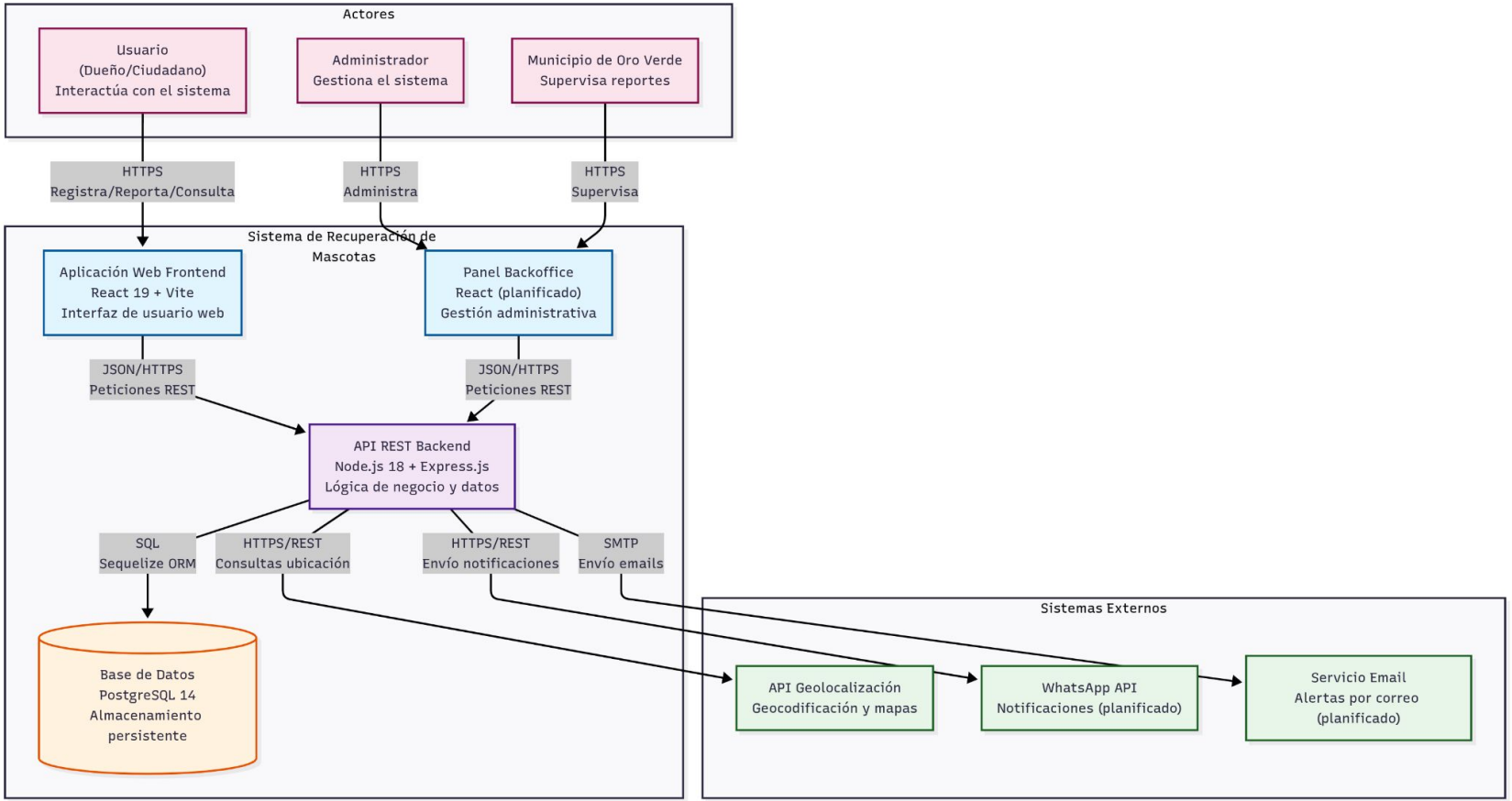


Diagrama de componentes

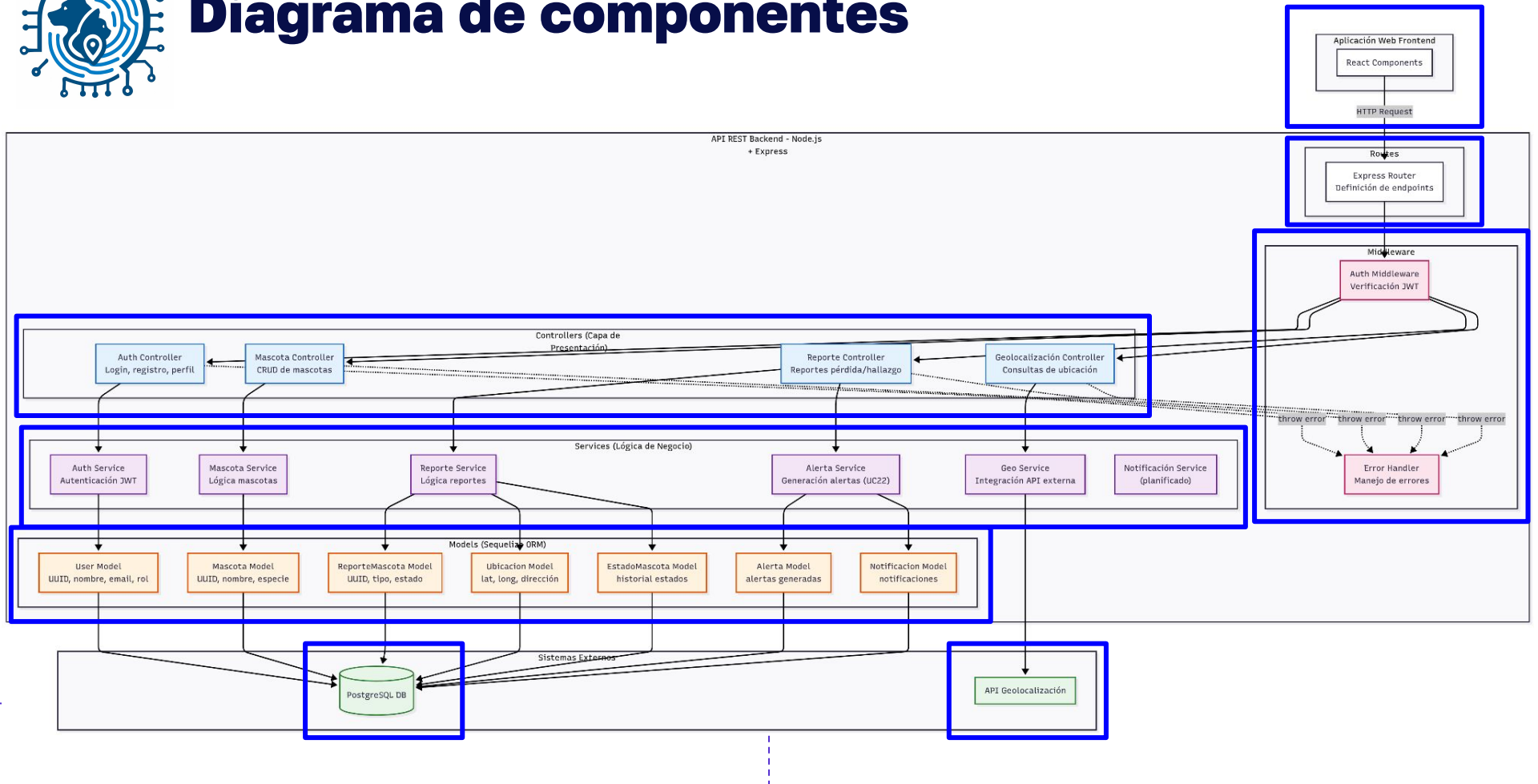


Diagrama de c ódigo

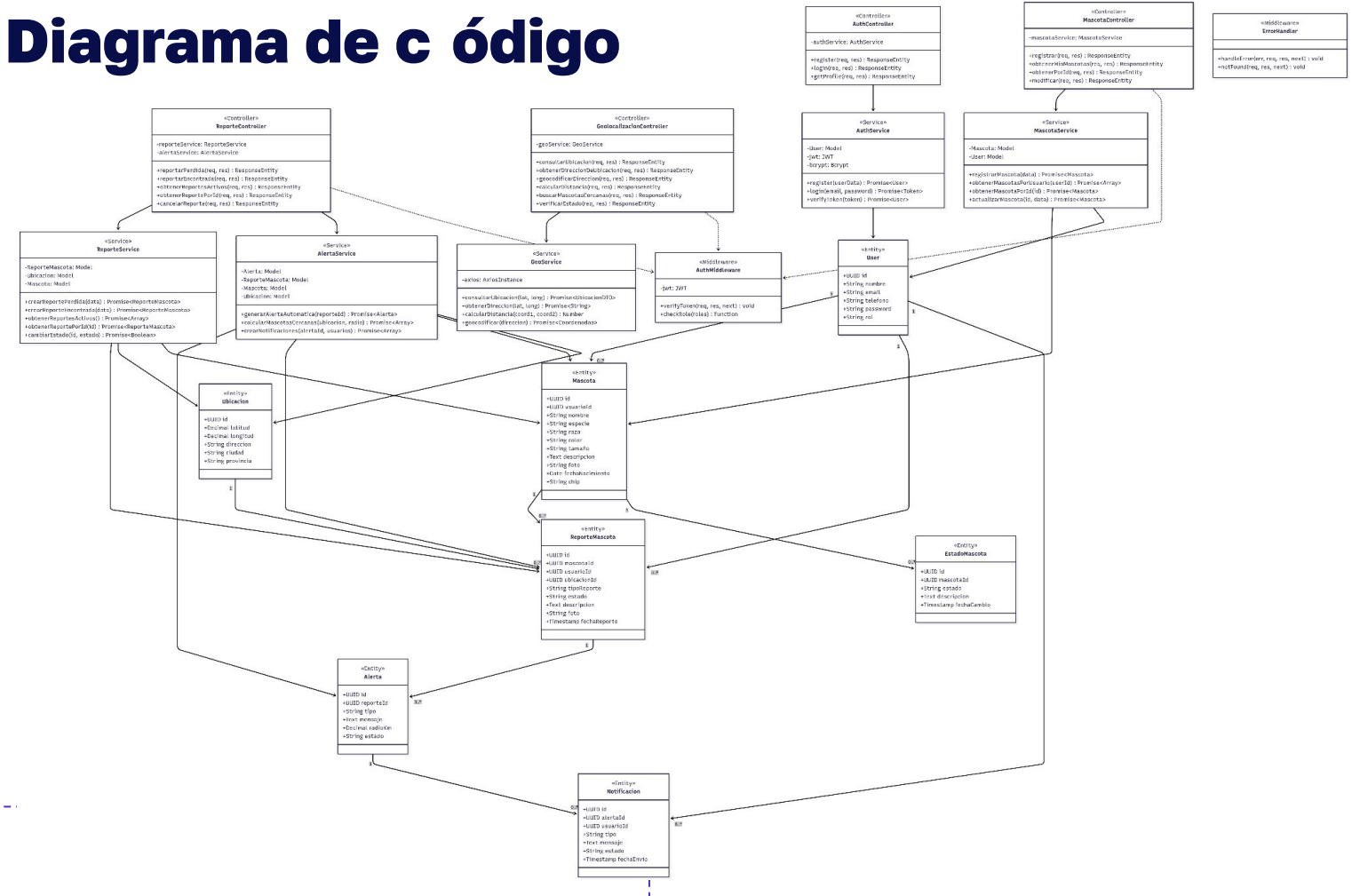
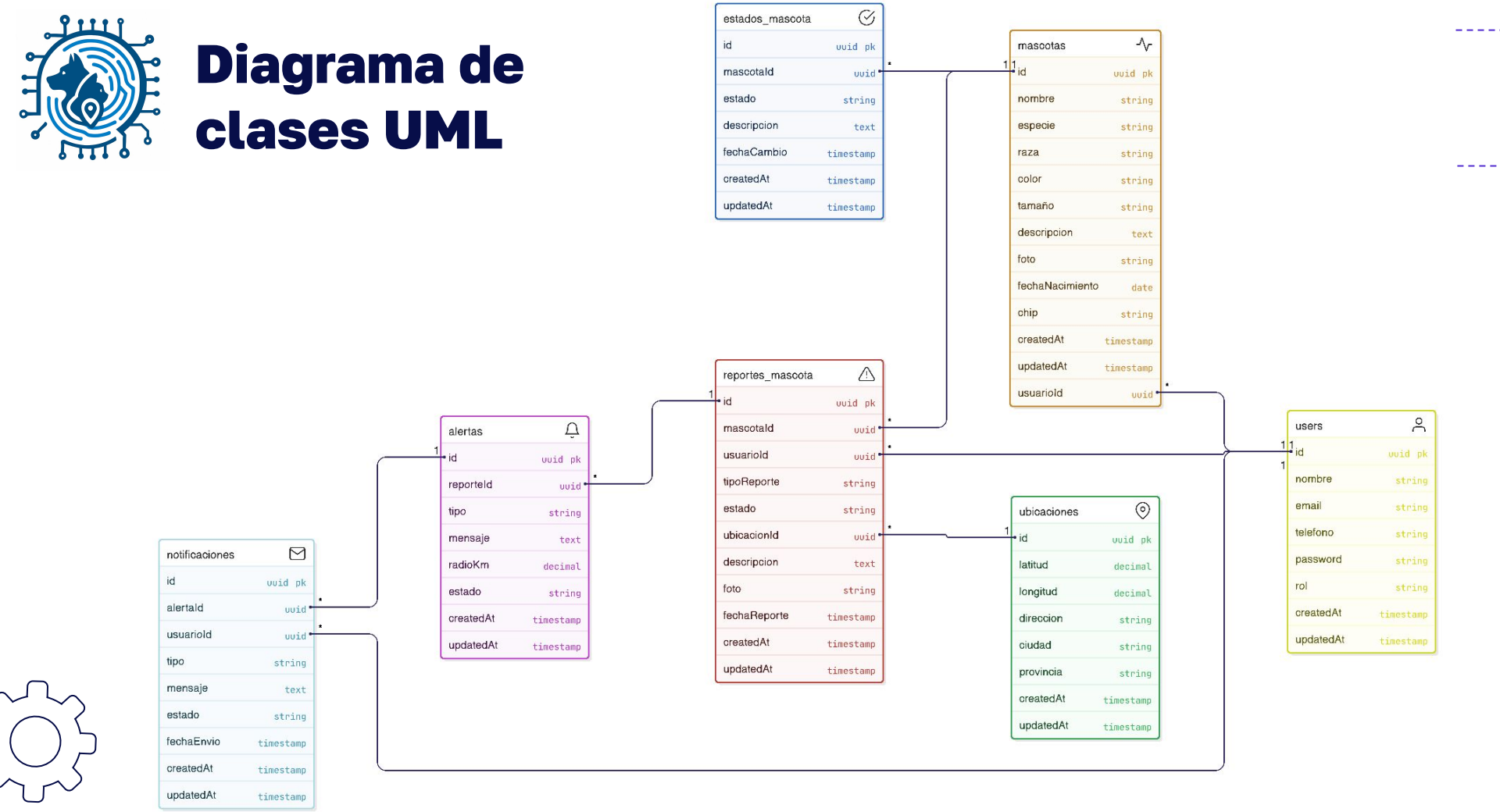


Diagrama de clases UML



Implementación

```
const { DataTypes } = require('sequelize');
const bcrypt = require('bcryptjs');
const { sequelize } = require('../config/database');

const User = sequelize.define('User', {
  id: {
    type: DataTypes.UUID,
    defaultValue: DataTypes.UUIDV4,
    primaryKey: true
  },
  nombre: {
    type: DataTypes.STRING,
    allowNull: false,
    validate: {
      notEmpty: {
        msg: 'El nombre es obligatorio'
      }
    }
  },
  apellido: {
    type: DataTypes.STRING,
    allowNull: false,
    validate: {
      notEmpty: {
        msg: 'El apellido es obligatorio'
      }
    }
  },
```

```
  tableName: 'users',
  timestamps: true,
  hooks: {
    // Hook para hashear la contraseña antes de crear
    beforeCreate: async (user) => {
      if (user.password) {
        const salt = await bcrypt.genSalt(10);
        user.password = await bcrypt.hash(user.password, salt);
      }
    },
    // Hook para hashear la contraseña antes de actualizar
    beforeUpdate: async (user) => {
      if (user.changed('password')) {
        const salt = await bcrypt.genSalt(10);
        user.password = await bcrypt.hash(user.password, salt);
      }
    }
  }
});

// Método de instancia para comparar contraseñas
User.prototype.compararPassword = async function(passwordIngresado) {
  return await bcrypt.compare(passwordIngresado, this.password);
};
```

Implementación

POST /api/auth/register - Registrar usuario
POST /api/auth/login - Iniciar sesión
GET /api/auth/me - Obtener usuario actual (protegido)
PUT /api/auth/perfil - Modificar perfil (protegido - UC4)

POST /api/mascotas - Registrar mascota (protegido - solo dueños - UC3)
GET /api/mascotas - Obtener mascotas del usuario (protegido)
GET /api/mascotas/:id - Obtener mascota por ID (protegido)
PUT /api/mascotas/:id - Modificar mascota (protegido - solo dueño - UC5)
GET /api/mascotas/:id/estado - Obtener estado de mascota (protegido - solo dueño - UC8)

POST /api/reportes/perdida - Reportar mascota perdida + Generar alerta automática (protegido - solo dueños - UC6 + UC22)
POST /api/reportes/encontrada - Reportar mascota encontrada (protegido - solo dueños - UC7)
GET /api/reportes/activos - Obtener reportes activos (público)
GET /api/reportes/mascota/:mascotaId/ubicacion - Obtener ubicación de mascota (protegido - solo dueño - UC9)

POST /api/geolocalizacion/consultar - UC23: Convierte coordenadas a dirección
GET /api/geolocalizacion/ubicacion/:ubicacionId - Obtiene dirección de ubicación almacenada
POST /api/geolocalizacion/geocodificar - Convierte dirección a coordenadas
POST /api/geolocalizacion/calcular-distancia - Calcula distancia entre dos puntos
POST /api/geolocalizacion/mascotas-cercanas - Busca mascotas perdidas cercanas
GET /api/geolocalizacion/estado - Verifica disponibilidad del servicio

Implementación

Demostración de funcionalidad de endpoints con:

Bruno

Reinventing the API Client

Registrar una mascota

POST http://localhost:3000/api/mascotas

Params Body • Headers 2 Auth • Vars Script Assert

Tests Docs Settings JSON Prettify

```
{
  "nombre": "Bruno",
  "raza": "Golden Retriever",
  "especie": "Perro",
  "fotoUrl": "https://example.com/bruno.jpg",
  "tamano": "mediano",
  "colores": "Dorado",
  "observaciones": "Tiene un collar rojo, es muy amistoso"
}
```

Response Headers 8 Timeline Tests

201 Created 2.4s 451B

```
{
  "success": true,
  "message": "Mascota registrada exitosamente",
  "data": {
    "id": "ee2f7cf1-b9c5-46a7-946d-db2127f381d0",
    "nombre": "Bruno",
    "raza": "Golden Retriever",
    "especie": "Perro",
    "fotoUrl": "https://example.com/bruno.jpg",
    "tamano": "mediano",
    "colores": "Dorado",
    "chip": null,
    "observaciones": "Tiene un collar rojo, es muy amistoso",
    "usuarioId": "9772457f-b66b-4622-a9e0-29da840ee63d",
    "updatedAt": "2025-11-04T09:53:30.117Z",
    "createdAt": "2025-11-04T09:53:30.117Z"
  }
}
```

Reportar una mascota como perdida

POST http://localhost:3000/api/reportes/perdida

Params Body * Headers 2 Auth * Vars Script Assert

Tests Docs Settings JSON Prettify

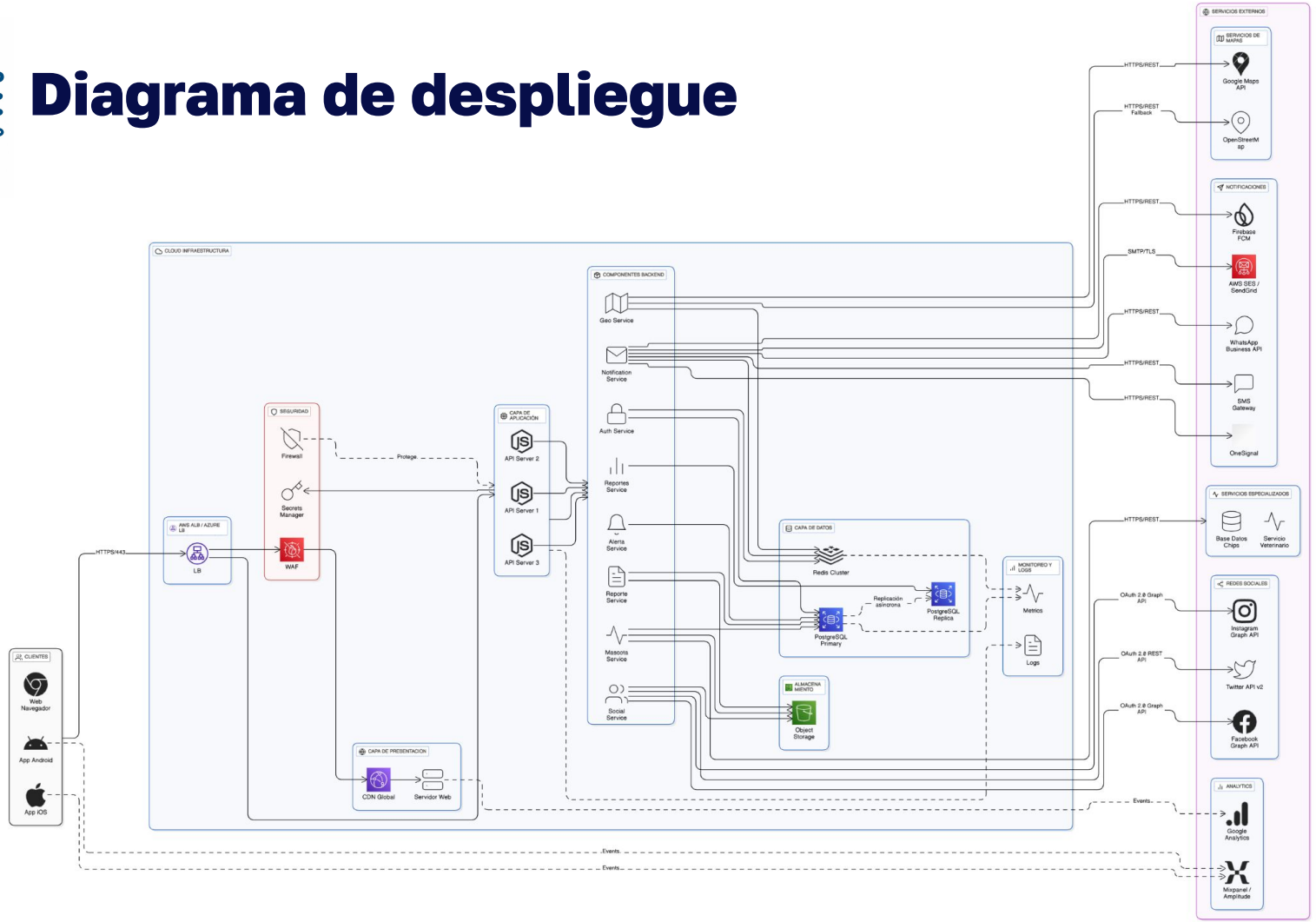
```
{
  "mascotaId": "376fcd79-5f95-40bc-bad0-d8bc966f2784",
  "latitud": -31.7833,
  "longitud": -60.5167,
  "descripcionLugar": "Polideportivo, Oro Verde",
  "descripcion": "Se perdió esta mañana cerca del polideportivo alrededor de las 9 AM. Última vez visto corriendo hacia el parque."
}
```

Response Headers 8 Timeline Tests

201 Created 5.14s 2.36KB


```
{
  "success": true,
  "message": "Mascota reportada como perdida y alerta generada exitosamente",
  "data": {
    "reporte": {
      "id": "571bd0f9-5b0e-4650-a263-d3009fa40518",
      "mascotaId": "376fcd79-5f95-40bc-bad0-d8bc966f2784",
      "usuarioId": "9772457f-b66b-4622-a9e0-29da840ee63d",
      "tipoReporte": "perdida",
      "estadoReporte": "activo",
      "ubicacionId": "ba290d7d-10b2-446b-89ac-d35dd74eb388",
      "descripcion": "Se perdió esta mañana cerca del polideportivo alrededor de las 9 AM. Última vez visto corriendo hacia el parque.",
      "fechaResolucion": null,
```

Diagrama de despliegue



Testing

```
// TEST 1: Reportar mascota perdida y generar alerta exitosamente
test('Debe reportar mascota perdida y generar alerta automáticamente', async () => {
  const reporteData = {
    mascotaId: mascotaId,
    latitud: -31.7833,
    longitud: -60.5167,
    descripcionLugar: 'Plaza de la Candelaria, Oro Verde',
    descripcion: 'Se perdió esta mañana cerca de la plaza'
  };

  const response = await request(app)
    .post('/api/reportes/perdida')
    .set('Authorization', `Bearer ${authToken}`)
    .send(reporteData);

  // Verificar respuesta HTTP
  expect(response.status).toBe(201);
  expect(response.body.success).toBe(true);
  expect(response.body.message).toContain('alerta generada');

  // Verificar que se creó el reporte
  expect(response.body.data.reporte).toBeDefined();
  expect(response.body.data.re  any  tipoReporte).toBe('perdida');
  expect(response.body.data.reporte.estadoReporte).toBe('activo');

  // Verificar que se creó la alerta
  expect(response.body.data.alerta).toBeDefined();
  expect(response.body.data.alerta.mensaje).toContain('MASCOTA PERDIDA');
  expect(response.body.data.alerta.mensaje).toContain('Max');
```

UC22

Generar alerta de mascota perdida

Caso de prueba 1:
Generar alerta de mascota perdida

```
// TEST 2: Rechazar reporte si faltan campos obligatorios
test('Debe rechazar reporte de mascota perdida si faltan campos obligatorios', async () =
  // Contar alertas antes del test
  const alertasAntes = await Alerta.count();

  const reporteIncompleto = {
    mascotaId: mascotaId,
    latitud: -31.7833
    // Falta longitud y descripcion
  };

  const response = await request(app)
    .post('/api/reportes/perdida')
    .set('Authorization', `Bearer ${authToken}`)
    .send(reporteIncompleto);

  // Verificar respuesta HTTP
  expect(response.status).toBe(400);
  expect(response.body.success).toBe(false);
  expect(response.body.message).toContain('campos obligatorios');

  // Verificar que NO se creó el reporte en base de datos
  const reportesCount = await ReporteMascota.count({
    where: { mascotaId: mascotaId, descripcion: null }
  });
  expect(reportesCount).toBe(0);

  // Verificar que NO se crearon alertas adicionales
  const alertasDespues = await Alerta.count();
  expect(alertasDespues).toBe(alertasAntes); // No debe haber alertas nuevas
});
```

UC22

Generar alerta de mascota perdida

Caso de prueba 2: Rechazar reporte si faltan campos obligatorios

```
// TEST 1: Consultar ubicación a sistema de geolocalización
test(' debería devolver una dirección válida cuando la API responde correctamente', async () =
  // Simulamos la respuesta de la API externa
  const mockResponse = {
    data: {
      direccion: 'Av. Los Eucaliptos 123, Oro Verde, Entre Ríos',
      latitud: -31.783,
      longitud: -60.516,
    },
  };

  // Mockeamos la respuesta de axios.post
  axios.post.mockResolvedValue(mockResponse);

  // Ejecutamos la función
  const resultado = await geoService.consultarUbicacion(-31.783, -60.516);

  // Verificamos el resultado
  expect(resultado.data.direccion).toBe('Av. Los Eucaliptos 123, Oro Verde, Entre Ríos');
  expect(resultado.data.latitud).toBeCloseTo(-31.783, 3);
  expect(resultado.data.longitud).toBeCloseTo(-60.516, 3);
});
```

UC23

Consultar ubicación a sistema de geolocalización

Caso de prueba 1:
Consultar ubicación a sistema de geolocalización

UC23

Consultar ubicación a sistema de geolocalización

Caso de prueba 2:
Manejo de error si la API de geolocalización falla

```
// TEST 2: Manejo de error si la API de geolocalización falla
test(' debería manejar correctamente un error si la API de geolocalización falla', async () =>
  // Simulamos un fallo de la API
  axios.post.mockRejectedValue(new Error('Servicio no disponible'));

  // Verificamos que la función lance una excepción
  await expect(geoService.consultarUbicacion(-31.783, -60.516))
    .rejects
    .toThrow('Servicio no disponible');
```

Conclusiones

⚙️ Principales decisiones

Arquitectura en capas: Cliente → Rutas → Controladores → Servicios → Modelos → BD

Selección de APIs REST estables (geolocalización y notificaciones)

Estandarización de nombres y estructuras en código, UML y documentación

⚠️ Dificultades y soluciones

Inconsistencia de nombres entre artefactos: se creó una nomenclatura común

Conflicto CommonJS / ES Modules: se eliminó "type": "module" del package.json

Ajustes en dependencias y configuración del entorno de Node

✨ Aprendizajes y mejoras futuras

Dominio del modelo C4 y diseño modular

Valor de la coherencia entre documentación y código en entornos colaborativos

Próximas mejoras:

- Integración con Amazon S3 / Cloudinary para almacenamiento de imágenes
- Incorporar chatbot con NLP para soporte automático 24/7

¿Preguntas?

Gracias por su atención