Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias.
Ingeniería de Sistemas y Computación.
Trabajo de Grado.

Prototipo de una plataforma de monitoreo geoespacial colaborativo para la detección y notificación de amenazas zoosanitarias urbanas en animales domésticos en la ciudad de Cali.

Laura Franco
Nestor Ortiz

Director: Ing. Juan Carlos Martinez Arias

Mayo 20 del 2026

Abstract

In a city such as Cali, where around 18 % of households have at least one pet, domestic animals are increasingly exposed to invisible health-related risks.

This project develops a prototype of a technological platform to monitor, geolocate, and report in real time the so-called Urban Zoosanitary Threats (UZT) in urban areas of Cali. Through active citizen participation and the integration of heterogeneous sources (simulated sensors, citizen reports, and official data), this tool enables the detection and categorization of threats from their emergence, the dissemination of alerts to the interested community, and the strengthening of decision-making processes aimed at animal welfare.

By combining geographic information systems, spatial databases, and a user-centered design, the prototype seeks not only to meet rigorous technical criteria, such as accuracy, latency, and reliability, but also key social indicators, such as usability and acceptance by pet guardians. Ultimately, the solution is positioned as a relevant component for public health and pet care in the city, providing a replicable model for urban environments in Latin America.

Keywords: Urban Zoosanitary Threats (UZT), collaborative geospatial monitoring, real-time geolocation, domestic animal health, citizen reporting, geographic information systems (GIS), spatial databases, citizen participation.

Resumen

En una ciudad como Cali, donde alrededor del 18 % de los hogares tienen al menos una mascota, se presenta una creciente exposición de los animales domésticos a riesgos sanitarios invisibles.

Este proyecto desarrolla un prototipo de plataforma tecnológica para monitorear, geolocalizar y reportar en tiempo real las llamadas Amenazas Zoosanitarias Urbanas (AZU) en zonas urbanas de Cali. Gracias a la participación activa de la ciudadanía y a la integración de fuentes heterogéneas (sensores simulados, reportes ciudadanos y datos oficiales), esta herramienta permite detectar y categorizar amenazas desde su aparición, difundir alertas a la comunidad interesada y potenciar la toma de decisiones orientadas al bienestar animal.

Al combinar sistemas de información geográfica, bases de datos espaciales y un diseño centrado en el usuario, el prototipo busca no solo cumplir con criterios técnicos rigurosos, tales como precisión, latencia y confiabilidad, sino también con indicadores sociales clave, como usabilidad y aceptación por parte de los tutores. En última instancia, la solución se posiciona como un componente relevante para la salud pública y el cuidado de las mascotas en la ciudad, aportando un modelo replicable para entornos urbanos en Latinoamérica.

Palabras clave: Amenazas zoosanitarias urbanas (AZU), monitoreo geoespacial colaborativo, geolocalización en tiempo real, salud de animales domésticos, reporte ciudadano, sistemas de información geográfica (GIS), bases de datos espaciales, participación ciudadana.

Índice general

1. Descripción del Problema **17**
- 1.1. Planteamiento del Problema . . . 17
  - 1.1.1. Formulación del Problema . . . 18
  - 1.1.2. Sistematización . . . 18
- 1.2. Objetivos . . . 18
  - 1.2.1. Objetivo General . . . 18
  - 1.2.2. Objetivos Específicos . . . 18
- 1.3. Justificación . . . 19
  - 1.3.1. Utilidad . . . 19
  - 1.3.2. Impacto . . . 19
  - 1.3.3. Viabilidad . . . 19
- 1.4. Alcance y Delimitación del Proyecto . . . 20
  - 1.4.1. Entregables . . . 20

2. Marco de Referencia **21**
- 2.1. Marco Teórico . . . 21
- 2.2. Conceptos Relevantes . . . 21
  - 2.2.1. Conceptos relevantes del dominio . . . 21
  - 2.2.2. Conceptos relevantes de la tecnología . . . 22
- 2.3. Trabajos relacionados . . . 23

3. Etapa 1: Análisis y Clasificación de Amenazas Zoosanitarias Y Zoosociales Urbanas en la ciudad de Cali **25**
- 3.1. Revisión documental y bibliográfica . . . 25
  - 3.1.1. Ampliación del concepto de amenazas urbanas . . . 26
- 3.2. Recolección de datos institucionales . . . 26
- 3.3. Validación con actores locales . . . 28
- 3.4. Análisis de soluciones existentes . . . 29
- 3.5. Selección y priorización de amenazas . . . 30
- 3.6. Conclusión del Capítulo . . . 33

4. Etapa 2: Análisis y Diseño del Sistema **35**
- 4.1. Diagnóstico de flujos de información existentes . . . 35
- 4.2. Análisis de requerimientos del sistema . . . 37
  - 4.2.1. Clasificación general de requerimientos . . . 37
  - 4.2.2. Modelado de casos de uso . . . 38
  - 4.2.3. Requerimientos funcionales . . . 39

4.2.4. Requerimientos no funcionales . . . 46
4.2.5. Requerimientos de interfaz . . . 49
4.2.6. Consideraciones de diseño a partir de los requerimientos . . . 51
4.2.7. Diseño de la arquitectura del sistema . . . 51
4.2.8. Justificación del uso del modelo C4 . . . 55
4.2.9. Justificación del enfoque monolítico . . . 56
4.2.10. Relación con los requerimientos del sistema . . . 56
4.3. Modelo de datos con soporte de geolocalización . . . 57
4.3.1. Enfoque general del modelo . . . 57
4.3.2. Representación del modelo relacional . . . 57
4.3.3. Representación geoespacial . . . 58
4.3.4. Normalización y catálogos . . . 59
4.3.5. Relaciones principales . . . 59
4.3.6. Validación colaborativa . . . 59
4.3.7. Persistencia local y conectividad . . . 59
4.3.8. Extensibilidad . . . 59
4.4. Estrategia de participación ciudadana . . . 59
4.4.1. Mecanismos de reporte colaborativo . . . 60
4.4.2. Validación de la información . . . 60
4.4.3. Importancia de la interacción social en la validación . . . 60
4.4.4. Beneficios del enfoque participativo . . . 61
4.4.5. Rol del usuario en el sistema . . . 61
4.4.6. Limitaciones y consideraciones . . . 61
4.4.7. Integración con el sistema propuesto . . . 62
4.4.8. Flujo de comunicación y monitoreo . . . 62
4.5. Diseño de interfaz y experiencia de usuario (UX/UI) . . . 63
4.5.1. Principios de diseño . . . 63
4.5.2. Estructura de la interfaz . . . 64
4.5.3. Diseño del flujo de interacción . . . 64
4.5.4. Prototipos de interfaz . . . 64
4.5.5. Modo claro y modo oscuro . . . 66
4.5.6. Consideraciones de usabilidad . . . 66
4.5.7. Relación con los requerimientos del sistema . . . 66
4.6. Conclusión del capítulo . . . 67

5. Etapa 3: Construcción e Implementación del Prototipo 69
5.1. Persistencia de datos y arquitectura híbrida . . . 69
5.2. Desarrollo del sistema de visualización geográfica . . . 70
5.3. Desarrollo del sistema de rutas y cálculo de trayectorias . . . 71
5.4. Implementación del módulo de mascotas . . . 72
5.5. Implementación del módulo de rutas . . . 73

5.6. Implementación del módulo de reporte colaborativo . . . 74
5.7. Selección de tecnologías y herramientas . . . 75
5.8. Integración backend y consumo de servicios remotos . . . 76
5.9. Integración de módulos y consolidación progresiva del prototipo . . . 76
5.10. Pruebas y validación parcial del sistema . . . 80
5.11. Expansión del alcance y evolución hacia problemáticas zoosociales . . . 80
5.12. Análisis comparativo entre el diseño arquitectónico y la implementación desarrollada 82
5.13. Conclusión del capítulo . . . 83

6. Etapa 4: Validación y Evaluación del Prototipo 85
6.1. Estrategia general de pruebas . . . 86
6.2. Ambiente de pruebas . . . 87
6.3. Ejecución de pruebas por fases . . . 89
6.4. Limitaciones de la evaluación . . . 92

7. Conclusiones y trabajos futuros 95
7.1. Conclusiones generales . . . 95
7.2. Trabajos futuros . . . 96
7.2.1. Fortalecimiento del enfoque de bienestar animal . . . 96
7.2.2. Implementación de rachas de paseo, hábitos saludables y comunidad . . . 96
7.2.3. Integración con mecanismos de alerta de la ciudad . . . 97

Bibliografía 99

A. Formularios de la Secretaría de Salud Pública de Cali 101
A.1. Formato de confidencialidad y protección de datos . . . 101
A.2. Formulario GC-1 para envío de proyectos . . . 106

B. Mockups diseñados en Figma 123
B.1. Catálogo de Interfaces . . . 123
B.1.1. Vista general de la aplicación . . . 123
B.1.2. Panel de vista del usuario sin iniciar sesión . . . 125
B.1.3. Panel de vista del usuario con sesión iniciada . . . 126
B.1.4. Panel de inicio de sesión . . . 127
B.1.5. Panel de registro de usuario . . . 128
B.1.6. Panel de gestión de mascotas . . . 129
B.1.7. Panel de gestión de rutas . . . 130
B.1.8. Panel de agregar mascota . . . 131
B.1.9. Panel de editar datos de mascota . . . 132
B.1.10. Panel de reportar un peligro . . . 133
B.1.11. Panel de iniciar viaje . . . 134
B.1.12. Panel de detalles de un peligro . . . 135

B.1.13. Panel de iniciar viaje con ruta seleccionada . . . 136
B.1.14. Panel de comentarios de un peligro . . . 137
B.1.15. Panel de viaje en curso con ruta evitando un peligro . . . 138
B.1.16. Panel de viaje completado . . . 139
B.1.17. Panel de selección de reporte . . . 140

C. Documento de Especificación de Requerimientos de Software 141
C.1. Especificación de Requerimientos de Software (SRS) . . . 141

D. Plan de Pruebas A4MPets 171

E. Casos de Prueba A4MPets 189

F. Reporte F0 221

G. Reporte F1 229

H. Reporte F1 - Cobertura 233

I. Reporte F2 235

J. Reporte F2 - HTML de ejecución 241
J.1. Reporte HTML de ejecución F2 . . . 241

K. Reporte F3 243

L. Reporte F3 - Cobertura 249

M. Reporte F4 251

N. Reporte F5 257

Ñ. Reporte F5 - Playwright 265

O. Reporte F6 267

P. Reporte F6 - Evidencias 269
P.1. Evidencias de ejecución de F6 . . . 269
P.1.1. Validación de mensajes de error en autenticación . . . 270
P.1.2. Validación de estados vacíos . . . 271
P.1.3. Creación y visualización de peligros . . . 273
P.1.4. Validación de permisos de ubicación . . . 274
P.1.5. Caracteres especiales y comentarios . . . 276
P.1.6. Gestión de mascotas . . . 277
P.1.7. Diferenciación visual de niveles de peligro . . . 278

P.1.8. Validación de fechas 279
P.1.9. Manejo de errores y datos completos 280
P.1.10. Legibilidad y contraste visual 281
P.1.11. Carga de archivos y panel de perfil 282

Q. Reporte F7 **283**

R. Reporte F8 - Regresión y Cierre del Proceso de Pruebas **293**

S. Consolidado de evidencias QA F0-F7 **301**

Introducción

En los entornos urbanos contemporáneos, la salud de los animales domésticos se ve comprometida por diversos factores ambientales y sanitarios que pasan inadvertidos en la cotidianidad. En ciudades como Cali, donde la densidad de mascotas supera las 200.000 registradas oficialmente, la identificación temprana de riesgos zoonóticos y ambientales se ha convertido en un reto tanto técnico como social. La falta de información centralizada y de mecanismos de alerta en tiempo real dificulta la prevención de brotes y la gestión oportuna de amenazas zoosanitarias urbanas (AZU).

En este contexto, el presente proyecto desarrolló un prototipo de plataforma de monitoreo geoespacial colaborativo que permita detectar, categorizar y difundir alertas sobre amenazas zoosanitarias que afecten a los animales domésticos de la ciudad. Al integrar tecnologías de geolocalización, sistemas de información geográfica (GIS) y bases de datos espaciales con un enfoque de participación ciudadana, esta iniciativa busca fortalecer la comunicación entre la comunidad y las entidades encargadas del control sanitario, promoviendo un entorno urbano más seguro y saludable para las mascotas y sus tutores.

Capítulo 1

Descripción del Problema

1.1. Planteamiento del Problema

En Colombia se estima que aproximadamente el 60 % de los hogares conviven con al menos un perro; esto sin contar otras especies de animales domésticos (e incluso exóticos) que se tienen como compañía y requieren cuidados y protección de amenazas conocidas y desconocidas para sus tutores. Según Infobae [1], esta estimación llevada a cabo por el Departamento Administrativo Nacional de Estadística (DANE), expresa que, de dicho porcentaje, Cali se muestra como la segunda ciudad del país con más presencia de mascotas en los hogares.

Sin embargo, en la urbe y de manera específica en Cali, las mascotas encuentran en el espacio público muchos más peligros de los que una persona común pueda percibir y anticipar, muchos de los cuales tienen un componente ambiental que potencia su impacto en la salud animal. Entre estas amenazas ambientales y zoosanitarias se incluyen: la aparición de especies invasoras como el caracol gigante africano (Achatina fulica), cuya presencia en distintas zonas de la ciudad puede representar una transmisión de parásitos que afectan en especial a un grupo selecto en el cual se incluyen las mascotas, cuyo contacto con superficies contaminadas representa un riesgo para la salud y el bienestar de estas [2]; la existencia de puntos críticos de contagio de enfermedades zoonóticas en la ciudad, cuya transmisión puede variar según el agente involucrado. Por ejemplo, en el caso particular de la Leptospirosis (cuya infección puede afectar no solo a los animales, sino también a los humanos), se ha determinado que su propagación se puede dar a través del contacto directo con orina de animales infectados o de manera indirecta luego de exponerse a aguas estancadas, suelos húmedos y superficies contaminadas; sin embargo, esto no es un hecho aislado a casos de leptospirosis, sino que varía dependiendo de las infecciones y puede representar un riesgo incluso mayor según el foco de infección y sus formas de transmisión. Estos componentes pueden verse amplificados en entornos urbanos debido a los factores ambientales que involucran acumulación de basuras, disposición de residuos y aguas residuales, que constituyen focos infecciosos para la comunidad en general y de manera especial para las mascotas [3].

A pesar de que se manifiesta como una problemática compleja, la información que se oferta no suele ser centralizada ni modificada o dada en tiempo real para los tutores de animales. La falta de estos mecanismos convierte la búsqueda de alertas e identificación de riesgos en un juego de azar, que solo logra prolongar la brecha en la protección animal, y, en consecuencia, en la salud pública.

En este sentido, se consideraó pertinente desarrollar el prototipo de una herramienta tecnológica que permitiera monitorear, identificar y reportar peligros presentes en la ciudad para los animales domésticos, de tal manera que los tutores pudiesen tomar decisiones informadas al momento de pasear, movilizar y proteger a sus mascotas.

1.1.1. Formulación del Problema

En la ciudad de Cali existe una creciente preocupación por el bienestar animal y además la proliferación de riesgos urbanos que intervienen en el mismo (como la presencia de caracoles africanos [2], focos de enfermedades zoonóticas como la leptospirosis [3] y la exposición a contaminantes ambientales [3]). Sin embargo, en la actualidad no se dispone de una herramienta tecnológica que permita alertar y notificar a los tutores sobre estos peligros en tiempo real. Ante este contexto, surge la siguiente pregunta de investigación:

¿Cómo desarrollar una herramienta tecnológica que permita monitorear, geolocalizar y reportar en tiempo real la presencia de amenazas zoosanitarias urbanas (AZU) en la zona urbana de la ciudad de Cali, con el fin de proteger la salud de los animales domésticos para contribuir consecuentemente al bienestar de los mismos?

1.1.2. Sistematización

Para resolver esta pregunta, se deben tener en cuenta algunas subpreguntas:

- ¿Cómo detectar, clasificar y verificar las amenazas zoosanitarias urbanas a las que se enfrentan los animales domésticos en la ciudad de Cali a partir de fuentes heterogéneas?
- ¿Qué limitaciones se presentan en el acceso a la información y la alerta sobre riesgos a la hora de buscar prevenir los riesgos zoosanitarios a los que se enfrentan las mascotas?
- ¿Qué características resulta importante tener en cuenta a la hora de modelar un mecanismo colaborativo y en tiempo real de información y alerta sobre posibles casos de contagio zoonótico y/o focos de infección?
- ¿Cómo evaluar técnica y socialmente la solución para asegurar su precisión, rapidez y aceptación por parte de los tutores?

1.2. Objetivos

1.2.1. Objetivo General

Desarrollar un prototipo de una herramienta tecnológica que permita monitorear, geolocalizar y reportar en tiempo real la presencia de amenazas zoosanitarias urbanas (AZU) en la zona urbana de la ciudad de Cali, con la finalidad de proteger la salud de los animales domésticos y contribuir a su bienestar mediante el reporte colaborativo de información y la detección temprana de alertas.

1.2.2. Objetivos Específicos

1. Identificar y clasificar las amenazas zoosanitarias urbanas que afectan a los animales domésticos en la ciudad de Cali, a partir de fuentes heterogéneas como sensores simulados, reportes ciudadanos y datos oficiales.

2. Analizar las limitaciones actuales en el acceso a la información y en los mecanismos de alerta sobre riesgos, con el fin de proponer una estrategia de comunicación y alerta temprana frente a las amenazas zoosanitarias urbanas (AZU).

3. Diseñar e implementar un prototipo tecnológico basado en un modelo colaborativo y en tiempo real, que integre funciones de monitoreo, geolocalización y notificación de amenazas zoosanitarias urbanas (AZU).

4. Evaluar el prototipo mediante un proceso de aseguramiento de la calidad (QA) que incluya la ejecución de pruebas funcionales, de integración y de validación de requisitos, con el fin de medir su desempeño en términos de precisión, confiabilidad y cumplimiento de especificaciones técnicas.

1.3. Justificación

1.3.1. Utilidad

Debido a la creciente proliferación de puntos críticos de riesgos zoosanitarios en la ciudad de Cali, informar a los tutores sobre las amenazas sanitarias a las que están expuestas sus mascotas se ha convertido si bien en una necesidad, también en una problemática que surge como el reflejo de una sociedad que brinda información no centralizada ni modificada o dada en tiempo real, lo cual dificulta la toma de decisiones oportunas. En este sentido, el desarrollo de un sistema que permita monitorear, identificar y reportar amenazas zoosanitarias urbanas (AZU) constituye una herramienta de gran utilidad para los tutores de mascotas, al brindarles la posibilidad de acceder a información confiable, tomar decisiones informadas y adoptar medidas preventivas que favorezcan el bienestar animal.

1.3.2. Impacto

La implementación de esta plataforma tendrá un impacto directo en la salud y calidad de vida de los animales domésticos, al disminuir su exposición a focos de riesgo y enfermedades zoonóticas. A nivel social, fomentará la participación ciudadana mediante el reporte colaborativo, fortaleciendo así la red comunitaria para la detección temprana de amenazas. Además, se proyecta como una contribución innovadora en el campo de los sistemas de monitoreo geoespacial, con potencial de convertirse en un modelo replicable y escalable en otras ciudades que enfrentan problemáticas similares.

1.3.3. Viabilidad

La viabilidad del proyecto se sustenta en la disponibilidad de tecnologías maduras como bases de datos geoespaciales, sistemas de información geográfica y herramientas de geolocalización que permiten gestionar de forma eficiente información en tiempo real. A esto se suma la posibilidad de acceder a datos publicados por entidades locales como el Centro de Zoonosis de Cali, la Secretaría

de Salud Pública y la CVC, los cuales incluyen reportes sobre especies invasoras, enfermedades zoonóticas y riesgos ambientales. El uso de estas fuentes de información oficial y de carácter público podría fortalecer (además de la confiabilidad de los datos) las probabilidades de que la plataforma sea acogida como apoyo en la formulación de políticas de salud pública y bienestar animal.

1.4. Alcance y Delimitación del Proyecto

Este proyecto tendrá como alcance el diseño y desarrollo de un prototipo tecnológico orientado a la gestión de información sobre la clasificación de amenazas zoosanitarias urbanas (AZU) en la ciudad de Cali. Este prototipo se enfocará en la clasificación y reporte de riesgos a partir de dos fuentes principales: (i) datos publicados por entidades gubernamentales y de carácter público, como el Centro de Zoonosis, la Secretaría de Salud y la CVC, y (ii) un modelo de participación ciudadana mediante el reporte colaborativo en tiempo real.

El sistema estará limitado al casco urbano de la ciudad de Cali y se centrará en las especies domésticas de mayor presencia en los hogares: perros y gatos. Su propósito es proporcionar alertas tempranas sobre riesgos zoonóticos, especies invasoras y focos de infección, promoviendo así la sensibilización de tutores y comunidades frente al impacto de las AZU. El prototipo incluirá funcionalidades de monitoreo, geolocalización, clasificación de riesgos y notificación en tiempo real, accesibles a través de un portal web y una aplicación móvil.

1.4.1. Entregables

- Documento de especificación de requerimientos (SRS): Definición detallada de los requerimientos funcionales, no funcionales y de arquitectura del sistema, elaborado bajo una metodología de ingeniería de software (como RUP), que garantice trazabilidad y claridad en el desarrollo del prototipo.
- Prototipo funcional del sistema de detección de amenazas zoosanitarias urbanas: Desarrollo de un sistema que integre un portal web y una aplicación móvil, a través de los cuales los usuarios podrán consultar información georreferenciada, reportar eventos, validar datos y recibir alertas en tiempo real.

Capítulo 2

Marco de Referencia

2.1. Marco Teórico

En la actualidad, las problemáticas relacionadas con la salud y el bienestar de los animales domésticos requieren ser abordadas desde una perspectiva holística con la cual sea posible fusionar la comprensión de riesgos ambientales y sanitarios que aquejan a las mascotas y el enfoque tecnológico propio de la era digital. La presencia de especies invasoras, puntos críticos de contagio y transmisión de enfermedades y parásitos y la exposición a superficies contaminadas se reflejan como un entorno poco propicio para una ciudad como Cali, cuya gran concentración de animales domésticos y la creciente preocupación por su bienestar demanda soluciones que permitan identificar, monitorear y representar esta posible exposición a riesgos en el espacio geográfico.

En este sentido, el presente trabajo pretende proponer el diseño de un prototipo de plataforma de monitoreo geoespacial y colaborativo, cuya intención está dirigida a detectar y prevenir AZU que afecten a los animales domésticos en la ciudad de Cali. Para poder sustentar de modo teórico y metodológico esta propuesta, se hace necesario constituir de manera formal un marco conceptual que articule de manera efectiva los principales componentes de la investigación.

Este marco se constituye desde conceptos del dominio de la aplicación hasta las tecnologías que harán posible el diseño funcional de la solución. Dentro del dominio del marco se abordan principios relacionados a la geolocalización, los sistemas de información geográfica (GIS), la identificación de amenazas zoosanitarias urbanas (AZU), y la dinámica de participación ciudadana en el proceso de generar reportes colaborativos. Por otro lado, y desde la escena tecnológica, se consideran elementos como bases de datos para la gestión y el almacenamiento de información, la ingeniería de requisitos como fundamento del desarrollo de software.

Finalmente, se incluye una revisión de trabajos relacionados, la cual constituye una parte importante para efectos del diagnóstico de aportes, limitaciones y oportunidades de mejora que permitan orientar el desarrollo de la propuesta a partir de los antecedentes en el campo de plataformas de monitoreo geoespacial y sistemas de alerta temprana (SAT).

2.2. Conceptos Relevantes

2.2.1. Conceptos relevantes del dominio

Amenazas Zoosanitarias Urbanas: Una amenaza zoosanitaria urbana se entiende como un conjunto de factores biológicos y ambientales que representan un riesgo para la salud de los animales domésticos que habitan en un determinado entorno urbano. En este proyecto, se consideran las

enfermedades reportadas por el Centro de Zoonosis de Cali [2], así como la presencia de especies foráneas disruptivas incluidas en el inventario de especies invasoras del Valle del Cauca de la CVC [3]. Estas amenazas se intensifican en condiciones de acumulación de basuras, aguas estancadas y deficiencias en la gestión de residuos, que constituyen focos de riesgo relevantes para el estudio.

Geolocalización: La geolocalización se define como un proceso que permite determinar la ubicación geográfica de un objeto, persona o dispositivo en la superficie terrestre, mediante coordenadas de latitud y longitud obtenidas a través de sistemas satelitales, redes móviles o direcciones IP [4]. En el contexto del proyecto, la geolocalización constituye la base para ubicar espacialmente la presencia de amenazas zoosanitarias urbanas y generar información precisa para tutores y entidades.

Sistemas de Información Geográfica (GIS): Los sistemas de información geográfica están diseñados para la gestión eficiente de datos geoespaciales [2]. En este proyecto se conciben como herramientas que permiten visualizar mapas, representar coordenadas y analizar datos espaciales, con el fin de identificar zonas de riesgo y caracterizar las condiciones ambientales asociadas a las amenazas zoosanitarias urbanas.

Participación Ciudadana y Reporte Colaborativo: La generación y validación de información por parte de múltiples ciudadanos a través de tecnologías digitales, cuando promueve la producción de conocimiento, se denomina crowdsourcing [5]. Goodchild introduce el concepto de "sensores sociales" para referirse a ciudadanos que generan datos espaciales [6], mientras que Benkler señala que la confianza en la información aumenta cuando proviene de pares o semejantes [7]. Estos principios fundamentan la dimensión colaborativa de la propuesta.

2.2.2. Conceptos relevantes de la tecnología

Bases de datos: Una base de datos es una recopilación organizada de información o datos estructurados, que normalmente se almacena de forma electrónica en un sistema informático y está gestionada por un sistema de gestión de bases de datos (DBMS) [8]. En este proyecto, las bases de datos resultan esenciales para la recolección, almacenamiento y gestión de los datos generados por usuarios y entidades.

Ingeniería de Requisitos: El proceso de ingeniería de requisitos constituye una herramienta clave para sistematizar el desarrollo de software, de modo que cumpla con los objetivos técnicos y funcionales planteados desde la etapa inicial. Según EBSCO [9], este proceso comprende varias etapas interrelacionadas:

- Incepción: definición del producto, objetivos y público.
- Elicitación: recopilación de expectativas de los interesados.
- Elaboración: modelado del comportamiento del sistema a partir de la información disponible.

- Negociación: priorización de requisitos y resolución de conflictos.
- Especificación: consolidación en un documento formal.
- Validación: aseguramiento de la calidad de los entregables.
- Gestión: análisis de cambios y proyecciones del proyecto.

2.3. Trabajos relacionados

Mosquito Alert: El proyecto Mosquito Alert se planteó como una iniciativa en España que permitió gestionar reportes de avistamientos del mosquito tigre (Aedes albopictus) mediante un aplicativo móvil. Este sistema implementó, entre otras cosas, un sistema de geolocalización aportado por los usuarios. Se valieron de dicho enfoque para demostrar que esta estrategia sería más económica y escalable que los métodos tradicionales, además de permitir la detección temprana en zonas aún no clasificadas oficialmente como riesgosas en el contexto [10].

The Tick App: La aplicación The Tick App, desarrollada en Estados Unidos para investigar la exposición de la población a garrapatas y la enfermedad de Lyme, recopiló información importante a través de encuestas diarias y reportes de los usuarios, lo cual permitió que se recopilaran datos de geolocalización, factores de riesgo y patrones de contagio en el entorno [11].

OIE/WAHIS - Tableau: En Kuwait, se realizó un sistema de vigilancia epidemiológica animal, que recopilaba datos históricos reportados a la OIE/WAHIS. Si bien en la actualidad este es más un visor de información oficial, los autores señalan al sistema como un buen comienzo para evolucionar hacia un sistema en tiempo real, que tenga este carácter colaborativo que se plantea en los demás proyectos [12].

En conjunto, estos trabajos relacionados aportan lecciones valiosas para el presente proyecto. *Mosquito Alert* y *The Tick App* demuestran la efectividad del reporte ciudadano y la ciencia colaborativa como mecanismos para la detección temprana de riesgos sanitarios, mientras que el sistema OIE/WAHIS resalta la importancia de contar con información oficial centralizada como soporte a la vigilancia epidemiológica. Sin embargo, a diferencia de estas iniciativas, el prototipo aquí propuesto se enfoca en un contexto urbano específico (la ciudad de Cali), en el cual integra tanto datos públicos institucionales como reportes colaborativos de la ciudadanía, orientados particularmente a la identificación de amenazas zoosanitarias urbanas que afectan a animales domésticos como perros y gatos. Este enfoque local, participativo y en tiempo real constituye su principal aporte diferencial, al combinar la confiabilidad de fuentes oficiales con la inmediatez del reporte ciudadano para la protección del bienestar animal.

Capítulo 3

Etapa 1: Análisis y Clasificación de Amenazas Zoosanitarias Y Zoosociales Urbanas en la ciudad de Cali

---

Introducción

En esta etapa se presentó el proceso de investigación y diagnóstico realizado para identificar las principales amenazas zoosanitarias urbanas (AZU) en la ciudad de Cali. Este análisis constituyó uno de los pilares para el desarrollo de la solución propuesta, ya que permitió no solo la comprensión del contexto de la problemática inicial, sino también la elaboración de un diagnóstico epidemiológico, ambiental y social que sustentó la estructura del plan de acción del proyecto.

Durante este proceso de análisis, se identificó que, si bien las amenazas zoosanitarias urbanas permiten clasificar de manera acertada los riesgos asociados a enfermedades, vectores y condiciones higiénico-sanitarias, existen otras problemáticas cuya falta de visibilización afecta significativamente el bienestar de los animales domésticos y cuya caracterización no es abordada desde el ámbito sanitario. Entre estas se encuentran situaciones como el robo, la desaparición y el abandono de mascotas, además de otras dinámicas de carácter más social que sanitario.

En consecuencia, se adoptó una ampliación conceptual del análisis para representar de manera más completa los riesgos presentes en el entorno urbano.

3.1. Revisión documental y bibliográfica

En esta sección se realizó el proceso de revisión documental y bibliográfica llevado a cabo con el fin de identificar y analizar las principales amenazas zoosanitarias urbanas (AZU) que afectan a los animales domésticos en la ciudad de Cali.

Para ello, se consultaron diversas fuentes de información, incluyendo literatura académica, artículos científicos, boletines epidemiológicos y reportes institucionales emitidos por entidades como la Secretaría de Salud, la Corporación Autónoma Regional del Valle del Cauca (CVC) y el Centro de Zoonosis de Cali. Estas fuentes permitieron obtener una visión fundamentada sobre la presencia de enfermedades zoonóticas, vectores, especies invasoras y condiciones ambientales que representan riesgos para la salud animal en contextos urbanos.

A partir de esta revisión, se logró identificar un conjunto de amenazas recurrentes en la ciudad, tales como enfermedades infecciosas de origen zoonótico, proliferación de especies invasoras como

el caracol africano (Achatina fulica), y la existencia de condiciones ambientales propicias para la propagación de vectores, como la acumulación de residuos y la presencia de aguas estancadas.

Adicionalmente, el análisis bibliográfico permitió comprender la relación entre factores ambientales, sociales y sanitarios en la generación de riesgos para los animales domésticos, evidenciando la necesidad de abordar estas problemáticas desde un enfoque integral que trascendiera lo estrictamente sanitario.

3.1.1. Ampliación del concepto de amenazas urbanas

A partir de la revisión documental y del análisis del contexto local, se evidenció que el concepto de amenazas zoosanitarias urbanas (AZU), aunque resulta adecuado para describir riesgos asociados a enfermedades, vectores y condiciones higiénico-sanitarias, no logra abarcar la totalidad de problemáticas que afectan el bienestar de los animales domésticos en entornos urbanos.

En particular, se identificaron situaciones que, si bien no corresponden directamente al ámbito sanitario, representan riesgos significativos para los animales, tales como el robo, la desaparición, el abandono y otras dinámicas de carácter social que inciden en su integridad y calidad de vida. Estas problemáticas, al no estar contempladas dentro de las categorías tradicionales de vigilancia zoosanitaria, tienden a carecer de visibilidad en los sistemas de información existentes.

De esta manera, se estableció una distinción entre las amenazas zoosanitarias urbanas, relacionadas con enfermedades, vectores y condiciones ambientales de riesgo, y otras amenazas de carácter zoosocial, asociadas a la interacción humana y a dinámicas sociales que impactan negativamente el bienestar animal.

Si bien esta ampliación conceptual se sustentó principalmente en la revisión documental y el análisis del contexto urbano, el proceso de contraste con actores institucionales no logró completarse en su totalidad, debido a los tiempos requeridos para la aprobación de acceso a información por parte de las entidades oficiales, los cuales excedieron el periodo disponible para el desarrollo del proyecto. No obstante, se planteó como línea futura la consolidación de alianzas institucionales que permitan validar, complementar y fortalecer esta clasificación en fases posteriores.

3.2. Recolección de datos institucionales

Como parte del proceso de identificación, verificación y contextualización de las amenazas que afectan a los animales domésticos en la ciudad de Cali, se desarrolló una estrategia de articulación con entidades institucionales del orden local, con el propósito de acceder a información oficial que permitiera complementar el análisis documental y fortalecer la base empírica del proyecto.

En primera instancia, se llevó a cabo un acercamiento con el Centro de Bienestar Animal de Cali, en el cual se sostuvo una reunión con su directora, Carmen Domínguez, con el objetivo de socializar el alcance del proyecto y explorar posibles fuentes de información relacionadas con problemáticas que afectan a los animales domésticos en el entorno urbano. A partir de este encuentro, se identificó que, si bien el Centro de Bienestar Animal gestiona casos asociados al bienestar y protección animal, la información de carácter epidemiológico y sanitario es administrada principalmente por el Centro

de Zoonosis, en articulación con la Secretaría de Salud Pública de Cali.

En consecuencia, se estableció contacto con un funcionario del Centro de Zoonosis, quien orientó el procedimiento institucional requerido para la solicitud formal de acceso a información. Como parte de este proceso, se indicó la necesidad de diligenciar el Formulario de Solicitud de Evaluación del Comité de Ética y de Investigaciones, así como la suscripción de un Acuerdo de Confidencialidad Individual y Protección de Datos, en cumplimiento de los lineamientos establecidos para el manejo de información en salud pública.

El formulario proporcionado por la Secretaría de Salud Pública constituyó un instrumento formal para la evaluación de proyectos de investigación que requieren acceso a información no disponible en repositorios de datos abiertos, e incluye apartados relacionados con la descripción del proyecto, justificación, objetivos, tipo de estudio, población, plan de gestión de datos, aspectos éticos y aportes esperados a nivel disciplinar y territorial. En este sentido, su diligenciamiento implicó la estructuración detallada de la propuesta investigativa, así como la definición de los mecanismos de manejo, almacenamiento y acceso a la información solicitada.

En el marco de esta solicitud, se definieron como requerimientos principales los siguientes tipos de datos:

- Datos agregados por año y mes sobre eventos zoonóticos reportados en el sistema de vigilancia en salud pública (SIVIGILA), correspondientes a la zona urbana de la ciudad de Cali, con el fin de analizar su comportamiento temporal.
- Distribución territorial de los casos a nivel de comuna o barrio, sin incluir información sensible o datos personales, con el propósito de identificar patrones espaciales y zonas de concentración de riesgo.
- Clasificación de los eventos sanitarios, incluyendo enfermedades zoonóticas de interés como leptospirosis, rabia, toxoplasmosis, brucelosis, así como accidentes rábicos u otros eventos priorizados por las autoridades sanitarias.
- Información consolidada sobre la presencia de especies de interés sanitario, particularmente el caracol africano (Achatina fulica), incluyendo su ubicación territorial aproximada y frecuencia de reporte.
- Datos históricos recientes, correspondientes a los últimos tres años epidemiológicos disponibles (2023-2025), con el objetivo de identificar tendencias, recurrencia de eventos y posibles patrones estacionales.

Cabe resaltar que la solicitud fue diseñada bajo criterios de anonimización y agregación de la información, evitando en todo momento el acceso a datos personales o sensibles, en coherencia con los principios éticos establecidos para la investigación en salud pública.

Aunque el proceso de solicitud fue completado conforme a los requerimientos institucionales, los tiempos de evaluación del Comité de Ética y de Investigaciones excedieron el periodo disponible para el desarrollo del proyecto, por lo cual no fue posible acceder efectivamente a la información solicitada dentro de esta fase.

No obstante, este proceso permitió establecer una ruta institucional clara para la obtención de datos oficiales, así como evidenciar la relevancia de los procedimientos éticos, legales y administrativos en el acceso a información epidemiológica. Asimismo, la gestión adelantada constituye un insumo metodológico significativo para el proyecto, al reflejar las condiciones reales de acceso a datos en el contexto local y las limitaciones asociadas a la disponibilidad de información en salud pública.

Finalmente, se considera pertinente incluir como anexos los documentos diligenciados durante este proceso, con el fin de evidenciar la intención de formalización de la solicitud y garantizar la trazabilidad de la gestión institucional realizada (Véase el apéndice A).

3.3. Validación con actores locales

En el marco del proceso de investigación y diagnóstico, se llevó a cabo una validación preliminar de la información recopilada mediante la interacción con actores institucionales del contexto local, con el propósito de contrastar los hallazgos obtenidos en la revisión documental y la clasificación de amenazas con la realidad operativa de las entidades involucradas en la gestión del bienestar animal y la salud pública en la ciudad de Cali.

Como se mencionó anteriormente, se estableció un primer acercamiento con el Centro de Bienestar Animal, a través de una reunión con su directora, Carmen Domínguez, en la cual se socializó el enfoque del proyecto y se discutieron las principales problemáticas que afectan a los animales domésticos en el entorno urbano. Este espacio permitió identificar que, desde la perspectiva institucional, existen múltiples factores que inciden en el bienestar animal, los cuales no se limitan únicamente a condiciones sanitarias, como la exposición a ambientes contaminados, la presencia de vectores y el contacto con otros animales potencialmente infectados, sino que también incluyen dinámicas sociales asociadas a la interacción humana. Entre estas se destacan situaciones como el abandono de animales en espacios públicos, la pérdida o extravío de mascotas, el robo de animales de compañía y la tenencia irresponsable, la cual puede manifestarse en la falta de vacunación, inadecuadas condiciones de alimentación o ausencia de control sobre la movilidad del animal.

También se mencionaron problemáticas relacionadas con la sobrepoblación de animales en condición de calle, la cual incrementa la exposición a enfermedades, accidentes y situaciones de maltrato, así como conflictos con la comunidad. De igual forma, se identificaron escenarios en los que los animales permanecen en condiciones inadecuadas de confinamiento, sin acceso suficiente a agua, alimento o atención veterinaria, lo que afecta directamente su calidad de vida.

Estas dinámicas, aunque no siempre son registradas dentro de los sistemas formales de vigilancia epidemiológica, generan impactos significativos tanto en la salud como en el bienestar de los animales domésticos, evidenciando la necesidad de incorporar este tipo de problemáticas dentro de un análisis más amplio del riesgo en el entorno urbano.

Además, la remisión hacia el Centro de Zoonosis evidenció la existencia de una estructura institucional diferenciada para la gestión de información sanitaria, lo cual permitió validar la pertinencia de considerar fuentes oficiales para el análisis de amenazas zoosanitarias. En este sentido, la orientación recibida respecto al proceso formal de solicitud de datos confirmó la relevancia de los eventos

zoonóticos previamente identificados, así como la necesidad de abordar el análisis desde un enfoque basado en información agregada y regulada por criterios éticos.

Aunque no fue posible acceder a los datos institucionales dentro del periodo de desarrollo del proyecto, la revisión documental permitió sustentar la pertinencia de las categorías definidas en la clasificación de amenazas. En particular, se identificó como relevante la inclusión de problemáticas como el abandono y la desaparición de mascotas, las cuales constituyen situaciones recurrentes en el contexto urbano y generan impactos directos en el bienestar animal, aun cuando no forman parte de los sistemas tradicionales de vigilancia epidemiológica.

Además, este proceso permitió identificar la necesidad de establecer mecanismos de articulación futura con dichas entidades, con el fin de fortalecer la validación de la información y mejorar la integración entre el componente ciudadano y el institucional dentro del sistema propuesto.

Se evidencia una propuesta coherente con la realidad operativa de las instituciones, en la medida en que entidades como el Centro de Bienestar Animal y el Centro de Zoonosis abordan problemáticas desde enfoques complementarios: mientras unas se centran en la atención y protección animal, otras gestionan eventos sanitarios y epidemiológicos. Esta fragmentación institucional refleja, a su vez, la naturaleza multidimensional del problema, lo que justifica la necesidad de una aproximación integradora como la planteada en el presente proyecto.

En conclusión, el estudio respalda la pertinencia del enfoque adoptado, al evidenciar que una clasificación limitada exclusivamente a lo zoosanitario resultaría insuficiente para representar la complejidad dimensional del problema. En consecuencia, el enfoque integral propuesto no solo responde a los hallazgos teóricos, sino que se fundamenta en la realidad observada en el territorio y en las necesidades identificadas por los actores institucionales, lo cual fortalece la coherencia y aplicabilidad de la solución tecnológica planteada.

3.4. Análisis de soluciones existentes

Como parte del proceso de diagnóstico, se retomaron y analizaron diversas iniciativas previamente revisadas en el marco teórico, con el objetivo de evaluar su aplicabilidad al contexto urbano de la ciudad de Cali y extraer elementos relevantes para el diseño del prototipo propuesto.

En primer lugar, se consideraron plataformas basadas en ciencia ciudadana que utilizan la participación activa de los usuarios como fuente principal de información. Este tipo de soluciones evidencian que el reporte colaborativo, apoyado en tecnologías móviles y geolocalización, permite ampliar significativamente la cobertura de los sistemas de vigilancia, especialmente en entornos urbanos donde los métodos tradicionales presentan limitaciones operativas y de escalabilidad. A partir de este análisis, se identificó como elemento clave la capacidad de generar información en tiempo casi real, lo cual resulta fundamental para la detección temprana de eventos.

De manera complementaria, se analizaron herramientas orientadas a la recolección de datos estructurados mediante la interacción constante con los usuarios, incorporando variables espaciales y comportamentales basadas en la interacción de la persona con su entorno inmediato. Este enfoque permitió evidenciar que no solo la localización de los eventos es relevante, sino también el contexto en el que estos ocurren, lo que aporta una comprensión integral de los factores asociados al riesgo.

Por otra parte, se estudiaron sistemas de vigilancia epidemiológica basados en fuentes oficiales, los cuales destacan por su confiabilidad y rigor en la validación de la información. Sin embargo, durante el análisis se identificó que estos sistemas suelen operar con datos agregados y con una periodicidad definida, lo que limita su capacidad de respuesta inmediata frente a eventos emergentes. Esta característica resulta especialmente relevante en contextos urbanos dinámicos, donde la rapidez en la detección y comunicación de riesgos puede incidir directamente en la prevención de afectaciones.

En el contexto local, se reconoció el papel de las entidades institucionales en la generación de información epidemiológica, particularmente a través de reportes y boletines oficiales. Estos insumos constituyen una base sólida para comprender la distribución y comportamiento de ciertas problemáticas sanitarias en la ciudad; no obstante, su formato y nivel de agregación dificultan su uso directo por parte de la ciudadanía y su integración en sistemas interactivos de consulta en tiempo real.

Adicionalmente, se identificó la existencia de dinámicas de reporte no institucionales relacionadas con problemáticas que afectan el bienestar animal, tales como la pérdida, el abandono o el robo de mascotas. Estas dinámicas, generalmente canalizadas a través de redes sociales o plataformas informales, evidencian una necesidad latente de herramientas que permitan organizar, estructurar y georreferenciar este tipo de información, la cual actualmente se encuentra dispersa y carece de mecanismos de validación.

A partir de este análisis, se concluye que, si bien existen soluciones que abordan de manera efectiva componentes específicos del problema, ya sea desde la vigilancia sanitaria, la participación ciudadana o la visualización de datos, ninguna integra de forma articulada estos elementos en un mismo sistema, ni contempla simultáneamente dimensiones sanitarias, sociales y ambientales en un contexto urbano localizado.

En consecuencia, el presente proyecto adopta un enfoque integrador que busca combinar la confiabilidad de la información institucional con la inmediatez del reporte ciudadano, incorporando además una clasificación ampliada de amenazas que trasciende lo estrictamente zoosanitario. Este planteamiento responde directamente a los hallazgos del análisis realizado y constituye el fundamento para el diseño de una solución más ajustada a las dinámicas reales de la ciudad de Cali.

3.5. Selección y priorización de amenazas

Como resultado de la revisión documental, la recolección de información institucional, la validación con actores locales y el análisis de soluciones existentes, se definió una selección priorizada de amenazas para la primera versión del sistema. Esta priorización no buscó agotar la totalidad de riesgos que podían afectar a los animales domésticos en la ciudad de Cali, sino que estableció una base de categorías suficientemente representativas, pertinentes y técnicamente gestionables dentro del prototipo.

La selección se realizó con base en cinco criterios principales: (i) relevancia para el bienestar de animales domésticos en el contexto urbano de Cali; (ii) presencia o trazabilidad en fuentes oficiales,

institucionales o comunitarias; (iii) posibilidad de representación geoespacial dentro del sistema; (iv) potencial de alerta o prevención temprana; y (v) posibilidad de ser comprendida y reportada por usuarios no expertos. A partir de estos criterios, se priorizaron amenazas sanitarias, ambientales y zoosociales que, en conjunto, permiten una lectura más integral del riesgo urbano. En el componente estrictamente zoosanitario, la selección inicial de amenazas se concentró en aquellas que presentan evidencia reciente de vigilancia, prevención o intervención institucional en la ciudad de Cali. En particular, se identificó un seguimiento continuo por parte de la Secretaría de Salud Pública frente a las agresiones y contactos con animales potencialmente transmisores de rabia (APTR), al punto de mantener boletines epidemiológicos específicos asociados a este evento [13]. Asimismo, durante 2025 la administración distrital reforzó las campañas de vacunación antirrábica y las acciones de vigilancia epidemiológica, destacando que Cali completó 36 años sin casos de rabia en perros o humanos, aunque manteniendo la prevención como una prioridad sanitaria permanente [14].

De igual forma, la leptospirosis se mantuvo dentro del conjunto de amenazas priorizadas, no solo por su histórico de vigilancia epidemiológica en el municipio, sino también por su presencia en acciones institucionales recientes relacionadas con control ambiental, manejo de roedores y seguimiento territorial desde el Consejo Territorial de Salud Ambiental [15]. Lo anterior permitió identificar que esta enfermedad continúa siendo epidemiológicamente relevante para la ciudad, especialmente debido a su relación con temporadas de lluvia, acumulación de residuos, aguas contaminadas y exposición ambiental urbana.

En el componente ambiental, se priorizó la presencia del caracol gigante africano (*Achatina fulica*), debido a que durante 2025 y comienzos de 2026 el Distrito de Cali y el DAGMA fortalecieron las acciones de control, educación ambiental y recuperación de zonas afectadas [16]. Durante el inicio de la temporada de lluvias de 2025, la entidad reportó un incremento en las solicitudes ciudadanas relacionadas con la presencia de esta especie, además de registrar más de doscientas intervenciones asociadas a brotes y operativos de control en distintos sectores de la ciudad. La continuidad de estas acciones institucionales evidenció la persistencia del problema y reforzó su pertinencia como amenaza prioritaria dentro de la primera versión funcional del sistema.

Ahora bien, el análisis realizado también evidenció que limitar la plataforma exclusivamente a amenazas sanitarias dejaba por fuera problemáticas urbanas que, aunque no forman parte de los sistemas tradicionales de vigilancia epidemiológica, afectan de manera significativa el bienestar animal. Por esta razón, se incorporaron amenazas de carácter zoosocial, particularmente el robo, la desaparición y el abandono de mascotas, ya que fueron identificadas de manera recurrente tanto en la revisión documental como en las interacciones preliminares con actores locales. Estas problemáticas reflejan dinámicas cotidianas de vulnerabilidad animal presentes en la ciudad y representan riesgos percibidos directamente por la comunidad.

En este sentido, las amenazas zoosociales no sustituyen el componente zoosanitario del proyecto, sino que amplían su alcance conceptual, permitiendo representar de manera más integral las condiciones que afectan a los animales domésticos en el entorno urbano.

Con base en lo anterior, la Tabla 3.1 presenta la selección priorizada de amenazas considerada para la primera versión funcional del sistema. En ella se distinguen las categorías generales, la amenaza específica, la justificación de inclusión y el nivel de prioridad definido para su implementación.

Cuadro 3.1: Priorización de amenazas para la primera versión funcional del sistema

Categoría general	**Amenaza priorizada**	**Justificación de inclusión**	**Prioridad**
Zoosanitaria	Agresiones y contactos con animales potencialmente transmisores de rabia (APTR)	Existe vigilancia epidemiológica específica en Cali y acciones recientes de prevención y vacunación antirrábica para perros y gatos.	Alta
Zoosanitaria	Riesgo por rabia	Aunque Cali no reporta casos en perros o humanos desde hace décadas, la continuidad de campañas y vigilancia oficial justificó su inclusión preventiva.	Alta
Zoosanitaria	Leptospirosis	Se mantiene como evento epidemiológico relevante y asociado a condiciones urbanas como aguas estancadas, residuos y control de roedores.	Alta
Ambiental / Zoosanitaria	Presencia del caracol gigante africano (*Achatina fulica*)	Tiene reportes recientes, atención institucional y riesgo ambiental/sanitario reconocido en Cali.	Alta
Ambiental	Acumulación de residuos / focos insalubres	Favorecen vectores, contaminación y persistencia de eventos sanitarios; además son georreferenciables y comprensibles para la ciudadanía.	Media-Alta
Ambiental	Aguas estancadas / condiciones propicias para proliferación de vectores	Tienen relación directa con riesgos epidemiológicos y pueden ser reportadas espacialmente por usuarios.	Media-Alta
Zoosocial	Robo de mascotas	Problemática urbana no sanitaria, pero de impacto directo sobre el bienestar animal y altamente relevante para usuarios.	Alta
Zoosocial	Desaparición de mascotas	Amenaza recurrente a nivel comunitario, cuya inclusión fortalece la utilidad social de la plataforma.	Alta
Zoosocial	Abandono de animales domésticos	Problema urbano que expone al animal a riesgos sanitarios, ambientales y de violencia.	Alta
Otras	Amenazas no previstas inicialmente	Permite capturar eventos emergentes o no contemplados inicialmente, sujetos a revisión y posible incorporación futura.	Estratégica

Como se observa en la Tabla 3.1, la priorización no depende únicamente de la gravedad intrínseca de una amenaza, sino también de su pertinencia operativa dentro del sistema. Es decir, una amenaza fue priorizada no solo por su impacto potencial, sino por la posibilidad de ser reportada por usuarios, representada geográficamente, entendida por la comunidad y utilizada para generar alertas o información útil dentro del prototipo.

En ese sentido, la categoría "Otras" cumple una función metodológica y tecnológica importante. Dado que no es posible prever de antemano la totalidad de amenazas presentes en un entorno urbano cambiante, se definió que el sistema incluirá la posibilidad de reportar eventos no contemplados en la clasificación inicial bajo esta categoría abierta. Posteriormente, dichos reportes podrán ser revisados y analizados para determinar si constituyen una amenaza recurrente y si deben incorporarse formalmente a la taxonomía base de la plataforma en futuras versiones. Esta decisión buscó responder a la naturaleza evolutiva del sistema y evitar que la herramienta quede rígidamente atada a un conjunto cerrado de categorías.

En consecuencia, la selección y priorización de amenazas realizada en esta etapa no representa una lista definitiva, sino una línea base funcional sobre la cual se construirá el sistema. Esta línea base permitió equilibrar el rigor conceptual con la viabilidad técnica del prototipo, al tiempo que deja abierta la posibilidad de ampliación progresiva conforme se obtengan nuevos datos, validaciones institucionales o reportes ciudadanos relevantes.

3.6. Conclusión del Capítulo

En este capítulo se desarrolló el proceso de investigación y diagnóstico orientado a la identificación, análisis y clasificación de las amenazas que afectan a los animales domésticos en la ciudad de Cali. A través de la revisión documental, la recolección de información institucional, la interacción preliminar con actores locales y el análisis de soluciones existentes, se logró consolidar una comprensión estructurada del problema, sustentada principalmente en la revisión documental y el análisis de fuentes institucionales disponibles.

Este análisis permitió evidenciar que las problemáticas que afectan el bienestar animal en entornos urbanos no pueden ser abordadas únicamente desde una perspectiva zoosanitaria, ya que existen dinámicas de carácter no estrictamente sanitario que influyen directamente en la exposición al riesgo. En este sentido, la ampliación del enfoque conceptual adoptado en el proyecto constituye un aporte relevante, al permitir representar de manera más completa la complejidad del contexto urbano.

Como resultado de este proceso, se definió una selección y priorización de amenazas que establece una base inicial para el desarrollo del sistema, sustentada tanto en evidencia institucional como en las dinámicas observadas en la comunidad. Esta selección no solo responde a criterios de relevancia y frecuencia, sino también a su viabilidad de representación geoespacial y a su potencial de ser reportadas y comprendidas por los usuarios.

Asimismo, se incorporó el concepto de categorías abiertas de reporte, reconociendo que el entorno urbano es dinámico y que no es posible anticipar la totalidad de amenazas existentes. Esta decisión fortalece el carácter evolutivo del sistema y permite su adaptación progresiva a nuevas

problemáticas identificadas en el territorio.

En consecuencia, los resultados obtenidos en esta etapa constituyen la base conceptual y empírica sobre la cual se estructura el diseño del sistema. A partir de esta selección priorizada se definieron las categorías iniciales de peligros que serán utilizadas en el desarrollo de los requisitos funcionales, la modelación de datos y la lógica de visualización geoespacial de la plataforma, estableciendo así un puente directo entre el diagnóstico del problema y la solución tecnológica propuesta.

Capítulo 4

Etapa 2: Análisis y Diseño del Sistema

Introducción

En este capítulo se presentó el proceso de análisis y diseño del sistema propuesto, el cual se fundamentó en los hallazgos obtenidos durante la etapa de investigación y diagnóstico. A partir de la comprensión de las problemáticas asociadas a las amenazas zoosanitarias y zoosociales en la ciudad de Cali, se procedió a estructurar una solución tecnológica que permitiera gestionar, visualizar y comunicar esta información de manera eficiente y accesible.

En primera instancia, se realizó un diagnóstico de los flujos de información existentes, con el fin de identificar las limitaciones actuales en los mecanismos de recolección, procesamiento y difusión de datos relacionados con el bienestar animal y la salud pública. Este análisis permitió evidenciar vacíos en términos de integración, disponibilidad y acceso oportuno a la información.

Posteriormente, se llevó a cabo la definición de los requerimientos del sistema, tanto funcionales como no funcionales, los cuales orientaron el diseño de la solución tecnológica y establecieron el margen bajo el cual se delimita el alcance del modelo propuesto. De esta manera, se garantizó que el sistema cumpliese con condiciones claras, verificables y alineadas con los objetivos del proyecto.

Estos requerimientos se estructuraron con base en las necesidades identificadas en el contexto y en los objetivos planteados, asegurando su claridad, trazabilidad y viabilidad de implementación.

Con base en estos insumos, se planteó la arquitectura del sistema, el modelo de datos geoespacial y los flujos de comunicación que permitieron soportar el monitoreo colaborativo de amenazas. Además, se definieron estrategias orientadas a la participación ciudadana, reconociendo su papel fundamental en la generación de información en tiempo real.

Finalmente, se presentó el diseño conceptual de la interfaz y la experiencia de usuario, apoyado en prototipos, los cuales permiten visualizar la interacción del usuario con el sistema y validar la usabilidad de la solución propuesta.

4.1. Diagnóstico de flujos de información existentes

En el marco del análisis del problema, se realizó un diagnóstico de los flujos de información actualmente disponibles relacionados con eventos zoosanitarios, ambientales y de bienestar animal en la ciudad de Cali, con el objetivo de identificar sus principales limitaciones en términos de acceso, integración, comprensión y oportunidad.

Como parte de este proceso, se analizaron diversas fuentes institucionales, particularmente los boletines epidemiológicos emitidos por la Secretaría de Salud Pública de Cali, los cuales constituyen uno de los principales mecanismos de difusión de información sobre eventos de interés en salud

pública. Estos boletines contienen datos relevantes sobre la ocurrencia de enfermedades zoonóticas, eventos asociados a riesgos sanitarios y acciones de vigilancia epidemiológica en la ciudad.

No obstante, a partir del análisis realizado, se identificó que, si bien estos instrumentos representan una fuente confiable y estructurada de información, presentan limitaciones significativas en términos de accesibilidad y uso por parte de la ciudadanía. En primer lugar, la información se encuentra dispersa en múltiples documentos publicados de manera periódica, lo que dificulta su consulta consolidada y el seguimiento continuo de los eventos en el tiempo. Esta fragmentación obliga a los usuarios a realizar una búsqueda manual y comparativa entre diferentes publicaciones, lo cual incrementa la complejidad del acceso a la información.

Adicionalmente, se evidenció que, aunque los boletines son publicados de manera constante, su nivel de difusión es limitado, lo que reduce su impacto como herramienta de comunicación hacia la comunidad. En muchos casos, la información permanece disponible en repositorios institucionales, pero no logra llegar de manera efectiva a los ciudadanos, quienes constituyen un actor clave en la prevención y detección de riesgos.

Otro aspecto relevante identificado corresponde a la forma en que se presenta la información. Los boletines epidemiológicos suelen estructurarse en formatos extensos, predominantemente textuales y técnicos, lo que puede dificultar su comprensión para usuarios no especializados. Diversos estudios en comunicación y comportamiento informacional han señalado que, en contextos donde predomina el contenido denso y altamente textual, los usuarios tienden a reducir su nivel de atención y procesamiento de la información. En particular, Levitin [17] plantea que la era digital expone a los individuos a un volumen de información que supera su capacidad de procesamiento, lo que afecta directamente la atención y la comprensión de los contenidos. En este sentido, se hace evidente la necesidad de explorar alternativas de comunicación más visuales, interactivas y centradas en el usuario, que faciliten la interpretación rápida de la información.

En cuanto a la disponibilidad de datos más detallados, se identificó que el sistema de vigilancia en salud pública (SIVIGILA) constituye una fuente potencialmente valiosa para el análisis de eventos epidemiológicos; pero, el acceso a esta información se encuentra restringido debido a consideraciones éticas y de protección de datos. Esto implica que, aunque existe información de alta calidad a nivel institucional, su disponibilidad para procesos de análisis externos y desarrollo tecnológico es limitada, lo que representa un desafío adicional para la construcción de soluciones basadas en datos.

Por otra parte, se evidenció una ausencia significativa de mecanismos de georreferenciación accesibles para la ciudadanía. La información disponible en los boletines y reportes institucionales suele presentarse de manera agregada, sin permitir una visualización clara de la distribución espacial de los eventos a nivel detallado (por ejemplo, por barrio o sector). Esta limitación impide que las personas que quieran acceder a esta información puedan identificar zonas de riesgo cercanas a su ubicación o comprender la dinámica territorial de las amenazas, reduciendo así la capacidad de toma de decisiones informadas.

Aunado a esto, se identificó que no existe una plataforma unificada que integre en un solo entorno la información proveniente de diferentes fuentes, ya sean institucionales o comunitarias. Esto genera un ecosistema fragmentado en el que los datos se encuentran distribuidos entre boletines oficiales, reportes internos, redes sociales y experiencias individuales, sin un mecanismo que permita su

consolidación, validación y visualización conjunta.

En conjunto, estos hallazgos evidencian que los flujos de información actuales presentan limitaciones en términos de centralización, accesibilidad, comprensibilidad y representación geoespacial, lo cual afecta su efectividad como herramienta para la prevención y monitoreo de riesgos en el entorno urbano. En consecuencia, se identifica la necesidad de una solución tecnológica que permita integrar estas fuentes de información, transformar los datos en representaciones visuales comprensibles y facilitar la participación de la ciudadanía en la generación y consulta de información en tiempo real.

Este diagnóstico constituye el punto de partida para la definición de los requerimientos del sistema y el diseño de una arquitectura que responda a las limitaciones identificadas, orientada a mejorar la disponibilidad, comprensión y utilidad de la información en el contexto del bienestar animal en la ciudad de Cali.

4.2. Análisis de requerimientos del sistema

A partir del diagnóstico de los flujos de información existentes y de la caracterización de las problemáticas asociadas al monitoreo de amenazas en la ciudad de Cali, se llevó a cabo el proceso de elicitación y definición de los requerimientos del sistema. Este proceso tuvo como propósito traducir las necesidades identificadas en especificaciones técnicas claras, medibles y verificables, que orientaran el diseño y desarrollo de la solución tecnológica propuesta.

La identificación de los requerimientos se fundamentó en múltiples fuentes, incluyendo los hallazgos de la revisión documental, la validación con actores institucionales, el análisis de soluciones existentes y la comprensión de las dinámicas de interacción de los usuarios con la información. De esta manera, se buscó garantizar que los requerimientos no solo respondieran a criterios técnicos, sino también a las condiciones reales del contexto urbano y a las necesidades de los usuarios finales.

Como resultado de este proceso, los requerimientos fueron clasificados en tres grandes categorías: requerimientos funcionales, requerimientos no funcionales y requerimientos de interfaz. Los requerimientos funcionales describen las capacidades específicas del sistema y las acciones que este debe ser capaz de ejecutar, los requerimientos no funcionales establecen criterios de calidad asociados al comportamiento del sistema, tales como usabilidad, disponibilidad, rendimiento y visualización y los requerimientos de interfaz definen las características visuales e interactivas de la aplicación.

4.2.1. Clasificación general de requerimientos

Con el fin de organizar y estructurar adecuadamente las especificaciones del sistema, se definió una clasificación que permite agrupar los requerimientos según su naturaleza y propósito dentro de la solución. La Tabla 4.1, presenta un resumen general de la distribución de los requerimientos identificados.

Cuadro 4.1: Clasificación de requerimientos del sistema

Tipo de requerimiento	**Descripción**	**Cantidad**
Requerimientos funcionales (RF)	Definen las funcionalidades del sistema, incluyendo visualización geoespacial, reporte de amenazas, interacción del usuario y gestión de información	45
Requerimientos no funcionales (RNF)	Establecen criterios de calidad del sistema, como usabilidad, rendimiento, disponibilidad y compatibilidad visual	20
Requerimientos de interfaz	Definen las características visuales e interactivas de la aplicación, incluyendo la organización de pantallas, controles de navegación, formularios, visualización del mapa, representación gráfica de amenazas y mecanismos de interacción del usuario con la interfaz.	15

4.2.2. Modelado de casos de uso

Con el fin de representar de manera visual las principales interacciones entre los usuarios y las funcionalidades definidas para el sistema, se elaboró un diagrama de casos de uso. Este modelo permite identificar los procesos principales soportados por la plataforma y evidenciar la relación entre los diferentes módulos funcionales definidos durante el análisis de requerimientos.

El diagrama se centra en el actor principal del sistema, correspondiente al usuario de la aplicación, quien interactúa con funcionalidades asociadas a la gestión de reportes, rutas, mascotas y viajes dentro del entorno geográfico de la ciudad de Cali.

Asimismo, el modelo evidencia relaciones de inclusión y extensión entre distintos casos de uso, particularmente en funcionalidades relacionadas con el reporte de amenazas, la configuración de viajes y la personalización de la experiencia de usuario. Esto permitió estructurar de manera más clara las dependencias funcionales entre módulos y visualizar el comportamiento esperado del sistema desde la perspectiva del usuario.

La Figura 4.1 presenta el diagrama de casos de uso correspondiente al sistema propuesto.

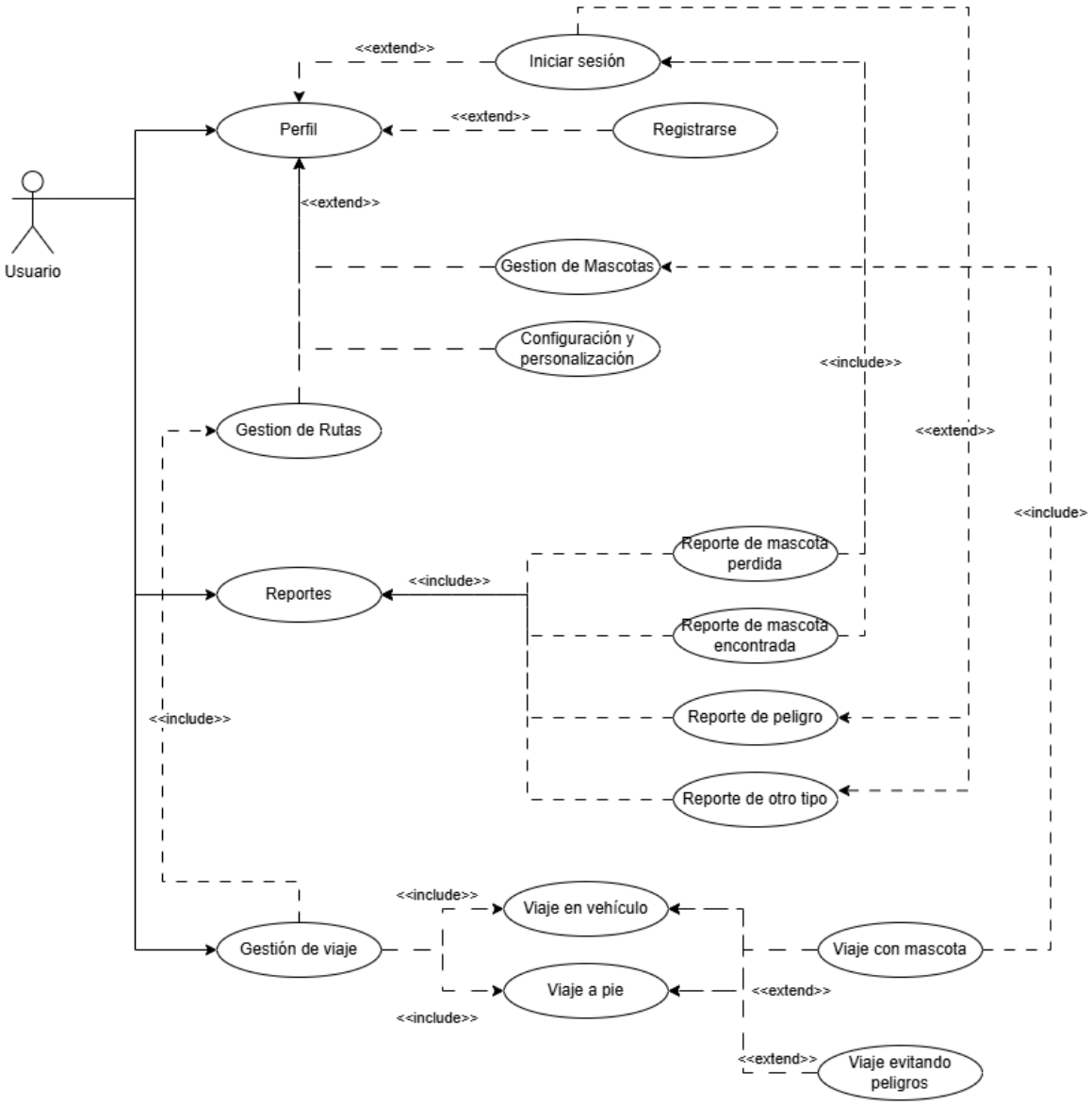


Figura 4.1: Diagrama de casos de uso del sistema propuesto.

4.2.3. Requerimientos funcionales

Los requerimientos funcionales constituyen el núcleo del sistema, ya que describen las funcionalidades necesarias para soportar el monitoreo geoespacial, la participación ciudadana y la gestión de amenazas en el entorno urbano.

Dado que la solución propuesta se fundamenta en la representación geográfica de información y en la construcción colaborativa de reportes urbanos, se definieron distintos grupos funcionales relacionados con visualización espacial, gestión de reportes, persistencia local parcial (orientada

al almacenamiento local de información del usuario) y consumo de servicios backend especializados para aquellas funcionalidades dependientes de conectividad, incluyendo consulta de amenazas, comentarios, reportes comunitarios, mapas y navegación.

4.2.3.1. *Visualización y navegación del mapa*

Uno de los componentes centrales del sistema corresponde a la representación del mapa y la interacción del usuario con el entorno geográfico. En este sentido, se definieron requerimientos orientados a garantizar la correcta visualización de la zona urbana de Cali, el control de navegación dentro del mapa y la restricción de los límites geográficos.

En este sentido, se definieron requerimientos asociados al comportamiento inicial del mapa, el manejo de permisos de ubicación y la navegación dentro del territorio urbano de Cali.

A continuación, se presentan los requerimientos de este componente:

- **RF1:** La aplicación debe mostrar en la interfaz principal un mapa base de la zona urbana de la ciudad de Cali.
- **RF2:** El mapa mostrado en la interfaz principal debe limitar su visualización a los límites geográficos de la zona urbana del Distrito de Santiago de Cali.
- **RF3:** Al abrir la página principal, si el usuario no ha concedido permisos de ubicación o estos no están disponibles, el sistema debe mostrar en pantalla el mapa con el nivel mínimo de acercamiento.
- **RF4:** Al abrir la página principal, si el usuario ha concedido permisos de ubicación y esta se encuentra disponible, el sistema debe centrar el mapa en la ubicación actual del usuario utilizando el nivel máximo de acercamiento definido por la aplicación.
- **RF5:** El mapa debe disponer de un control que permita aumentar el nivel de acercamiento del mapa hasta alcanzar el nivel máximo de acercamiento definido por la aplicación dentro de los límites de la zona urbana del Distrito de Santiago de Cali.
- **RF6:** El mapa debe disponer de un control que permita reducir el nivel de acercamiento del mapa hasta alcanzar el nivel mínimo de acercamiento, en el cual se visualice la totalidad de la zona urbana del Distrito de Santiago de Cali.
- **RF7:** El sistema debe impedir que el usuario desplace o aumente el nivel de acercamiento del mapa por fuera de los límites geográficos definidos.

4.2.3.2. *Gestión y visualización de amenazas*

El sistema incorpora un componente orientado a la representación de amenazas urbanas georreferenciadas, permitiendo visualizar peligros reportados por los usuarios y consultar información

asociada a cada evento registrado. Este componente constituye uno de los ejes funcionales principales de la plataforma, debido a que transforma reportes individuales en información visual accesible para la comunidad.

En este contexto, se definieron requerimientos asociados a la representación gráfica de amenazas, consulta de información detallada y definición estructurada de atributos relacionados con cada peligro reportado.

A continuación, se presentan los requerimientos de este componente:

- **RF9:** En la interfaz principal el sistema debe mostrar los peligros reportados mediante elementos gráficos ubicados en sus respectivas coordenadas geográficas sobre el mapa.
- **RF10:** El sistema debe permitir al usuario seleccionar un peligro mostrado en el mapa para consultar su información detallada.
- **RF11:** La información detallada de un peligro debe incluir categoría del peligro, descripción textual del reporte, fecha de creación, duración aproximada, nivel de riesgo, radio de afectación y comentarios asociados.
- **RF12:** El radio de afectación de un peligro debe definirse mediante un conjunto de valores permitidos configurados en el sistema.

4.2.3.3. *Configuración e interfaz de usuario*

Con el propósito de mejorar la experiencia de uso y permitir la personalización básica de la plataforma, se incorporaron funcionalidades relacionadas con configuración visual y acceso a herramientas de administración del usuario.

En este sentido, se definieron requerimientos asociados a controles de navegación hacia módulos internos del sistema y opciones de adaptación visual de la interfaz.

A continuación, se presentan los requerimientos de este componente:

- **RF13:** El sistema debe proporcionar un botón en la interfaz principal que permita alternar entre modo claro y modo oscuro.
- **RF14:** El sistema debe disponer de un botón accesible desde la página principal que permita ingresar a la página de configuración del usuario.
- **RF15:** El sistema debe disponer de una página de configuración que permita acceder a las funcionalidades de gestión de rutas y gestión de mascotas.

4.2.3.4. *Gestión de rutas*

El sistema incorpora un módulo de gestión de rutas que permite almacenar trayectorias definidas por el usuario dentro del mapa de la ciudad de Cali. Estas rutas funcionan como base para la configuración de viajes y el monitoreo contextual de amenazas durante los desplazamientos.

En este contexto, se definieron requerimientos asociados al almacenamiento, edición y representación de trayectorias geográficas.

A continuación, se presentan los requerimientos de este componente:

- **RF16:** El sistema debe permitir al usuario crear, visualizar, actualizar y eliminar rutas dentro de la página de gestión de rutas.
- **RF17:** Cada ruta registrada en el sistema debe estar conformada por una secuencia ordenada de ubicaciones geográficas dentro de la zona urbana del Distrito de Santiago de Cali.

4.2.3.5. *Gestión de mascotas*

Con el fin de incorporar información contextual asociada al usuario y fortalecer funcionalidades relacionadas con viajes y reportes, el sistema incluye un módulo de gestión de mascotas.

En este sentido, se definieron requerimientos asociados al almacenamiento y administración de información básica de animales domésticos registrados por el usuario.

A continuación, se presentan los requerimientos de este componente:

- **RF18:** El sistema debe permitir al usuario crear, visualizar, actualizar y eliminar mascotas dentro de la página de gestión de mascotas.
- **RF19:** Cada mascota registrada en el sistema debe incluir obligatoriamente un nombre y opcionalmente una fotografía asociada.

4.2.3.6. *Reportes colaborativos y monitoreo ciudadano*

Uno de los principales aportes funcionales de la plataforma corresponde al sistema de reportes colaborativos, mediante el cual los usuarios pueden registrar diferentes tipos de eventos asociados al bienestar animal y al entorno urbano.

En este sentido, se definieron requerimientos asociados a la creación de distintos tipos de reportes y al acceso centralizado al módulo de monitoreo ciudadano.

A continuación, se presentan los requerimientos de este componente:

- **RF20:** El sistema debe disponer de un botón denominado “Reportar” en la interfaz principal, el cual debe desplegar un panel que permita seleccionar el tipo de reporte que el usuario desea registrar.
- **RF21:** El sistema debe permitir seleccionar entre los siguientes tipos de reporte: peligro, mascota perdida, mascota encontrada y otro reporte.
- **RF22:** El sistema debe permitir registrar reportes mediante formularios georreferenciados asociados a una ubicación específica dentro del mapa de la ciudad de Cali.

4.2.3.7. *Reportes de peligro*

El sistema incorpora un formulario especializado para el registro de peligros asociados a amenazas zoosanitarias y ambientales. Este componente permite capturar información estructurada y georreferenciada relacionada con riesgos presentes en el entorno urbano.

En este contexto, se definieron requerimientos asociados a validación de formularios, categorías de peligro y representación espacial de áreas de riesgo.

A continuación, se presentan los requerimientos de este componente:

- **RF23:** El sistema debe permitir registrar un reporte de tipo peligro mediante un formulario que incluya ubicación geográfica, categoría, descripción textual, nivel de riesgo, radio de afectación y opcionalmente duración aproximada e imagen asociada.
- **RF24:** Las categorías disponibles para los reportes de peligro deben pertenecer a un conjunto predefinido y configurable por el sistema.
- **RF25:** El sistema debe validar que todos los campos obligatorios del reporte de peligro estén completos antes de permitir su envío.
- **RF26:** El sistema debe representar gráficamente en el mapa los reportes de peligro mediante marcadores georreferenciados y áreas de influencia asociadas a su radio de afectación.

4.2.3.8. *Reportes de mascota perdida*

El sistema incorpora funcionalidades orientadas al registro de mascotas desaparecidas dentro del entorno urbano. Estos reportes permiten asociar información descriptiva y geográfica para facilitar procesos de localización comunitaria.

En este sentido, se definieron requerimientos asociados al registro y visualización de reportes de mascotas perdidas.

A continuación, se presentan los requerimientos de este componente:

- **RF27:** El sistema debe permitir registrar reportes de mascota perdida mediante un formulario que incluya último lugar de avistamiento, nombre de la mascota, descripción, fecha de desaparición y número de contacto.
- **RF28:** El sistema debe visualizar en el mapa los reportes de mascota perdida mediante elementos gráficos asociados a su última ubicación registrada.

4.2.3.9. *Reportes de mascota encontrada*

De manera complementaria, el sistema permite registrar animales encontrados sin supervisión dentro del entorno urbano.

En este contexto, se definieron requerimientos asociados al registro y representación geográfica de mascotas encontradas.

A continuación, se presentan los requerimientos de este componente:

- **RF29:** El sistema debe permitir registrar reportes de mascota encontrada mediante un formulario que incluya lugar de encuentro, descripción de la situación, fecha del encuentro y opcionalmente número de contacto e imagen asociada.
- **RF30:** El sistema debe visualizar en el mapa los reportes de mascota encontrada mediante elementos gráficos asociados a la ubicación registrada.

4.2.3.10. *Otros reportes urbanos*

Con el fin de evitar limitar el sistema exclusivamente a categorías predefinidas, se incorporó un mecanismo de reportes abiertos para eventos no contemplados inicialmente.

En este sentido, se definieron requerimientos asociados al registro flexible de problemáticas urbanas emergentes.

A continuación, se presentan los requerimientos de este componente:

- **RF31:** El sistema debe permitir registrar reportes de tipo "otro" mediante un formulario que incluya ubicación geográfica, descripción del evento, fecha del reporte, nivel de riesgo y opcionalmente imagen asociada.

4.2.3.11. *Comentarios y validación colaborativa*

El sistema incorpora mecanismos de retroalimentación comunitaria mediante comentarios asociados a reportes existentes. Esta funcionalidad permite complementar información previamente registrada y fortalecer procesos de validación colaborativa.

En este contexto, se definieron requerimientos asociados a la creación y almacenamiento de comentarios vinculados a reportes urbanos.

A continuación, se presentan los requerimientos de este componente:

- **RF32:** El sistema debe permitir a los usuarios agregar comentarios a los reportes existentes con el fin de complementar, actualizar o validar la información reportada.
- **RF33:** Cada comentario asociado a un reporte debe incluir identificador del usuario, contenido textual y fecha y hora de publicación generada por el sistema.

4.2.3.12. *Persistencia local y manejo de conectividad*

El sistema se concibe como una arquitectura híbrida con persistencia local parcial. En consecuencia, se redefinieron las funcionalidades relacionadas con almacenamiento local y manejo de conectividad, delimitando el alcance de los datos conservados en el dispositivo (información propia del usuario) y estableciendo conectividad requerida para las funciones que dependen de mapas, navegación y servicios remotos del backend.

En este sentido, se definieron requerimientos asociados al almacenamiento local de información del usuario, manejo de funcionalidades dependientes de conectividad y gestión de contenido multimedia dentro de la plataforma.

A continuación, se presentan los requerimientos de este componente:

- **RF34:** El sistema debe almacenar localmente en el dispositivo la información propia del usuario que no requiere publicación comunitaria inmediata, incluyendo rutas, mascotas y configuraciones básicas.
- **RF35:** El sistema debe permitir registrar reportes en modalidad anónima cuando el usuario no haya iniciado sesión, siempre que exista conectividad para enviar la información a la plataforma.
- **RF36:** El sistema debe informar al usuario cuando una funcionalidad no pueda completarse por ausencia de conectividad, especialmente en mapa, reportes, viajes, comentarios y consulta de amenazas.
- **RF37:** Las imágenes asociadas a reportes y mascotas deben almacenarse o referenciarse de acuerdo con el mecanismo disponible para cada módulo, manteniendo vínculo con el registro correspondiente.

4.2.3.13. *Viajes y monitoreo en tiempo real*

Finalmente, el sistema incorpora un módulo de monitoreo de recorridos orientado al acompañamiento del usuario durante sus desplazamientos dentro de la ciudad. Este componente integra rutas, ubicación del usuario y peligros reportados para generar una experiencia de monitoreo contextual en tiempo real.

En este sentido, se definieron requerimientos asociados a la configuración de viajes, visualización de rutas y notificación de amenazas durante el recorrido.

A continuación, se presentan los requerimientos de este componente:

- **RF38:** El sistema debe disponer de un botón de iniciar viaje que permita acceder a la configuración de un recorrido.
- **RF39:** El sistema debe permitir configurar un viaje mediante la selección de un punto de inicio y un punto de destino dentro del mapa.
- **RF40:** Durante la configuración de un viaje, el sistema debe permitir seleccionar una ruta previamente guardada o generar automáticamente una nueva ruta entre el punto de inicio y el punto de destino.
- **RF41:** Una vez configurado el viaje, el sistema debe mostrar en el mapa la ruta calculada entre el punto de inicio y el destino.
- **RF42:** Durante un viaje activo, el sistema debe mostrar la ubicación actual del usuario en tiempo real dentro del mapa.
- **RF43:** El sistema debe visualizar sobre la ruta los peligros reportados que se encuentren dentro del trayecto calculado.

- **RF44:** Durante un viaje activo, el sistema debe notificar al usuario cuando este se encuentre dentro del radio de afectación de un peligro reportado.

- **RF45:** Al finalizar un viaje, el sistema debe permitir almacenar la ruta recorrida dentro del historial del usuario y asociarla opcionalmente a una mascota registrada.

4.2.4. Requerimientos no funcionales

Los requerimientos no funcionales establecen las condiciones de calidad que debe cumplir el sistema para garantizar su correcto funcionamiento, usabilidad, rendimiento, seguridad y mantenibilidad. A diferencia de los requerimientos funcionales, estos no describen acciones específicas realizadas por el usuario, sino propiedades y restricciones asociadas al comportamiento general de la aplicación durante su operación.

Dado que el sistema propuesto se fundamenta en la visualización geoespacial, la interacción constante con dispositivos móviles y el consumo de servicios backend especializados (con conectividad requerida en módulos clave), se definieron distintos grupos de requerimientos no funcionales relacionados con compatibilidad visual, rendimiento, experiencia de usuario, persistencia local parcial, privacidad y escalabilidad.

4.2.4.1. *Visualización y compatibilidad*

Uno de los aspectos más importantes dentro del sistema corresponde a la correcta visualización de la interfaz y del mapa interactivo en distintos tamaños de pantalla y condiciones de uso. Debido a que la aplicación fue diseñada principalmente para dispositivos móviles Android, se establecieron requerimientos orientados a garantizar legibilidad, coherencia visual y facilidad de interacción táctil.

En este sentido, se definieron requerimientos relacionados con resoluciones soportadas, adaptación visual y consistencia gráfica de los componentes de interfaz.

A continuación, se presentan los requerimientos de este componente:

- **RNF1:** El elemento visual del mapa debe verse correctamente en resoluciones de pantalla entre 600x400 y 3840x2160.

- **RNF2:** La aplicación debe mantener consistencia visual y funcional en dispositivos móviles Android con acceso a geolocalización y conexión a internet.

- **RNF3:** La aplicación debe permitir la lectura adecuada de textos, botones e íconos tanto en modo claro como en modo oscuro.

- **RNF4:** Los botones, formularios y elementos interactivos deben mantener una distribución y tamaño adecuados para su uso táctil en dispositivos móviles.

- **RNF5:** Las fotografías opcionales asociadas a mascotas o peligros deben almacenarse y visualizarse sin distorsión apreciable dentro de la aplicación.

4.2.4.2. *Rendimiento y tiempo de respuesta*

El sistema incorpora múltiples funcionalidades dependientes de interacción geográfica y actualización de información obtenida desde servicios backend especializados. Debido a esto, se establecieron restricciones asociadas a tiempos máximos de respuesta y latencia de actualización bajo condiciones normales de red, con el objetivo de garantizar una experiencia fluida para el usuario.

En este apartado se definieron requerimientos relacionados con velocidad de carga, respuesta de la interfaz y actualización de ubicación durante recorridos activos.

A continuación, se presentan los requerimientos de este componente:

- **RNF6:** La aplicación debe responder a las interacciones principales del usuario en un tiempo máximo de 3 segundos bajo condiciones normales de conexión.
- **RNF7:** La carga inicial del mapa de la ciudad de Cali no debe tardar más de 5 segundos en condiciones normales de red.
- **RNF8:** La actualización de la ubicación del usuario durante un viaje activo debe reflejarse en pantalla con una latencia máxima de 5 segundos.

4.2.4.3. *Usabilidad y experiencia de usuario*

Debido a que el sistema está orientado a usuarios con distintos niveles de experiencia tecnológica, se establecieron requerimientos enfocados en simplicidad de interacción, claridad de navegación y retroalimentación frente a errores.

En este sentido, se priorizó la reducción de pasos para tareas frecuentes y la presentación de mensajes comprensibles ante fallos relacionados con conectividad o carga de información.

A continuación, se presentan los requerimientos de este componente:

- **RNF9:** La interfaz de usuario debe ser intuitiva y permitir que un usuario pueda reportar un peligro en un máximo de 5 pasos desde la página principal.
- **RNF10:** El sistema debe manejar errores de conexión mostrando mensajes claros al usuario cuando no sea posible cargar el mapa, la ubicación o la información de peligros.

4.2.4.4. *Persistencia local y manejo de conectividad*

Aunque el sistema incorpora mecanismos de almacenamiento local de información del usuario, el funcionamiento general de la plataforma depende parcialmente de conectividad requerida debido al uso de servicios geoespaciales y a la consulta/actualización de información comunitaria y de amenazas mediante el backend.

Por esta razón, los requerimientos no funcionales asociados a este componente se orientaron principalmente a garantizar la persistencia local parcial de información permitida y la correcta comunicación al usuario cuando determinadas funcionalidades no puedan ejecutarse por ausencia de conexión.

A continuación, se presentan los requerimientos de este componente:

- **RNF11:** El sistema debe almacenar de forma persistente en el dispositivo las rutas, mascotas y configuraciones locales permitidas, de modo que no se pierdan al cerrar la aplicación.

- **RNF12:** La aplicación debe indicar de forma comprensible qué funciones requieren conectividad cuando el usuario intente utilizarlas sin acceso a red.

4.2.4.5. *Seguridad, privacidad y permisos*

Debido a que la aplicación gestiona información relacionada con ubicación geográfica y datos asociados a usuarios, se definieron requerimientos orientados a limitar la recolección de información personal y proteger los datos almacenados localmente.

Asimismo, se establecieron condiciones específicas para el uso de permisos de ubicación dentro del sistema.

A continuación, se presentan los requerimientos de este componente:

- **RNF13:** El sistema debe proteger la información almacenada localmente mediante mecanismos que eviten accesos no autorizados a los datos del usuario.

- **RNF14:** El sistema no debe solicitar ni almacenar datos personales no requeridos para el funcionamiento de la aplicación.

- **RNF15:** La aplicación debe solicitar permiso de ubicación únicamente cuando este sea necesario para las funcionalidades de mapa, viaje o reporte.

4.2.4.6. *Integridad geográfica y consistencia de información*

Debido a que el sistema trabaja sobre información georreferenciada y combina persistencia local parcial con datos obtenidos desde servicios backend especializados, se establecieron requerimientos orientados a garantizar coherencia territorial y consistencia temporal de la información mostrada al usuario.

En este componente se incluyeron restricciones relacionadas con el alcance geográfico del sistema y el manejo uniforme de información temporal.

A continuación, se presentan los requerimientos de este componente:

- **RNF16:** El sistema debe garantizar que los peligros mostrados en el mapa correspondan únicamente a ubicaciones comprendidas dentro de los límites geográficos de la ciudad de Cali.

- **RNF17:** La información mostrada al usuario debe corresponder a la última actualización obtenida desde la plataforma o a la información local permitida para el módulo consultado.

- **RNF18:** El sistema debe registrar fechas y horas de creación de reportes, comentarios y rutas utilizando un formato consistente en toda la aplicación.

4.2.4.7. *Mantenibilidad y escalabilidad*

Finalmente, se establecieron requerimientos relacionados con la evolución futura del sistema, considerando la posibilidad de ampliar funcionalidades, incorporar nuevos tipos de reportes y aumentar progresivamente el volumen de información procesada.

Para ello, se definieron requerimientos asociados a modularidad arquitectónica y capacidad de crecimiento funcional.

A continuación, se presentan los requerimientos de este componente:

- **RNF19:** El sistema debe permitir la mantenibilidad del software mediante una arquitectura modular que separe los componentes de mapa, rutas, mascotas, reportes y configuración.
- **RNF20:** El sistema debe permitir la escalabilidad suficiente para soportar el crecimiento futuro en el número de reportes, comentarios, rutas y categorías de peligros.

4.2.5. Requerimientos de interfaz

Como parte del proceso de definición de requerimientos del sistema, se establecieron los requerimientos de interfaz, los cuales describen la manera en que el usuario interactúa visualmente con la plataforma y cómo la información es presentada dentro del sistema. Estos requerimientos se enfocan en aspectos relacionados con distribución visual, navegación, representación gráfica y retroalimentación de interacción.

Dado que la solución propuesta se fundamenta en la visualización geoespacial y en la interacción constante con mapas interactivos, formularios y elementos gráficos, se definieron distintos componentes de interfaz orientados a garantizar una experiencia de usuario clara, accesible y consistente.

4.2.5.1. *Interfaz cartográfica*

Uno de los componentes principales de la interfaz corresponde al mapa interactivo, el cual constituye el eje central de visualización del sistema. Este componente permite representar amenazas georreferenciadas, visualizar rutas y mostrar la ubicación actual del usuario dentro del entorno urbano de Cali.

En este apartado se definieron requerimientos relacionados con representación espacial, visualización de elementos geográficos y despliegue contextual de información.

A continuación, se presentan los requerimientos para este componente:

- **RI1:** La interfaz principal de la aplicación debe mostrar un mapa interactivo de la ciudad de Cali ocupando la mayor parte de la pantalla.
- **RI2:** La página principal debe mostrar claramente la ubicación actual del usuario en el mapa mediante un marcador visual diferenciado.
- **RI3:** Los peligros reportados deben representarse en el mapa mediante íconos visuales que permitan identificar su ubicación aproximada.

- **RI4:** Al seleccionar un ícono de peligro en el mapa, la interfaz debe desplegar un panel con la información detallada del peligro reportado.
- **RI5:** Durante un viaje activo, la interfaz debe resaltar la ruta seleccionada en el mapa mediante una línea visual que permita distinguirla de otros elementos.
- **RI6:** La interfaz debe mostrar notificaciones visuales cuando el usuario se acerque a un peligro dentro del radio de influencia definido.

4.2.5.2. *Interacción y navegación*

Con el fin de facilitar el acceso a las funcionalidades principales del sistema, se definieron requerimientos relacionados con navegación entre módulos y acceso rápido a herramientas de interacción.

Estos requerimientos buscan garantizar una estructura de navegación consistente y reducir la complejidad de uso de la aplicación.

A continuación, se presentan los requerimientos para este componente:

- **RI7:** La interfaz debe proporcionar botones visibles para realizar acciones principales como reportar un peligro, iniciar un viaje y acceder a la configuración del usuario.
- **RI8:** La página de gestión de rutas debe permitir visualizar las rutas existentes del usuario y proporcionar opciones para crear, editar o eliminar rutas.
- **RI9:** La página de gestión de mascotas debe mostrar una lista de las mascotas registradas por el usuario y permitir visualizar o editar la información de cada una.
- **RI10:** La interfaz de configuración de viaje debe permitir seleccionar el punto de inicio y el punto de destino mediante interacción directa con el mapa o mediante campos de búsqueda.
- **RI11:** La interfaz debe permitir desplazarse entre las diferentes páginas de la aplicación mediante un sistema de navegación consistente.

4.2.5.3. *Formularios y retroalimentación visual*

Debido a que gran parte de la interacción del usuario ocurre mediante formularios de reporte y configuración, se definieron requerimientos orientados a mejorar claridad visual, validación de información y retroalimentación frente a errores de ingreso de datos.

En este apartado se contemplaron los siguientes requerimientos:

- **RI12:** La interfaz de reporte de peligro debe presentarse en forma de formulario claro y estructurado que permita seleccionar el tipo de peligro, añadir una descripción y adjuntar opcionalmente una fotografía.
- **RI13:** Los formularios de la aplicación deben incluir mensajes claros de validación cuando un campo obligatorio no haya sido completado correctamente.

4.2.5.4. *Consistencia visual y adaptabilidad*

Finalmente, se establecieron requerimientos relacionados con consistencia gráfica y adaptación visual de la aplicación en distintos contextos de uso.

En este apartado se contemplaron aspectos relacionados con modos de visualización, coherencia gráfica y uniformidad de componentes interactivos.

A continuación, se presentan los requerimientos para este componente:

- **RI14:** La interfaz debe permitir alternar entre modo claro y modo oscuro mediante un botón accesible desde la página principal.
- **RI15:** Los íconos, botones y elementos interactivos deben ser claramente identificables y mantener consistencia visual en toda la aplicación.

La especificación completa de requerimientos del sistema se presenta en el Apéndice C, donde se documentan detalladamente los requerimientos funcionales, no funcionales y de interfaz definidos para la solución propuesta.

4.2.6. Consideraciones de diseño a partir de los requerimientos

Los requerimientos definidos no solo permiten describir el comportamiento esperado del sistema, sino que también influyen directamente en las decisiones de diseño relacionadas con la arquitectura, el modelo de datos y la interfaz de usuario.

En particular, la necesidad de integrar información geoespacial, soportar interacción en tiempo real y permitir el reporte colaborativo llevó a considerar una arquitectura que combine componentes de visualización, almacenamiento y comunicación de datos de manera eficiente.

Finalmente, la estructura de los requerimientos permitió definir las entidades principales del sistema, como amenazas, usuarios, comentarios y rutas, lo cual se verá reflejado en el modelado de la base de datos en las siguientes secciones.

4.2.7. Diseño de la arquitectura del sistema

A partir de los requerimientos funcionales y no funcionales definidos previamente, se diseñó la arquitectura del sistema con el propósito de establecer una estructura tecnológica que soporte de manera eficiente las funcionalidades de monitoreo geoespacial, reporte colaborativo y visualización de amenazas en tiempo real.

La arquitectura propuesta adopta un enfoque de tipo monolítico, en el cual los diferentes módulos del sistema se integran dentro de una única unidad lógica de despliegue. Esta decisión se fundamenta en el alcance del proyecto, el cual corresponde al desarrollo de un prototipo funcional, priorizando la simplicidad en la implementación, la facilidad de mantenimiento y la coherencia en la gestión de la lógica de negocio.

El sistema contempla un cliente que puede ser desplegado tanto en dispositivos móviles como en entornos web, permitiendo a los usuarios interactuar con la plataforma, visualizar información

geoespacial y reportar amenazas desde diferentes contextos de uso. Este enfoque contribuye a mejorar la accesibilidad y el alcance de la solución dentro del entorno urbano de la ciudad de Cali.

4.2.7.1. *Enfoque arquitectónico*

La arquitectura del sistema se organiza en tres componentes principales:

- **Cliente (Frontend):** encargado de la interacción con el usuario, la visualización del mapa, la representación de amenazas y la captura de información mediante formularios.
- **Servidor (Backend):** responsable de la lógica de negocio, la validación de datos, la gestión de reportes y la comunicación con los servicios externos.
- **Base de datos geoespacial:** destinada al almacenamiento estructurado de la información, incluyendo datos espaciales asociados a amenazas, rutas y usuarios.

Esta separación de responsabilidades permite estructurar el sistema de manera clara y facilita su evolución futura hacia arquitecturas más desacopladas si el crecimiento del sistema lo requiere.

4.2.7.2. *Representación arquitectónica mediante el modelo C4*

Para la representación de la arquitectura del sistema, se adoptó el modelo C4 propuesto por Brown (2018), el cual proporciona una forma estructurada de visualizar sistemas de software a diferentes niveles de abstracción: contexto, contenedores, componentes y código.

El modelo C4 permite describir la arquitectura desde una perspectiva progresiva, facilitando la comprensión del sistema tanto para perfiles técnicos como no técnicos. En el presente proyecto, se utilizaron tres niveles principales:

- **Nivel de contexto:** describe la interacción del sistema con los usuarios (ciudadanos, administradores) y entidades externas como servicios de mapas o almacenamiento. La Figura 4.2 presenta el diagrama correspondiente al nivel de contexto del modelo C4.

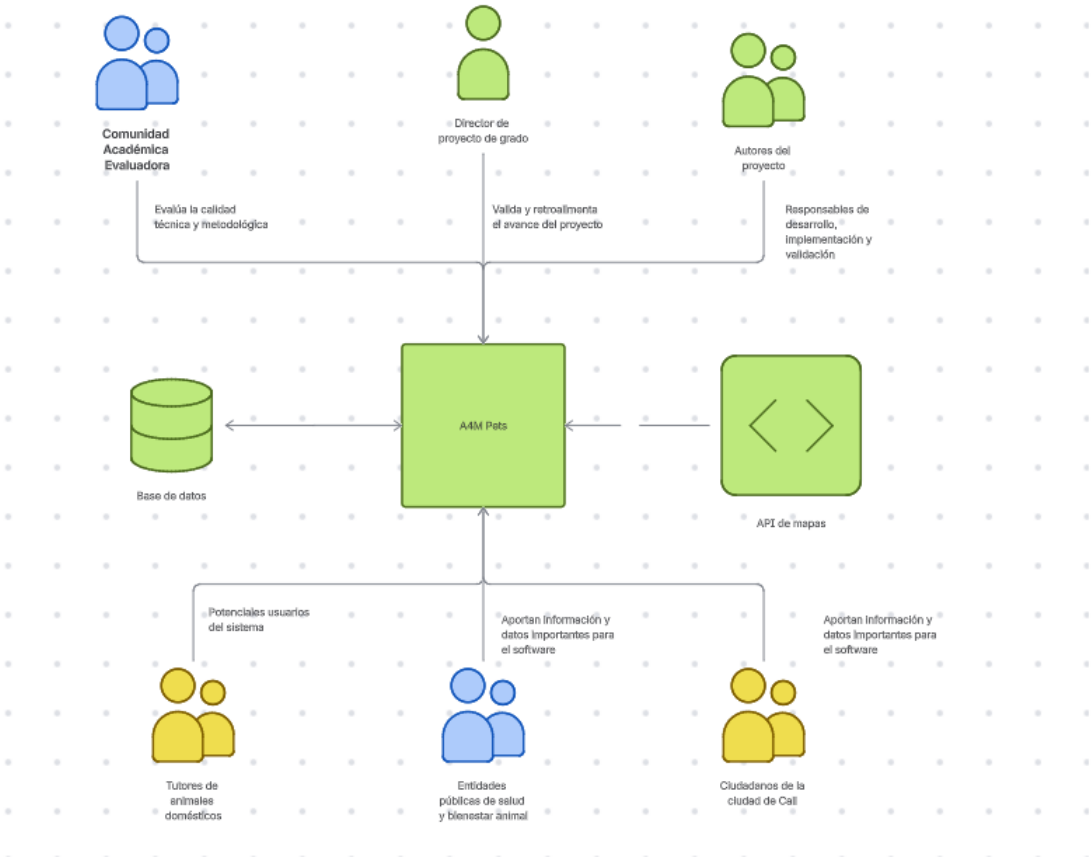


Figura 4.2: Diagrama de contexto del sistema utilizando el modelo C4.

- **Nivel de contenedores:** representa los principales bloques tecnológicos del sistema, incluyendo la aplicación cliente, el servidor y la base de datos.

La Figura 4.3 presenta el diagrama correspondiente al nivel de contenedores del modelo C4.

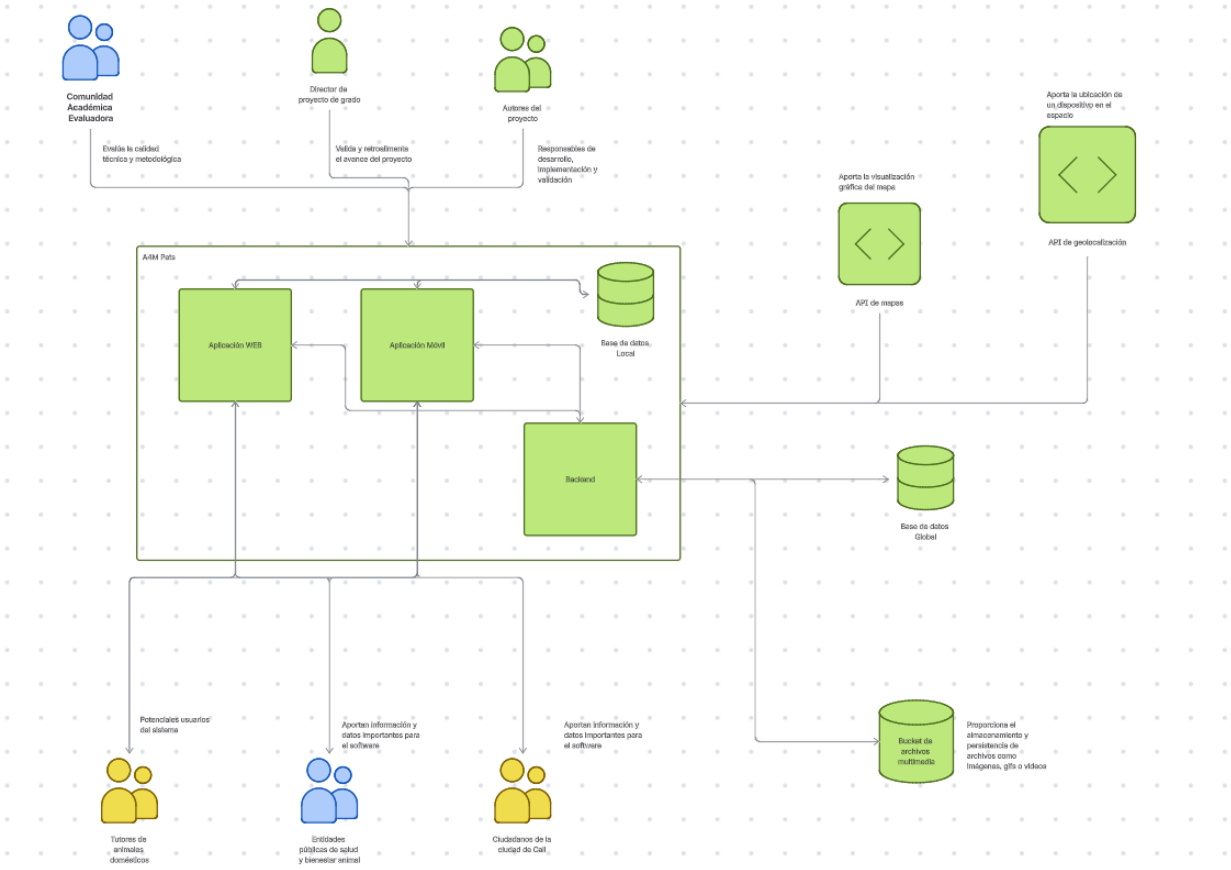


Figura 4.3: Diagrama de contenedores del sistema utilizando el modelo C4.

- **Nivel de componentes:** detalla los módulos internos del sistema, tales como gestión de amenazas, geolocalización, rutas, notificaciones y manejo de usuarios. La Figura 4.4 presenta el diagrama correspondiente al nivel de componentes del modelo C4.

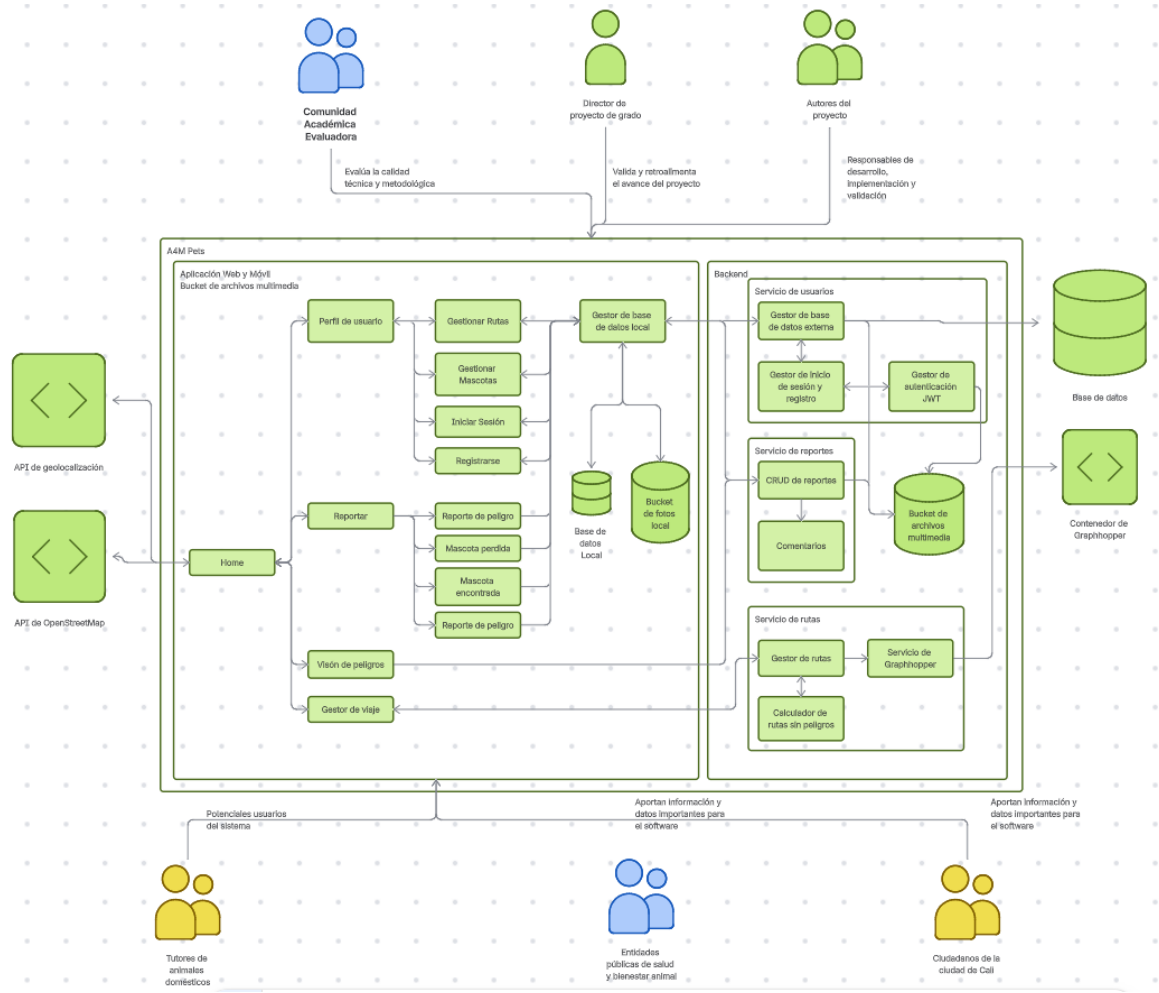


Figura 4.4: Diagrama de componentes del sistema utilizando el modelo C4.

4.2.8. Justificación del uso del modelo C4

La elección del modelo C4 se debe a su capacidad para representar sistemas complejos de manera clara, jerárquica y progresiva. A diferencia de otros enfoques tradicionales, este modelo permite:

- Facilitar la comunicación entre actores técnicos y no técnicos
- Organizar la arquitectura en diferentes niveles de detalle
- Mejorar la comprensión de las relaciones entre componentes
- Mantener coherencia en la documentación del sistema

Para efectos del presente proyecto, el modelo C4 resulta especialmente útil debido a la integración de múltiples elementos como la geolocalización, el procesamiento de datos en tiempo real y la interacción con el usuario, los cuales requieren una representación clara y estructurada.

A pesar de lo mencionado, el nivel de código (nivel 4) no fue abordado en esta etapa, dado que este implica un nivel de detalle orientado a la implementación específica de clases, métodos y estructuras internas del software. Considerando el alcance del proyecto y su enfoque en la definición arquitectónica y funcional del sistema, se determinó que la inclusión de este nivel no aportaría significativamente a la comprensión global de la solución.

En su lugar, se priorizó una representación arquitectónica clara y comprensible, que permitiera evidenciar la organización del sistema y su comportamiento a nivel estructural, sin incurrir en un nivel de detalle que excediera los objetivos del proyecto.

4.2.9. Justificación del enfoque monolítico

La arquitectura monolítica fue seleccionada considerando el alcance y los objetivos del proyecto. Dado que se trata de un prototipo funcional, este enfoque permite:

- Reducir la complejidad del desarrollo inicial
- Facilitar la integración de funcionalidades
- Simplificar el despliegue y la configuración del sistema
- Mantener una gestión centralizada de la lógica de negocio

Aunque arquitecturas como microservicios ofrecen ventajas en términos de escalabilidad, estas implican un mayor nivel de complejidad que no resulta necesario en esta etapa del proyecto. No obstante, la solución se diseñó de forma modular, permitiendo su evolución hacia arquitecturas distribuidas en trabajos futuros.

4.2.10. Relación con los requerimientos del sistema

La arquitectura definida responde directamente a los requerimientos identificados en la sección anterior, garantizando la trazabilidad entre las necesidades del sistema y su implementación tecnológica.

En particular:

- Los requerimientos de visualización geoespacial son soportados por el frontend y su integración con servicios de mapas.
- El reporte colaborativo es gestionado por el backend, el cual procesa y valida la información enviada por los usuarios.
- La persistencia de datos se implementó mediante una base de datos relacional con soporte para información georreferenciada, permitiendo almacenar coordenadas, radios de influencia y rutas asociadas a los reportes del sistema.
- Los requerimientos de usabilidad y accesibilidad se reflejan en el diseño de la interfaz y la experiencia de usuario.

Esta alineación asegura que cada componente del sistema cumple una función específica dentro de la solución propuesta.

4.3. Modelo de datos con soporte de geolocalización

El diseño del modelo de datos constituye un componente fundamental dentro de la arquitectura del sistema, ya que permite estructurar de manera coherente la información asociada a los usuarios, sus interacciones y los eventos geolocalizados de riesgo. En este sentido, se adoptó un enfoque de modelado relacional normalizado hasta tercera forma normal (3FN), con el fin de garantizar consistencia, integridad y reducción de redundancias en el almacenamiento de datos.

El modelo integra tanto entidades de carácter funcional como estructuras orientadas a la representación espacial, permitiendo soportar la visualización y análisis de amenazas en un contexto geográfico urbano.

4.3.1. Enfoque general del modelo

El sistema se basa en un esquema centrado en la interacción entre usuarios y reportes de amenazas, donde cada evento registrado se encuentra asociado a una ubicación geográfica específica y a un conjunto de atributos que permiten su clasificación, interpretación y validación.

En este contexto, la entidad principal del modelo es HazardReport, la cual representa un evento de riesgo zoosanitario o zoosocial geolocalizado. Cada instancia contiene información espacial, temporal y descriptiva del incidente.

4.3.2. Representación del modelo relacional

Con el fin de representar de manera estructurada las tablas, atributos y relaciones que conforman el sistema, se elaboró el modelo relacional. Este modelo permitió formalizar la organización de los datos asociados a usuarios, reportes, comentarios, mascotas, rutas y catálogos de clasificación, así como evidenciar las relaciones necesarias para soportar la lógica colaborativa y geolocalizada de la plataforma.

La Figura 4.5 presenta el MR del sistema, en el cual se observa la relación entre los usuarios y los distintos tipos de reportes gestionados por la aplicación. En particular, el modelo incluye reportes de peligro, reportes de mascota perdida, reportes de mascota encontrada y reportes genéricos, todos asociados a información de ubicación mediante coordenadas geográficas. Asimismo, incorpora tablas de soporte como comentarios, niveles de riesgo, categorías, duraciones estimadas, rutas, puntos de ruta, mascotas y tokens de autenticación.

Este modelo también refleja la evolución funcional del sistema, al integrar no solo la gestión de amenazas geolocalizadas, sino también mecanismos de participación ciudadana y validación colaborativa, apoyados por servicios backend especializados y por almacenamiento local de información del usuario en los módulos donde aplica. De esta manera, el MR constituye una representación técnica de la estructura de información que soporta las funcionalidades principales del prototipo.

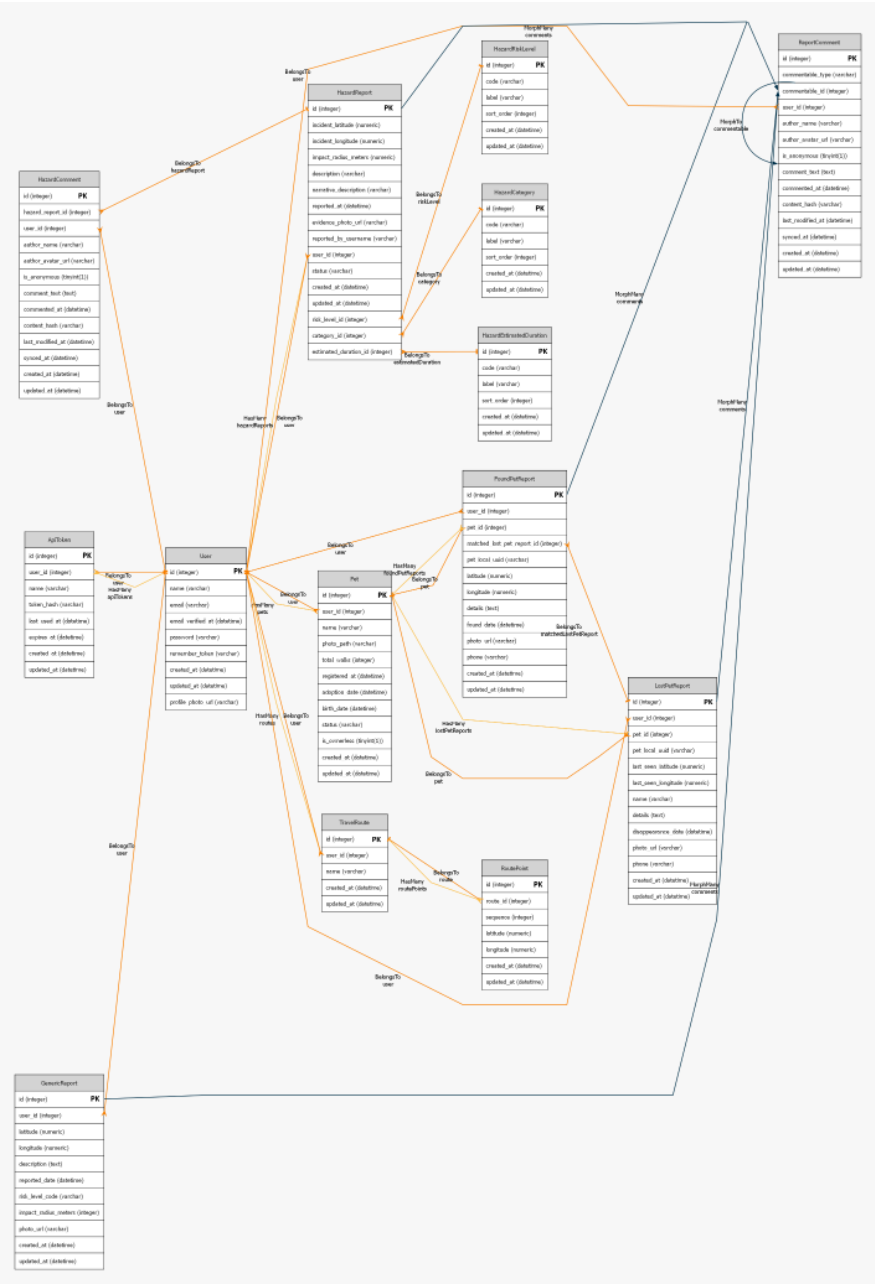


Figura 4.5: Modelo relacional del sistema

4.3.3. Representación geoespacial

Se utilizan coordenadas geográficas (latitud y longitud) para ubicar eventos, junto con un radio de impacto que define el área afectada. Esto permite representar tanto puntos como zonas de riesgo en el mapa.

4.3.4. Normalización y catálogos

Se incorporan tablas de catálogo para niveles de riesgo, categorías y duración estimada, lo que evita redundancia y mejora la consistencia de los datos.

4.3.5. Relaciones principales

Un usuario puede generar múltiples reportes, y cada reporte puede tener múltiples comentarios. Esto permite estructurar la interacción colaborativa del sistema.

4.3.6. Validación colaborativa

Los comentarios permiten validar la información mediante participación ciudadana, incluyendo anonimato y evidencia.

4.3.7. Persistencia local y conectividad

El modelo contempla mecanismos de persistencia local para almacenar información propia del usuario, como rutas, mascotas y configuraciones básicas. Asimismo, incorpora validaciones relacionadas con conectividad para gestionar adecuadamente funcionalidades que dependen de servicios remotos, visualización cartográfica y consulta de información comunitaria en tiempo real.

4.3.8. Extensibilidad

El modelo fue diseñado de manera modular, permitiendo la incorporación progresiva de nuevas funcionalidades relacionadas con monitoreo geoespacial, navegación asistida, gestión de amenazas y personalización de la experiencia de usuario.

4.4. Estrategia de participación ciudadana

El sistema propuesto se fundamenta en un enfoque de monitoreo colaborativo, en el cual los ciudadanos desempeñan un rol activo en la generación, validación y consulta de información relacionada con amenazas zoosanitarias, zoosociales y ambientales en la ciudad de Cali. Este enfoque se enmarca en el paradigma de *ciencia ciudadana (Citizen Science)*, el cual ha demostrado ser una estrategia efectiva para la recolección de datos en contextos donde la cobertura institucional es limitada o insuficiente.

Diversos estudios han evidenciado que la participación de ciudadanos en procesos de recolección de información permite ampliar significativamente la cobertura geográfica y temporal de los datos, facilitando la identificación de patrones que serían difíciles de detectar mediante métodos tradicionales[18].En particular, en el ámbito de la vigilancia epidemiológica y ambiental, la ciencia ciudadana ha sido utilizada como un mecanismo complementario que permite obtener información sobre eventos distribuidos en amplios territorios y en tiempo real.

4.4.1. Mecanismos de reporte colaborativo

El sistema incorpora un módulo de reporte colaborativo que permite a los usuarios registrar amenazas identificadas en el entorno urbano mediante formularios estructurados. Estos reportes recopilan información relevante para su posterior visualización y análisis, incluyendo la categoría de la amenaza, una descripción textual, el nivel de riesgo percibido, la ubicación geográfica y, de manera opcional, evidencia visual asociada al evento.

Este tipo de mecanismos ha sido utilizado en plataformas de monitoreo colaborativo, donde la participación ciudadana contribuye a la detección temprana de riesgos distribuidos en el territorio. Un ejemplo de ello se observa en sistemas de vigilancia de vectores transmisores de enfermedades, en los cuales los reportes realizados por ciudadanos permiten ampliar la cobertura de observación y complementar los mecanismos tradicionales de monitoreo [19]. De manera similar, el sistema propuesto busca aprovechar la capacidad de observación de los usuarios para fortalecer la identificación de amenazas en el contexto urbano de Cali.

Además, el uso de dispositivos móviles facilita la captura de datos geolocalizados, permitiendo asociar cada reporte con una posición específica en el mapa. Esta característica favorece el análisis de la distribución espacial de los eventos y fortalece los procesos de monitoreo, especialmente en escenarios donde la información puede variar según la ubicación y el momento en que se registra [20].

4.4.2. Validación de la información

Uno de los principales desafíos de los sistemas basados en participación ciudadana corresponde a la calidad y confiabilidad de la información reportada. Diversas investigaciones han señalado que los datos generados por ciudadanos pueden presentar inconsistencias, subjetividad o variaciones en precisión, debido a diferencias en la percepción, interpretación y experiencia de los usuarios [18].

En este contexto, el sistema incorpora estrategias de validación basadas en la interacción entre usuarios, tales como:

- Comentarios sobre reportes existentes
- Visualización pública de los eventos

Este enfoque se inspira en modelos de validación colaborativa, en los cuales la comunidad contribuye a mejorar la calidad de los datos mediante procesos de retroalimentación continua.

4.4.3. Importancia de la interacción social en la validación

La inclusión de un sistema de comentarios dentro de cada amenaza responde a la necesidad de enriquecer la información y permitir procesos de verificación colectiva. A través de estos mecanismos, los usuarios pueden confirmar, complementar o cuestionar la información reportada, generando un proceso dinámico de construcción de conocimiento.

Este tipo de interacción transforma los reportes individuales en información más robusta, disminuyendo la incertidumbre asociada a los datos iniciales y fortaleciendo la confiabilidad del sistema.

4.4.4. Beneficios del enfoque participativo

El uso de estrategias de participación ciudadana aporta múltiples beneficios al sistema:

- Mayor cobertura territorial
- Actualización en tiempo real
- Reducción de costos
- Fortalecimiento de la conciencia ciudadana

Se ha demostrado que este tipo de enfoques permite identificar áreas de riesgo no detectadas por sistemas tradicionales, complementando la información institucional existente [21].

4.4.5. Rol del usuario en el sistema

Dentro del sistema propuesto, los ciudadanos dejan de ser únicamente receptores de información para convertirse en actores activos en la construcción del conocimiento. A través de la plataforma, los usuarios pueden:

- Reportar amenazas identificadas en su entorno
- Complementar información existente mediante comentarios
- Consultar eventos cercanos a su ubicación
- Contribuir a la actualización dinámica de la información

Este enfoque participativo se alinea con el uso de aplicaciones móviles en iniciativas de ciencia ciudadana, donde los usuarios actúan como sensores distribuidos capaces de capturar información contextual en tiempo real [22].

4.4.6. Limitaciones y consideraciones

A pesar de sus ventajas, el enfoque colaborativo presenta desafíos que deben ser considerados, tales como la variabilidad en la calidad de los datos, la posible concentración de reportes en ciertas zonas y la participación desigual entre usuarios.

Por esta razón, el sistema se plantea como un complemento a las fuentes institucionales, integrando información oficial con datos generados por los ciudadanos para lograr una visión más completa del entorno.

4.4.7. Integración con el sistema propuesto

La estrategia de participación ciudadana se encuentra directamente integrada con los componentes del sistema, particularmente con el módulo de reporte, la visualización geoespacial, los mecanismos de alerta y el sistema de comentarios.

Esta integración permite establecer un flujo continuo de información desde los usuarios hacia la plataforma y de regreso hacia la comunidad, fortaleciendo los procesos de monitoreo y toma de decisiones.

4.4.8. Flujo de comunicación y monitoreo

El sistema propuesto se estructura a partir de un flujo continuo de comunicación y procesamiento de la información, el cual permite transformar los reportes ciudadanos en datos útiles para la visualización, análisis y generación de alertas en tiempo real. Este flujo se compone de cuatro etapas principales: reporte ciudadano, validación, visualización y alerta, las cuales se encuentran interconectadas y permiten el funcionamiento integral de la plataforma.

En primer lugar, el proceso inicia con el reporte ciudadano, mediante el cual un usuario registra una amenaza identificada en su entorno a través del aplicativo. Este reporte se realiza mediante un formulario estructurado que permite capturar información relevante como la categoría de la amenaza, una descripción, el nivel de riesgo percibido, la ubicación geográfica y, de manera opcional, evidencia visual como fotografías. Esta etapa constituye el punto de entrada de la información al sistema y es fundamental para garantizar la actualización constante de los datos.

Posteriormente, se desarrolla la fase de validación de la información, la cual se basa en un enfoque colaborativo. A diferencia de sistemas tradicionales que dependen exclusivamente de validaciones institucionales, el sistema propuesto incorpora mecanismos de validación social que permiten a otros usuarios interactuar con los reportes. Esta validación se materializa principalmente a través de dos elementos: los comentarios y la evidencia visual asociada al reporte.

Por un lado, los comentarios permiten que otros usuarios corroboren, amplíen o cuestionen la información registrada, generando un proceso de retroalimentación que contribuye a mejorar la calidad de los datos. Por otro lado, las fotografías asociadas al reporte permiten ofrecer evidencia directa del evento, facilitando su verificación por parte de la comunidad. De esta manera, la validación no se concibe como un proceso único o centralizado, sino como una dinámica distribuida que se construye a partir de la interacción entre los usuarios.

Una vez que la información ha sido registrada y comienza a ser validada, se da paso a la etapa de visualización, en la cual los reportes son representados en el mapa de la ciudad de Cali mediante elementos gráficos georreferenciados. Cada amenaza es ubicada según sus coordenadas y puede ser consultada por los usuarios, quienes pueden acceder a su información detallada, incluyendo descripción, nivel de riesgo, comentarios y evidencia asociada.

Esta visualización permite no solo identificar eventos individuales, sino también reconocer patrones espaciales y zonas con mayor concentración de riesgos, facilitando una comprensión más integral del entorno.

Finalmente, el flujo culmina con la etapa de alerta, en la cual el sistema notifica al usuario cuando

se encuentra dentro del radio de influencia de una amenaza reportada. Estas alertas se generan a partir de la relación entre la ubicación del usuario y la localización de los eventos registrados, permitiendo advertir de manera oportuna sobre posibles riesgos en el entorno inmediato.

Este mecanismo de alerta transforma la información en una herramienta preventiva, ya que no solo informa sobre eventos pasados, sino que permite anticipar situaciones de riesgo durante el desplazamiento del usuario en la ciudad.

En conjunto, este flujo de comunicación y monitoreo establece un ciclo continuo en el cual la información es generada, validada, visualizada y utilizada para la toma de decisiones, fortaleciendo el carácter dinámico, participativo y contextual del sistema propuesto. Además, este enfoque permite integrar de manera efectiva la colaboración ciudadana con herramientas tecnológicas, contribuyendo a la construcción de un sistema de monitoreo urbano más accesible, actualizado y alineado con las necesidades reales del entorno.

4.5. Diseño de interfaz y experiencia de usuario (UX/UI)

El diseño de la interfaz y la experiencia de usuario del sistema se desarrolló con el objetivo de garantizar una interacción clara, intuitiva y eficiente, considerando las características del público objetivo y el contexto de uso en entornos urbanos. Dado que el sistema se basa en la consulta y reporte de información geolocalizada, el diseño se centró en priorizar la visualización del mapa como elemento principal de interacción.

Para la construcción de los prototipos de interfaz se utilizó la herramienta Figma, la cual permitió diseñar y validar de manera iterativa las diferentes pantallas del sistema, así como evaluar la organización de los elementos visuales y la navegabilidad entre secciones.

4.5.1. Principios de diseño

El diseño de la interfaz se fundamentó en los siguientes principios:

- **Simplicidad:** minimizar la carga cognitiva del usuario, evitando saturación de información en pantalla.
- **Claridad visual:** uso de elementos gráficos comprensibles que faciliten la interpretación de la información.
- **Accesibilidad:** garantizar que la interfaz sea usable en diferentes condiciones de visualización.
- **Consistencia:** mantener una estructura uniforme en todas las pantallas del sistema.
- **Enfoque contextual:** priorizar la información relevante según la ubicación del usuario.

Estos principios permiten que el sistema sea fácilmente utilizable por diferentes tipos de usuarios, independientemente de su nivel de experiencia tecnológica.

4.5.2. Estructura de la interfaz

La interfaz del sistema se organiza en torno a una pantalla principal centrada en el mapa, desde la cual se accede a las funcionalidades principales del sistema. Esta pantalla integra:

- Visualización de amenazas georreferenciadas
- Ubicación del usuario (cuando está disponible)
- Acceso al reporte de eventos
- Navegación hacia módulos adicionales (configuración, rutas, mascotas)

El mapa actúa como el eje central de la interacción, permitiendo que el usuario interprete la información de manera espacial y contextual.

4.5.3. Diseño del flujo de interacción

El diseño de la experiencia de usuario se alinea directamente con el flujo de comunicación definido previamente (reporte → validación → visualización → alerta). En este sentido:

- El proceso de reporte se realiza mediante formularios simples y estructurados
- La visualización se presenta de manera directa en el mapa
- La validación se facilita mediante la interacción con comentarios
- Las alertas se integran como elementos contextuales durante la navegación

Esta coherencia entre el flujo del sistema y la interfaz permite reducir la fricción en la interacción y mejorar la experiencia del usuario.

4.5.4. Prototipos de interfaz

Los prototipos de la interfaz fueron desarrollados en Figma, permitiendo simular la interacción del usuario con el sistema antes de su implementación. Estos prototipos incluyen las principales pantallas del sistema, entre las cuales se destacan:

- **Pantalla principal con mapa**

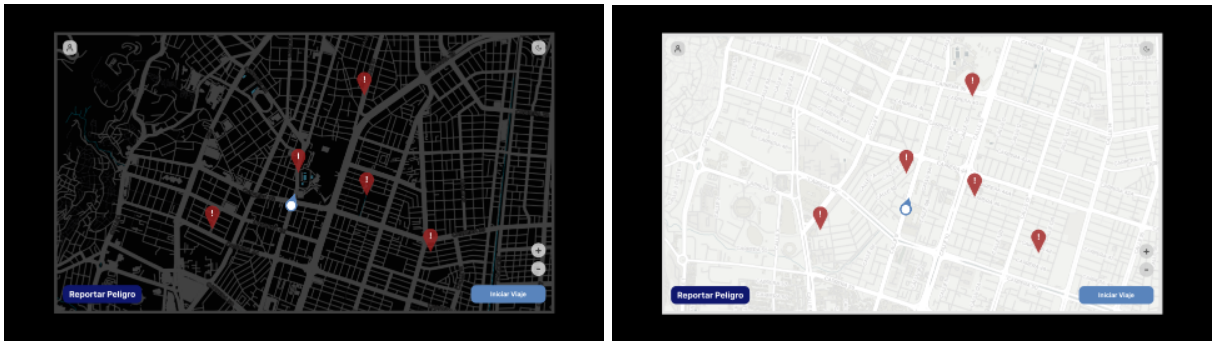


Figura 4.6: Interfaces correspondientes a la navegación principal y visualización cartográfica.

- **Vista detallada de un evento**

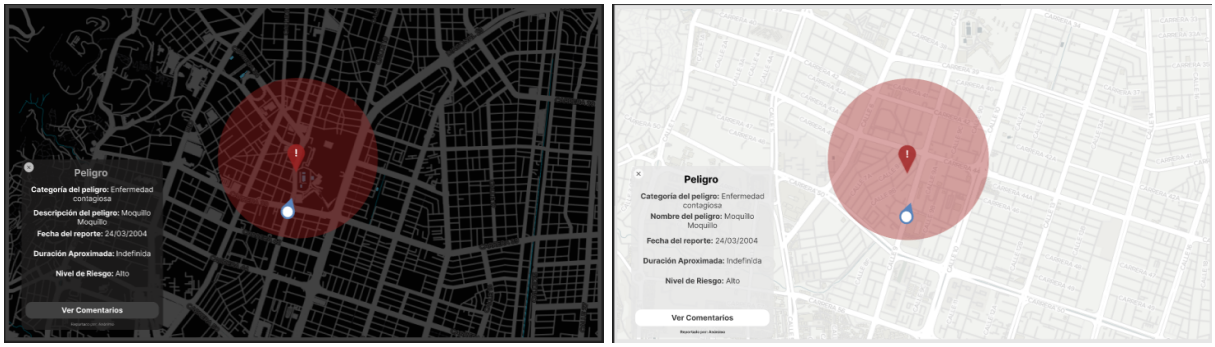


Figura 4.7: Interfaces correspondientes a la vista detallada de la información de un vector en el mapa

- **Gestión de Rutas**

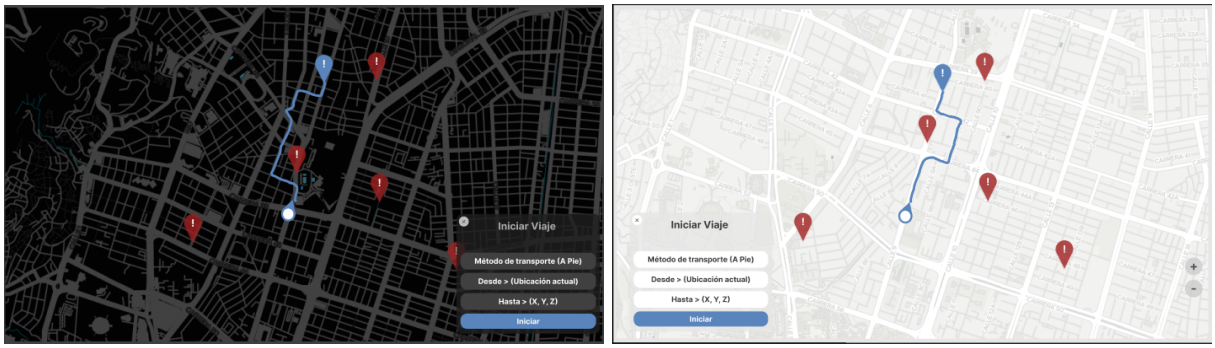


Figura 4.8: Interfaces correspondientes a la vista de inicio de la gestión de rutas.

Los prototipos diseñados se incluyen en los anexos del documento, permitiendo evidenciar la estructura visual y funcional del sistema. (Véase el apéndice B).

4.5.5. Modo claro y modo oscuro

Con el fin de mejorar la accesibilidad y adaptabilidad del sistema a diferentes condiciones de uso, se incorporó la posibilidad de alternar entre modo claro y modo oscuro. Esta funcionalidad permite:

- Mejorar la visibilidad en condiciones de alta luminosidad (modo claro)
- Reducir la fatiga visual en entornos con poca iluminación (modo oscuro)
- Ofrecer una experiencia personalizada según las preferencias del usuario

El cambio entre modos se realiza mediante un control accesible desde la interfaz principal, garantizando una transición inmediata y consistente en toda la aplicación.

4.5.6. Consideraciones de usabilidad

El diseño de la interfaz tuvo en cuenta aspectos clave de usabilidad, tales como:

- Reducción del número de pasos para realizar un reporte
- Uso de iconografía clara para representar amenazas
- Ubicación estratégica de botones de acción
- Priorización de información relevante en pantalla

Con esto se buscó que la interacción con el sistema fuera lo más natural posible, permitiendo que el usuario comprenda rápidamente cómo navegar, reportar y consultar información sin necesidad de instrucciones complejas.

4.5.7. Relación con los requerimientos del sistema

El diseño de la interfaz se encuentra directamente alineado con los requerimientos funcionales definidos, especialmente aquellos relacionados con:

- Visualización del mapa
- Interacción con amenazas
- Reporte de eventos
- Navegación entre módulos

Esta correspondencia garantiza que la interfaz no solo cumpla una función estética, sino que responda de manera efectiva a las necesidades del sistema.

4.6. Conclusión del capítulo

En este capítulo se desarrolló el proceso de análisis y diseño del sistema propuesto, estableciendo las bases conceptuales, funcionales y tecnológicas necesarias para la construcción del prototipo. A partir del diagnóstico de las limitaciones existentes en el acceso y difusión de información sobre amenazas en el entorno urbano de Cali, se identificó la necesidad de una solución que integre múltiples fuentes de datos y facilite su interpretación mediante herramientas accesibles para la ciudadanía.

En este sentido, se definieron los requerimientos funcionales y no funcionales del sistema, garantizando que estos fueran claros, medibles y alineados con los objetivos del proyecto. Estos requerimientos permitieron estructurar las funcionalidades principales del sistema, incluyendo la visualización de información geolocalizada, el reporte colaborativo de amenazas y la generación de alertas contextuales.

Posteriormente, se diseñó la arquitectura del sistema bajo un enfoque monolítico, adecuado para el alcance del prototipo, permitiendo integrar de manera coherente los componentes de interfaz, lógica de negocio y almacenamiento de datos. Así mismo, se estableció un modelo de datos que incorpora información descriptiva y de geolocalización, facilitando la representación de los eventos dentro del mapa y su consulta por parte de los usuarios.

Adicionalmente, se definió una estrategia de participación ciudadana basada en principios de ciencia colaborativa, en la cual los usuarios desempeñan un rol activo en la generación y validación de la información. Este enfoque se materializa mediante mecanismos como el reporte de eventos, la inclusión de evidencia visual y el uso de comentarios, los cuales permiten mejorar la calidad y confiabilidad de los datos a través de la interacción entre los usuarios.

De igual forma, se estructuró el flujo de comunicación del sistema, estableciendo un proceso continuo que abarca desde el reporte ciudadano hasta la generación de alertas, pasando por etapas de validación y visualización. Este flujo garantiza que la información no solo sea registrada, sino también procesada y utilizada de manera efectiva para apoyar la toma de decisiones en el entorno urbano.

Finalmente, se diseñó la interfaz de usuario, priorizando la claridad, la accesibilidad y la interacción centrada en el mapa como elemento principal. La incorporación de prototipos desarrollados en Figma, así como la inclusión de funcionalidades como el modo claro y oscuro, permiten evidenciar un enfoque orientado a la usabilidad y la experiencia del usuario.

En conjunto, los elementos desarrollados en este capítulo consolidan una propuesta de sistema coherente, viable y alineada con las necesidades identificadas en el contexto de estudio. Este diseño integral constituye la base sobre la cual se desarrollará la etapa de implementación del prototipo, en la cual se materializarán las funcionalidades definidas y se evaluará su desempeño en un entorno controlado.

Capítulo 5

Etapa 3: Construcción e Implementación del Prototipo

Introducción

La etapa de construcción y prototipado correspondió a la materialización progresiva de los componentes definidos durante las fases de análisis y diseño, permitiendo consolidar un prototipo funcional orientado al monitoreo colaborativo de amenazas zoosanitarias urbanas. En esta fase se implementaron e integraron los distintos módulos del sistema, incorporando capacidades de geolocalización, persistencia local, servicios backend especializados y visualización interactiva de información geoespacial.

El desarrollo se llevó a cabo bajo un enfoque incremental, iniciando desde una base mínima funcional y evolucionando gradualmente hacia una arquitectura más compleja. Este proceso permitió validar decisiones de diseño en contextos reales de uso, identificar limitaciones técnicas y ajustar la solución de manera iterativa hasta alcanzar un sistema coherente, modular y escalable.

5.1. Persistencia de datos y arquitectura híbrida

El desarrollo del sistema incorporó desde etapas tempranas mecanismos de persistencia local; sin embargo, el prototipo no se concibe como una plataforma de operación completamente offline. En su lugar, se consolidó una arquitectura híbrida con persistencia local parcial, orientada principalmente al almacenamiento local de información propia del usuario (por ejemplo, mascotas, rutas y configuraciones), mientras que múltiples funcionalidades dependen de conectividad requerida hacia servicios backend especializados.

Sobre esta base, se diseñó un componente centralizado de acceso a datos que permitió desacoplar la lógica de negocio de la capa de persistencia, facilitando la implementación de operaciones CRUD sobre las entidades principales del sistema. Este diseño permitió estructurar de manera ordenada la manipulación de datos locales y, al mismo tiempo, preparar la arquitectura para el consumo de endpoints remotos cuando existe conectividad.

Es importante precisar que la visualización del mapa no se encuentra disponible sin conexión. Esto se debe a que el servicio cartográfico, basado en OpenStreetMap, depende de la carga dinámica de teselas (*tiles*) desde servidores externos, lo cual requiere conectividad a internet.

En este sentido, por alcance del proyecto, no se implementó un mecanismo de almacenamiento local de mapas ni un sistema de caché avanzado de teselas, ya que esto implicaría un incremento

significativo en la complejidad del sistema y en los requerimientos de almacenamiento del dispositivo.

5.2. Desarrollo del sistema de visualización geográfica

El mapa interactivo se consolidó como el componente central de la interfaz del sistema, implementado mediante OpenStreetMap y librerías especializadas en Flutter. En una etapa inicial, el mapa funcionaba únicamente como un elemento visual; sin embargo, su funcionalidad se amplió progresivamente hasta convertirse en el eje principal de interacción del usuario.

Figura 5.1: Primera implementación funcional del mapa interactivo basado en OpenStreetMap dentro del sistema

Como se observa en la Figura 5.1, la primera versión del mapa aún no incorporaba representación de peligros ni funcionalidades avanzadas de interacción.

Se implementaron mecanismos de representación geoespacial que permitieron visualizar eventos mediante marcadores y áreas de influencia, estas últimas modeladas como geometrías circulares cuyo comportamiento depende del nivel de riesgo asociado al evento.

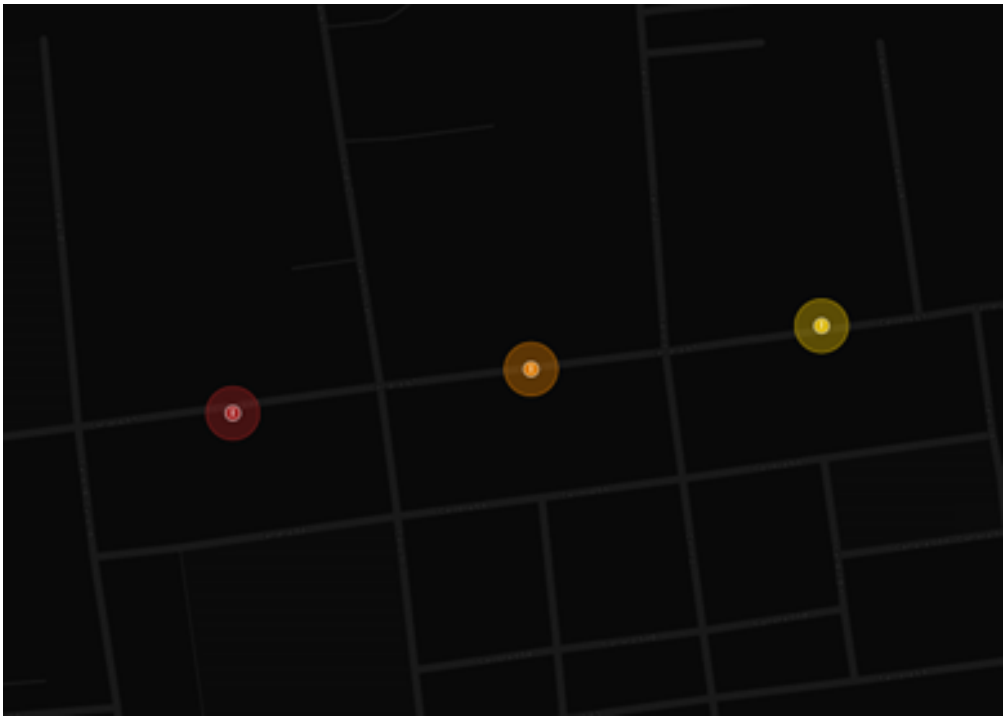


Figura 5.2: Representación de los eventos mediante marcadores y áreas de influencia

Gracias a la representación de la Figura 5.2, se permitió no solo ubicar amenazas en el territorio, sino también comunicar su impacto de manera intuitiva. Al mismo tiempo, se incorporaron funcionalidades de navegación controlada, centrado en la ubicación del usuario y actualización dinámica de los elementos visuales, mejorando la experiencia de interacción con el entorno geográfico.

5.3. Desarrollo del sistema de rutas y cálculo de trayectorias

Durante el desarrollo del sistema, se identificó una limitación en las soluciones de enrutamiento convencionales, las cuales no permitían modificar dinámicamente los criterios de cálculo de rutas, en particular la capacidad de evitar zonas de riesgo.

Para superar esta limitación, se implementó una solución basada en GraphHopper, desplegada mediante contenedores Docker. Esta decisión permitió tener control sobre el algoritmo de enrutamiento, incorporando parámetros personalizados como la evasión de áreas peligrosas y el método de transporte.

Como resultado, el sistema no solo permite calcular rutas, sino también adaptarlas en función de las condiciones del entorno, integrando la información geoespacial generada por los usuarios

dentro del proceso de navegación.

5.4. Implementación del módulo de mascotas

Como parte de la evolución funcional del sistema, se desarrolló un módulo orientado a la gestión de mascotas registradas por el usuario. Este componente permitió almacenar información básica asociada a los animales domésticos, incluyendo nombre, fecha de nacimiento, fecha de adopción y fotografía opcional, como se puede observar en la Figura 5.3

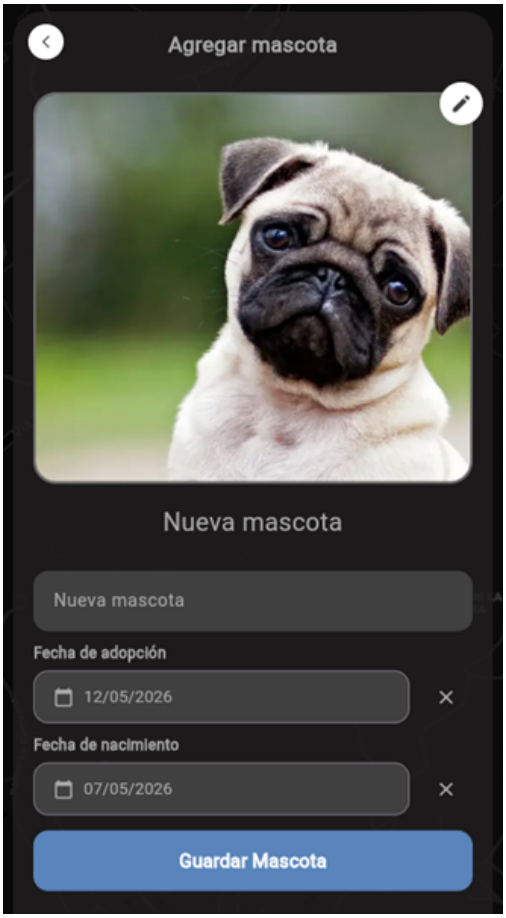


Figura 5.3: Visualización de la interfaz móvil del módulo de mascotas

Aunque inicialmente este módulo podría interpretarse únicamente como un mecanismo de personalización dentro de la aplicación, su incorporación respondió también a una estrategia orientada al fortalecimiento del vínculo entre el usuario y la plataforma. La posibilidad de registrar mascotas propias permitió contextualizar las funcionalidades del sistema dentro de la experiencia cotidiana de cuidado animal, incentivando la participación continua de los usuarios y aumentando el nivel de apropiación sobre las herramientas de monitoreo y prevención implementadas.

Desde el punto de vista técnico, este módulo representó una de las primeras implementaciones completas de operaciones CRUD persistentes sobre la base de datos local, permitiendo validar el funcionamiento de la persistencia local parcial (para información propia del usuario) y consolidar la estructura de almacenamiento utilizada posteriormente por otros componentes del sistema.

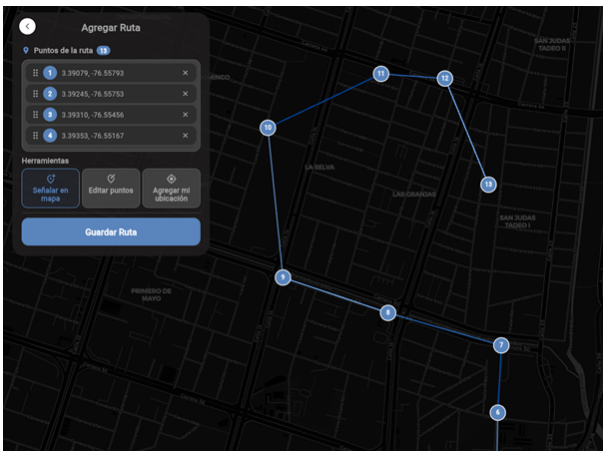
De la misma manera, el módulo incorporó mecanismos de almacenamiento de imágenes mediante referencias locales y estructuras desacopladas de persistencia, sentando las bases para el manejo posterior de evidencia fotográfica dentro de los reportes de amenazas.

5.5. Implementación del módulo de rutas

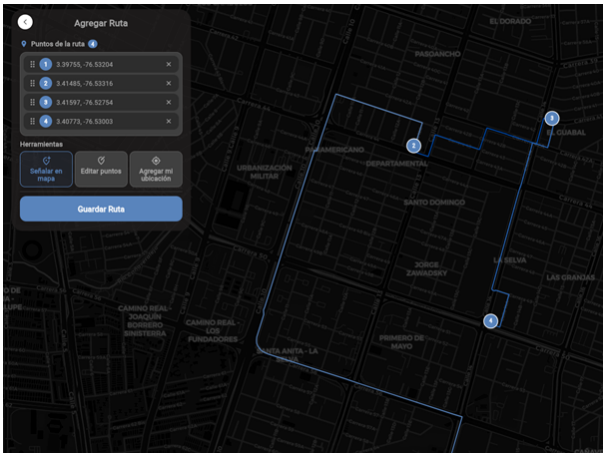
Posteriormente, se desarrolló el módulo de gestión de rutas, el cual permitió a los usuarios crear, almacenar y reutilizar trayectorias geográficas dentro del mapa de la ciudad de Cali. Estas rutas se definieron mediante secuencias de puntos geográficos y fueron integradas directamente sobre la representación cartográfica del sistema.

Inicialmente, el módulo permitió únicamente almacenar recorridos definidos manualmente por el usuario. Sin embargo, a medida que evolucionó la arquitectura del sistema, este componente adquirió un rol central dentro de la lógica preventiva de la plataforma.

La incorporación posterior del sistema de peligros transformó funcionalmente el propósito de las rutas, ya que estas dejaron de representar únicamente trayectorias geográficas para convertirse en mecanismos de navegación asistida orientados a la evasión de zonas de riesgo.



(a) Creación inicial de rutas mediante selección manual de puntos geográficos



(b) Visualización del trazado generado sobre el mapa interactivo

Figura 5.4: Implementación del módulo de gestión de rutas dentro del sistema

Como se observa en la Figura 5.4a, las primeras versiones del módulo permitían definir trayectorias manuales mediante puntos geográficos seleccionados directamente sobre el mapa. La Figura 5.4b evidencia la representación visual del trazado generado dentro del entorno cartográfico de la aplicación.

Esta necesidad llevó a integrar servicios especializados de cálculo de rutas mediante GraphHopper, permitiendo generar recorridos optimizados considerando la ubicación de amenazas reportadas por la comunidad. Como resultado, el sistema fue capaz de calcular rutas que evitaran zonas peligrosas definidas dentro del mapa, fortaleciendo el carácter preventivo y contextual de la aplicación.

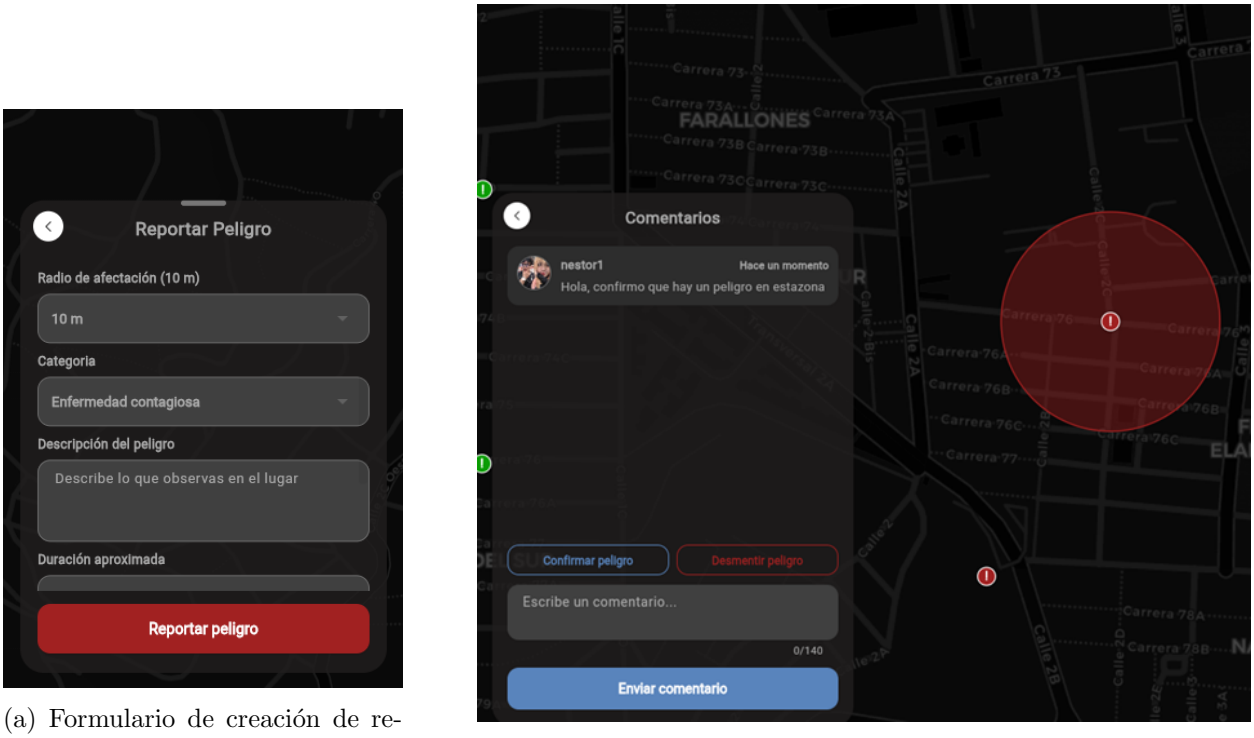
Desde una perspectiva arquitectónica, este módulo también permitió consolidar la interacción entre persistencia local, representación geoespacial y servicios externos de procesamiento de rutas, convirtiéndose en uno de los componentes más complejos dentro de la evolución funcional del sistema.

5.6. Implementación del módulo de reporte colaborativo

La implementación del módulo de reportes de amenazas representó una transición clave en la arquitectura del sistema, al introducir por primera vez un modelo de información compartida entre múltiples usuarios.

Este módulo permitió registrar peligros geolocalizados vinculados a amenazas zoosanitarias urbanas, mediante atributos estructurados como categoría, nivel de riesgo, duración estimada, descripción textual, radio de influencia y evidencia fotográfica. A diferencia de los módulos de mascotas y rutas, cuya información se encuentra asociada principalmente al usuario que la registra, los peligros reportados fueron concebidos como información de carácter compartido, disponible para la comunidad dentro del sistema. Esta decisión implicó la necesidad de integrar un servidor de backend que permitiera centralizar la información y habilitar consulta/actualización de reportes comunitarios cuando existe conectividad. Como resultado, se consolidó una arquitectura híbrida: se mantiene persistencia local parcial para información del usuario, mientras que la información comunitaria y la consulta de amenazas dependen de servicios backend especializados.

La Figura 5.5 presenta algunas de las interfaces implementadas para el registro y validación colaborativa de amenazas geolocalizadas dentro del mapa interactivo.



(a) Formulario de creación de reportes de peligro

(b) Visualización del peligro y sistema de comentarios colaborativos

Figura 5.5: Implementación del módulo colaborativo de reportes de amenazas

Como se observa en la Figura 5.5a, el sistema permitió registrar peligros mediante formularios estructurados con atributos asociados al riesgo y ubicación geográfica. Como extensión de este módulo, se incorporó un sistema de comentarios asociado a cada reporte, permitiendo validar, complementar o cuestionar la información, fortaleciendo así la confiabilidad del sistema a través de la interacción entre usuarios.

5.7. Selección de tecnologías y herramientas

Durante el proceso de construcción del sistema, se realizó la selección de las tecnologías utilizadas con base en criterios de compatibilidad, escalabilidad, facilidad de integración y alineación con los requerimientos definidos en etapas anteriores.

Para el desarrollo del frontend se utilizó Flutter, debido a su capacidad para construir interfaces multiplataforma con un único código base, así como su integración con librerías de visualización geográfica como `flutter_map`, lo que facilitó la implementación del componente cartográfico basado en OpenStreetMap.

En cuanto a la persistencia local, se seleccionó SQLite como motor de base de datos embebido, debido a su ligereza, bajo consumo de recursos y soporte nativo en dispositivos móviles, lo cual resulta adecuado para implementar persistencia local parcial de información propia del usuario.

Para la gestión de la información compartida, se implementó un backend utilizando el framework Laravel en PHP. Esta elección se fundamentó en su robustez para el desarrollo de APIs REST, su estructura modular y su facilidad para gestionar autenticación, control de acceso y persistencia de datos en entornos distribuidos.

Adicionalmente, para el cálculo de rutas se incorporó GraphHopper, desplegado mediante contenedores Docker, lo que permitió controlar la lógica de enrutamiento y adaptarla a las necesidades específicas del sistema, particularmente en lo relacionado con la evasión de zonas de riesgo.

En conjunto, estas herramientas permitieron construir una arquitectura híbrida que combina procesamiento local y servicios backend especializados, alineada con los objetivos funcionales y no funcionales del proyecto.

5.8. Integración backend y consumo de servicios remotos

Con la introducción de datos compartidos, el sistema evolucionó hacia una arquitectura distribuida. Se implementó un servidor de backend basado en Laravel encargado de gestionar la persistencia global de la información, incluyendo reportes de amenazas, comentarios y autenticación de usuarios.

En este contexto, la persistencia local se mantuvo como un mecanismo acotado al almacenamiento local de información propia del usuario. Por su parte, la consulta, publicación y actualización de información comunitaria se realiza mediante endpoints del backend, lo que introduce conectividad requerida para módulos como reportes, comentarios y consulta de amenazas.

De esta manera, el prototipo no se apoya en un mecanismo automático complejo de reconciliación entre almacenamiento local y remoto como eje principal; en su lugar, combina persistencia local parcial con acceso bajo demanda a servicios backend especializados, de acuerdo con la disponibilidad de red.

5.9. Integración de módulos y consolidación progresiva del prototipo

En etapas posteriores del desarrollo, los distintos módulos implementados fueron integrados progresivamente en una experiencia unificada. Esta integración no consistió únicamente en agrupar funcionalidades aisladas, sino en articular los componentes de interfaz, persistencia local, visualización geográfica y lógica de rutas dentro de un mismo flujo de uso.

Uno de los primeros elementos que permitió estructurar esta integración fue el sistema de paneles modulares. Este componente se diseñó como una base reutilizable para presentar funcionalidades mediante paneles tipo modal, manteniendo una organización visual consistente dentro de la aplicación. A través de este mecanismo se gestionó la navegación entre vistas internas, la apertura y cierre de paneles, el control del estado visual y la adaptación de la interfaz a diferentes tamaños de pantalla. Esta decisión permitió evitar la duplicación de estructuras de interfaz y facilitó la incorporación progresiva de nuevos módulos, como mascotas, rutas, reportes y configuración.

Posteriormente, se implementó el módulo de gestión de mascotas, el cual permitió validar el funcionamiento completo de la persistencia local. Este módulo incorporó operaciones de creación, visualización, actualización y eliminación de registros, asociando cada mascota con datos básicos como nombre e imagen. Para el manejo de imágenes se utilizó inicialmente un sistema de almacenamiento local, de manera que la aplicación pudiera conservar referencias a los archivos asociados en el dispositivo. Este componente permitió comprobar que la persistencia local parcial es viable para información propia del usuario y consolidó el patrón de acceso a datos mediante SQLite.

La introducción del módulo de reportes de peligros generó una primera expansión significativa del sistema, ya que este tipo de información no podía permanecer limitada al dispositivo del usuario. A diferencia de las mascotas o rutas locales, los peligros reportados debían ser visibles para la comunidad, por lo que fue necesario integrar un backend que permitiera almacenar y distribuir información compartida. En esta etapa se incorporó un servidor basado en Laravel para gestionar reportes, comentarios y posteriormente autenticación de usuarios. Esta decisión marcó la transición desde una aplicación local hacia un sistema híbrido, capaz de combinar persistencia local parcial con servicios backend especializados.

El backend evolucionó de manera progresiva conforme aumentaron las necesidades funcionales del sistema. En un primer momento, su propósito principal fue recibir y devolver reportes de peligros. Luego, se amplió para soportar comentarios asociados a cada reporte, permitiendo fortalecer la validación colaborativa de la información. Más adelante, se incorporó la autenticación, lo que permitió asociar la información del usuario con una cuenta remota y acceder a servicios protegidos. Esta evolución permitió mantener una arquitectura incremental, donde cada expansión respondió a una necesidad funcional identificada durante el desarrollo.

El sistema de rutas también representó un componente relevante dentro de la consolidación del prototipo. Inicialmente, las rutas se gestionaron como secuencias de puntos geográficos seleccionados por el usuario dentro del mapa. Cada ruta podía estar compuesta por un conjunto ordenado de ubicaciones, lo que permitía construir trayectos básicos y almacenarlos para su posterior consulta. Sin embargo, al integrar el módulo de peligros, surgió la necesidad de calcular trayectorias que no solo conectaran un punto de origen con un destino, sino que también consideraran la presencia de zonas de riesgo.

Posteriormente, el sistema evolucionó hacia un esquema de planificación de viajes asistida por peligros, permitiendo que el usuario seleccionara criterios de navegación según el nivel de riesgo presente en el entorno. Para ello, se incorporaron filtros asociados a categorías de peligros y niveles de severidad, integrando la visualización simultánea de amenazas dentro del trayecto generado. De esta manera, el sistema no solo permite definir rutas, sino también apoyar procesos de movilidad considerando condiciones zoosanitarias reportadas por la comunidad.

La Figura 5.6 presenta un ejemplo del módulo de planificación de viaje, donde se visualiza una ruta generada junto con distintos peligros distribuidos sobre el mapa y los filtros configurables asociados al recorrido.

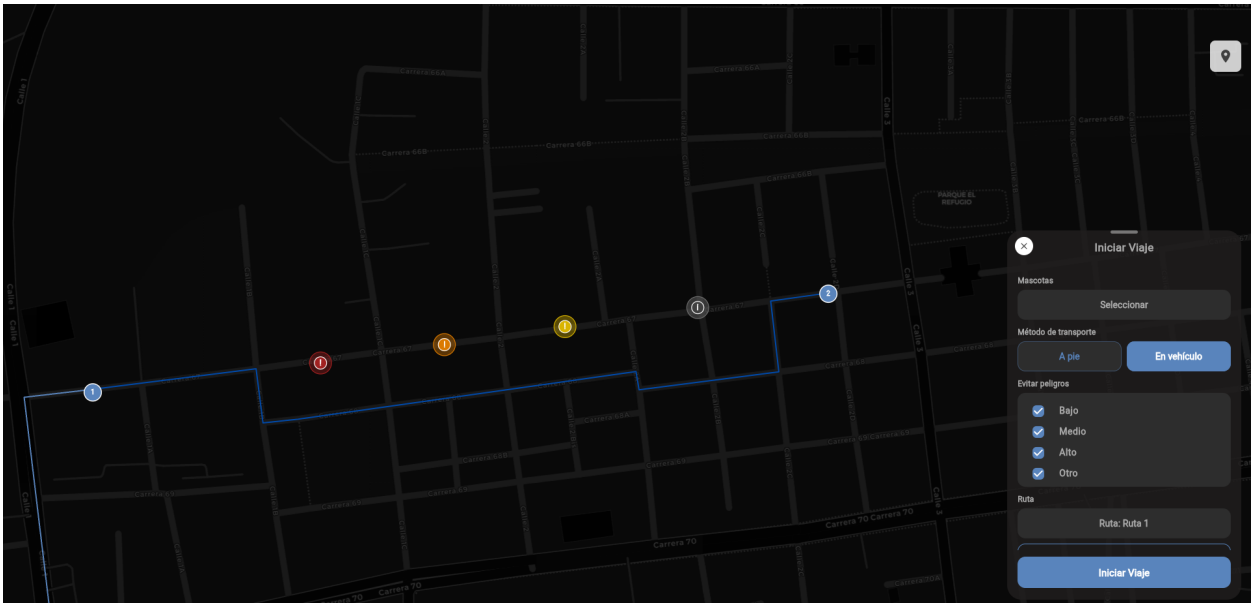
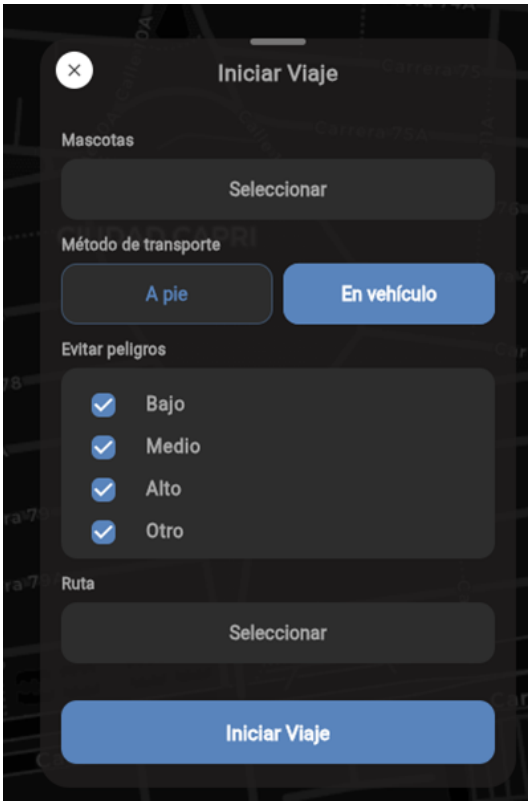


Figura 5.6: Módulo de planificación de viaje con visualización de peligros y filtros de navegación.

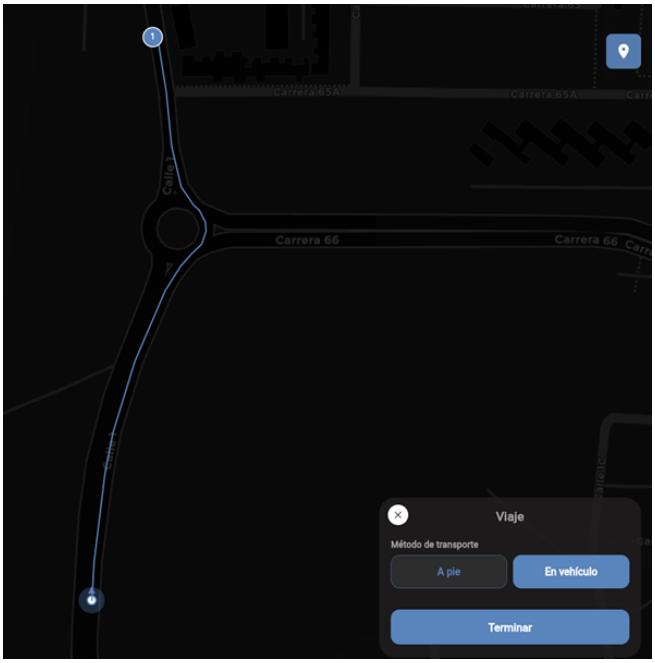
Para resolver esta necesidad, se incorporó GraphHopper como motor de enrutamiento desplegado mediante Docker. Esta integración permitió delegar el cálculo de rutas a un servicio especializado y controlar parámetros relevantes para el sistema, como el método de transporte y la posibilidad de evitar áreas peligrosas. De esta manera, las rutas dejaron de ser únicamente recorridos almacenados por el usuario y pasaron a formar parte de una lógica de navegación contextual, en la cual la información de peligros influye en la generación del trayecto.

Posteriormente, se desarrolló el módulo de viaje, el cual integró funcionalidades previamente construidas. Este módulo permitió configurar un recorrido a partir de un origen, un destino, una ruta guardada o una ruta generada, considerando además la información de peligros disponible en el mapa. Durante este proceso, el sistema articuló la visualización geográfica, la ubicación del usuario, los reportes de peligro y la lógica de rutas, consolidando un flujo orientado al acompañamiento del usuario durante sus desplazamientos.

La Figura 5.7 presenta algunas de las interfaces implementadas para la configuración y ejecución de recorridos dentro del sistema, incluyendo la selección de parámetros de navegación y la visualización activa del trayecto sobre el mapa interactivo.



(a) Configuración inicial de parámetros para el inicio de un viaje



(b) Visualización activa de un recorrido sobre el mapa interactivo

Figura 5.7: Implementación del módulo de viaje y navegación asistida

Como se observa en la Figura 5.7a, el sistema permitió configurar parámetros asociados al recorrido, incluyendo método de transporte, selección de rutas y criterios de evasión de peligros. Por otra parte, la Figura 5.7b evidencia la representación visual del recorrido activo sobre el mapa, integrando navegación geoespacial y seguimiento contextual del usuario. Finalmente, se implementó el sistema de autenticación con backend, una vez que los módulos principales ya habían sido construidos y validados de forma local o híbrida. Esta decisión permitió integrar la gestión de cuentas sin interrumpir el desarrollo inicial de funcionalidades. A partir de la autenticación, la información propia del usuario almacenada localmente pudo asociarse a un usuario autenticado, y la aplicación pudo habilitar el acceso a endpoints protegidos. Del mismo modo, al cerrar sesión, el sistema pudo aislar la información local para evitar inconsistencias entre usuarios y mantener coherencia con el estado remoto gestionado por el backend.

Como resultado de este proceso, se consolidó una arquitectura híbrida compuesta por varios niveles de operación: los reportes de peligros y sus comentarios se gestionan como información global a través del backend; los datos propios del usuario, como mascotas y rutas, se almacenan localmente como persistencia local parcial; las imágenes se manejan mediante referencias a alma-

cenamiento externo o local según el estado de integración; y el cálculo de rutas se delega a un servicio especializado basado en GraphHopper. Esta integración permitió equilibrar continuidad de uso para información local del usuario con funcionalidades dependientes de conectividad para consulta y participación comunitaria.

En conjunto, la consolidación del prototipo evidenció una evolución técnica progresiva: desde una interfaz inicial basada en un mapa y botones sin funcionalidad, hacia un sistema modular y orientado al monitoreo colaborativo, soportado por servicios backend especializados. Esta evolución permitió validar decisiones arquitectónicas y ajustar la implementación conforme aparecieron nuevas necesidades funcionales durante el desarrollo.

5.10. Pruebas y validación parcial del sistema

Durante el desarrollo se realizaron pruebas unitarias y validaciones parciales sobre distintos componentes del sistema, con el fin de verificar progresivamente la estabilidad de los módulos implementados y la correcta interacción entre frontend, backend, persistencia local y servicios externos.

Estas validaciones permitieron identificar inconsistencias asociadas a persistencia local, manejo de estados en la interfaz, navegación geoespacial, consumo de endpoints del backend y comportamiento de componentes distribuidos, contribuyendo al fortalecimiento progresivo de la robustez general de la aplicación.

No obstante, debido a que estas pruebas formaron parte del proceso iterativo de desarrollo, la ejecución formal del plan de pruebas, los resultados detallados, las métricas de cobertura, las pruebas de integración completas, las validaciones end-to-end y los reportes técnicos asociados se presentan de manera estructurada en la Etapa 4 del proyecto, correspondiente al proceso de validación y evaluación del sistema.

5.11. Expansión del alcance y evolución hacia problemáticas zoosociales

A medida que el desarrollo del sistema avanzó y se consolidaron las funcionalidades base de monitoreo y navegación geoespacial, el proyecto entró en una etapa de reevaluación conceptual impulsada por el análisis epidemiológico, ambiental y social realizado durante la investigación sobre amenazas urbanas en la ciudad de Cali.

Inicialmente, el sistema había sido concebido principalmente como una plataforma orientada al reporte y visualización de amenazas zoosanitarias urbanas. Sin embargo, durante el proceso de revisión documental y contraste contextual, se evidenció que múltiples problemáticas que afectan directamente el bienestar animal no se encontraban representadas dentro de los sistemas tradicionales de vigilancia sanitaria. Entre estas situaciones se identificaron particularmente el robo, la desaparición y el abandono de mascotas, fenómenos recurrentes en entornos urbanos y altamente percibidos por la comunidad, pero usualmente excluidos de las taxonomías epidemiológicas

formales.

Este hallazgo generó un punto de inflexión dentro del proyecto, ya que implicó ampliar el alcance funcional y conceptual del sistema hacia un modelo de reporte comunitario más integral. Como resultado, la plataforma evolucionó desde un enfoque centrado exclusivamente en riesgos sanitarios hacia una estructura híbrida capaz de representar amenazas zoosanitarias, ambientales y zoosociales de manera unificada.

A partir de esta expansión conceptual, se definió una nueva clasificación funcional de amenazas priorizadas dentro del sistema, organizada en cuatro grandes grupos:

- Amenazas zoosanitarias, como leptospirosis y riesgos asociados a animales potencialmente transmisores de rabia.
- Amenazas ambientales, incluyendo aguas estancadas, acumulación de residuos y presencia de caracol gigante africano.
- Amenazas zoosociales, orientadas principalmente a reportes de mascotas perdidas, animales encontrados y situaciones de abandono.
- Reportes genéricos u "otros", diseñados para permitir el registro de eventos emergentes no contemplados inicialmente dentro de la taxonomía base.

La incorporación de esta nueva estructura implicó modificaciones profundas tanto en la arquitectura frontend como en la lógica de persistencia local y conectividad del sistema.

Desde el punto de vista de interfaz, el antiguo botón dedicado exclusivamente al reporte de peligros fue reemplazado por un módulo general de reportes colaborativos. Este componente introdujo un selector de tipos de reporte que permitió desplegar flujos independientes para peligros sanitarios, mascotas perdidas, mascotas encontradas y reportes genéricos. Cada flujo fue implementado mediante paneles modulares reutilizables construidos sobre la arquitectura de interfaces previamente consolidada durante el desarrollo del sistema.

La incorporación de reportes de mascotas perdidas representó una expansión significativa de la lógica colaborativa del sistema. Este módulo permitió registrar información específica asociada a desapariciones, incluyendo nombre de la mascota, ubicación del último avistamiento, fecha de desaparición, fotografía y datos de contacto del propietario. De manera similar, el módulo de mascotas encontradas fue diseñado para facilitar reportes rápidos de animales sin supervisión observados en vía pública, reduciendo la fricción de interacción para favorecer la participación ciudadana inmediata.

Adicionalmente, se implementó un módulo de reportes genéricos orientado a capturar situaciones no tipificadas previamente dentro del sistema. Este componente introdujo una estructura flexible basada en descripciones abiertas y niveles de riesgo parametrizables, permitiendo que la plataforma mantuviera capacidad de adaptación frente a problemáticas emergentes identificadas por los usuarios.

Con el objetivo de evitar duplicación de lógica entre formularios, se desacopló el mecanismo de selección geográfica en un componente reutilizable independiente encargado de capturar coordenadas directamente desde el mapa interactivo. Esta decisión permitió mantener consistencia funcional

entre todos los tipos de reporte implementados y simplificó la extensibilidad futura de la arquitectura frontend.

La expansión funcional también requirió una evolución importante del modelo de datos de la plataforma. En lugar de reutilizar una única tabla genérica para todos los reportes, se diseñaron estructuras específicas para cada categoría de información, permitiendo tipado fuerte y persistencia especializada para mascotas perdidas, mascotas encontradas y reportes genéricos. Esta decisión facilitó la validación contextual de formularios, el manejo independiente de cada módulo y la administración especializada de atributos asociados a cada tipo de amenaza o reporte comunitario.

En paralelo, la arquitectura de persistencia y gestión de datos fue extendida para soportar múltiples tipos de reportes y módulos funcionales dentro de la plataforma. Asimismo, el backend en Laravel incorporó nuevos servicios especializados y controladores independientes para cada categoría de reporte, permitiendo separar la lógica de negocio asociada a mascotas perdidas, mascotas encontradas, peligros y reportes genéricos. De igual manera, se mantuvieron mecanismos de almacenamiento local para información propia del usuario, junto con validaciones relacionadas con conectividad y acceso a funcionalidades dependientes de servicios remotos.

Finalmente, la incorporación masiva de nuevos eventos geolocalizados generó un impacto directo sobre el sistema de navegación y *feedback* al usuario. El aumento significativo de alertas presentes en el mapa provocó inicialmente problemas relacionados con solapamiento de mensajes de voz, saturación de instrucciones y bloqueos asociados al sistema de síntesis de audio. Como respuesta, se desarrolló un motor de retroalimentación completamente asíncrono basado en colas de audio y priorización dinámica de eventos críticos.

Esta evolución permitió integrar de manera fluida instrucciones de navegación, alertas de peligros y notificaciones asociadas a problemáticas zoosociales dentro de una única experiencia de interacción continua, consolidando el sistema no solo como una plataforma de monitoreo geoespacial, sino como una herramienta colaborativa orientada a la protección integral del bienestar animal urbano.

5.12. Análisis comparativo entre el diseño arquitectónico y la implementación desarrollada

Una vez finalizada la etapa de construcción y prototipado, fue posible contrastar la arquitectura inicialmente planteada durante la etapa de diseño con la implementación real desarrollada a lo largo del proyecto. Este análisis permitió identificar tanto la efectividad de las decisiones arquitectónicas adoptadas como aquellos aspectos cuya complejidad práctica no había sido completamente anticipada durante la planeación inicial.

En términos generales, la estructura arquitectónica definida mediante el modelo C4 se mantuvo estable durante todo el desarrollo. Como se observa en la Figura 4.3, la separación entre frontend, backend, persistencia local y servicios especializados permitió construir una solución modular y progresivamente escalable, facilitando la evolución incremental del sistema y la incorporación de nuevas funcionalidades sin requerir una reestructuración completa de la plataforma.

No obstante, durante la implementación surgieron desafíos técnicos que no habían sido contemplados con suficiente nivel de detalle en la fase de diseño. Uno de los casos más relevantes estuvo relacionado con la persistencia local de datos en entornos web. Inicialmente, se había previsto el uso homogéneo de SQLite como mecanismo de persistencia local; sin embargo, durante el desarrollo se evidenció que Flutter Web no permite utilizar SQLite de la misma manera que en dispositivos móviles. Como consecuencia, fue necesario adaptar parcialmente la estrategia de persistencia para utilizar mecanismos de almacenamiento propios del navegador, lo cual introdujo diferencias importantes entre plataformas y aumentó la complejidad de implementación de la persistencia local parcial.

De manera similar, el diseño arquitectónico inicial no contempló algunos componentes internos específicos del backend que posteriormente resultaron fundamentales para garantizar estabilidad, seguridad y mantenibilidad del sistema. Entre estos elementos se encontraron middlewares, validadores especializados, servicios desacoplados, gestores de persistencia local y mecanismos de control de autenticación más complejos de lo previsto inicialmente.

Además, varios de los componentes internos representados inicialmente en la Figura 4.4 evolucionaron significativamente durante la implementación, especialmente aquellos relacionados con persistencia local, navegación geoespacial, planificación de viajes y gestión colaborativa de reportes. Esta evolución no implicó una ruptura de la arquitectura propuesta, sino una expansión progresiva de responsabilidades y relaciones internas entre componentes previamente definidos de manera más general durante la etapa de diseño.

Durante la etapa de planeación se consideró la posibilidad de integrar activamente entidades públicas relacionadas con salud y bienestar animal, particularmente para el acceso a información epidemiológica y validación institucional de amenazas urbanas. Sin embargo, aunque se realizaron acercamientos y procesos formales de solicitud, estas entidades no participaron directamente durante la implementación y desarrollo operativo del prototipo, principalmente debido a restricciones temporales y tiempos institucionales de respuesta que excedían el alcance académico del proyecto.

A pesar de estas diferencias entre el modelo previsto y la implementación final, se concluye que los principios arquitectónicos definidos durante la etapa de diseño fueron respetados de manera consistente a lo largo del desarrollo. La modularidad, la separación de responsabilidades, la persistencia local parcial y la distribución de servicios constituyeron pilares fundamentales para la construcción progresiva de la plataforma y permitieron sostener la evolución funcional del sistema incluso frente a cambios importantes en el alcance conceptual y técnico del proyecto.

5.13. Conclusión del capítulo

La etapa de construcción y prototipado permitió materializar los lineamientos definidos en la fase de análisis y diseño, consolidando un sistema funcional que integra capacidades de geolocalización, monitoreo colaborativo y gestión de información geoespacial. A través de un enfoque de desarrollo incremental, se logró evolucionar desde una arquitectura inicial centrada principalmente en persistencia local hacia una solución híbrida que combina almacenamiento de información propia del usuario, servicios backend especializados y procesamiento de información en tiempo real.

Uno de los principales aportes de esta etapa radica en la implementación de un modelo de interacción colaborativa, en el cual los usuarios no solo consumen información, sino que participan activamente en la generación, validación y actualización de reportes de amenazas. Este enfoque permitió trasladar el problema desde un modelo tradicional de vigilancia centralizada hacia un esquema distribuido, más acorde con las dinámicas urbanas y la disponibilidad de tecnologías móviles.

De esta manera, las decisiones técnicas adoptadas, como la incorporación de persistencia local parcial para información del usuario, la integración de servicios backend especializados para la gestión global de datos y la incorporación de motores de enrutamiento personalizados, permitieron abordar limitaciones identificadas en soluciones existentes, particularmente en lo relacionado con la adaptabilidad del sistema a condiciones reales del entorno y la capacidad de respuesta ante riesgos geolocalizados.

No obstante, también se identificaron limitaciones inherentes al alcance del proyecto, como la dependencia de conectividad para la visualización cartográfica, lo cual representa una oportunidad de mejora en futuras iteraciones del sistema.

En conjunto, esta etapa no solo permitió validar la viabilidad técnica de la propuesta, sino también establecer una base sólida para la fase de evaluación, en la cual se analizará el desempeño del sistema en términos de usabilidad, precisión y pertinencia frente al contexto urbano de la ciudad de Cali.

Capítulo 6

Etapa 4: Validación y Evaluación del Prototipo

Validación y evaluación del prototipo

La etapa de validación y evaluación del prototipo correspondió al proceso de validación técnica, funcional y operativa del sistema desarrollado durante las etapas anteriores, con el propósito de determinar el grado de cumplimiento de los requerimientos definidos, la estabilidad de los componentes implementados y la viabilidad general de la solución propuesta.

A diferencia de las etapas previas, centradas en la investigación, el diseño y la construcción progresiva del sistema, esta fase se orientó a la ejecución sistemática de pruebas sobre el prototipo funcional, evaluando tanto el comportamiento individual de los componentes como la interacción entre los distintos módulos que conforman la arquitectura de la aplicación.

El proceso de validación se estructuró mediante un plan de pruebas dividido en múltiples fases, abarcando desde la preparación del entorno de ejecución hasta pruebas unitarias, pruebas de integración, pruebas funcionales, pruebas end-to-end, pruebas manuales exploratorias, validaciones de rendimiento y procesos de regresión final. Esta organización permitió establecer una estrategia de evaluación progresiva y trazable, donde cada fase abordó distintos niveles de complejidad y diferentes perspectivas de análisis sobre el sistema.

Inicialmente, se realizó la validación del ambiente de pruebas y de las dependencias tecnológicas necesarias para la ejecución del proyecto, verificando el funcionamiento coordinado del backend en Laravel, el frontend desarrollado en Flutter, la base de datos utilizada durante las pruebas, los servicios de planificación de rutas mediante GraphHopper desplegado en Docker y los mecanismos de conectividad externa empleados durante las pruebas de rendimiento y acceso remoto. Esta etapa permitió garantizar que la infraestructura técnica se encontrara en condiciones adecuadas antes de iniciar los procesos de evaluación funcional.

Posteriormente, se llevaron a cabo pruebas unitarias tanto en el backend como en el frontend, con el objetivo de validar de manera aislada el comportamiento de modelos, servicios, proveedores de estado, reglas de validación, componentes reutilizables y lógica de negocio. Durante este proceso se identificó la necesidad de reorganizar la clasificación de las pruebas implementadas inicialmente, separando formalmente las pruebas unitarias puras de aquellas que involucraban integración entre múltiples capas del sistema, particularmente en el backend. Como resultado, se redefinió la estructura de pruebas para distinguir adecuadamente las pruebas unitarias de las pruebas de integración backend/API, mejorando la trazabilidad y precisión de las métricas de cobertura obtenidas.

En una fase posterior, se ejecutaron pruebas de integración orientadas a validar el comportamiento de los endpoints HTTP, middleware, controladores, modelos, mecanismos de autenticación

y persistencia de datos. Estas pruebas permitieron comprobar el correcto funcionamiento de los contratos de comunicación entre frontend y backend, así como la respuesta del sistema frente a escenarios válidos, inválidos y situaciones de error controlado.

De manera complementaria, se realizaron pruebas del sistema integrado local, evaluando el funcionamiento conjunto de los componentes que conforman la arquitectura del prototipo. En esta etapa se validó la interacción entre el mapa interactivo, el backend, la persistencia local parcial (orientada al almacenamiento local de información propia del usuario), el cálculo de rutas mediante GraphHopper y la representación geoespacial de amenazas dentro del entorno de la aplicación, considerando que varias funcionalidades dependen de conectividad requerida hacia servicios externos y endpoints del backend.

Asimismo, se incorporaron pruebas end-to-end ejecutadas desde navegador mediante Playwright, las cuales permitieron simular flujos completos de interacción desde la perspectiva del usuario final. Estas pruebas abarcaron procesos como autenticación, gestión de mascotas, creación de reportes, visualización de amenazas, comentarios, rutas y navegación dentro de la interfaz. Paralelamente, se desarrollaron pruebas manuales exploratorias enfocadas en aspectos de usabilidad, validación visual, mensajes de error, estados vacíos, comportamiento responsive y manejo de permisos del dispositivo.

Finalmente, se ejecutaron pruebas no funcionales orientadas a evaluar el comportamiento del sistema frente a múltiples solicitudes y condiciones de conectividad variables, utilizando herramientas como Postman y Cloudflare Tunnel para analizar tiempos de respuesta, estabilidad general y comportamiento de recuperación ante fallos. Posteriormente, se realizó una fase de regresión y cierre QA, consolidando las evidencias obtenidas durante todo el proceso de validación y verificando que las correcciones implementadas no introdujeran nuevos errores en los flujos críticos del sistema.

En conjunto, esta etapa permitió validar el funcionamiento integral del prototipo desarrollado, identificar limitaciones y hallazgos relevantes, y consolidar evidencia técnica sobre la estabilidad, coherencia funcional y capacidad operativa de la solución propuesta dentro del alcance definido para el proyecto.

6.1. Estrategia general de pruebas

Con el propósito de validar el funcionamiento del prototipo desarrollado, se diseñó e implementó una estrategia de pruebas estructurada por fases, orientada a evaluar de manera progresiva los distintos componentes del sistema y las interacciones existentes entre ellos. Esta estrategia permitió abordar el proceso de validación desde múltiples niveles de análisis, incluyendo pruebas unitarias, pruebas de integración, pruebas funcionales, pruebas de sistema, pruebas end-to-end, validaciones manuales y pruebas no funcionales relacionadas con rendimiento y estabilidad.

Como resultado de este proceso se consolidó un plan de pruebas (Véase el Anexo D) y un conjunto estructurado de casos de prueba ejecutables organizados por fase (Véase el Apéndice E), junto con sus respectivos resultados y evidencias técnicas documentadas en los apartados de la presente etapa.

La planificación general se definió bajo un enfoque incremental, en el cual cada fase dependía

parcialmente de la estabilidad alcanzada en las fases anteriores. De esta forma, la validación inició con la preparación y verificación del entorno de ejecución y evolucionó progresivamente hacia escenarios más complejos que involucraron interacción simultánea entre frontend, backend, persistencia de datos, servicios de rutas y navegación geoespacial.

Para garantizar trazabilidad, las pruebas fueron organizadas en fases identificadas mediante la nomenclatura F0–F8. Esta segmentación permitió aislar responsabilidades dentro del proceso de testing, facilitando tanto la ejecución como el análisis posterior de resultados.

- **F0**: Preparación y validación del ambiente.
- **F1**: Pruebas unitarias del backend.
- **F2**: Pruebas unitarias del frontend.
- **F3**: Pruebas de integración backend/API.
- **F4**: Validación del sistema integrado local.
- **F5**: Pruebas end-to-end.
- **F6**: Pruebas manuales y exploratorias.
- **F7**: Pruebas de rendimiento y validaciones no funcionales.
- **F8**: Regresión y cierre QA.

6.2. Ambiente de pruebas

Antes de iniciar la ejecución de las pruebas funcionales, de integración y de rendimiento, se realizó una validación preliminar del ambiente de ejecución con el fin de garantizar que todos los componentes tecnológicos del sistema se encontraran correctamente configurados y operativos. Esta verificación correspondió a la fase F0 del plan de pruebas y permitió establecer una base estable para las etapas posteriores de evaluación.

El ambiente de pruebas fue construido sobre una arquitectura distribuida compuesta por un frontend desarrollado en Flutter Web, un backend implementado en Laravel, un servicio de cálculo de rutas mediante GraphHopper desplegado en Docker y un conjunto de mecanismos de persistencia y servicios backend especializados que soportan las funcionalidades del sistema.

Backend (Laravel)

El backend del sistema fue desarrollado utilizando Laravel, un framework PHP ampliamente utilizado para el desarrollo de aplicaciones web y APIs RESTful debido a su estructura modular, sistema de rutas, mecanismos de autenticación y herramientas integradas para pruebas automatizadas [23].

Dentro del ambiente de pruebas, el backend fue ejecutado localmente mediante el servidor integrado de Laravel, exponiendo los endpoints necesarios para autenticación, gestión de reportes, comentarios, mascotas y rutas, así como las operaciones requeridas por funcionalidades dependientes de conectividad (por ejemplo, consulta de amenazas y reportes comunitarios). Asimismo, se verificó el correcto funcionamiento de middleware, migraciones, controladores y mecanismos de persistencia utilizados durante las pruebas posteriores.

Durante la validación inicial del entorno también se comprobó la correcta ejecución de dependencias PHP y paquetes gestionados mediante Composer, así como la disponibilidad de los endpoints principales del sistema mediante solicitudes HTTP de prueba.

Frontend (Flutter Web)

La interfaz del sistema fue ejecutada utilizando Flutter Web, aprovechando la capacidad multiplataforma del framework para desplegar aplicaciones orientadas inicialmente a dispositivos móviles dentro de entornos de navegador [24].

El frontend fue utilizado como cliente principal durante las pruebas funcionales y end-to-end, permitiendo validar componentes visuales, formularios, navegación y visualización geoespacial, además de la comunicación con el backend para el acceso a servicios que requieren conectividad activa. Dentro del ambiente de pruebas se verificó la correcta carga de dependencias y el funcionamiento de los módulos principales antes de iniciar las fases automatizadas de evaluación.

Persistencia de datos y entorno local

El sistema utilizó una arquitectura híbrida con persistencia local parcial y persistencia remota. En particular, el almacenamiento local se orientó a conservar información propia del usuario (por ejemplo, rutas, mascotas y configuraciones) y a soportar continuidad de uso en escenarios acotados, mientras que múltiples funcionalidades del sistema requieren conectividad activa, especialmente aquellas asociadas a mapas, navegación, comentarios, reportes comunitarios y consulta de amenazas mediante servicios backend especializados.

En el backend se verificó la correcta ejecución de migraciones y la disponibilidad de las tablas principales del sistema, incluyendo usuarios, reportes, mascotas, rutas, comentarios y estructuras auxiliares relacionadas con autenticación y operación de los servicios backend.

GraphHopper y servicios de rutas

El cálculo de rutas fue soportado mediante GraphHopper, un motor de enrutamiento geoespacial basado en OpenStreetMap, desplegado localmente mediante contenedores Docker [25]. Durante la fase de preparación del ambiente se validó la correcta configuración de perfiles de navegación, puertos de comunicación y carga del archivo cartográfico utilizado por el sistema. Asimismo, se ejecutaron rutas de prueba utilizando múltiples puntos geográficos para verificar el correcto funcionamiento del servicio antes de las pruebas de integración y navegación.

La utilización de Docker permitió aislar el servicio de rutas del resto de componentes del sistema, facilitando la reproducibilidad del entorno de pruebas y garantizando consistencia en la ejecución de los escenarios evaluados [26].

Navegador de pruebas

Las pruebas funcionales y end-to-end fueron ejecutadas principalmente sobre navegadores basados en Chromium, incluyendo Google Chrome y los entornos automatizados utilizados por Playwright. Esta selección permitió validar compatibilidad con tecnologías web modernas, renderizado del mapa interactivo y comportamiento de la interfaz en condiciones cercanas a un entorno real de usuario.

Adicionalmente, el uso de Playwright permitió automatizar interacciones complejas dentro del navegador, incluyendo autenticación, navegación, formularios y validación visual de componentes [27].

Cloudflare Tunnel y pruebas externas

Para las pruebas de conectividad y validaciones no funcionales realizadas durante la fase F7, se utilizó Cloudflare Tunnel como mecanismo temporal de exposición segura del backend hacia redes externas [28]. Este servicio permitió generar URLs públicas temporales para validar conectividad remota, comportamiento de endpoints y tiempos de respuesta bajo escenarios controlados, sin necesidad de desplegar permanentemente el sistema en un servidor público.

Validación del ambiente

En conjunto, la preparación del ambiente permitió verificar que todos los componentes requeridos para la ejecución del sistema se encontraran correctamente configurados antes de iniciar las pruebas profundas definidas en las fases posteriores del plan de validación. Esta validación inicial redujo errores asociados a configuración de dependencias, conectividad entre servicios y disponibilidad de componentes externos, garantizando un entorno estable para la ejecución de pruebas unitarias, de integración, funcionales y de rendimiento.

6.3. Ejecución de pruebas por fases

La ejecución del proceso de validación se desarrolló de manera progresiva a través de las fases de prueba F0–F8. Cada fase fue diseñada para evaluar componentes específicos del sistema, permitiendo validar desde el ambiente base de ejecución hasta el comportamiento integral del prototipo bajo escenarios funcionales y no funcionales. Esta organización permitió mantener trazabilidad sobre los objetivos evaluados, las herramientas utilizadas, las evidencias obtenidas y las decisiones tomadas durante el proceso de aseguramiento de calidad.

F0: Preparación y validación del ambiente

La fase F0 estuvo orientada a la preparación y validación del ambiente de ejecución utilizado durante el proceso de pruebas. En esta etapa se verificó la correcta instalación y funcionamiento de dependencias, servicios y herramientas necesarias para la ejecución del sistema, así como la conectividad entre componentes distribuidos.

El informe completo correspondiente a esta fase se presenta en el Apéndice F.

F1: Pruebas unitarias del backend

La fase F1 correspondió a la ejecución de pruebas unitarias sobre los componentes internos del backend desarrollado en Laravel. Estas pruebas estuvieron enfocadas en validar de manera aislada modelos, servicios, reglas de validación, middleware y lógica de negocio.

Durante el proceso se identificó la necesidad de separar formalmente las pruebas unitarias puras de aquellas asociadas a integración backend/API. Como resultado, las pruebas de integración fueron reorganizadas dentro de la fase F3, permitiendo una clasificación más precisa de la cobertura obtenida.

La cobertura generada por las herramientas automatizadas fue exportada en formato HTML para facilitar su análisis detallado. Debido a la extensión y naturaleza navegable de estos reportes, dichos artefactos fueron incluidos como anexos digitales complementarios.

El informe detallado de la fase F1 se presenta en el Apéndice G. De manera complementaria, el reporte de cobertura generado por las herramientas automatizadas de PHPUnit fue conservado en su formato HTML original como se menciona en el Apéndice H y se adjunta como archivo pdf bajo el nombre `coverageF1.pdf`, intentando preservar la estructura navegable y las métricas completas de cobertura obtenidas durante la ejecución de las pruebas.

F2: Pruebas unitarias del frontend

La fase F2 estuvo orientada a pruebas unitarias y validaciones sobre componentes del frontend desarrollado en Flutter Web. En esta etapa se evaluó el comportamiento de widgets, providers, servicios, formularios y lógica de interacción visual.

Estas pruebas permitieron validar manejo de estados, renderizado de componentes, persistencia local y comportamiento de los módulos de interfaz sin depender directamente de servicios backend externos.

Los reportes de cobertura y resultados en formato HTML fueron incorporados como anexos digitales debido a su estructura interactiva y extensión. El informe correspondiente a esta fase se presenta en el Apéndice I, mientras que el reporte de resultados se encuentra en el Apéndice J.

F3: Pruebas de integración backend/API

La fase F3 se enfocó en validar la integración entre componentes del backend y los mecanismos de comunicación HTTP utilizados por la aplicación. En esta etapa se evaluaron endpoints, middleware,

autenticación, persistencia de datos y contratos de comunicación entre cliente y servidor, verificando escenarios válidos, inválidos y respuestas frente a errores controlados.

El informe detallado correspondiente a esta fase se presenta en el Apéndice K, mientras que el reporte de cobertura asociado se incluye en el Apéndice L.

F4: Validación del sistema integrado local

La fase F4 estuvo orientada a validar el funcionamiento conjunto de los principales componentes del sistema dentro de un entorno local integrado. En esta fase se evaluó la interacción simultánea entre frontend, backend, persistencia local parcial de información del usuario, visualización geoespacial y cálculo de rutas mediante GraphHopper, además de la comunicación con los servicios backend especializados que requieren conectividad.

Durante esta etapa se ejecutaron escenarios relacionados con autenticación, creación de reportes, navegación sobre el mapa y operación general de la arquitectura distribuida, considerando la existencia de funcionalidades dependientes de conectividad para la consulta y actualización de información comunitaria y de amenazas.

El informe correspondiente a esta fase se presenta en el Apéndice M.

F5: Pruebas end-to-end

La fase F5 correspondió a pruebas end-to-end ejecutadas mediante Playwright, orientadas a simular flujos completos de interacción desde la perspectiva del usuario final. Estas pruebas incluyeron procesos de autenticación, gestión de mascotas, creación de reportes, navegación geoespacial, comentarios y planificación de rutas dentro de la interfaz web.

La automatización mediante Playwright permitió validar el comportamiento integral del sistema dentro de un navegador basado en Chromium, comprobando interacción entre frontend, backend y servicios externos bajo escenarios de uso cercanos a condiciones reales. Las evidencias visuales capturadas durante la ejecución de las pruebas fueron incorporadas como anexos complementarios.

El informe técnico correspondiente a esta fase se presenta en el Apéndice N, mientras que el reporte generado automáticamente por Playwright se incluye en el Apéndice Ñ.

F6: Pruebas manuales y exploratorias

La fase F6 estuvo orientada a pruebas manuales y exploratorias enfocadas en experiencia de usuario, comportamiento visual y validación funcional de componentes difíciles de evaluar mediante automatización.

Durante esta etapa se verificaron formularios, mensajes de error, navegación, comportamiento responsive, validaciones visuales, estados vacíos y manejo de permisos asociados a geolocalización y almacenamiento local.

El informe correspondiente a esta fase se presenta en el Apéndice O. Junto a las capturas y evidencias visuales asociadas se presentan en el Apéndice P.

F7: Pruebas de rendimiento y validaciones no funcionales

La fase F7 estuvo enfocada en evaluar el comportamiento del sistema bajo condiciones relacionadas con rendimiento, conectividad y estabilidad de endpoints. Para ello se utilizaron herramientas como Postman y Cloudflare Tunnel, permitiendo ejecutar pruebas sobre tiempos de respuesta, acceso remoto controlado y validación de endpoints protegidos mediante autenticación.

El informe asociado a esta fase se presenta en el Apéndice Q.

F8: Regresión y cierre QA

La fase F8 correspondió al proceso de regresión y cierre de calidad del sistema. En esta etapa se consolidaron los resultados obtenidos durante todas las fases anteriores, verificando que las correcciones implementadas no introdujeran nuevos errores sobre funcionalidades previamente validadas. Asimismo, se revisó el estado final de defectos identificados y se establecieron criterios de cierre respecto a la estabilidad del sistema.

El informe final de regresión y cierre QA se presenta en el Apéndice R, mientras que las evidencias consolidadas de ejecución correspondientes a las fases F0 a F7 se incluyen en el Apéndice S.

6.4. Limitaciones de la evaluación

Si bien el proceso de validación permitió obtener evidencia significativa sobre el funcionamiento técnico y funcional del prototipo, el alcance de la evaluación se mantuvo delimitado por restricciones temporales, técnicas y metodológicas definidas para el desarrollo del proyecto. En consecuencia, algunas validaciones especializadas y escenarios de despliegue reales quedaron fuera del alcance de esta etapa.

En primer lugar, no se realizó una auditoría formal de seguridad basada en estándares OWASP. Aunque durante las pruebas se verificaron aspectos relacionados con autenticación, protección de endpoints y manejo básico de errores, el proyecto no contempló la ejecución de pruebas especializadas de seguridad ofensiva, análisis de vulnerabilidades avanzadas ni procesos de *pentesting*.

De igual manera, no se llevó a cabo una validación cartográfica exhaustiva sobre la totalidad de calles, rutas y geometrías urbanas presentes en la ciudad de Cali. Las pruebas de navegación y cálculo de rutas se realizaron sobre escenarios representativos definidos durante la validación funcional del sistema, utilizando sectores específicos del mapa y rutas seleccionadas para los casos evaluados.

Asimismo, las pruebas ejecutadas no utilizaron datos productivos reales provenientes de entidades oficiales o sistemas institucionales de vigilancia epidemiológica. Los escenarios de evaluación fueron construidos con datos de prueba, registros simulados y reportes generados dentro del entorno de desarrollo, debido a restricciones de acceso a información institucional y a consideraciones éticas relacionadas con datos reales de salud pública.

Por otro lado, aunque durante las pruebas se evaluaron aspectos básicos de usabilidad, legibilidad visual y navegación, no se realizó un proceso formal de certificación de accesibilidad bajo lineamientos WCAG (Web Content Accessibility Guidelines). En consecuencia, el sistema no fue

sometido a evaluaciones especializadas de accesibilidad orientadas a certificación o validaciones exhaustivas para usuarios con condiciones específicas de discapacidad.

En cuanto a las pruebas de rendimiento, estas fueron ejecutadas sobre un entorno controlado de desarrollo utilizando Cloudflare Tunnel como mecanismo temporal de exposición externa del backend. Por esta razón, los resultados obtenidos corresponden a un entorno de pruebas local y no deben interpretarse como métricas equivalentes a escenarios de producción con infraestructura distribuida, balanceadores de carga o tráfico concurrente de usuarios reales.

Adicionalmente, el proceso de validación no contempló el despliegue oficial de la aplicación en tiendas móviles ni pruebas sobre múltiples dispositivos físicos en condiciones masivas de uso. Aunque el sistema fue probado sobre navegadores basados en Chromium y entornos controlados de ejecución, no se realizó una validación comercial multiplataforma a gran escala.

Estas limitaciones se encuentran alineadas con el alcance definido inicialmente dentro del plan de pruebas del proyecto, el cual priorizó la validación funcional, arquitectónica y técnica del prototipo dentro de un contexto académico y experimental. No obstante, los resultados obtenidos permiten establecer una base para futuras etapas de expansión, endurecimiento de seguridad, validación institucional y despliegue en escenarios reales de operación.

Capítulo 7

Conclusiones y trabajos futuros

7.1. Conclusiones generales

El desarrollo del prototipo permitió consolidar una plataforma geoespacial colaborativa orientada al reporte, visualización y notificación de amenazas zoosanitarias urbanas en animales domésticos en la ciudad de Cali. A partir de los objetivos planteados, se logró construir una solución funcional que integra componentes de geolocalización, gestión de reportes, visualización cartográfica, administración de mascotas, rutas y mecanismos de interacción comunitaria.

En términos funcionales, el sistema permitió materializar una propuesta tecnológica centrada en la participación ciudadana, en la cual los usuarios pueden reportar situaciones de riesgo, consultar amenazas cercanas y acceder a información relevante para la toma de decisiones durante sus recorridos. Esto evidencia el potencial de las plataformas colaborativas como apoyo a la identificación temprana de problemáticas urbanas relacionadas con animales domésticos.

Desde el punto de vista técnico, el proyecto permitió validar una arquitectura híbrida basada en persistencia local parcial, servicios backend especializados y funcionalidades dependientes de conectividad. Esta aproximación resultó adecuada para el alcance del prototipo, ya que permitió combinar almacenamiento de información propia del usuario con servicios remotos para la gestión comunitaria de reportes, comentarios, autenticación y visualización de amenazas.

La implementación del prototipo evidenció la importancia de definir requerimientos claros, trazables y verificables desde las etapas iniciales del proyecto. La evolución de los requerimientos durante el desarrollo permitió ajustar el alcance funcional del sistema, especialmente en aspectos relacionados con conectividad, persistencia, rutas, reportes y experiencia de usuario.

El proceso de pruebas permitió verificar progresivamente el comportamiento del sistema mediante una estrategia organizada por fases, abarcando pruebas unitarias, de integración, funcionales, manuales, end-to-end y no funcionales. Esta validación contribuyó a identificar errores, ajustar componentes críticos y fortalecer la estabilidad general del prototipo.

Finalmente, el proyecto permitió demostrar que una solución geoespacial colaborativa puede aportar valor en contextos urbanos al facilitar la identificación, consulta y notificación de amenazas zoosanitarias. Aunque el sistema corresponde a un prototipo, su desarrollo establece una base técnica y conceptual para futuras ampliaciones orientadas al bienestar animal, la prevención comunitaria y la integración con mecanismos institucionales de alerta.

7.2. Trabajos futuros

A partir del desarrollo del prototipo, se identifican diversas oportunidades de mejora y ampliación que permitirían fortalecer el alcance funcional, social y tecnológico de la plataforma. Estas líneas futuras buscan consolidar el sistema no solo como una herramienta de reporte y visualización de amenazas zoosanitarias urbanas, sino también como un medio de apoyo al bienestar animal, la prevención comunitaria y la articulación con mecanismos institucionales de alerta.

7.2.1. Fortalecimiento del enfoque de bienestar animal

Como trabajo futuro, se propone ampliar el concepto de bienestar animal dentro de la plataforma mediante funcionalidades orientadas al seguimiento de hábitos saludables, el cuidado preventivo y el acompañamiento responsable de animales domésticos en entornos urbanos. Durante el proceso de diseño e implementación se identificaron funcionalidades potenciales que podrían fortalecer este enfoque en versiones posteriores del sistema, especialmente aquellas relacionadas con el registro estructurado de paseos, el almacenamiento del historial de recorridos, el cálculo de métricas asociadas a frecuencia y continuidad de actividad física, la generación de indicadores de exposición a zonas de riesgo y la emisión de alertas preventivas basadas en la ubicación del usuario, la presencia de amenazas georreferenciadas y las características del trayecto seleccionado.

De esta manera, la plataforma podría evolucionar desde un sistema centrado en el reporte, visualización y notificación de amenazas hacia una herramienta de apoyo integral para tutores de mascotas, promoviendo prácticas responsables de cuidado, prevención y bienestar animal en contextos urbanos.

7.2.2. Implementación de rachas de paseo, hábitos saludables y comunidad

Otra línea de mejora corresponde a la incorporación de un módulo orientado al seguimiento de hábitos de paseo y bienestar animal, en el cual los usuarios puedan registrar la frecuencia, duración y características de los recorridos realizados con sus mascotas. Este módulo podría incluir un sistema de rachas de paseo que permita visualizar la constancia de la actividad física mediante indicadores progresivos, metas semanales, historial de recorridos, recordatorios y retroalimentación visual sobre el cumplimiento de rutinas saludables.

Además de incentivar la actividad física mediante elementos de gamificación, este módulo podría integrar contenido educativo elaborado o validado por especialistas vinculados a la iniciativa, tales como médicos veterinarios, etólogos, profesionales en bienestar animal o entidades relacionadas con salud pública y protección animal. Esta información podría presentarse en forma de recomendaciones preventivas, pautas de cuidado durante los paseos, alertas sobre exposición a riesgos, orientación sobre socialización responsable y sugerencias asociadas a edad, tamaño, condición física o necesidades particulares de cada mascota.

Asimismo, el módulo podría fortalecer el componente colaborativo de la plataforma mediante espacios de interacción entre tutores de mascotas. Por ejemplo, se podrían habilitar comentarios comunitarios, recomendaciones sobre rutas seguras, valoración de zonas de paseo, reportes de expe-

riencias, grupos por sectores de la ciudad o dinámicas de participación asociadas a retos colectivos de bienestar animal. De esta manera, la plataforma no solo permitiría consultar amenazas, sino también construir conocimiento comunitario alrededor del cuidado responsable de animales domésticos en entornos urbanos.

La implementación de esta funcionalidad contribuiría a aumentar la participación recurrente de los usuarios, fortalecer el vínculo entre tutor y mascota y consolidar una comunidad activa alrededor de la prevención, el bienestar animal y la apropiación responsable del espacio público. En este sentido, las rachas de paseo no se limitarían a un mecanismo de motivación individual, sino que funcionarían como una estrategia para promover hábitos saludables, circulación de información especializada y colaboración ciudadana.

7.2.3. Integración con mecanismos de alerta de la ciudad

Finalmente, se plantea como trabajo futuro la integración de la plataforma con mecanismos de alerta, información y gestión urbana existentes en la ciudad. Esta línea de ampliación permitiría que los reportes generados por la comunidad no permanezcan únicamente dentro de la aplicación, sino que puedan convertirse en insumos útiles para procesos institucionales relacionados con salud pública, bienestar animal, control de riesgos, gestión ambiental y atención de problemáticas zoosanitarias urbanas.

Desde una perspectiva técnica, esta integración podría desarrollarse mediante mecanismos de interoperabilidad con entidades locales, sistemas de información geográfica institucionales, servicios de datos abiertos, plataformas de atención ciudadana o canales oficiales de notificación. Para ello, sería necesario definir protocolos de intercambio de información, criterios de validación de reportes, formatos estandarizados de datos geoespaciales y niveles de acceso diferenciados según el tipo de actor involucrado. De esta manera, los reportes ciudadanos podrían organizarse, filtrarse y priorizarse antes de ser compartidos con entidades competentes.

Asimismo, la plataforma podría incorporar un módulo de alertas tempranas basado en concentración espacial de reportes, recurrencia de amenazas en determinadas zonas, incremento inusual de eventos o cercanía de los usuarios a áreas previamente identificadas como riesgosas. Este tipo de funcionalidad permitiría generar notificaciones preventivas para la comunidad y, al mismo tiempo, ofrecer información agregada que apoye la identificación de sectores críticos dentro del territorio urbano.

La integración con mecanismos de alerta de la ciudad también permitiría fortalecer la trazabilidad de los reportes, diferenciando entre información comunitaria preliminar, reportes en proceso de validación y eventos que hayan sido escalados a entidades responsables. Esto contribuiría a mejorar la confiabilidad del sistema y a evitar que la información reportada sea interpretada como confirmación oficial sin un proceso previo de revisión.

En conjunto, esta evolución permitiría ampliar el impacto del prototipo hacia un modelo de colaboración entre ciudadanía, especialistas y entidades públicas. Así, la plataforma podría convertirse en una herramienta de apoyo para la toma de decisiones, la priorización de zonas de atención, la comunicación preventiva y la respuesta oportuna frente a amenazas zoosanitarias urbanas en la

ciudad de Cali.

Bibliografía

[1] Infobae, "El 60 % de los colombianos tienen perro, según cifras del dane," July 2024. En línea. Disponible en: https://www.infobae.com/colombia/2024/07/21/el-60-de-los-colombianos-tienen-perro-segun-cifras-del-dane/.

[2] Alcaldía de Santiago de Cali, "Comunicaciones secretaría de salud pública," s.f. En línea. Disponible en: https://www.cali.gov.co.

[3] Secretaría de Salud de Cali, "Boletín de salud pública," tech. rep., 2025.

[4] M. F. Goodchild, "Giscience, geography, form, and process," *International Journal of Geographical Information Science*, vol. 32, no. 1, pp. 1–15, 2018.

[5] J. Howe, "The rise of crowdsourcing," *Wired Magazine*, vol. 14, no. 6, pp. 1–4, 2006. En línea. Disponible en: https://www.wired.com/2006/06/crowds/.

[6] M. F. Goodchild, "Citizens as sensors: The world of volunteered geography," *GeoJournal*, vol. 69, no. 4, pp. 211–221, 2007.

[7] Y. Benkler, *The Wealth of Networks: How Social Production Transforms Markets and Freedom*. New Haven, CT: Yale University Press, 2006.

[8] O. A. Latina, "¿qué es una base de datos?." https://www.oracle.com/lad/database/what-is-database/, 2020. Recuperado en 2025.

[9] EBSCO, "Research starters: Ingeniería de requisitos." En línea. Disponible en: https://www.ebsco.com/research-starters.

[10] J. R. Palmer, A. Oltra, F. Collantes, J. A. Delgado, J. Lucientes, S. Delacour, and F. Bartumeus, "Citizen science provides a reliable and scalable tool to track disease-carrying mosquitoes," *Nature Communications*, vol. 8, p. 916, 2017.

[11] M. Fernández, G. M. Bron, O. Omodior, and C. B. Graham, "The tick app: An innovative tool for tick exposure prevention and disease surveillance," *International Journal of Environmental Research and Public Health*, vol. 16, no. 13, p. 2347, 2019.

[12] I. Abdulrahman, A. Al-Azemi, Y. Al-Kandari, and S. Al-Sabah, "Animal health surveillance using wahis data and tableau visualization in kuwait," *International Journal of Veterinary Science*, vol. 10, no. 3, pp. 177–183, 2021.

[13] Secretaría de Salud Pública de Cali, "Boletines de eventos de interés en salud pública." [En línea], 2025. Consultado: mayo 6, 2026.

[14] Alcaldía de Santiago de Cali, "Vacunar, observar y actuar: claves para evitar riesgos por rabia." [En línea], 2025. Consultado: mayo 6, 2026.

[15] Alcaldía de Santiago de Cali, "Leptospirosis." [En línea], 2025. Consultado: mayo 6, 2026.

[16] Alcaldía de Santiago de Cali, "El dagma refuerza acciones para el control del caracol gigante africano." [En línea], 2025. Consultado: mayo 6, 2026.

[17] D. J. Levitin, *The Organized Mind: Thinking Straight in the Age of Information Overload*. Dutton Penguin, 2014.

[18] M. Vohland *et al.*, *The Science of Citizen Science*. Springer, 2021.

[19] J. Palmer *et al.*, "Citizen science provides a reliable and scalable tool to track disease-carrying mosquitoes," *Nature Communications*, vol. 8, 2017.

[20] European Commission, "Preventing mosquito-borne diseases with ai and citizen science." CORDIS, 2021.

[21] A. M. Kullenberg and D. Kasperowski, "What is citizen science? a scientometric meta-analysis," *PLOS ONE*, vol. 11, no. 1, 2016.

[22] J. C. Newman *et al.*, "The future of citizen science: Emerging technologies and shifting paradigms," *Frontiers in Ecology and the Environment*, vol. 10, no. 6, pp. 298–304, 2012.

[23] Laravel LLC, "Laravel documentation," 2026. Accedido: 2026-05-13.

[24] Google, "Flutter documentation," 2026. Accedido: 2026-05-13.

[25] GraphHopper GmbH, "Graphhopper routing engine documentation," 2026. Accedido: 2026-05-13.

[26] Docker Inc., "Docker documentation," 2026. Accedido: 2026-05-13.

[27] Microsoft, "Playwright documentation," 2026. Accedido: 2026-05-13.

[28] Cloudflare Inc., "Cloudflare tunnel documentation," 2026. Accedido: 2026-05-13.

Apéndice A

Formularios de la Secretaría de Salud Pública de Cali

En este apéndice se presentan los documentos institucionales diligenciados durante el proceso de gestión y solicitud de información ante la Secretaría de Salud Pública de Cali y entidades asociadas.

Estos documentos corresponden a los formatos requeridos para la evaluación ética y administrativa del proyecto, así como a los acuerdos relacionados con confidencialidad y tratamiento de información.

A.1. Formato de confidencialidad y protección de datos

El siguiente documento corresponde al acuerdo de confidencialidad y protección de datos requerido para el proceso de solicitud de acceso a información institucional asociada al proyecto.

ALCALDÍA DE SANTIAGO DE CALI GESTION TECNOLOGICA Y DE LA INFORMACIÓN ADMINISTRACIÓN DE LAS TECNOLOGIAS DE LA INFORMACIÓN Y LAS COMUNICACIONES	MODELO INTEGRADO DE PLANEACIÓN Y GESTIÓN (MIPG) ACUERDO DE CONFIDENCIALIDAD INDIVIDUALY PROTECCIÓN DE DATOS	MAGT04.04.03.P003.F-SS001	
		VERSIÓN	001

Entre los suscritos a saber: Laura Camila Franco Valencia,mayor de edad y domiciliado(a) en la ciudad de Cali, identificado(a) como aparece al pie de su respectiva firma, quien en adelante y para los efectos del presente contrato se denominará EL INVESTIGADOR; y El Distrito de Santiago de Cali – Secretaría de Salud del Distrito Especial de Santiago de Cali, quien en adelante y para los efectos del presente acuerdo se denominará EL DISTRITO.

Declaro y reconozco que por la índole de mis conocimientos, obligaciones y actividades, que me califican como un tercero lo cual implica necesariamente el acceso, manejo y la utilización de información, documentos y/o materiales que representan datos reservados y sensibles de responsabilidad del Distrito de Santiago de Cali – Secretaría de Salud Pública, cuya información sea de carácter confidencial, reservada frente a terceros, relacionada con datos clínicos, estadísticos, epidemiológicos, software y/o material informático, documentos, guías y material interno referente del evento de interés en salud pública con ocasión de los principios rectores de la Ley estatutaria 1581 de 2012 (Ley de Protección de Datos Personales) principio de confidencialidad, La Ley 1712 de 2014 “Ley de transparencia y del derecho al acceso a la información pública nacional”.

Las partes han convenido en suscribir el presente acuerdo de confidencialidad, previas las siguientes consideraciones:

Marco Constitucional. Artículo 15 de la Constitución Política: “Todas las personas tienen derecho a su intimidad personal y familiar y a su buen nombre, y el Estado debe respetarlos y hacerlos respetar. De igual modo, tienen derecho a conocer, actualizar y rectificar las informaciones que se hayan recogido sobre ellas en bancos de datos y en archivos de entidades públicas y privadas”. “En la recolección, tratamiento y circulación de datos se respetarán la libertad y demás garantías consagradas en la Constitución.”

Ley 1581 de 2012: ¿Qué es un dato personal? Cualquier información vinculada o que pueda asociarse a una o varias personas naturales determinadas o determinables.

Clasificación de Datos Personales (Ley 1266 de 2008 y Decreto 1377 de 2013)

- *Dato Público*: Calificado como tal en la ley. Dato que no es semiprivado, privado o sensible (Ej. datos relativos al estado civil de las personas, su profesión u oficio, su calidad de comerciante o servidor público y aquellos que pueden obtenerse sin reserva alguna).
- *Dato semiprivado*: Dato que no tiene naturaleza íntima, reservada, ni pública y cuyo conocimiento interesa al titular y a cierto sector o grupo de personas o a la sociedad en general (Ej. datos financieros y crediticios, dirección, teléfono, correo electrónico,).
- *Dato privado*: Dato que solo es relevante para su titular (Ej. fotografías, videos, datos relacionados con su estilo de vida.)
- *Datos sensibles:* Aquellos que afectan la intimidad de las personas o cuyo uso indebido puede generar discriminación. (Origen racial, convicciones religiosas, datos de salud, vida sexual, entre otros).

Este documento es propiedad de la Administración Central del Distrito de Santiago de Cali. Prohibida su alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

- *Datos personales de menores:* Toda la información de menores de 18 años puede ser tratada, siempre y cuando no se ponga en riesgo la prevalencia de sus derechos fundamentales y la información sea de naturaleza pública.

Que de acuerdo a la naturaleza de El Distrito de Santiago de Cali – Secretaría de Salud del Distrito Especial de Santiago de Cali, y dentro de las acciones para cumplir con su misión, en muchos de sus procesos se lleva a cabo el registro, recolección y tratamientos de datos personales, datos sensibles y reservados; motivo por el cual se hace necesario establecer un acuerdo de confidencialidad para garantizar la política de tratamiento de datos y los derechos de los individuos, acuerdo que se regirá por las siguientes clausulas:

CONSIDERACIONES

1. Que, entre el INVESTIGADOR y El Distrito de Santiago de Cali – Secretaría de Salud del Distrito Especial de Santiago de Cali, existe una relación de orden civil, comercial, reglamentario o laboral, conforme al contrato suscrito por las partes.
2. Que para realizar las labores para las cuales fue asignado el INVESTIGADOR, tendrá acceso a información y se generará nueva información, toda ella, con carácter especial de reserva y confidencialidad.
3. Que El Distrito de Santiago de Cali – Secretaría de Salud del Distrito Especial de Santiago de Cali en cumplimiento de compromisos legales y contractuales está obligado para con terceros a garantizar que la información entregada para gestión y la que se genere en desarrollo del contrato no sea copiada, editada, reproducida por cualquier medio, y no sea revelada a cualquier persona, total o parcialmente.
4. Para los efectos del presente contrato se define como *información confidencial*: toda aquella información que con ocasión de las labores contratadas se la ha puesto al conocimiento del INVESTIGADOR, puede ser técnica, financiera, comercial, estratégica y cualquier información relacionada con las gestiones pasadas, presentes y futuras, o condiciones financieras de El Distrito de Santiago de Cali – Secretaría de Salud del Distrito Especial de Santiago de Cali, o información sobre datos personales de personas con cualquier tipo de vínculo, bien sea que dicha información sea escrita, oral o visual, magnética, complementariamente, incluyendo aplicativos y/o las bases de datos personales que se le entreguen al INVESTIGADOR, en el desarrollo de la labor encomendada. Adicionalmente, cualquier información suministrada, previa a la celebración del presente acuerdo, mediante capacitaciones, de forma oral o escrita, se considerará como confidencial y estará sujeta a los términos de la presente clausula.

ACUERDO ADICIONAL DE CONFIDENCIALIDAD Y CONFIDENCIALIDAD DE DATOS

En desarrollo de la labor encomendada, el INVESTIGADOR tendrá acceso de manera directa e indirecta a información de propiedad exclusiva de El Distrito de Santiago de Cali – Secretaría de Salud del Distrito

Este documento es propiedad de la Administración Central del Distrito de Santiago de Cali. Prohibida su alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

ALCALDÍA DE SANTIAGO DE CALI GESTION TECNOLOGICA Y DE LA INFORMACIÓN ADMINISTRACIÓN DE LAS TECNOLOGIAS DE LA INFORMACIÓN Y LAS COMUNICACIONES	MODELO INTEGRADO DE PLANEACIÓN Y GESTIÓN (MIPG) ACUERDO DE CONFIDENCIALIDAD INDIVIDUALY PROTECCIÓN DE DATOS	MAGT04.04.03.P003.F-SS001	
		VERSIÓN	001

Especial de Santiago de Cali y/o de sus relacionados, información que goza de protección especial, atendiendo el grado de confidencialidad de la misma, es por ello que se consagran como obligaciones especiales del INVESTIGADOR las siguientes:

1. Toda información otorgada es de propiedad exclusiva de El Distrito de Santiago de Cali – Secretaría de Salud del Distrito Especial de Santiago de Cali. En consecuencia, EL INVESTIGADOR no utilizará esta información para su propio uso o el de terceros sin autorización.
2. EL INVESTIGADOR se obliga a no copiar, editar, transformar, extraer, revelar, divulgar, exhibir, mostrar, comunicar, utilizar y/o emplear para sí, en forma alguna, de manera directa o indirecta, pública o privadamente, a ninguna persona natural o jurídica, o entidad alguna de carácter público (salvo por orden, mandato judicial o autorización expresa), información considerada como confidencial, pública clasificada, información pública reservada y de protección de datos personales. o la información que le ha sido entregada con ocasión de las labores encomendadas o aquella a la que haya tenido acceso por cualquier causa, que sea de propiedad de El Distrito de Santiago de Cali – Secretaría de Salud del Distrito Especial de Santiago de Cali y/o de sus relacionados.
3. El usuario y contraseña asignado para el acceso a los sistemas de información, es de carácter personal, la contraseña debe ser cambiada periódicamente, usando una combinación adecuada de caracteres que la haga lo menos vulnerable y no divulgarla o suministrarla a otras personas.
4. La información sólo podrá ser utilizada para desarrollar productos y servicios propios de El Distrito de Santiago de Cali – Secretaría de Salud del Distrito Especial de Santiago de Cali; adicionalmente sólo podrá reproducirse dicha información confidencial si ello resulta necesario para cumplir tal finalidad y solo podrá darse a conocer a aquellos empleados, trabajadores o asesores, que hayan firmado este mismo acuerdo de confidencialidad de manera individual.
5. El INVESTIGADOR se obliga en consecuencia a mantenerla reservada y privada y a proteger dicha información para evitar su divulgación no autorizada, ejerciendo sobre ésta, el máximo grado de diligencia y cuidado, faltando a esta obligación por acción o por omisión.
6. EL INVESTIGADOR se obliga a responder por todos los documentos, claves de acceso, mercancías, herramientas, software y hardware que le ha sido entregado para el ejercicio de su labor, y en general con la información confidencial que maneje en el desempeño del cargo.
7. EL INVESTIGADOR se hace responsable por los perjuicios que pudieren causarse al Distrito de Santiago de Cali – Secretaría de Salud del Distrito Especial de Santiago de Cali o sus relacionados, en virtud del no cumplimiento de estas obligaciones.
8. Todo ello en virtud y armonía de las obligaciones del INVESTIGADOR previstas en los contratos celebrados, los manuales, políticas o procedimientos establecidos por El Distrito de Santiago de Cali – Secretaría de Salud del Distrito Especial de Santiago de Cali
9. El INVESTIGADOR manifiesta conocer las políticas internas que se establecen dentro de El Distrito de Santiago de Cali – Secretaría de Salud del Distrito Especial de Santiago de Cali alusivas a lo consagrado en la ley 1266 de 2008, ley estatutaria 1581 de 2012 y decreto reglamentario 1377 de 2007 que contienen las normas sobre protección y manejo de datos personales.
10. Las obligaciones de confidencialidad previstas en el presente acuerdo tendrán una vigencia indefinida aún después de finalizado el vínculo entre las partes.

Este documento es propiedad de la Administración Central del Distrito de Santiago de Cali. Prohibida su alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

ALCALDÍA DE SANTIAGO DE CALI GESTION TECNOLOGICA Y DE LA INFORMACIÓN ADMINISTRACIÓN DE LAS TECNOLOGIAS DE LA INFORMACIÓN Y LAS COMUNICACIONES	MODELO INTEGRADO DE PLANEACIÓN Y GESTIÓN (MIPG) ACUERDO DE CONFIDENCIALIDAD INDIVIDUALY PROTECCIÓN DE DATOS	MAGT04.04.03.P003.F-SS001	
		VERSIÓN	001

DATOS DEL USUARIO

Nombre (persona natural o jurídica)	Laura Camila Franco Valencia				
CC. No.	1119150211				
ENTIDAD					
Entidad	Pontificia Universidad Javeriana de Cali				
Dependencia / Área / Grupo	Facultad de Ingeniería y Ciencias Programa de Ingeniería de Sistemas y Computación				
Cargo que desempeña	Estudiante – Investigadora Principal (Trabajo de Grado)				
Dirección	Calle 18 No. 118-250, Cali, Valle del Cauca				
Teléfono	3232403308	Correo Electrónico	lcfranco@javerianacali.edu.co		
Vinculación con la Secretaría de Salud Pública *(si aplica)*					
Número de contrato					
Fecha de inicio de contrato:	dd/mmm/aaaa	Fecha de terminación de contrato:	dd/mmm/aaaa		
Directa (de planta)		Prestación de servicios		Contrato interadministrativo	

Para constancia, y en señal de aceptación, se firma el presente acuerdo en el Distrito de Santiago de Cali, a los _____ días del mes de _____ del año _____.
Firma del suscriptor en señal de aceptación y conocimiento:

Nombre:	Laura Camila Franco Valencia		
CC.	1119150211	Entidad:	
Cargo:	Estudiante – Investigadora Principal (Trabajo de Grado)		Secretaria de Salud Pública del Distrito Especial de Santiago de Cali

Este documento es propiedad de la Administración Central del Distrito de Santiago de Cali. Prohibida su alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

A.2. Formulario GC-1 para envío de proyectos

El siguiente documento corresponde al formulario institucional GC-1 diligenciado para la evaluación del proyecto ante el Comité de Ética e Investigaciones de la Secretaría de Salud Pública de Cali.

ALCALDÍA DE SANTIAGO DE CALI DESARROLLO SOCIAL SERVICIO DE SALUD PÚBLICA	MODELO INTEGRADO DE PLANEACIÓN Y GESTIÓN (MIPG) ENVÍO DE PROYECTOS PARA EVALUACIÓN	MEDE01.05.02.P025.F001-SS	
		VERSIÓN	001

FORMULARIO DE SOLICITUD DE EVALUACIÓN COMITÉ DE ÉTICA Y DE INVESTIGACIONES

Estimado(a) investigador(a):

Este formato se utiliza con el fin de tramitar los avales éticos e investigativos de los proyectos de investigación que realizan algún tipo de solicitud a la Secretaría Distrital de Salud Pública de Cali. Le recomendamos leer el formulario en su totalidad y diligenciar todo aquello que compete de acuerdo con las características de su proyecto. **No elimine ninguna sección del formato original** para evitar reprocesos.

Tenga en cuenta que si está solicitando datos a la Secretaría Distrital de Salud Pública de Cali es porque su proyecto tiene impacto en la ciudad, de lo contrario, no es aplicable para revisión.

Finalmente, no se tendrán en cuenta propuestas que no estén relacionadas con proyectos investigativos en salud.

1. **FECHA DE SOMETIMIENTO:** 24 de febrero del 2026

2. **TIPO DE SOLICITUD:**

- Aval por comité de investigaciones y ética para el desarrollo de investigación generada por alguna de las áreas de la Secretaría Distrital de Salud Pública. ☐

- Aval por comité de investigaciones y ética para el desarrollo de investigación colaborativa con la Secretaría Distrital de Salud Pública. ☐

- Aval por comité de ética e investigaciones para acceder a información que no se encuentra en datos abiertos de la Secretaría Distrital de Salud Pública con propósitos de investigación y generación de conocimiento. ☐

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

- Financiación para el desarrollo de investigación de interés de la Secretaría Distrital de Salud Pública □

- Otras. □ ¿Cuáles? Haga clic o pulse aquí para escribir texto.

3. POR FAVOR, ESPECIFÍQUE LA SOLICITUD QUE DESEA HACERLE A LA SECRETARÍA DE SALUD:

La presente solicitud tiene como finalidad acceder a información epidemiológica agregada y anonimizada relacionada con eventos zoonóticos y sanitarios de interés en salud pública que afecten animales domésticos en la ciudad de Cali, en el marco del proyecto de grado titulado “Prototipo de una plataforma de monitoreo geoespacial colaborativo para la detección y notificación de Amenazas Zoosanitarias Urbanas en animales domésticos en la ciudad de Cali”. En este sentido, se solicita información agregada correspondiente al período comprendido entre 2023 y 2025 (últimos tres años epidemiológicos completos o el año más reciente disponible), con el fin de realizar análisis espaciales y temporales que permitan identificar patrones de recurrencia, concentración territorial y comportamiento estacional de dichos eventos en la zona urbana de la ciudad.

Específicamente, se solicita:

- Datos agregados por año y mes sobre eventos zoonóticos reportados en el sistema SIVIGILA con ocurrencia en la zona urbana de Cali.
- Distribución territorial de los casos por comuna o barrio (sin direcciones exactas ni datos personales).
- Clasificación del tipo de evento (por ejemplo: leptospirosis, rabia, brucelosis, toxoplasmosis, accidentes rábicos u otros eventos priorizados por la Secretaría).

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

- Información consolidada sobre reportes de presencia de especies de interés sanitario como el caracol africano (*Achatina fulica*), con ubicación territorial aproximada (comuna o barrio).

Se aclara expresamente que no se solicita información nominal, direcciones exactas, datos clínicos individuales ni ningún dato personal identificable. La información será utilizada exclusivamente con fines académicos y de investigación, bajo criterios de anonimización, agregación estadística y cumplimiento de la normatividad vigente en protección de datos.

El propósito del análisis es identificar patrones espaciales de riesgo y apoyar el diseño de una herramienta tecnológica de monitoreo geoespacial que contribuya a la detección temprana y comunicación de amenazas zoosanitarias urbanas en beneficio de la salud animal y la salud pública.

El proyecto no implica intervención directa sobre personas ni animales, y se limita al análisis de datos secundarios agregados.

4. ÁMBITO DE PRESENTACIÓN DEL PROYECTO

- Para presentar a convocatoria nacional □
- Para presentar a convocatoria internacional □
- Para convocatoria interna de institución universitaria □
- Para presentar a revista científica □
- Otros □

¿Cuáles? Haga clic o pulse aquí para escribir texto.

5. INFORMACIÓN DEL PROYECTO

4.1 Título:

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

"Prototipo de una plataforma de monitoreo geoespacial colaborativo para la detección y notificación de Amenazas Zoosanitarias Urbanas (AZU) en animales domésticos en la ciudad de Cali."

4.2 Institución(es):
Pontificia Universidad Javeriana Cali
Facultad de Ingeniería y Ciencias
Programa de Ingeniería de Sistemas y Computación
4.3 Investigador principal (nombres y apellidos, filiación y correo):
Laura Camila Franco Valencia
Estudiante de Ingeniería de Sistemas y Computación
Pontificia Universidad Javeriana Cali
lcfranco@javerianacali.edu.co
4.4 Coinvestigadores (nombres y apellidos, filiación y correo):
Nestor Mauricio Ortiz Montenegro
Estudiante de Ingeniería de Sistemas y Computación
Pontificia Universidad Javeriana Cali
nmortizm@javerianacali.edu.co
Director del proyecto:
Juan Carlos Martínez Arias
Ingeniero de Sistemas – Docente e investigador
Pontificia Universidad Javeriana Cali
juancmartinez@javerianacali.edu.co

4.5 Nombre y área del funcionario de la SDS vinculado al proyecto en caso de que aplique:

4.6 Introducción del proyecto:
En la ciudad de Cali, donde aproximadamente el 60 % de los hogares conviven con al menos un perro y una proporción significativa cuenta también con otras especies domésticas, los animales están expuestos a múltiples riesgos ambientales y sanitarios que no siempre son visibles para sus tutores. Entre estos se encuentran enfermedades zoonóticas, focos de contaminación ambiental y la presencia de especies invasoras como el caracol gigante africano (Achatina fulica).

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

Actualmente, la información relacionada con estos eventos no se encuentra centralizada ni disponible en tiempo real para la ciudadanía, lo que limita la prevención y la toma de decisiones informadas. En este contexto, el proyecto propone el desarrollo de un prototipo de plataforma tecnológica de monitoreo geoespacial colaborativo que permita detectar, categorizar y notificar Amenazas Zoosanitarias Urbanas (AZU) en la zona urbana de Cali, integrando datos oficiales e información reportada por la comunidad.

La herramienta busca fortalecer la comunicación entre ciudadanía e instituciones, contribuyendo al bienestar animal y apoyando estrategias de salud pública desde un enfoque tecnológico e innovador.

4.7 Problema/Justificación:

En Cali existe una creciente exposición de animales domésticos a amenazas sanitarias y ambientales tales como leptospirosis, rabia, brucelosis, toxoplasmosis, accidentes rábicos y presencia de especies invasoras. Sin embargo, no se dispone de un sistema tecnológico que permita consolidar, visualizar y difundir esta información de manera georreferenciada y accesible en tiempo real para los tutores.

La ausencia de mecanismos de alerta temprana y monitoreo espacial dificulta la prevención de riesgos y la gestión oportuna de eventos sanitarios. Por ello, se justifica el desarrollo de una plataforma que integre información epidemiológica agregada, datos territoriales y reporte colaborativo, con el fin de generar análisis espaciales y temporales que apoyen la toma de decisiones y fortalezcan la protección del bienestar animal en la ciudad.

4.8 Pregunta de investigación:

¿Cómo desarrollar una herramienta tecnológica que permita monitorear, geolocalizar y reportar en tiempo real la presencia de Amenazas Zoosanitarias Urbanas (AZU) en la zona urbana de la ciudad de Cali, con el fin de proteger la salud de los animales domésticos y contribuir al bienestar animal mediante un modelo colaborativo de información?

4.9 Marco teórico (máx. página y media):

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

El proyecto se fundamenta en la integración de conceptos del dominio zoosanitario urbano y tecnologías geoespaciales. Las Amenazas Zoosanitarias Urbanas (AZU) se definen como factores biológicos y ambientales presentes en entornos urbanos que representan un riesgo para la salud de los animales domésticos. Estas incluyen enfermedades zoonóticas, focos de contaminación ambiental y especies invasoras.

Desde el componente tecnológico, la propuesta integra sistemas de información geográfica (GIS), bases de datos espaciales y herramientas de geolocalización que permiten representar información territorial mediante coordenadas geográficas. La geolocalización posibilita ubicar espacialmente eventos sanitarios, mientras que los GIS facilitan el análisis y visualización de patrones espaciales y zonas de riesgo.

El proyecto también se sustenta en el enfoque de participación ciudadana y crowdsourcing, donde los ciudadanos actúan como “sensores sociales” generando datos espaciales relevantes. Esta combinación entre datos oficiales agregados y reporte colaborativo fortalece la vigilancia y la detección temprana de riesgos.

4.10 **Objetivo General:**
Desarrollar un prototipo de una herramienta tecnológica que permita monitorear, geolocalizar y reportar en tiempo real la presencia de amenazas zoosanitarias urbanas (AZU) en la zona urbana de la ciudad de Cali, con la finalidad de proteger la salud de los animales domésticos y contribuir a su bienestar mediante el reporte colaborativo de información y la detección temprana de alertas.

4.11 Objetivos Específicos:

1. Identificar y clasificar las amenazas zoosanitarias urbanas que afectan a los animales domésticos en la ciudad de Cali, a partir de fuentes heterogéneas como sensores simulados, reportes ciudadanos y datos oficiales.
2. Analizar las limitaciones actuales en el acceso a la información y en los mecanismos de alerta sobre riesgos, con el fin de proponer una estrategia

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

de comunicación y alerta temprana frente a las amenazas zoosanitarias urbanas (AZU).
3. Diseñar e implementar un prototipo tecnológico basado en un modelo colaborativo y en tiempo real, que integre funciones de monitoreo, geolocalización y notificación de amenazas zoosanitarias urbanas (AZU).
4. Evaluar el prototipo mediante un proceso de aseguramiento de la calidad (QA) que incluya la ejecución de pruebas funcionales, de integración y de validación de requisitos, con el fin de medir su desempeño en términos de precisión, confiabilidad y cumplimiento de especificaciones técnicas.

4.12 Tipo de estudio:
El presente proyecto se desarrolla bajo un enfoque exploratorio-aplicado, ya que busca caracterizar Amenazas Zoosanitarias Urbanas (AZU) en la ciudad de Cali mediante la creación de un prototipo tecnológico de monitoreo geoespacial colaborativo, que permita gestionar información sobre la detección, categorización y difusión comunitaria de las AZU.
4.13 Población:
La población de interés corresponde a los eventos zoonóticos y sanitarios asociados a animales domésticos reportados en el área urbana de Cali, así como la comunidad de tutores que potencialmente interactuarán con la plataforma de monitoreo geoespacial.
4.14 Muestra, muestreo y fuente de información:

La muestra estará constituida por los registros agregados de eventos zoonóticos y sanitarios reportados en el período 2023–2025 (o el año más reciente disponible), con ocurrencia en la zona urbana de Cali y desagregados territorialmente por comuna o barrio. Se empleará un muestreo no probabilístico de tipo censal, considerando la totalidad de registros disponibles que cumplan con los criterios temporales y territoriales definidos.

Las fuentes de información incluirán datos agregados y anonimizados del sistema SIVIGILA suministrados por la Secretaría de Salud Pública de Cali, reportes oficiales del Centro de Zoonosis, así como reportes colaborativos generados mediante el prototipo tecnológico.

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

4.15 En caso de ser un estudio de intervención, incluir punto de quiebre:
El presente proyecto no corresponde a un estudio de intervención clínica o experimental sobre sujetos humanos o animales. Se trata del desarrollo y validación de un prototipo tecnológico de monitoreo geoespacial basado en información agregada y anonimizada. Por tanto, no aplica la definición de punto de quiebre.

4.16 Criterios de inclusión (justifique el plan de análisis)
Se incluirán en el análisis:

• Registros agregados y anonimizado de eventos zoonóticos reportados en el sistema SIVIGILA con ocurrencia en la zona urbana de Cali durante el período 2023–2025 (o el año más reciente disponible).
• Información territorial desagregada por comuna o barrio, sin direcciones exactas ni datos personales.
• Eventos relacionados con enfermedades zoonóticas y riesgos sanitarios que puedan afectar animales domésticos (perros y gatos), tales como leptospirosis, rabia, brucelosis, toxoplasmosis, accidentes rábicos u otros priorizados por la Secretaría de Salud.
• Reportes oficiales sobre presencia de especies invasoras de interés sanitario como Achatina fulica.

Estos criterios permiten realizar análisis espaciales y temporales orientados a identificar patrones de recurrencia, concentración territorial y comportamiento estacional de las amenazas zoosanitarias urbanas.

4.17 Criterios de exclusión (justifique el plan de análisis):
Se excluirán del análisis:

• Registros que contengan información personal identificable (nombres, direcciones exactas, números de identificación u otros datos sensibles).
• Eventos ocurridos fuera del área urbana del Distrito de Santiago de Cali.
• Registros incompletos que no contengan información territorial mínima para su análisis (comuna o barrio).

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

• Eventos no relacionados con riesgos zoosanitarios que afecten animales domésticos.

Estos criterios garantizan el cumplimiento de principios de confidencialidad y protección de datos personales, así como la pertinencia técnica del análisis.

4.18 Cuestionarios y/o pruebas a utilizar durante la investigación (justifique):
Durante la etapa de validación del prototipo se emplearán:

• Pruebas funcionales para verificar el correcto funcionamiento de los módulos del sistema (base de datos, visualización geoespacial y reporte colaborativo).
• Pruebas de integración para evaluar la interoperabilidad entre los componentes tecnológicos.
• Pruebas de desempeño para medir tiempos de respuesta, latencia y estabilidad del sistema.
• Pruebas de usabilidad con un grupo piloto de tutores voluntarios, mediante encuestas estructuradas de percepción (escala Likert) para evaluar facilidad de uso, claridad de la información y utilidad percibida.

Estas pruebas permitirán evaluar tanto el desempeño técnico como la aceptación social del prototipo.

4.19 Plan de gestión de datos:
La gestión de datos se realizará bajo principios de confidencialidad, anonimización y uso responsable de la información.

• Solo se trabajará con datos agregados y anonimizados suministrados por entidades oficiales.
• No se almacenarán datos personales identificables.
• La información será utilizada exclusivamente con fines académicos en el marco del proyecto de grado.
• Los datos serán almacenados en entornos seguros con control de acceso restringido a los investigadores.

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

• Los resultados se presentarán de forma agregada, evitando cualquier posibilidad de reidentificación.

El tratamiento de la información cumplirá con la Ley 1581 de 2012 sobre protección de datos personales en Colombia.

6. ASPECTOS ÉTICOS (RELACIONADO CON LA NORMATIVA 8430 DE 1993)

El presente proyecto se clasifica como una investigación **sin riesgo**, de acuerdo con la Resolución 8430 de 1993 del Ministerio de Salud de Colombia, ya que no implica intervención directa sobre seres humanos ni animales, ni manipulación de variables biológicas, clínicas o comportamentales.

La investigación se basa en el análisis de datos agregados y anonimizados, así como en el desarrollo de un prototipo tecnológico con fines académicos.

5.**1 Justifique el manejo, almacenamiento y acceso de datos:**

El proyecto trabajará exclusivamente con datos agregados y anonimizados suministrados por entidades oficiales (por ejemplo, SIVIGILA, Secretaría de Salud, Centro de Zoonosis y CVC), sin acceso a información personal identificable.

Los datos serán almacenados en entornos digitales seguros, con acceso restringido únicamente a los investigadores responsables del proyecto.

La información será utilizada exclusivamente con fines académicos y de investigación, garantizando la confidencialidad, integridad y uso responsable de los datos conforme a la Ley 1581 de 2012 sobre protección de datos personales.

Requiere consentimiento informado. Sí □ No □

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

5.2 En caso de que su respuesta sea negativa justifique:

No se requiere consentimiento informado debido a que el estudio no involucra intervención directa sobre personas ni animales, ni acceso a datos personales identificables. La investigación se realiza sobre información agregada y anonimizada, y en el caso de pruebas de usabilidad, se solicitará participación voluntaria sin recolección de datos sensibles.

5.3 ¿El proyecto cuenta con financiación? en caso de que la respuesta sea positiva especifique el nombre de la institución que lo financia:

No. El proyecto se desarrolla en el marco académico del programa de Ingeniería de Sistemas y Computación de la Pontificia Universidad Javeriana Cali y no cuenta con financiación externa.

5.4 ¿Alguno o varios de los investigadores tienen algún conflicto de interés que declarar?

Sí □ No □

5.5 En caso de que su respuesta sea positiva especifique el tipo de conflicto:

5.6 Presupuesto:

El presupuesto contempla principalmente la dedicación horaria de los investigadores y del director del proyecto, así como recursos tecnológicos (equipo de cómputo, conexión a internet, herramientas de desarrollo y almacenamiento).

El valor estimado corresponde a la dedicación académica de los investigadores y del director, conforme al presupuesto presentado en el anteproyecto.

	Valor unitario	Semanas	Unidades por semana	Subtotal	Descripción
Director del proyecto de grado	$85,000	24	1	$ 2,040,000	Reunión semanal con el director del proyecto
Autores del proyecto de grado	$30,000	24	5	$ 3,600,000	Trabajo conjunto de los autores durante 12 horas a lo largo del día por 5 días a la semana
Alimentación	$60,000	24	5	$ 7,200,000	Alimentación estimada para los 5 días a la semana en los que se trabajará en el proyecto
Energía	$2,080	24	5	$ 249,600	Costo diario del consumo eléctrico estimado para el año 2026
Internet	$3,000	24	5	$ 360,000	Costo diario estimado del costo del internet para el año 2026
Equipos	$3,000,000	N/A	2	$ 6,000,000	Computador Portátil
Total				$19,449,600	

5.7 Cronograma:

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

El proyecto se desarrollará en un período aproximado de 16 semanas académicas, distribuidas en cuatro etapas:

1. Investigación y diagnóstico.
2. Análisis y diseño del sistema.
3. Construcción y prototipado.
4. Validación y socialización.

Etapa	Meses	1				2				3				4			
	Actividades/Semana	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Investigación y Diagnóstico	Analizar literatura académica y reportes institucionales relevantes.																
	Recopilar información oficial de entidades locales (Secretaría de Salud, CVC, Centro de Zoonosis de Cali).																
	Clasificar enfermedades zoonóticas y especies invasoras que afectan a los animales domésticos.																
	Validar y contextualizar la información recopilada con actores institucionales.																
	Estudiar experiencias previas de plataformas colaborativas (Mosquito Alert, The Tick App).																
Análisis y Diseño del Sistema	Examinar canales y procesos de recolección y difusión de datos zoosanitarios en Cali																
	Definir requisitos funcionales (módulos de geolocalización, reportes, visualización), no funcionales (seguridad, disponibilidad, usabilidad) y de información.																
	Establecer la arquitectura tecnológica general de la plataforma.																
	Diseñar el modelo conceptual de la base de datos geoespacial.																
	Diseñar estrategias de participación ciudadana																
	Diseñar el flujo de comunicación y monitoreo																
	Elaborar conceptualmente la interfaz y experiencia de usuario																
Construcción y Prototipado	Implementar la base de datos georreferenciada con soporte espacial.																
	Desarrollar los mapas interactivos para la representación de zonas de riesgo.																
	Integrar el sistema de reportes ciudadanos en la plataforma.																
	Realizar pruebas unitarias y de integración parcial de los módulos implementados.																
Validación y Socialización	Ejecutar pruebas funcionales para comprobar el desempeño de cada módulo.																
	Analizar la experiencia de uso mediante la experiencia de actores seleccionados																
	Validar los requisitos del sistema frente a los objetivos del proyecto.																
	Realizar pruebas de aceptación con actores institucionales (Centro de Zoonosis, Secretaría de Salud, CVC).																
	Realizar ajustes y mejoras en el sistema con base en los resultados de las pruebas																
	Documentar resultados, conclusiones y recomendaciones																
	Realizar reuniones con el director del trabajo de grado																
	Realizar documento de entrega final																
	Preparación para la sustentación																
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

7. APORTES Y RELEVANCIA

6.1 Aportes de la investigación a la disciplina:

El proyecto aporta a la disciplina de Ingeniería de Sistemas y Computación mediante la integración de tecnologías geoespaciales, bases de datos espaciales y modelos de reporte colaborativo aplicados al campo de la salud pública veterinaria.

Desde el enfoque de ingeniería de software, el trabajo articula procesos de ingeniería de requisitos, diseño arquitectónico y validación mediante aseguramiento

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

de la calidad (QA), aplicados a un sistema con componente territorial y en tiempo real.

Asimismo, contribuye al desarrollo de soluciones tecnológicas orientadas a la vigilancia epidemiológica urbana, proponiendo un modelo replicable de monitoreo geoespacial colaborativo para amenazas zoosanitarias urbanas (AZU).

6.2 Aportes de la investigación al contexto regional o nacional:

A nivel regional, el proyecto propone una herramienta tecnológica orientada específicamente al contexto urbano de Cali, integrando datos oficiales y participación ciudadana para fortalecer la detección temprana de riesgos sanitarios que afectan animales domésticos.

En el contexto nacional, la propuesta puede servir como modelo replicable en otras ciudades que enfrentan problemáticas similares relacionadas con zoonosis, especies invasoras y riesgos ambientales urbanos.

El enfoque de integración entre datos institucionales y reporte ciudadano contribuye a la modernización de estrategias de vigilancia epidemiológica apoyadas en tecnologías digitales.

6.3 Aportes para la Secretaría Distrital de Salud:

El proyecto puede aportar a la Secretaría Distrital de Salud mediante:

• La visualización geoespacial consolidada de eventos zoonóticos y riesgos ambientales en la zona urbana.
• La identificación de patrones territoriales y temporales que podrían apoyar procesos de planeación y priorización de intervenciones.
• El fortalecimiento de la comunicación con la ciudadanía a través de un canal tecnológico complementario de alerta y difusión de información.
• La generación de un modelo piloto que articule vigilancia institucional con participación comunitaria.

La plataforma no reemplaza los sistemas oficiales existentes, sino que actúa como herramienta complementaria de apoyo tecnológico.

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

6.4 Resultados esperados:

Se espera obtener:

• Un prototipo funcional de plataforma de monitoreo geoespacial colaborativo.
• Un modelo de base de datos espacial estructurado para eventos zoonóticos urbanos.
• Mapas interactivos que representen zonas de riesgo por comuna o barrio.
• Un sistema de reporte colaborativo con mecanismos de validación.
• Un análisis espacial y temporal preliminar de amenazas zoosanitarias urbanas en Cali.
• Documentación técnica y académica del proceso metodológico y de validación.

6.5 Impactos esperados:

Entre los impactos esperados se encuentran:

• Mejora en el acceso ciudadano a información territorial sobre riesgos sanitarios para animales domésticos.
• Fortalecimiento de la cultura de reporte y participación comunitaria.
• Apoyo a procesos de toma de decisiones informadas por parte de tutores de mascotas.
• Potencial contribución a estrategias locales de vigilancia epidemiológica y bienestar animal.
• Generación de conocimiento aplicable en el ámbito de la ingeniería de software con enfoque en salud pública urbana.

8. OBSERVACIONES Y/O COMENTARIOS ADICIONALES:

El proyecto se desarrolla con fines académicos en el marco del programa de Ingeniería de Sistemas y Computación de la Pontificia Universidad Javeriana Cali. La información solicitada será utilizada exclusivamente con fines investigativos y no tendrá uso comercial.

La propuesta busca articular esfuerzos entre academia e institucionalidad, respetando los lineamientos de confidencialidad y protección de datos establecidos en la normativa vigente

9. REFERENCIAS:

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

ALCALDÍA DE SANTIAGO DE CALI DESARROLLO SOCIAL SERVICIO DE SALUD PÚBLICA	MODELO INTEGRADO DE PLANEACIÓN Y GESTIÓN (MIPG) ENVÍO DE PROYECTOS PARA EVALUACIÓN	MEDE01.05.02.P025.F001-SS	
		VERSIÓN	001

Las referencias bibliográficas y fuentes técnicas utilizadas se encuentran detalladas en el documento de anteproyecto adjunto, incluyendo literatura científica sobre geolocalización, sistemas de información geográfica, crowdsourcing y vigilancia epidemiológica.

Este documento es propiedad de la Administración Central del Distrito Santiago de Cali. Prohibido su copia, alteración o modificación por cualquier medio, sin previa autorización del Alcalde.

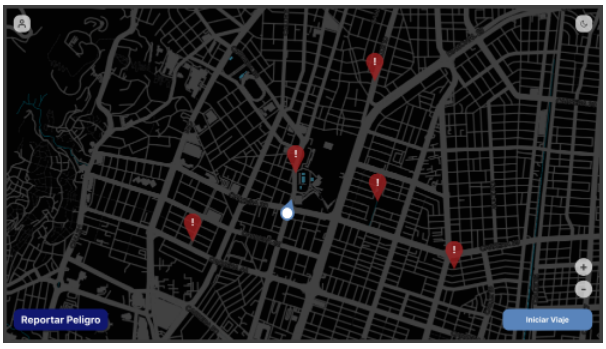
Apéndice B

Mockups diseñados en Figma

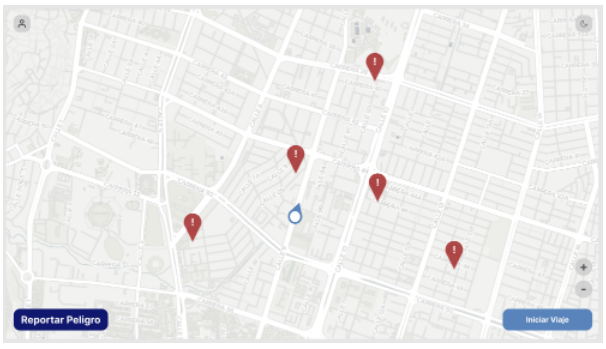
B.1. Catálogo de Interfaces

Este apartado contiene las capturas de pantalla de la aplicación organizadas por panel funcional, tipo de dispositivo y modo de visualización. Para cada panel se presentan las versiones web y móvil en modo oscuro y claro.

B.1.1. Vista general de la aplicación

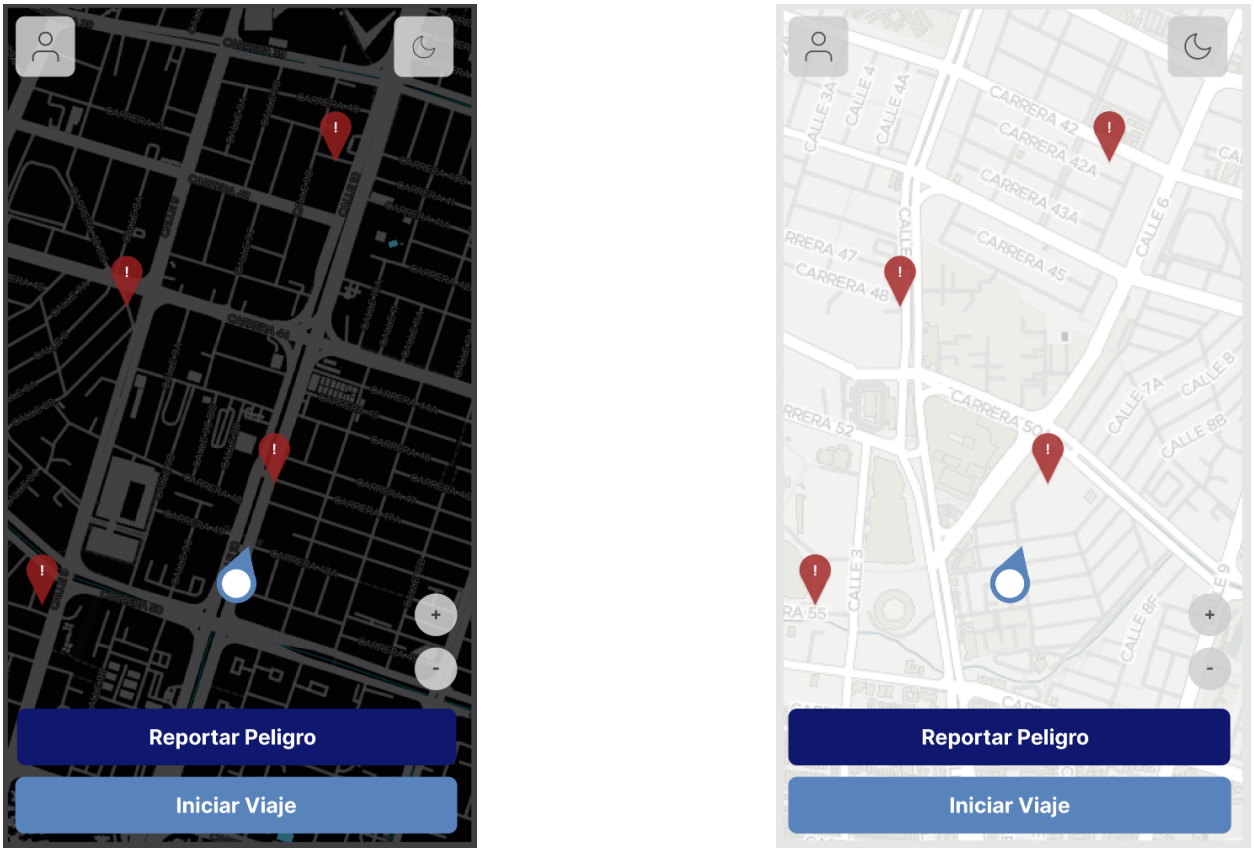


(a) Web - modo oscuro



(b) Web - modo claro

Figura B.1: Vista general de la aplicación en versión web

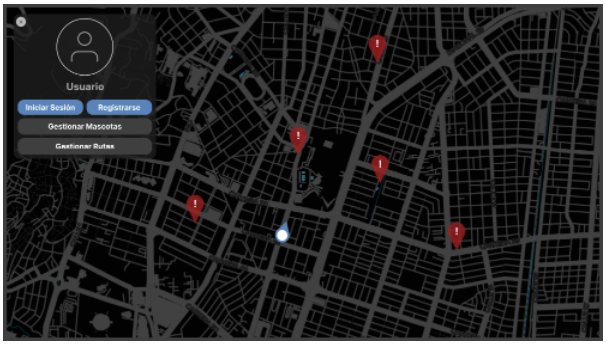


(a) Móvil - modo oscuro

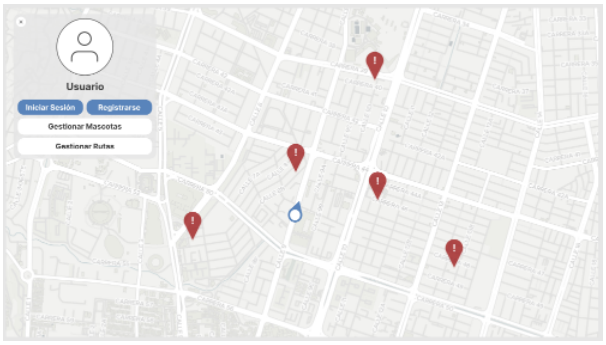
(b) Móvil - modo claro

Figura B.2: Vista general de la aplicación en versión móvil

B.1.2. Panel de vista del usuario sin iniciar sesión



(a) Web - modo oscuro



(b) Web - modo claro

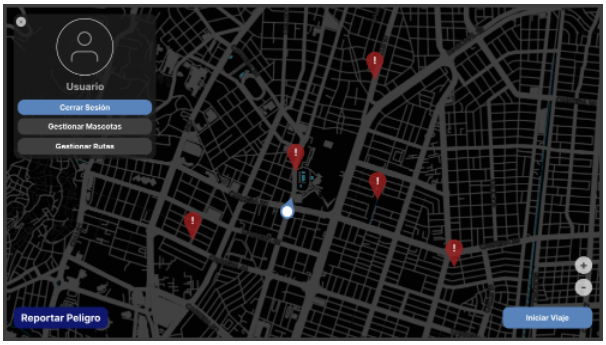
Figura B.3: Panel de vista del usuario sin iniciar sesión en versión web

(a) Móvil - modo oscuro

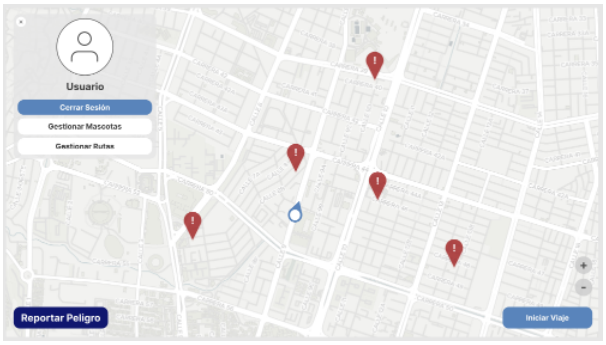
(b) Móvil - modo claro

Figura B.4: Panel de vista del usuario sin iniciar sesión en versión móvil

B.1.3. Panel de vista del usuario con sesión iniciada



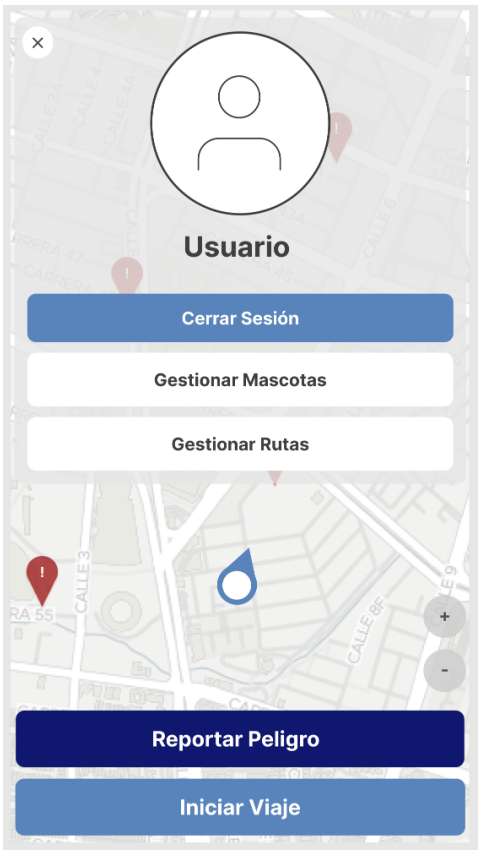
(a) Web - modo oscuro



(b) Web - modo claro

Figura B.5: Panel de vista del usuario con sesión iniciada en versión web

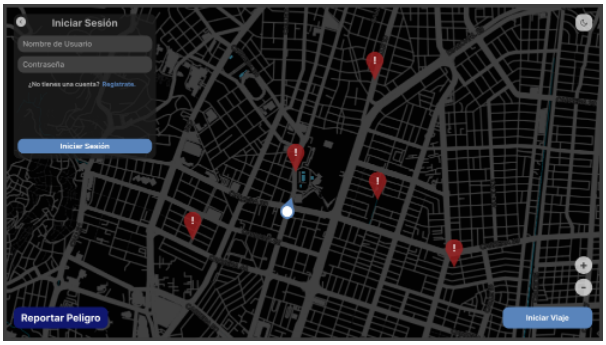
(a) Móvil - modo oscuro



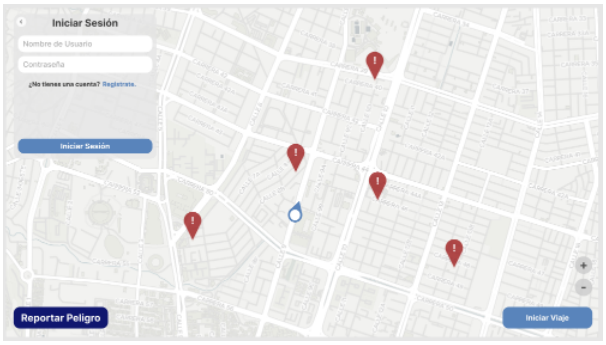
(b) Móvil - modo claro

Figura B.6: Panel de vista del usuario con sesión iniciada en versión móvil

B.1.4. Panel de inicio de sesión

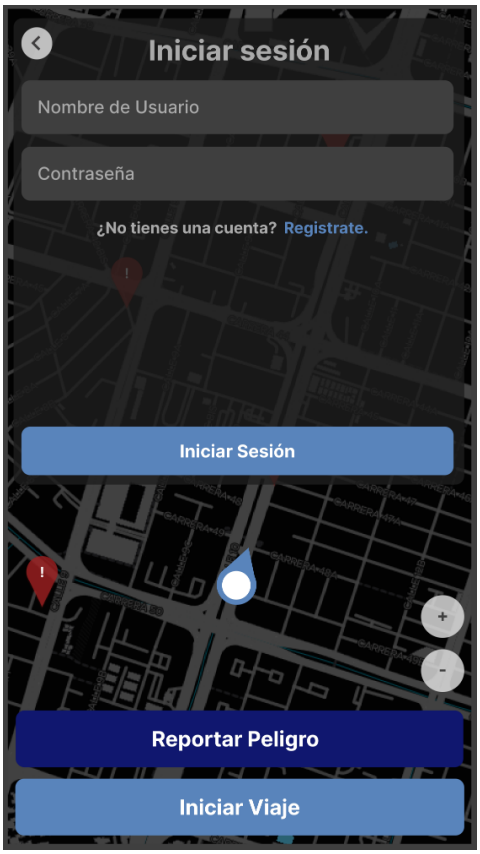


(a) Web - modo oscuro

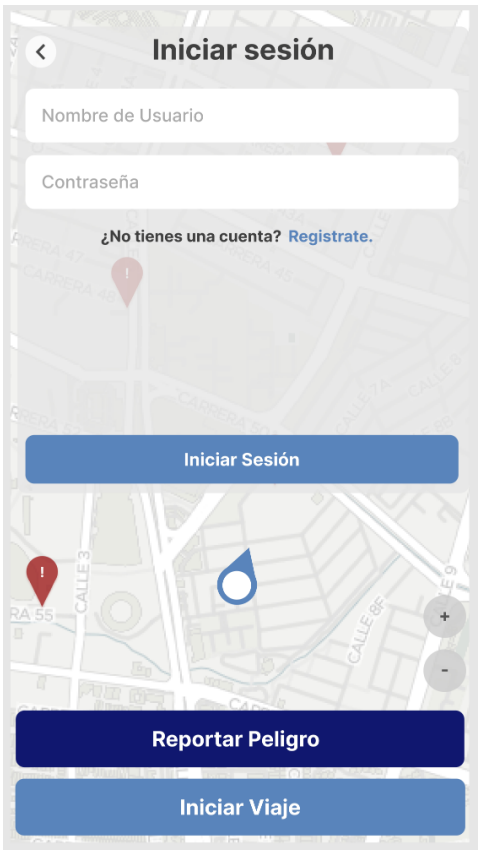


(b) Web - modo claro

Figura B.7: Panel de inicio de sesión en versión web



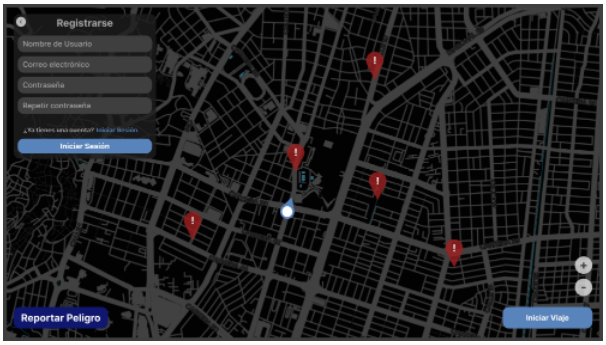
(a) Móvil - modo oscuro



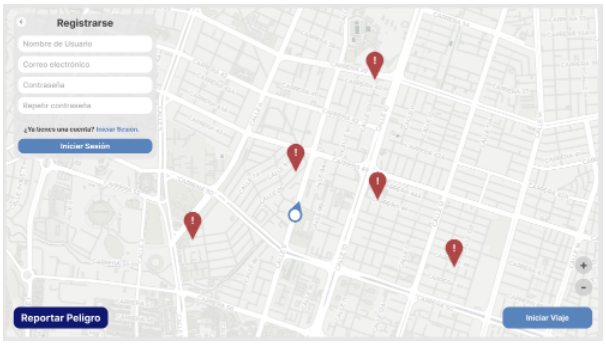
(b) Móvil - modo claro

Figura B.8: Panel de inicio de sesión en versión móvil

B.1.5. Panel de registro de usuario

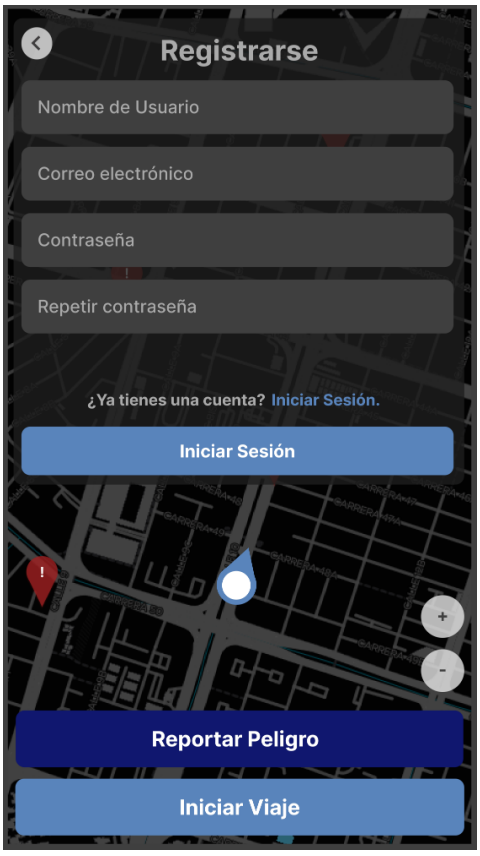


(a) Web - modo oscuro

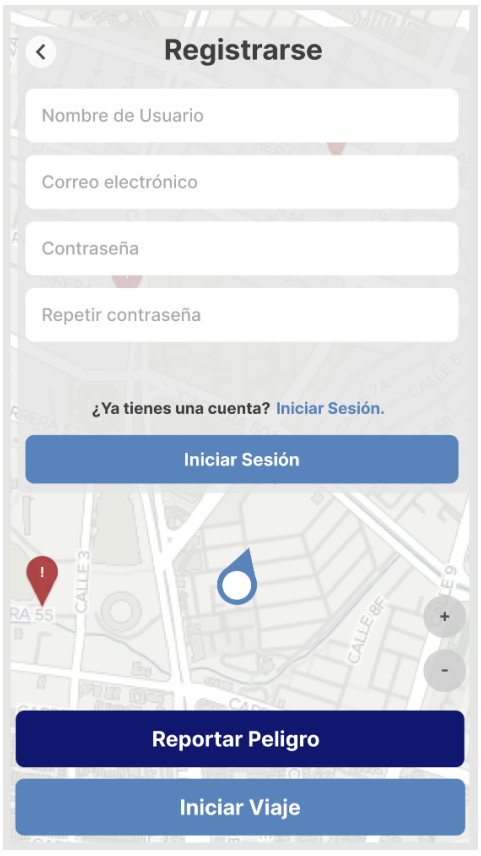


(b) Web - modo claro

Figura B.9: Panel de registro de usuario en versión web



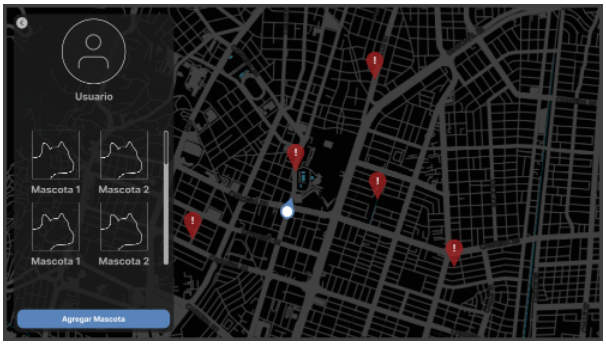
(a) Móvil - modo oscuro



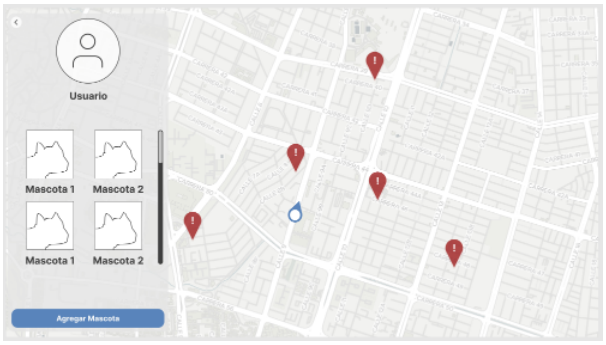
(b) Móvil - modo claro

Figura B.10: Panel de registro de usuario en versión móvil

B.1.6. Panel de gestión de mascotas



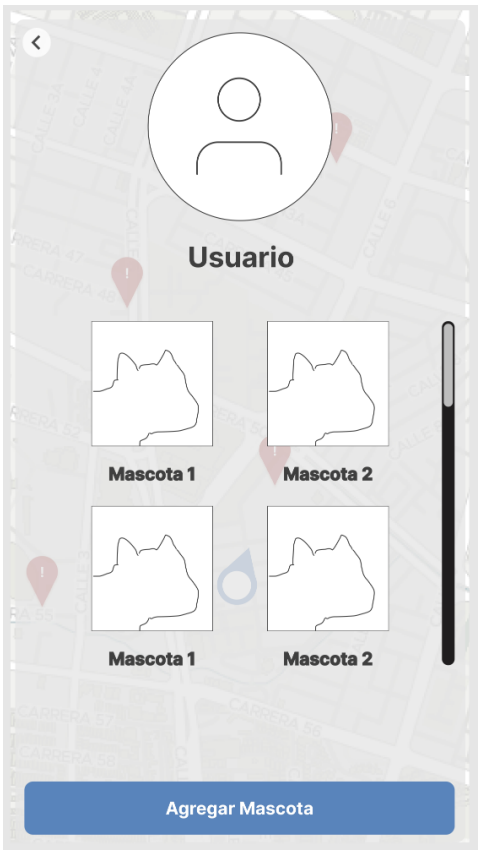
(a) Web - modo oscuro



(b) Web - modo claro

Figura B.11: Panel de gestión de mascotas en versión web

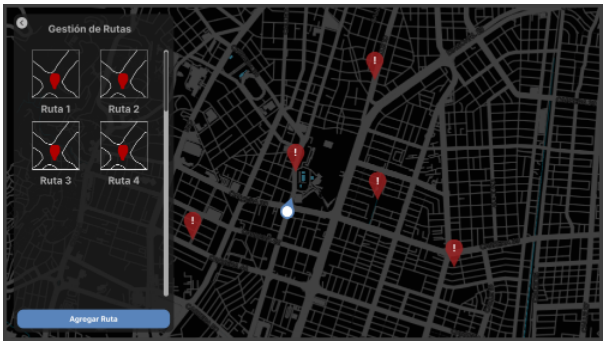
(a) Móvil - modo oscuro



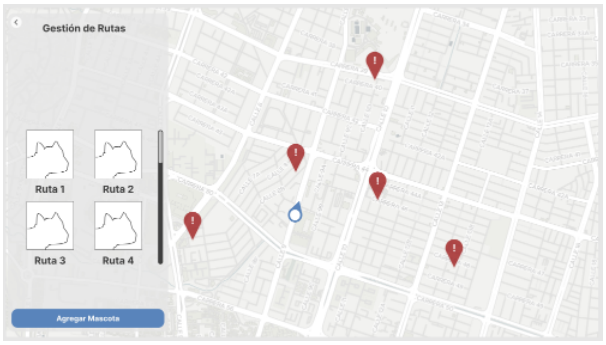
(b) Móvil - modo claro

Figura B.12: Panel de gestión de mascotas en versión móvil

B.1.7. Panel de gestión de rutas



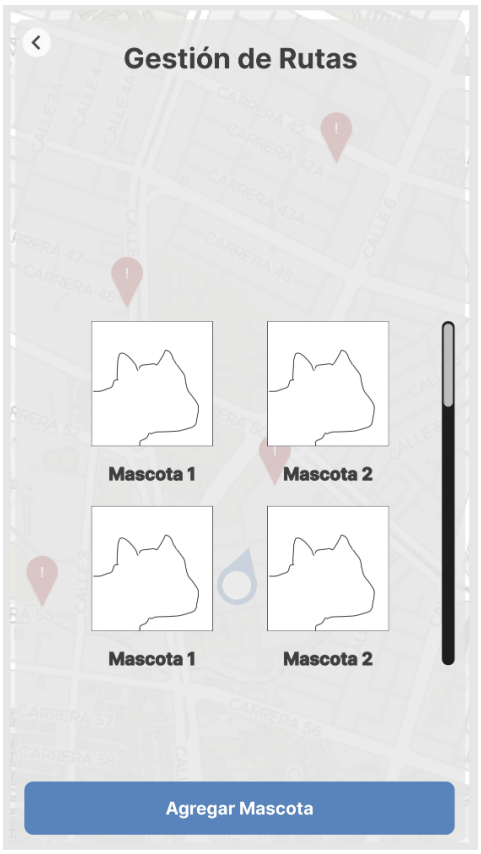
(a) Web - modo oscuro



(b) Web - modo claro

Figura B.13: Panel de gestión de rutas en versión web

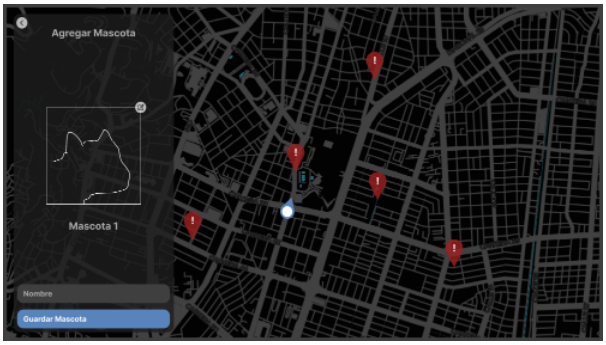
(a) Móvil - modo oscuro



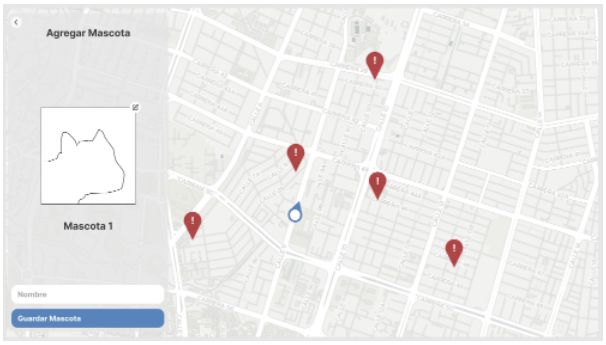
(b) Móvil - modo claro

Figura B.14: Panel de gestión de rutas en versión móvil

B.1.8. Panel de agregar mascota



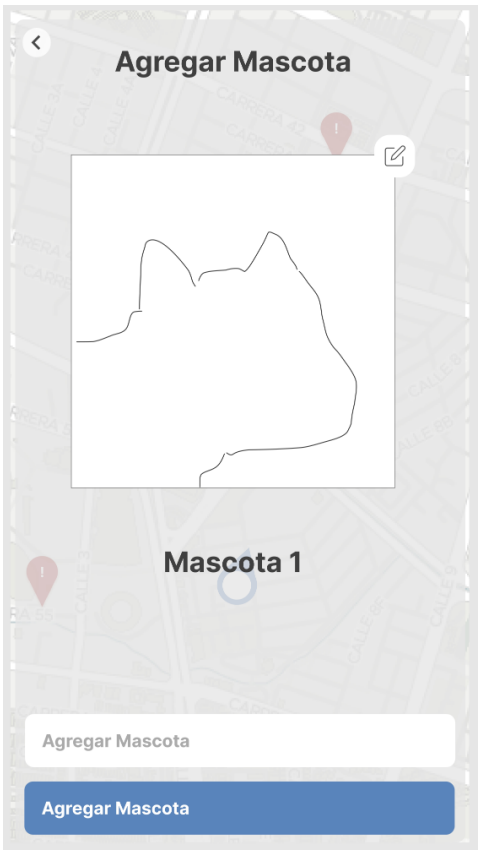
(a) Web - modo oscuro



(b) Web - modo claro

Figura B.15: Panel de agregar mascota en versión web

(a) Móvil - modo oscuro



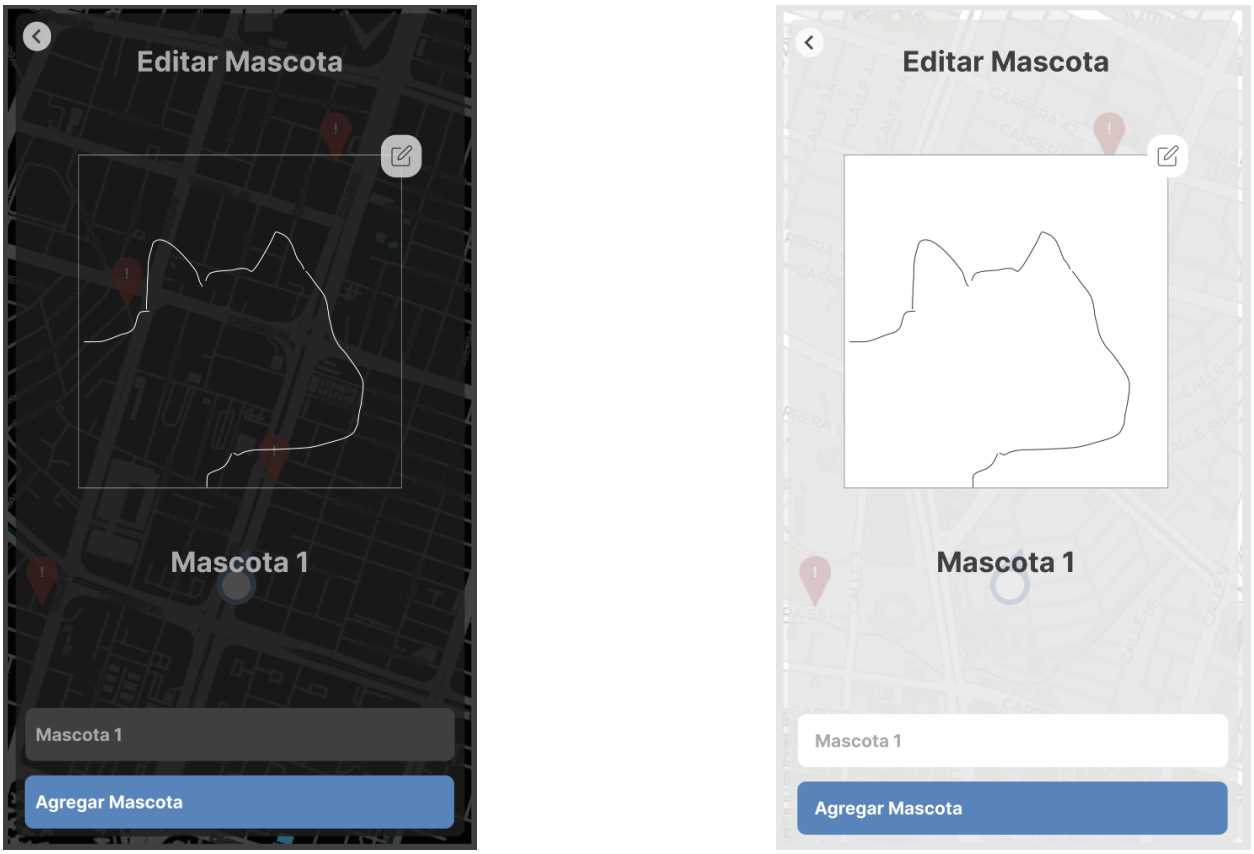
(b) Móvil - modo claro

Figura B.16: Panel de agregar mascota en versión móvil

B.1.9. Panel de editar datos de mascota

(a) Web - modo oscuro (b) Web - modo claro

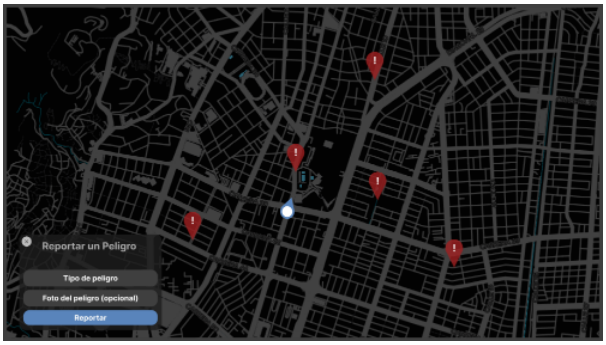
Figura B.17: Panel de edición de datos de mascota en versión web



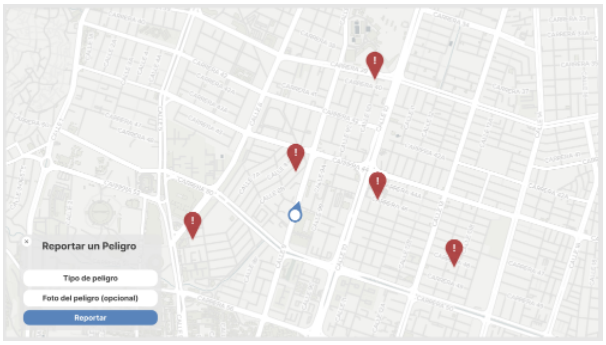
(a) Móvil - modo oscuro (b) Móvil - modo claro

Figura B.18: Panel de edición de datos de mascota en versión móvil

B.1.10. Panel de reportar un peligro

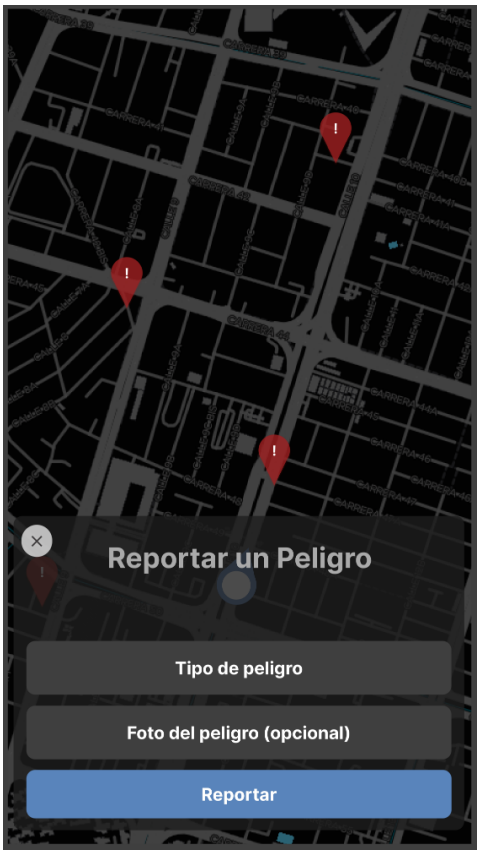


(a) Web - modo oscuro

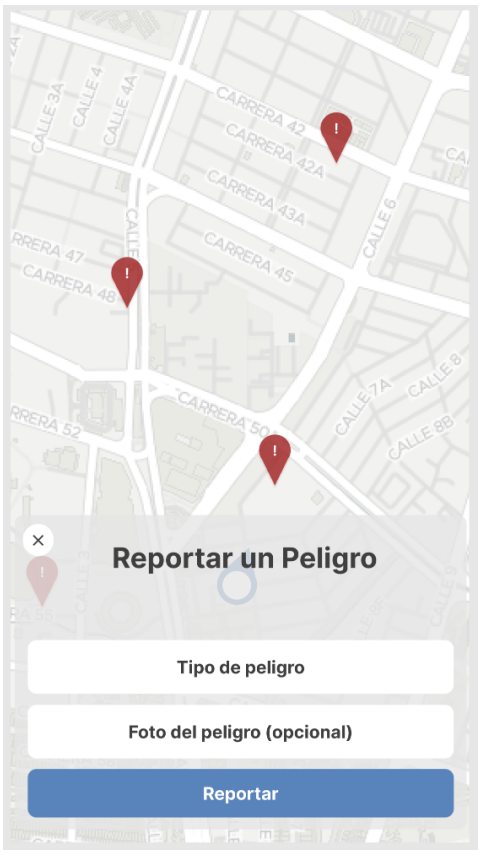


(b) Web - modo claro

Figura B.19: Panel de reporte de peligro en versión web



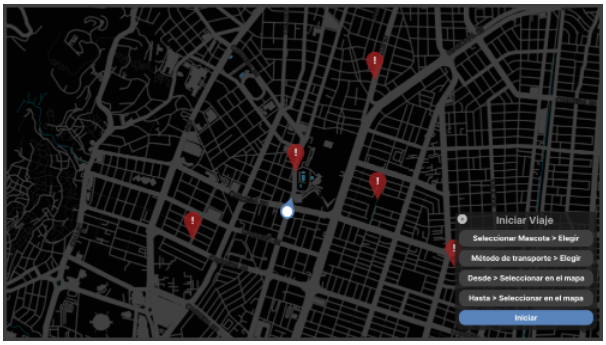
(a) Móvil - modo oscuro



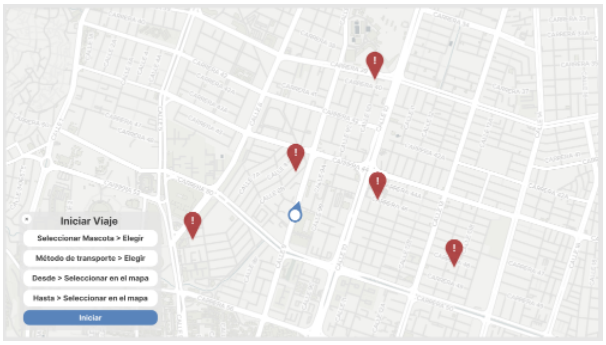
(b) Móvil - modo claro

Figura B.20: Panel de reporte de peligro en versión móvil

B.1.11. Panel de iniciar viaje

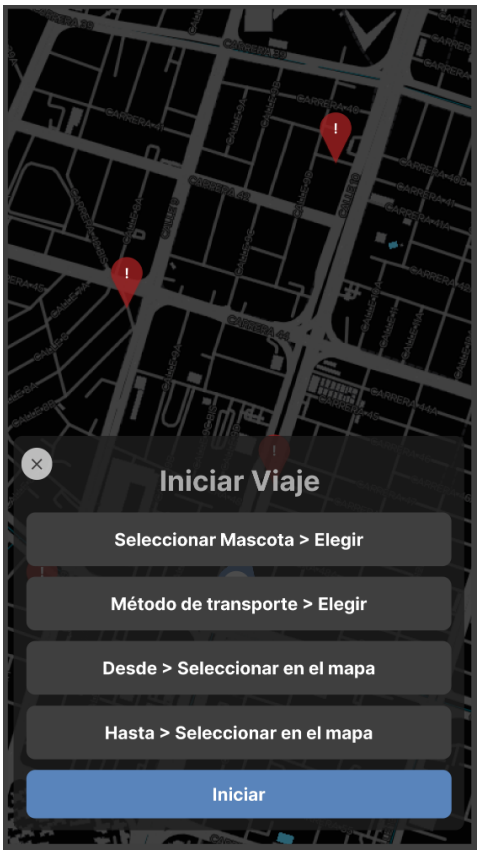


(a) Web - modo oscuro



(b) Web - modo claro

Figura B.21: Panel de inicio de viaje en versión web

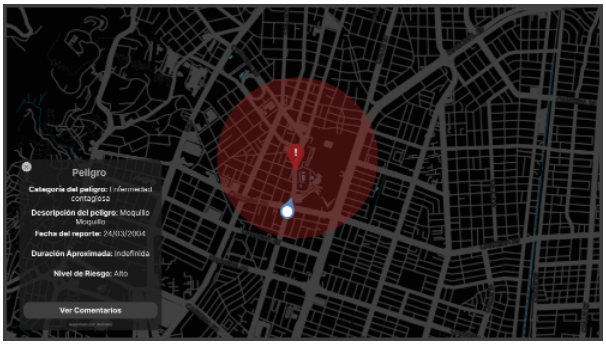


(a) Móvil - modo oscuro

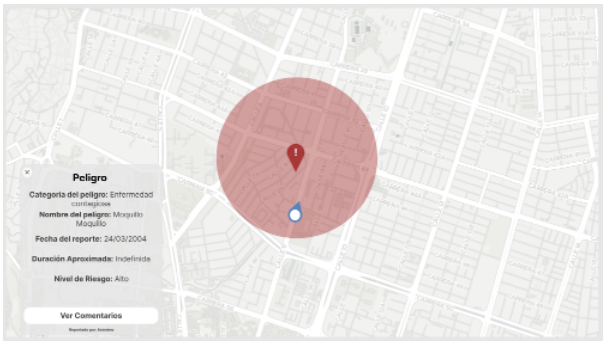
(b) Móvil - modo claro

Figura B.22: Panel de inicio de viaje en versión móvil

B.1.12. Panel de detalles de un peligro



(a) Web - modo oscuro



(b) Web - modo claro

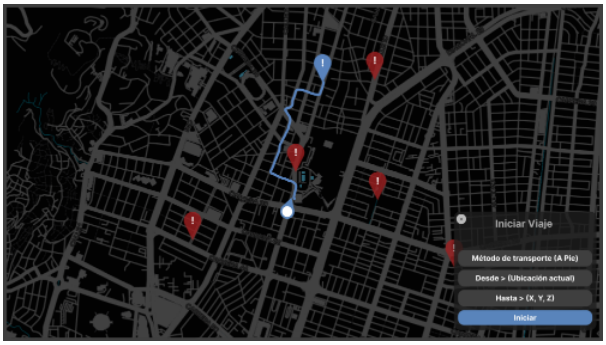
Figura B.23: Panel de detalles de un peligro en versión web

(a) Móvil - modo oscuro

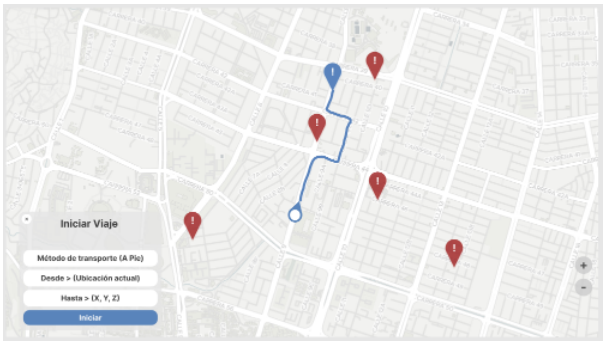
(b) Móvil - modo claro

Figura B.24: Panel de detalles de un peligro en versión móvil

B.1.13. Panel de iniciar viaje con ruta seleccionada

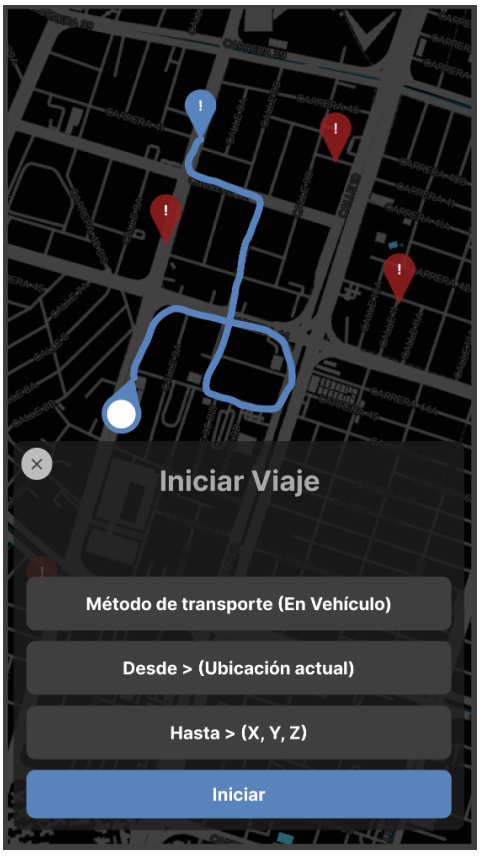


(a) Web - modo oscuro



(b) Web - modo claro

Figura B.25: Panel de inicio de viaje con ruta seleccionada en versión web

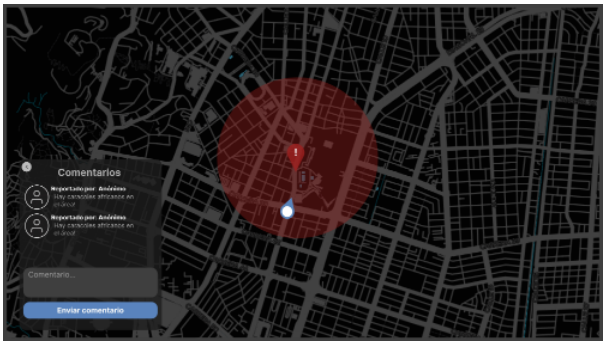


(a) Móvil - modo oscuro

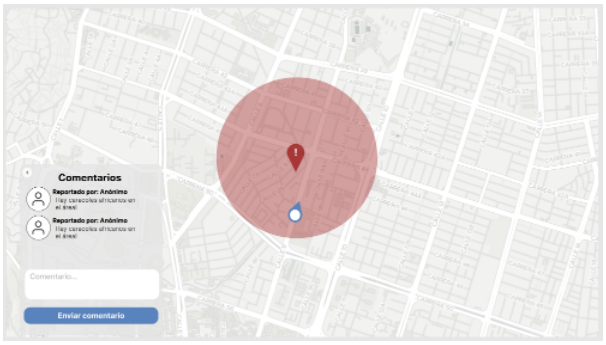
(b) Móvil - modo claro

Figura B.26: Panel de inicio de viaje con ruta seleccionada en versión móvil

B.1.14. Panel de comentarios de un peligro

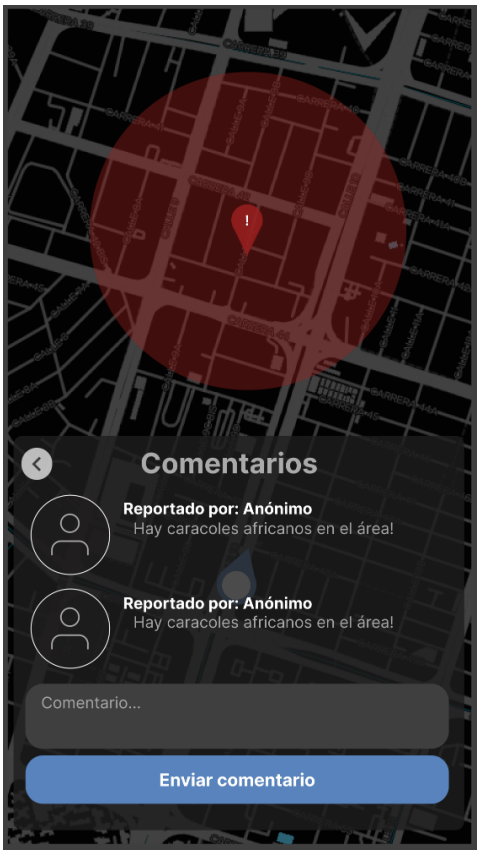


(a) Web - modo oscuro

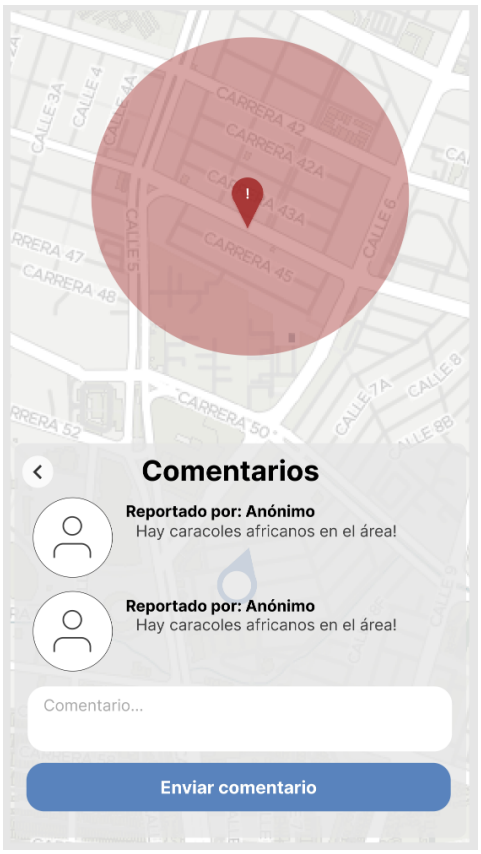


(b) Web - modo claro

Figura B.27: Panel de comentarios de un peligro en versión web



(a) Móvil - modo oscuro

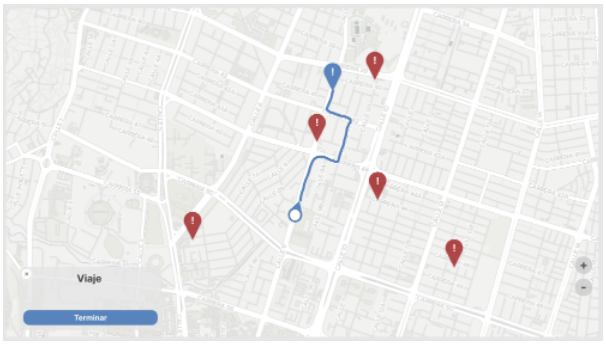


(b) Móvil - modo claro

Figura B.28: Panel de comentarios de un peligro en versión móvil

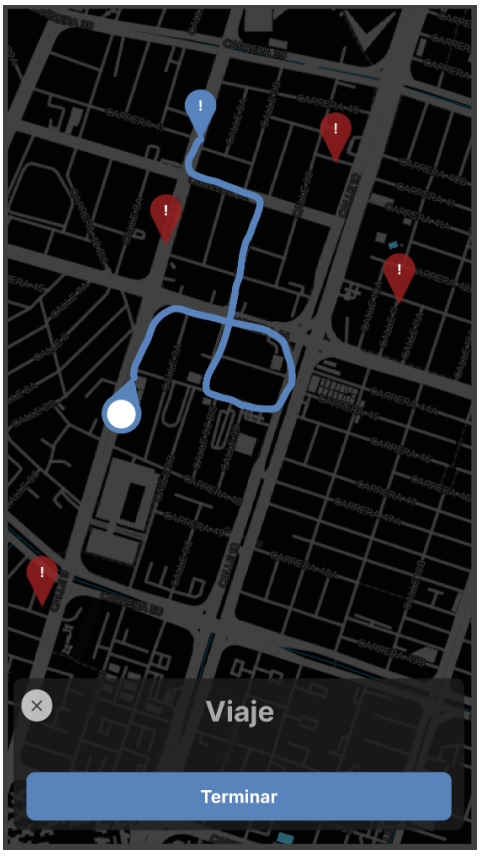
B.1.15. Panel de viaje en curso con ruta evitando un peligro

(a) Web - modo oscuro

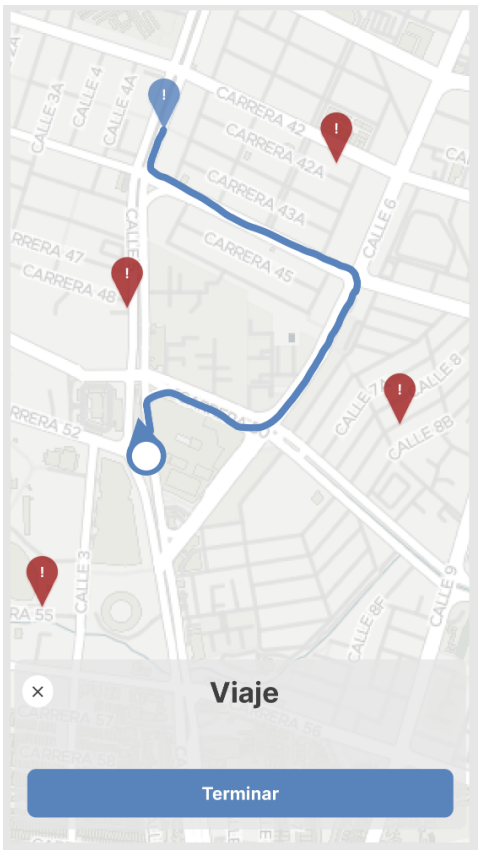


(b) Web - modo claro

Figura B.29: Panel de viaje en curso con ruta evitando un peligro en versión web



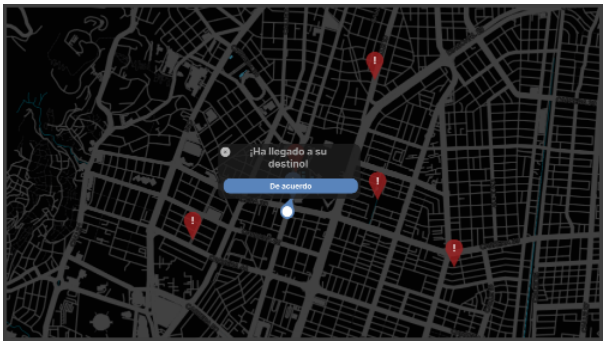
(a) Móvil - modo oscuro



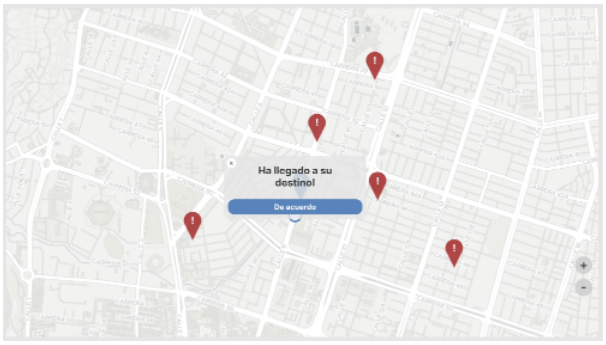
(b) Móvil - modo claro

Figura B.30: Panel de viaje en curso con ruta evitando un peligro en versión móvil

B.1.16. Panel de viaje completado

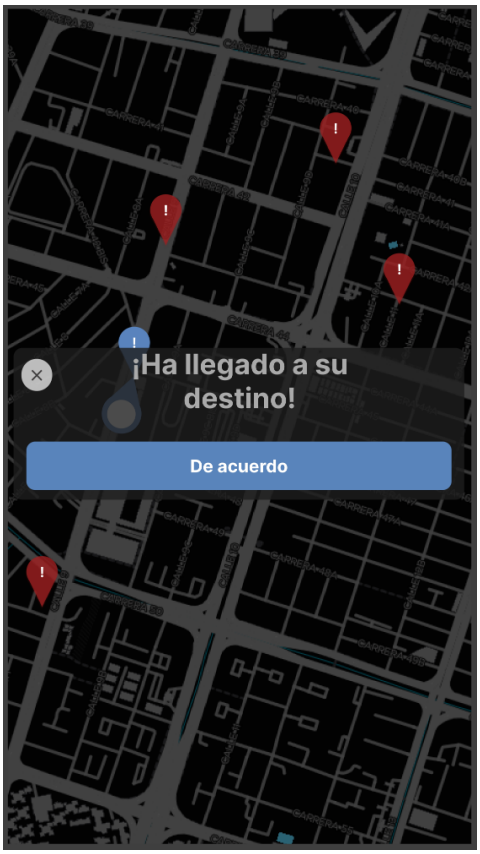


(a) Web - modo oscuro

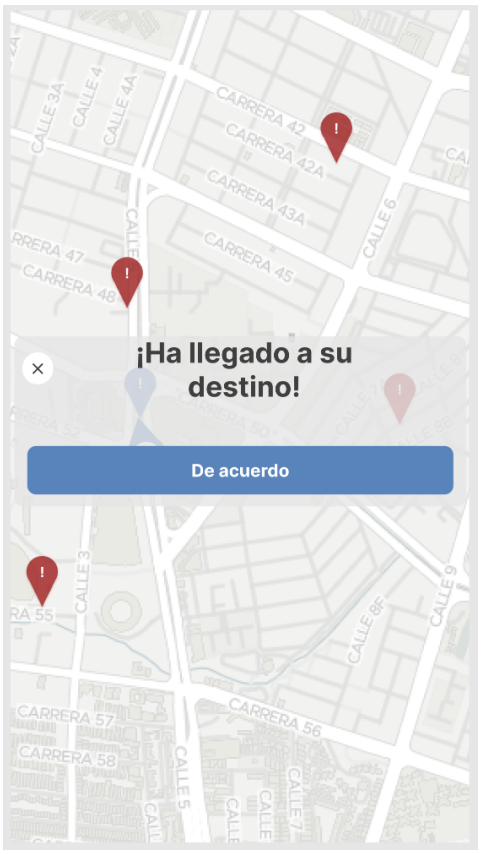


(b) Web - modo claro

Figura B.31: Panel de viaje completado en versión web



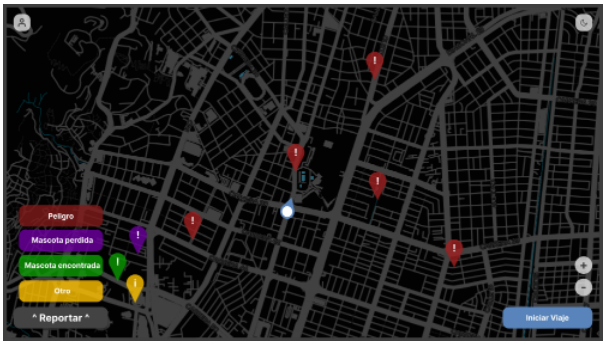
(a) Móvil - modo oscuro



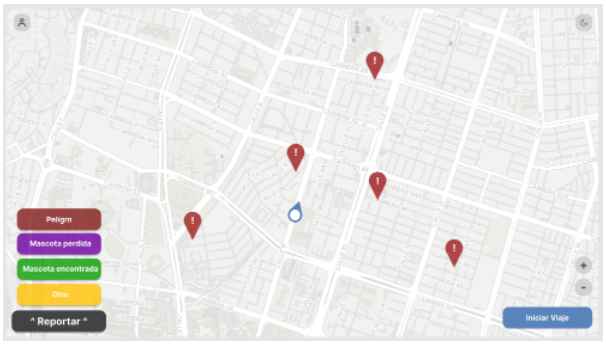
(b) Móvil - modo claro

Figura B.32: Panel de viaje completado en versión móvil

B.1.17. Panel de selección de reporte



(a) Web - modo oscuro



(b) Web - modo claro

Figura B.33: Panel de selección de reporte en versión web

(a) Móvil - modo oscuro

(b) Móvil - modo claro

Figura B.34: Panel de selección de reporte en versión móvil

Apéndice C

Documento de Especificación de Requerimientos de Software

C.1. Especificación de Requerimientos de Software (SRS)

El presente anexo corresponde al documento de Especificación de Requerimientos de Software (*Software Requirements Specification – SRS*) desarrollado para el sistema A4MPets.

Este documento consolida formalmente los requerimientos funcionales, no funcionales, de interfaz y arquitectónicos definidos durante las etapas iniciales del proyecto, sirviendo como base para el diseño, implementación y validación del sistema.

Dentro del SRS se describen las funcionalidades principales asociadas al monitoreo geoespacial de amenazas zoosanitarias urbanas, la gestión de mascotas, la generación de rutas preventivas, el sistema colaborativo de reportes y los mecanismos de interacción entre usuarios y componentes cartográficos.

Asimismo, el documento establece restricciones técnicas, criterios de calidad, requerimientos de usabilidad, lineamientos de persistencia de datos y consideraciones relacionadas con la arquitectura híbrida implementada durante el desarrollo del prototipo.

El SRS constituyó una referencia central para la trazabilidad del proyecto, permitiendo mantener coherencia entre los requerimientos definidos, los componentes implementados y las actividades de validación ejecutadas posteriormente durante las fases de pruebas QA.

Documento de Especificación de Requisitos de Software SRS

A4MPets

Sistema de monitoreo colaborativo de amenazas urbanas para animales domésticos

Índice

1. Introducción **3**
1.1. Propósito . . . 3
1.2. Alcance . . . 3
1.3. Glosario, siglas y abreviaturas . . . 3
1.4. Referencias . . . 5

2. Descripción General **5**
2.1. Descripción del contexto . . . 5
2.2. Perspectiva del Producto . . . 5
2.3. Características del Usuario . . . 5
2.4. Suposiciones y Dependencias . . . 6
2.5. Modelo C4 . . . 6
2.5.1. Diagrama de contexto . . . 6
2.5.2. Diagrama de contenedores . . . 7
2.5.3. Diagrama de componentes . . . 8
2.6. Riesgos . . . 10
2.7. Reglas del negocio . . . 11

3. Definición y Especificación de Requisitos **11**
3.1. Atributos de calidad . . . 11
3.2. Escenarios de calidad . . . 13
3.3. Modelo Entidad-Relación . . . 13
3.4. Diagrama de casos de uso . . . 14
3.5. Requisitos funcionales . . . 15
3.6. Requisitos no funcionales . . . 19
3.7. Requisitos de interfaz . . . 21
3.8. Mockups . . . 22

4. Otros requisitos **23**
4.1. Requisitos por implementar en versiones futuras . . . 23
4.2. Requisitos eliminados . . . 23

5. Proceso de construcción del documento de requisitos **24**

5.1. Estrategias de obtención de requisitos 24
5.2. Estrategias de priorización de requisitos 24
5.3. Estrategias y resultados de la validación de requisitos 24

6. Requisitos de las Pruebas **25**
6.1. Pruebas Unitarias 25
6.2. Pruebas de Integración 25
6.3. Pruebas de Sistema 25
6.4. Pruebas E2E 26
6.5. Pruebas Manuales 26
6.6. Pruebas de Rendimiento 26
6.7. Pruebas de regresión 27

1. Introducción

El presente documento corresponde a la especificación de requisitos de software del sistema A4MPets, una plataforma orientada al monitoreo, visualización y reporte colaborativo de amenazas urbanas que pueden afectar a animales domésticos en la ciudad de Cali. El documento organiza las necesidades funcionales, no funcionales, de interfaz y de validación del sistema, de manera que sirva como base para su diseño, implementación, verificación y evolución posterior.

La especificación se elaboró a partir del análisis del contexto urbano y la revisión documental sobre amenazas zoosanitarias, ambientales y zoosociales.

1.1. Propósito

El propósito de este SRS es definir de manera clara, verificable y trazable los requisitos que debe cumplir A4MPets para apoyar a tutores de animales domésticos en la identificación de riesgos presentes en el entorno urbano. El documento busca establecer qué debe hacer el sistema, bajo qué condiciones debe operar, qué restricciones debe respetar y qué criterios deben utilizarse para validar su cumplimiento.

1.2. Alcance

A4MPets tiene como alcance el desarrollo de un prototipo tecnológico para la ciudad de Cali que permita consultar un mapa interactivo, visualizar amenazas georreferenciadas, registrar reportes colaborativos, gestionar mascotas y rutas, configurar viajes y recibir alertas contextuales durante recorridos. El sistema se enfoca en riesgos que afectan a animales domésticos, principalmente perros y gatos, incluyendo amenazas zoosanitarias, ambientales y zoosociales.

El prototipo contempla la participación ciudadana como fuente de información, permitiendo que los usuarios registren reportes de peligros, mascotas perdidas, mascotas encontradas y otros eventos urbanos. Estos reportes se representan sobre el mapa y pueden enriquecerse mediante comentarios o evidencia visual.

1.3. Glosario, siglas y abreviaturas

A4MPets: Nombre del sistema especificado en este documento. Corresponde a una plataforma de monitoreo colaborativo de amenazas urbanas para animales domésticos.

Amenaza ambiental: Evento o condición del entorno físico que puede afectar la salud o bienestar de animales domésticos, como aguas estancadas, residuos o presencia de especies invasoras.

Amenaza urbana: Situación georreferenciada dentro de la ciudad que puede representar riesgo para animales domésticos. Puede ser zoosanitaria, ambiental o zoosocial.

Amenaza zoosanitara urbana (AZU): Riesgo asociado a enfermedades, vectores, agentes infecciosos o condiciones higiénico-sanitarias que pueden afectar animales domésticos en contextos urbanos.

Amenaza zoosocial: Problemática urbana no estrictamente sanitaria que afecta el bienestar animal, como desaparición, abandono o robo de mascotas.

APTR: Animales potencialmente transmisores de rabia.

C4: Modelo de documentación arquitectónica que permite representar un sistema mediante diagramas de contexto, contenedores y componentes.

CRUD: Conjunto de operaciones básicas sobre datos: crear, consultar, actualizar y eliminar.

CVC: Corporación Autónoma Regional del Valle del Cauca.

DAGMA: Departamento Administrativo de Gestión del Medio Ambiente de Cali.

E2E: Pruebas end-to-end. Validan flujos completos de usuario desde la interfaz hasta los servicios relacionados.

Geolocalización: Proceso de obtención o uso de coordenadas geográficas para ubicar usuarios, rutas o reportes.

MER: Modelo entidad-relación. Representa entidades, atributos y relaciones de la información manejada por el sistema.

Mockup: Representación visual preliminar de una pantalla o flujo de interfaz antes del desarrollo del sistema.

OpenStreetMap: Fuente cartográfica utilizada para la visualización del mapa base.

QA: Aseguramiento de calidad. Conjunto de actividades orientadas a verificar el cumplimiento funcional y técnico del sistema.

Reporte colaborativo: Registro creado por un usuario para comunicar una amenaza o evento urbano dentro del mapa.

RF: Requisito funcional.

RI: Requisito de interfaz.

RNF: Requisito no funcional.

SIVIGILA: Sistema de vigilancia en salud pública.

SRS: Software Requirements Specification. Documento de especificación de requisitos de software.

Teselas: Fragmentos gráficos que conforman la representación visual de un mapa digital.

Usuario anónimo: Usuario que puede realizar ciertas acciones sin cuenta registrada, siempre dentro de las restricciones de conectividad y validación definidas para el sistema.

1.4. Referencias

- ISO/IEC/IEEE 29148:2018. *Systems and software engineering – Life cycle processes – Requirements engineering.*

2. Descripción General

2.1. Descripción del contexto

En Cali existe una presencia significativa de animales domésticos en los hogares, especialmente perros y gatos. Esta realidad convive con un entorno urbano en el que las mascotas pueden exponerse a enfermedades zoonóticas, focos ambientales de riesgo, especies invasoras, acumulación de residuos, aguas estancadas y situaciones sociales como pérdida, abandono o robo.

La información disponible sobre estos eventos suele encontrarse distribuida entre boletines institucionales, reportes oficiales, redes sociales y experiencias individuales. Esta dispersión dificulta que los tutores consulten riesgos cercanos a su ubicación o tomen decisiones informadas al pasear, movilizar o proteger a sus mascotas.

A4MPets surge como una solución tecnológica que integra visualización geoespacial, participación ciudadana y alertas contextuales para transformar reportes individuales en información útil para la comunidad. El sistema no sustituye los canales institucionales, pero puede complementar la vigilancia y comunicación de riesgos mediante una herramienta accesible y centrada en el usuario.

2.2. Perspectiva del Producto

A4MPets se concibe como una herramienta de consulta y reporte para tutores de animales domésticos. Su pantalla principal estará centrada en un mapa interactivo de la ciudad de Cali, desde el cual el usuario podrá identificar amenazas cercanas, revisar información detallada, registrar reportes, iniciar recorridos y acceder a opciones relacionadas con rutas, mascotas y configuración personal.

Desde la perspectiva del usuario, el producto funcionará como una aplicación de apoyo para tomar decisiones durante paseos o desplazamientos con mascotas. El sistema permitirá consultar información compartida por la comunidad, aportar nuevos reportes y recibir avisos contextuales cuando exista una amenaza cerca de la ubicación o de la ruta seleccionada. Para estas acciones principales será necesario contar con conexión a internet o datos.

2.3. Características del Usuario

El usuario principal del sistema es el tutor o cuidador de animales domésticos que requiere consultar riesgos cercanos, reportar eventos y planificar recorridos con mayor información

contextual. No se asume que este usuario tenga conocimientos técnicos, epidemiológicos o cartográficos avanzados, por lo que la interfaz debe ser clara, directa y orientada a acciones frecuentes.

También se consideran usuarios comunitarios que, aun sin una cuenta registrada, desean contribuir mediante reportes anónimos o consultar información sobre amenazas urbanas. Adicionalmente, el sistema puede ser de interés para actores institucionales, académicos o comunitarios que busquen comprender patrones de riesgo y participación ciudadana.

2.4. Suposiciones y Dependencias

- El usuario dispone de un dispositivo con navegador compatible o entorno móvil capaz de ejecutar la aplicación.
- Las funcionalidades principales requieren conexión a internet o datos.
- La visualización del mapa depende de servicios externos de mapas.
- El cálculo de rutas depende de la disponibilidad del servicio de rutas configurado para el prototipo.
- La precisión de ubicación depende del dispositivo, permisos concedidos y condiciones de geolocalización.
- La calidad de los reportes ciudadanos depende de la claridad de los formularios, evidencia disponible y validación comunitaria.
- La información institucional disponible puede ser limitada por restricciones éticas, administrativas o de acceso a datos.
- El almacenamiento local se limita a información propia del usuario, como rutas, mascotas y preferencias básicas en el dispositivo.

2.5. Modelo C4

El modelo C4 se utiliza para representar el sistema en distintos niveles de detalle. En este SRS se incluyen los niveles de contexto, contenedores y componentes, suficientes para comprender quién interactúa con A4MPets, qué partes principales lo conforman y cómo se organizan sus funciones internas.

2.5.1. Diagrama de contexto

El diagrama de contexto presenta la relación entre A4MPets, los usuarios ciudadanos y los apoyos externos necesarios para consultar mapas, calcular rutas y conservar información compartida.

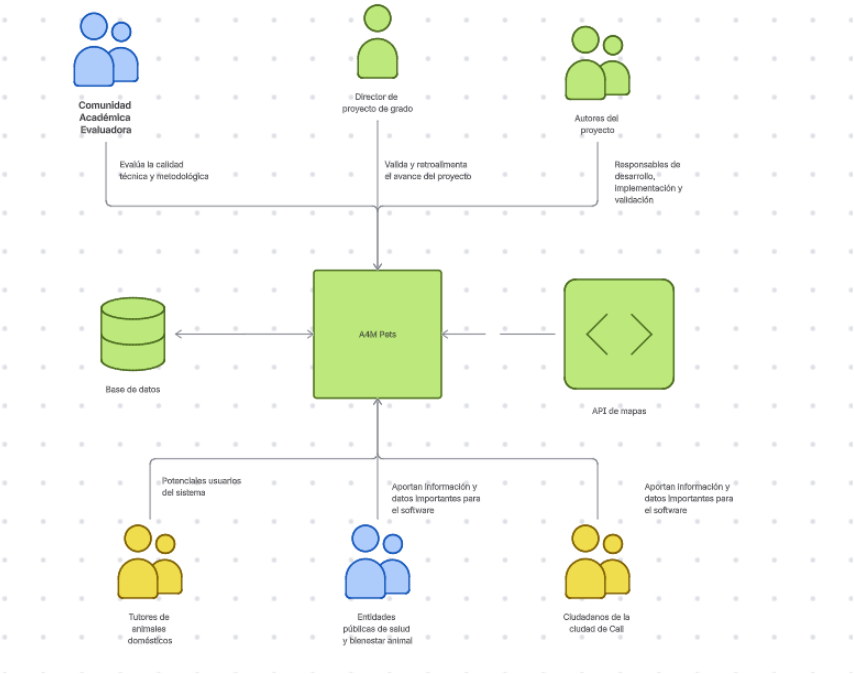


Figura 1: Diagrama de contexto del sistema A4MPets.

2.5.2. Diagrama de contenedores

El diagrama de contenedores muestra las partes principales que permiten que el usuario interactúe con la plataforma, consulte información, registre reportes y reciba respuestas del sistema. Esta separación ayuda a entender la organización general de la solución.

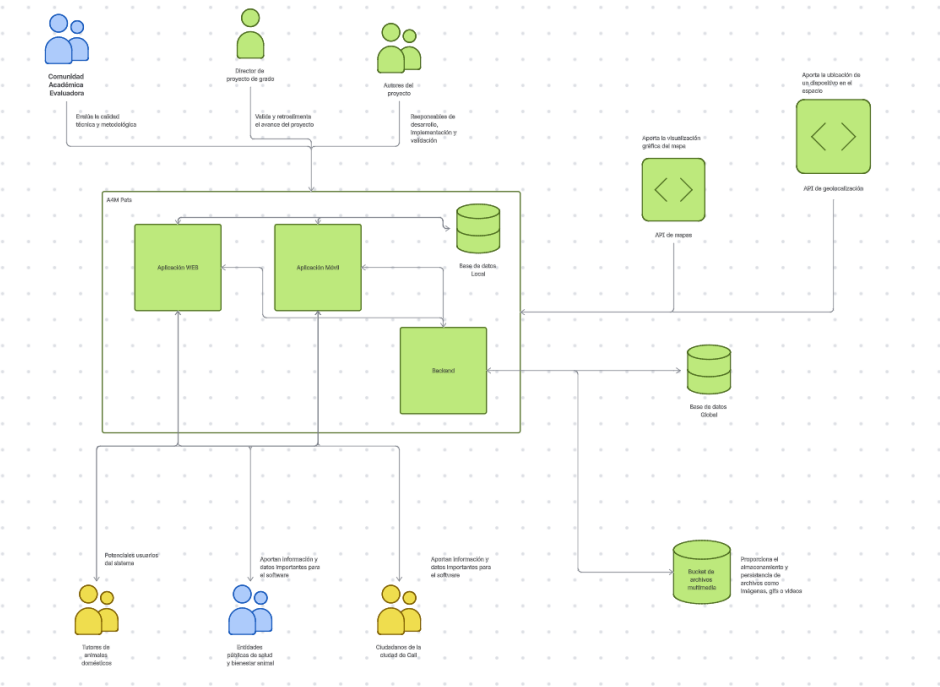


Figura 2: Diagrama de contenedores del sistema A4MPets.

2.5.3. Diagrama de componentes

El diagrama de componentes detalla las funciones internas previstas para el sistema, incluyendo mapa, reportes, rutas, mascotas, autenticación, comentarios y comunicación con servicios de soporte.

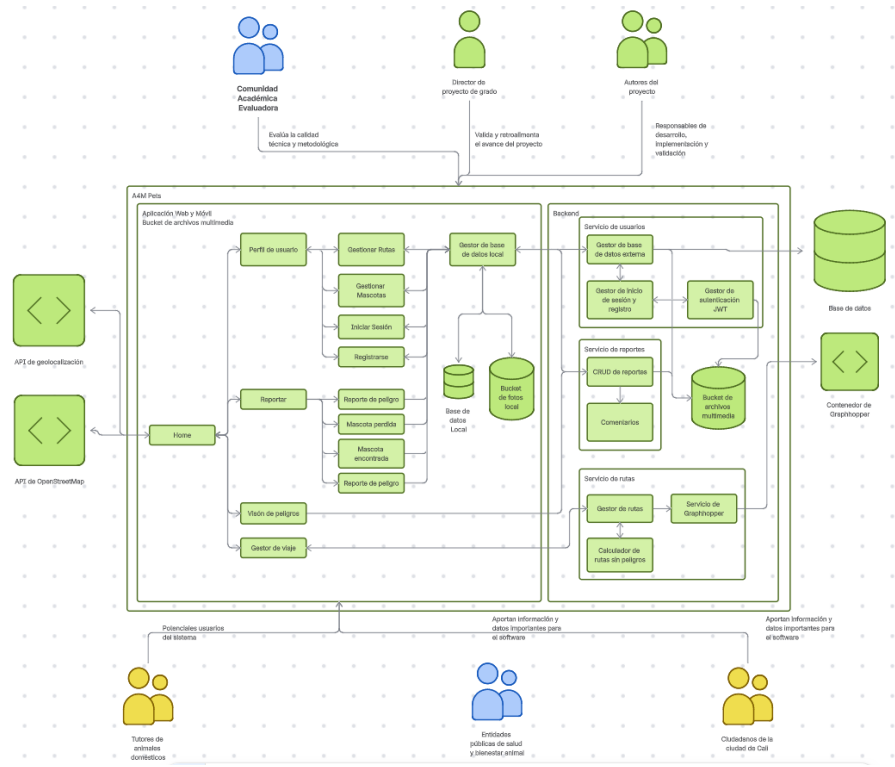


Figura 3: Diagrama de componentes del sistema A4MPets.

2.6. Riesgos

Tabla 1: Riesgos principales del sistema

Riesgo	**Descripción**	**Mitigación**
Dependencia de conectividad	El sistema requiere red para consultar mapas, registrar reportes, calcular rutas y actualizar información compartida.	Mostrar mensajes claros ante fallos de conexión y conservar datos locales permitidos.
Calidad variable de reportes	La información ciudadana puede contener errores, duplicados o baja precisión.	Formularios estructurados, comentarios, evidencia opcional y validación comunitaria.
Limitaciones de datos oficiales	El acceso a información institucional puede tardar o no estar disponible.	Usar fuentes públicas, reportes ciudadanos y dejar abierta la integración institucional futura.
Precisión geográfica limitada	La ubicación puede variar por dispositivo, permisos o condiciones del entorno.	Validar límites geográficos y permitir selección manual de ubicación en formularios.
Escalabilidad futura	El aumento de reportes y comentarios puede afectar rendimiento.	Mantener una organización modular que permita crecer sin afectar las funciones principales.
Seguridad y privacidad	El sistema maneja ubicación, imágenes y datos de contacto en algunos reportes.	Reducir datos obligatorios, proteger el acceso a la información y limitar información personal solicitada.

2.7. Reglas del negocio

1. El sistema se limita geográficamente a la zona urbana del Distrito de Santiago de Cali.
2. Los reportes deben estar asociados a una ubicación geográfica válida dentro de los límites definidos.
3. Las categorías de amenaza iniciales incluyen amenazas zoosanitarias, ambientales, zoosociales y reportes genéricos.
4. Los usuarios pueden realizar reportes en modalidad anónima cuando no han iniciado sesión, siempre que exista conectividad.
5. Los reportes de peligro deben incluir información mínima para permitir su interpretación: ubicación, categoría, descripción, nivel de riesgo y radio de afectación cuando aplique.
6. Las mascotas y rutas pueden almacenarse localmente en el dispositivo para usuarios no registrados.
7. La aplicación debe solicitar permisos de ubicación únicamente cuando sean necesarios para mapa, reporte o viaje.
8. La información mostrada en el mapa debe corresponder a datos obtenidos desde la plataforma o a información local permitida para el usuario.

3. Definición y Especificación de Requisitos

3.1. Atributos de calidad

Los atributos de calidad priorizados para A4MPets responden al contexto de uso del sistema: una aplicación georreferenciada, colaborativa y orientada a usuarios no necesariamente técnicos.

Tabla 2: Atributos de calidad priorizados

Atributo	**Descripción**
Usabilidad	La interfaz debe permitir consultar, reportar y navegar con baja carga cognitiva.
Rendimiento	Las acciones principales deben responder en tiempos adecuados bajo condiciones normales de red.
Disponibilidad funcional	Las funciones principales dependen de conectividad, pero el sistema debe manejar fallos de red de forma comprensible.
Mantenibilidad	La organización interna del sistema debe separar mapa, reportes, rutas, mascotas y configuración.
Seguridad y privacidad	El sistema debe limitar la recolección de datos personales y proteger la información gestionada.
Consistencia geográfica	Los eventos, rutas y ubicaciones deben mantenerse dentro del alcance territorial definido.
Adaptabilidad visual	La interfaz debe funcionar en distintos tamaños de pantalla y soportar modo claro y oscuro.

3.2. Escenarios de calidad

Tabla 3: Escenarios de calidad

Atributo	Estímulo	Respuesta esperada	Criterio
Usabilidad	Usuario crea un reporte de peligro.	El sistema guía mediante formulario claro y validaciones visibles.	Reporte completado en máximo 5 pasos desde la pantalla principal.
Rendimiento	Usuario abre la aplicación con red estable.	El mapa y controles principales cargan sin bloqueo prolongado.	Carga inicial no superior a 5 segundos bajo red normal.
Conectividad	Falla la red durante consulta de mapa.	La aplicación muestra mensaje comprensible y evita estados inconsistentes.	El usuario entiende que debe recuperar conexión.
Privacidad	Usuario registra reporte sin cuenta.	El sistema permite modalidad anónima y solicita solo datos necesarios.	No se exige autenticación para reportar cuando la función lo permita.
Consistencia geográfica	Usuario intenta reportar fuera de Cali.	El sistema restringe o advierte que la ubicación está fuera del área soportada.	El reporte no queda asociado a una ubicación inválida.
Mantenibilidad	Se agrega una nueva categoría de reporte.	La organización del sistema permite extender formularios y reglas asociadas sin reescribir módulos completos.	El cambio se integra manteniendo separación de responsabilidades.

3.3. Modelo Entidad-Relación

El modelo entidad-relación organiza la información del sistema alrededor de usuarios, reportes, comentarios, mascotas, rutas, puntos de ruta y catálogos de clasificación. Este modelo permite soportar información georreferenciada y datos necesarios para visualización, consulta y validación colaborativa.

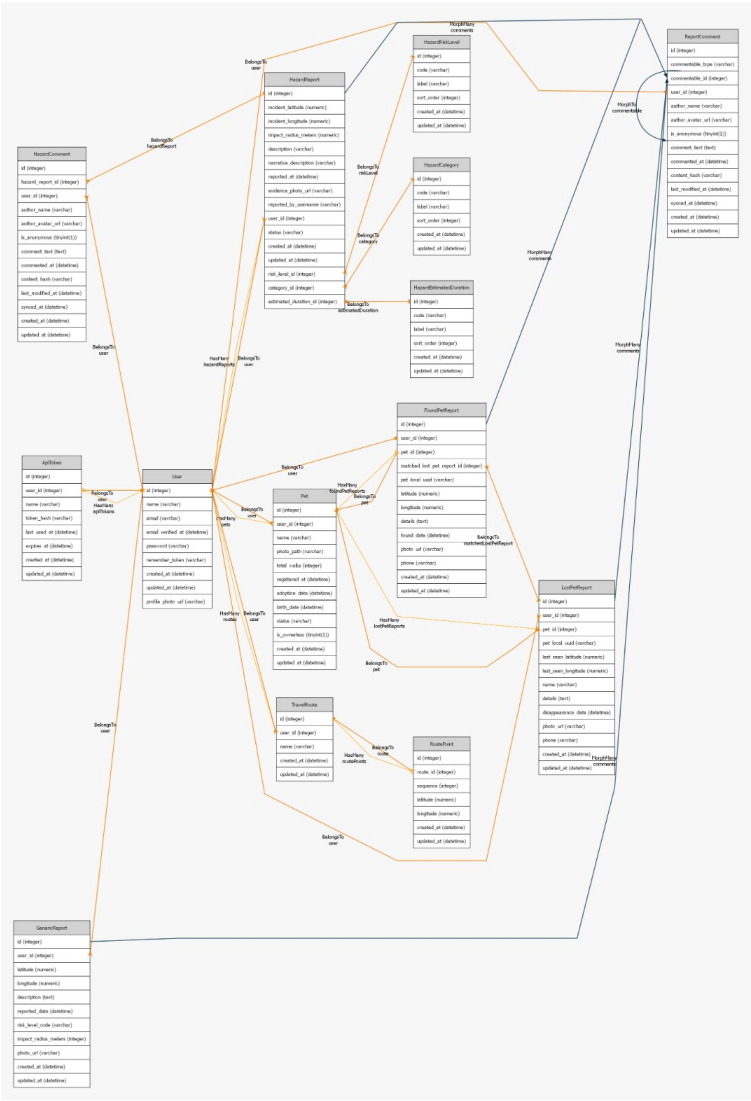


Figura 4: Modelo entidad-relación del sistema A4MPets.

3.4. Diagrama de casos de uso

El diagrama de casos de uso representa las principales interacciones del usuario con la plataforma: visualizar amenazas, registrar reportes, gestionar rutas, gestionar mascotas, configurar

viajes y consultar información asociada a eventos.

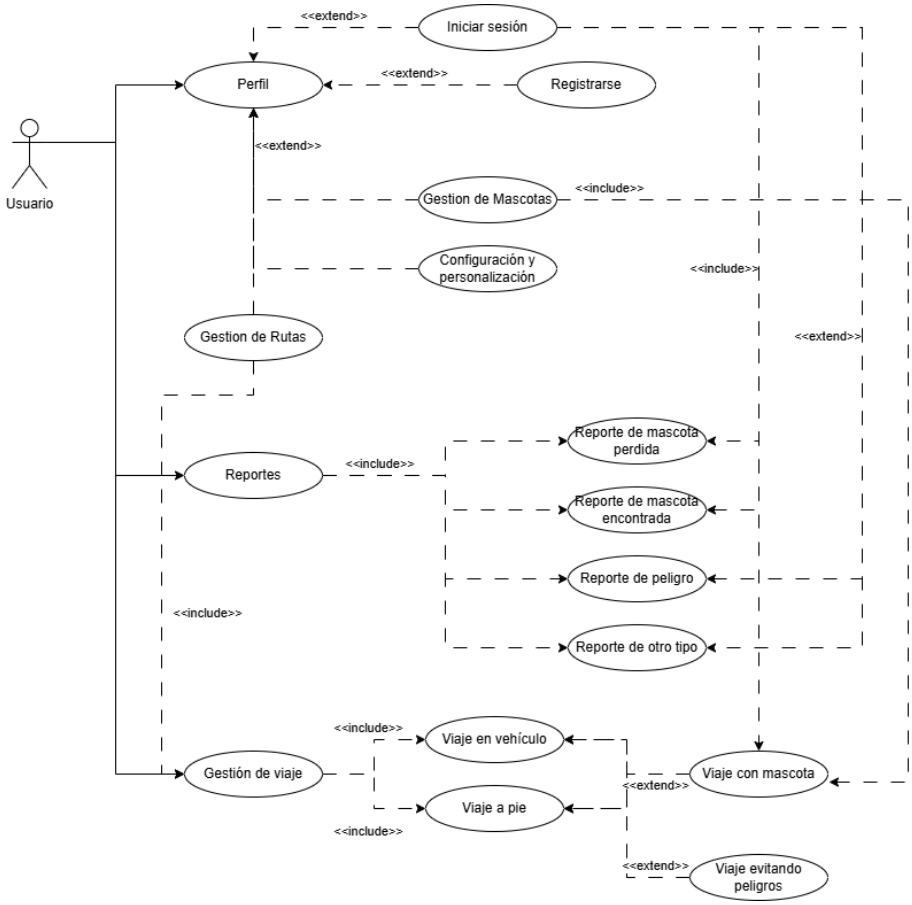


Figura 5: Diagrama de casos de uso de A4MPets.

3.5. Requisitos funcionales

Los requisitos funcionales definen las capacidades que el sistema debe ofrecer al usuario. Se organizan por grupos funcionales para facilitar su trazabilidad.

Visualización y navegación del mapa

1. **RF1:** La aplicación debe mostrar en la interfaz principal un mapa base de la zona urbana de la ciudad de Cali obtenido a partir de OpenStreetMap.

2. **RF2:** El mapa mostrado en la interfaz principal debe limitar su visualización a los límites geográficos de la zona urbana del Distrito de Santiago de Cali.
3. **RF3:** Al abrir la página principal, si el usuario no ha concedido permisos de ubicación o estos no están disponibles, el sistema debe mostrar el mapa con un nivel de acercamiento que permita visualizar la zona urbana definida.
4. **RF4:** Al abrir la página principal, si el usuario ha concedido permisos de ubicación y esta se encuentra disponible, el sistema debe centrar el mapa en la ubicación actual del usuario.
5. **RF5:** El mapa debe disponer de un control para aumentar el nivel de acercamiento dentro de los límites geográficos definidos.
6. **RF6:** El mapa debe disponer de un control para reducir el nivel de acercamiento hasta visualizar la zona urbana soportada.
7. **RF7:** El sistema debe impedir que el usuario desplace el mapa o aumente el acercamiento por fuera de los límites geográficos definidos.
8. **RF8:** El sistema debe mostrar la ubicación actual del usuario únicamente cuando los permisos de ubicación estén habilitados y la ubicación corresponda al área soportada.

Visualización y gestión de amenazas

9. **RF9:** El sistema debe mostrar en el mapa los peligros reportados mediante elementos gráficos georreferenciados ubicados en sus coordenadas.
10. **RF10:** El sistema debe permitir seleccionar un peligro mostrado en el mapa para consultar su información detallada.
11. **RF11:** La información detallada de un peligro debe incluir categoría, descripción, fecha de creación, duración aproximada, nivel de riesgo, radio de afectación y comentarios asociados cuando existan.
12. **RF12:** El radio de afectación de un peligro debe definirse mediante un conjunto de valores permitidos configurados en el sistema.

Configuración e interfaz de usuario

13. **RF13:** El sistema debe proporcionar un botón en la interfaz principal que permita alternar entre modo claro y modo oscuro.
14. **RF14:** El sistema debe disponer de un botón accesible desde la página principal para ingresar a la página de configuración del usuario.
15. **RF15:** El sistema debe disponer de una página de configuración que permita acceder a las funcionalidades de gestión de rutas y gestión de mascotas.

Gestión de rutas

16. **RF16:** El sistema debe permitir al usuario crear, visualizar, actualizar y eliminar rutas dentro de la página de gestión de rutas.

17. **RF17:** Cada ruta registrada debe estar conformada por una secuencia ordenada de ubicaciones geográficas dentro de la zona urbana de Cali.

Gestión de mascotas

18. **RF18:** El sistema debe permitir al usuario crear, visualizar, actualizar y eliminar mascotas dentro de la página de gestión de mascotas.

19. **RF19:** Cada mascota registrada debe incluir obligatoriamente un nombre y opcionalmente una fotografía asociada.

Reportes colaborativos y monitoreo ciudadano

20. **RF20:** El sistema debe disponer de un botón denominado "Reportar" en la interfaz principal, el cual debe desplegar un panel para seleccionar el tipo de reporte.

21. **RF21:** El sistema debe permitir seleccionar entre reportes de peligro, mascota perdida, mascota encontrada y otro reporte.

22. **RF22:** El sistema debe permitir registrar reportes mediante formularios georreferenciados asociados a una ubicación específica dentro del mapa de Cali.

Reportes de peligro

23. **RF23:** El sistema debe permitir registrar un reporte de tipo peligro mediante un formulario que incluya ubicación geográfica, categoría, descripción textual, nivel de riesgo, radio de afectación y opcionalmente duración aproximada e imagen asociada.

24. **RF24:** Las categorías disponibles para reportes de peligro deben pertenecer a un conjunto predefinido y configurable por el sistema.

25. **RF25:** El sistema debe validar que todos los campos obligatorios del reporte de peligro estén completos antes de permitir su envío.

26. **RF26:** El sistema debe representar gráficamente en el mapa los reportes de peligro mediante marcadores georreferenciados y áreas de influencia asociadas a su radio de afectación.

Reportes de mascota perdida

27. **RF27:** El sistema debe permitir registrar reportes de mascota perdida mediante un formulario que incluya último lugar de avistamiento, nombre de la mascota, descripción, fecha de desaparición y número de contacto.

28. **RF28:** El sistema debe visualizar en el mapa los reportes de mascota perdida mediante elementos gráficos asociados a su última ubicación registrada.

Reportes de mascota encontrada

29. **RF29:** El sistema debe permitir registrar reportes de mascota encontrada mediante un formulario que incluya lugar de encuentro, descripción de la situación, fecha del encuentro y opcionalmente número de contacto e imagen asociada.

30. **RF30:** El sistema debe visualizar en el mapa los reportes de mascota encontrada mediante elementos gráficos asociados a la ubicación registrada.

Otros reportes urbanos

31. **RF31:** El sistema debe permitir registrar reportes de tipo "otro" mediante un formulario que incluya ubicación geográfica, descripción del evento, fecha del reporte, nivel de riesgo y opcionalmente imagen asociada.

Comentarios y validación colaborativa

32. **RF32:** El sistema debe permitir a los usuarios agregar comentarios a reportes existentes con el fin de complementar, actualizar o validar la información reportada.

33. **RF33:** Cada comentario asociado a un reporte debe incluir identificador del usuario cuando exista, contenido textual y fecha y hora de publicación generada por el sistema.

Persistencia local y conectividad

34. **RF34:** El sistema debe almacenar localmente en el dispositivo la información propia del usuario que no requiere publicación comunitaria inmediata, incluyendo rutas, mascotas y configuraciones básicas.

35. **RF35:** El sistema debe permitir registrar reportes en modalidad anónima cuando el usuario no haya iniciado sesión, siempre que exista conectividad para enviar la información a la plataforma.

36. **RF36:** El sistema debe informar al usuario cuando una funcionalidad no pueda completarse por ausencia de conectividad, especialmente en mapa, reportes, viajes, comentarios y consulta de amenazas.

37. **RF37:** Las imágenes asociadas a reportes y mascotas deben almacenarse o referenciarse de acuerdo con el mecanismo disponible para cada módulo, manteniendo vínculo con el registro correspondiente.

Viajes y monitoreo en tiempo real

38. **RF38:** El sistema debe disponer de un botón de iniciar viaje que permita acceder a la configuración de un recorrido.
39. **RF39:** El sistema debe permitir configurar un viaje mediante la selección de un punto de inicio y un punto de destino dentro del mapa.
40. **RF40:** Durante la configuración de un viaje, el sistema debe permitir seleccionar una ruta previamente guardada o generar una nueva ruta entre el punto de inicio y el punto de destino.
41. **RF41:** Una vez configurado el viaje, el sistema debe mostrar en el mapa la ruta calculada entre el punto de inicio y el destino.
42. **RF42:** Durante un viaje activo, el sistema debe mostrar la ubicación actual del usuario en tiempo real dentro del mapa.
43. **RF43:** El sistema debe visualizar sobre la ruta los peligros reportados que se encuentren dentro del trayecto calculado.
44. **RF44:** Durante un viaje activo, el sistema debe notificar al usuario cuando se encuentre dentro del radio de afectación de un peligro reportado.
45. **RF45:** Al finalizar un viaje, el sistema debe permitir almacenar la ruta recorrida dentro del historial del usuario y asociarla opcionalmente a una mascota registrada.

3.6. Requisitos no funcionales

1. **RNF1:** El elemento visual del mapa debe verse correctamente en resoluciones de pantalla entre 600x400 y 3840x2160.
2. **RNF2:** La aplicación debe mantener consistencia visual y funcional en dispositivos móviles Android con acceso a geolocalización y conexión a internet.
3. **RNF3:** La aplicación debe permitir la lectura adecuada de textos, botones e íconos tanto en modo claro como en modo oscuro.
4. **RNF4:** Los botones, formularios y elementos interactivos deben mantener distribución y tamaño adecuados para uso táctil en dispositivos móviles.
5. **RNF5:** Las fotografías opcionales asociadas a mascotas o peligros deben visualizarse sin distorsión apreciable dentro de la aplicación.

6. **RNF6:** La aplicación debe responder a las interacciones principales del usuario en un tiempo máximo de 3 segundos bajo condiciones normales de conexión.
7. **RNF7:** La carga inicial del mapa de Cali no debe tardar más de 5 segundos bajo condiciones normales de red.
8. **RNF8:** La actualización de ubicación del usuario durante un viaje activo debe reflejarse con latencia máxima de 5 segundos bajo condiciones normales de red y geolocalización.
9. **RNF9:** La interfaz debe permitir que un usuario pueda reportar un peligro en máximo 5 pasos desde la página principal.
10. **RNF10:** El sistema debe manejar errores de conexión mostrando mensajes claros cuando no sea posible cargar el mapa, la ubicación, los reportes o la información de peligros.
11. **RNF11:** El sistema debe almacenar de forma persistente en el dispositivo las rutas, mascotas y configuraciones locales permitidas, de modo que no se pierdan al cerrar la aplicación.
12. **RNF12:** La aplicación debe indicar de forma comprensible qué funciones requieren conectividad cuando el usuario intente utilizarlas sin acceso a red.
13. **RNF13:** El sistema debe proteger la información almacenada localmente mediante mecanismos que eviten accesos no autorizados desde la aplicación.
14. **RNF14:** El sistema no debe solicitar ni almacenar datos personales no requeridos para el funcionamiento de la aplicación.
15. **RNF15:** La aplicación debe solicitar permiso de ubicación únicamente cuando sea necesario para funcionalidades de mapa, viaje o reporte.
16. **RNF16:** El sistema debe garantizar que los peligros mostrados en el mapa correspondan únicamente a ubicaciones comprendidas dentro de los límites geográficos de Cali.
17. **RNF17:** La información mostrada al usuario debe corresponder a la última actualización obtenida desde la plataforma o a la información local permitida para el módulo consultado.
18. **RNF18:** El sistema debe registrar fechas y horas de creación de reportes, comentarios y rutas utilizando un formato consistente.
19. **RNF19:** El sistema debe permitir la mantenibilidad del software mediante una arquitectura modular que separe mapa, rutas, mascotas, reportes y configuración.
20. **RNF20:** El sistema debe permitir crecimiento futuro en número de reportes, comentarios, rutas y categorías de peligros.

3.7. Requisitos de interfaz

Interfaz cartográfica

1. **RI1:** La interfaz principal debe mostrar un mapa interactivo de Cali ocupando la mayor parte de la pantalla.
2. **RI2:** La página principal debe mostrar claramente la ubicación actual del usuario mediante un marcador visual diferenciado cuando el permiso esté activo.
3. **RI3:** Los peligros reportados deben representarse en el mapa mediante íconos visuales que permitan identificar su ubicación aproximada.
4. **RI4:** Al seleccionar un ícono de peligro, la interfaz debe desplegar un panel con la información detallada del reporte.
5. **RI5:** Durante un viaje activo, la interfaz debe resaltar la ruta seleccionada mediante una línea visual distinguible.
6. **RI6:** La interfaz debe mostrar notificaciones visuales cuando el usuario se acerque a un peligro dentro del radio de influencia definido.

Interacción y navegación

7. **RI7:** La interfaz debe proporcionar botones visibles para reportar, iniciar viaje y acceder a configuración.
8. **RI8:** La página de gestión de rutas debe permitir visualizar rutas existentes y ofrecer opciones para crear, editar o eliminar rutas.
9. **RI9:** La página de gestión de mascotas debe mostrar una lista de mascotas registradas y permitir visualizar o editar la información de cada una.
10. **RI10:** La interfaz de configuración de viaje debe permitir seleccionar punto de inicio y destino mediante interacción directa con el mapa o campos de búsqueda.
11. **RI11:** La interfaz debe permitir desplazarse entre páginas mediante un sistema de navegación consistente.

Formularios y retroalimentación visual

12. **RI12:** La interfaz de reporte de peligro debe presentarse como formulario claro y estructurado que permita seleccionar tipo de peligro, añadir descripción y adjuntar opcionalmente una fotografía.
13. **RI13:** Los formularios deben incluir mensajes claros de validación cuando un campo obligatorio no haya sido completado correctamente.

Consistencia visual y adaptabilidad

14. **RI14:** La interfaz debe permitir alternar entre modo claro y modo oscuro mediante un botón accesible desde la página principal.

15. **RI15:** Los íconos, botones y elementos interactivos deben ser claramente identificables y mantener consistencia visual en toda la aplicación.

3.8. Mockups

Los mockups e interfaces del prototipo muestran la evolución visual de la aplicación y su relación con los requisitos de mapa, reportes, rutas, mascotas y viajes.

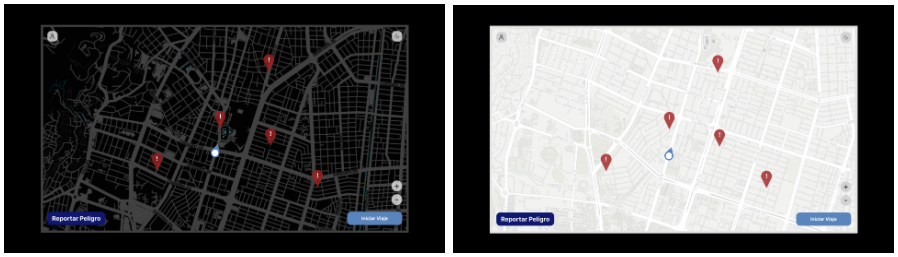


Figura 6: Mockups de navegación principal y visualización cartográfica.

Figura 7: Mockups de vista detallada de evento y consulta de información.

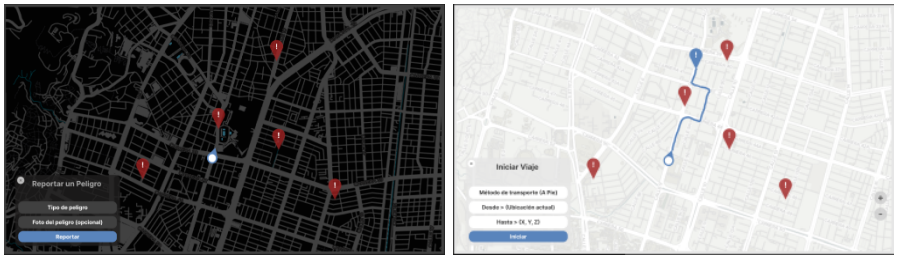


Figura 8: Mockups asociados a gestión de rutas.

4. Otros requisitos

4.1. Requisitos por implementar en versiones futuras

- Integración directa con entidades institucionales para incorporar datos oficiales validados sobre amenazas sanitarias y ambientales.
- Panel de administración para moderar reportes, revisar categorías emergentes y gestionar contenido inapropiado.
- Métricas territoriales para identificar concentración de amenazas por sector, comuna o periodo de tiempo.
- Mejoras avanzadas de accesibilidad alineadas con lineamientos WCAG.
- Despliegue productivo con infraestructura escalable, monitoreo operativo y políticas formales de respaldo.
- Validación con usuarios reales en recorridos urbanos y pruebas sobre múltiples dispositivos físicos.

4.2. Requisitos eliminados

Durante la definición de esta especificación se descartaron requisitos inicialmente considerados, debido a que no corresponden al alcance funcional que se desea documentar para el sistema:

- Uso completo de la aplicación sin acceso a red.
- Envío diferido y automático de datos locales hacia el servidor al recuperar conectividad.
- Uso del almacenamiento local para reportes comunitarios como mecanismo principal de publicación posterior.

- Asociación automática de datos locales de rutas y mascotas con una cuenta remota después de iniciar sesión.

Estos elementos fueron reemplazados por una definición más precisa del alcance: A4MPets requiere conectividad para mapa, reportes, viajes, comentarios y consulta de información compartida; conserva almacenamiento local limitado para rutas, mascotas y configuraciones propias del usuario en el dispositivo.

5. Proceso de construcción del documento de requisitos

5.1. Estrategias de obtención de requisitos

Los requisitos fueron obtenidos mediante revisión documental, análisis del contexto urbano de Cali, consulta de fuentes institucionales, acercamientos con actores locales y revisión de soluciones existentes. Estos insumos permitieron definir una especificación orientada al sistema propuesto y a las necesidades identificadas para sus usuarios.

La revisión documental permitió identificar amenazas zoosanitarias y ambientales como leptospirosis, riesgos asociados a rabia, presencia de caracol gigante africano, aguas estancadas y acumulación de residuos. El análisis del contexto y las interacciones preliminares con actores locales permitieron ampliar el enfoque hacia amenazas zoosociales como desaparición, robo y abandono de mascotas.

5.2. Estrategias de priorización de requisitos

La priorización se realizó considerando impacto sobre el bienestar animal, viabilidad técnica, posibilidad de representación geográfica, utilidad para usuarios no expertos y relación con los objetivos del prototipo. Se priorizaron primero las funcionalidades que habilitan el flujo principal del sistema: mapa, reportes, visualización de amenazas, rutas, viajes y gestión de mascotas.

También se priorizaron requisitos de conectividad, privacidad, rendimiento y usabilidad debido a que afectan directamente la experiencia del usuario final y la confiabilidad del sistema.

5.3. Estrategias y resultados de la validación de requisitos

La validación de requisitos se plantea como una actividad previa y paralela al desarrollo, orientada a revisar que cada requisito sea necesario, verificable, consistente y comprensible. Para ello se propone una estrategia de revisión por criterios de calidad, escenarios de uso, trazabilidad con objetivos del sistema y validación con actores relacionados con el dominio.

Como resultado esperado, los requisitos deben mantenerse alineados con el alcance definido para A4MPets. En particular, los requisitos relacionados con persistencia y conectividad deben evitar atribuir al sistema capacidades fuera de alcance.

6. Requisitos de las Pruebas

6.1. Pruebas Unitarias

Las pruebas unitarias deben validar partes aisladas del sistema. En la capa de servicios se deben verificar modelos, reglas de validación y lógica de negocio. En la interfaz se deben validar formularios, elementos visuales, servicios locales y manejo de estados.

Criterios mínimos:

- Validar creación, consulta, actualización y eliminación de entidades principales cuando aplique.
- Validar reglas de obligatoriedad en formularios.
- Validar cálculo y representación básica de estados de interfaz.
- Validar que los módulos locales de rutas y mascotas conserven la información permitida.

6.2. Pruebas de Integración

Las pruebas de integración deben verificar la comunicación entre la interfaz, los servicios del sistema, el almacenamiento de datos y los apoyos externos necesarios. Deben cubrir reportes, comentarios, autenticación, rutas, mascotas y consulta de amenazas.

Criterios mínimos:

- Validar respuestas exitosas y errores controlados de los servicios.
- Verificar que la información enviada por la interfaz sea interpretada correctamente por el sistema.
- Probar flujos con usuario autenticado y usuario anónimo.
- Validar integración del cálculo de rutas con el servicio de rutas.

6.3. Pruebas de Sistema

Las pruebas de sistema deben validar el comportamiento conjunto del prototipo en un ambiente local integrado. Deben cubrir mapa, reportes, comentarios, rutas, mascotas, viajes, permisos y conectividad.

Criterios mínimos:

- Verificar carga inicial del mapa.
- Verificar visualización de amenazas dentro de Cali.

- Verificar creación y consulta de reportes.
- Verificar configuración y ejecución de viajes.
- Verificar mensajes ante ausencia de conectividad.

6.4. Pruebas E2E

Las pruebas end-to-end deben simular flujos completos desde la perspectiva del usuario final mediante navegador. Se recomienda utilizar una herramienta de automatización de navegador para validar navegación e interacción con formularios.

Flujos mínimos:

- Ingresar a la aplicación y visualizar el mapa.
- Crear un reporte de peligro en modalidad anónima.
- Consultar detalle de una amenaza y agregar comentario.
- Registrar una mascota y una ruta.
- Configurar un viaje y visualizar el trayecto.

6.5. Pruebas Manuales

Las pruebas manuales deben complementar la automatización en aspectos visuales, exploratorios y de experiencia de usuario. Deben enfocarse en claridad de formularios, navegación, mensajes de error, comportamiento responsive, permisos de ubicación y consistencia entre modo claro y oscuro.

6.6. Pruebas de Rendimiento

Las pruebas de rendimiento deben medir tiempos de respuesta bajo condiciones normales de red y escenarios controlados de múltiples solicitudes. Se deben verificar tiempos de carga del mapa, respuesta de los servicios, actualización de ubicación y cálculo de rutas.

Criterios mínimos:

- Carga inicial del mapa en máximo 5 segundos bajo condiciones normales de red.
- Respuesta a interacciones principales en máximo 3 segundos.
- Actualización de ubicación durante viaje en máximo 5 segundos.
- Endpoints principales con tiempos estables en escenarios controlados.

6.7. Pruebas de regresión

Las pruebas de regresión deben ejecutarse después de correcciones o cambios relevantes para asegurar que los flujos previamente validados no se vean afectados. Deben incluir mapa, reportes, comentarios, rutas, mascotas, viajes, autenticación cuando aplique y manejo de errores de conectividad.

Apéndice D

Plan de Pruebas A4MPets

El presente anexo contiene el Plan de Pruebas desarrollado para la plataforma A4MPets, elaborado con el propósito de definir de manera estructurada las estrategias, criterios, herramientas y procedimientos utilizados durante el proceso de validación del sistema.

Este documento establece las actividades de verificación aplicadas sobre los diferentes componentes de la arquitectura, incluyendo el frontend desarrollado en Flutter Web, el backend implementado en Laravel, los servicios de enrutamiento soportados mediante GraphHopper y los mecanismos de integración asociados al modelo de persistencia y sincronización de datos. Asimismo, se describen los distintos niveles de pruebas ejecutados, tales como pruebas unitarias, pruebas de integración, pruebas funcionales, pruebas de sistema y pruebas end-to-end, junto con sus respectivos criterios de entrada, salida y evidencias esperadas.

El plan fue diseñado con un enfoque incremental, permitiendo validar progresivamente tanto el comportamiento individual de los módulos como la interacción completa entre los componentes de la plataforma. De igual manera, el documento define los ambientes de prueba utilizados, los recursos tecnológicos requeridos y las consideraciones relacionadas con estabilidad, manejo de errores, conectividad y comportamiento del sistema en diferentes escenarios de uso.

Adicionalmente, este anexo sirvió como base para la organización de la etapa de evaluación presentada en el documento principal, facilitando la trazabilidad entre los casos de prueba ejecutados, los resultados obtenidos y las evidencias asociadas a cada fase del proceso de validación.

1. Plan de Pruebas A4MPets v2

Campo	Valor
Proyecto	A4MPets
Documento	Plan de pruebas detallado
Versión	2.0 - guía operacional sin ejecución
Fecha de inicio	2026-05-05
Fecha de cierre planeada	2026-05-14
Responsable	Equipo QA
Repositorio base	`C:\projects\Web\A4MPets`
Stack bajo prueba	Frontend Flutter, Backend Laravel 12, GraphHopper, Docker Compose y Cloudflare Tunnel

Este documento define cómo se van a planear, ejecutar, evidenciar y cerrar las pruebas de A4MPets. No representa una certificación de calidad por sí mismo. Cada actividad queda pendiente hasta que el equipo QA la ejecute, registre resultado real, adjunte evidencia y documente incidencias cuando corresponda.

2. Objetivo del testing

El objetivo principal del testing es comprobar, con evidencia trazable, que A4MPets permite a una persona usuaria registrarse, iniciar sesión, administrar mascotas, reportar mascotas perdidas, reportar mascotas encontradas, reportar peligros, comentar reportes, cargar imágenes, consultar información en mapa, crear rutas y planear recorridos usando GraphHopper sin romper la consistencia entre frontend, backend y datos persistidos.

Los objetivos específicos son verificables y deben revisarse contra evidencia real:

Objetivo	Explicación detallada	Evidencia mínima esperada
Confirmar funcionalidad central	Validar los flujos que sostienen el producto: autenticación, perfil, mascotas, reportes, comentarios, medios, rutas y mapa. Cada flujo debe probarse con datos válidos, datos inválidos y condiciónes de permisos.	Capturas, respuestas JSON, salidas de pruebas automatizadas y matriz de casos actualizada.
Confirmar contratos de API	Verificar que cada endpoint de Laravel responda con código HTTP, estructura JSON, validaciónes y permisos acordes con el comportamiento esperado por el frontend.	Logs de PHPUnit, colección Postman o respuestas exportadas.
Confirmar comportamiento visual	Validar que la UI en Flutter web muestre mensajes claros, paneles consistentes, formularios utilizables, marcadores correctos y ausencia de errores visibles.	Capturas por viewport, consola sin errores severos y observaciónes manuales.
Confirmar integración local	Validar que Laravel, Flutter, GraphHopper, Docker y Cloudflare Tunnel funcionen juntos bajo un ambiente controlado.	Health checks, URL de túnel, logs de servicios y evidencia de flujo integrado.

Objetivo	Explicación detallada	Evidencia mínima esperada
Confirmar tolerancia ante condiciónes de fallo	Validar respuestas cuando falta token, hay payload inválido, GraphHopper no responde, se pierde conexión o el usuario intenta modificar recursos ajenos.	Casos fallidos controlados, mensajes de error y ausencia de caídas no manejadas.
Confirmar estabilidad básica	Ejecutar solicitudes repetidas mediante Postman contra el ambiente local expuesto por Cloudflare para observar latencia, errores HTTP y comportamiento del servidor.	Reporte de Postman Runner con iteraciones, tiempos y errores.

3. Alcance

3.1. Funcionalidades incluidas

Módulo	Qué se prueba	Detalles que no deben quedar implícitos
Autenticación	Registro, login por correo, login por nombre de usuario, perfil autenticado, actualización de perfil y logout.	Se debe verificar emisión de token, rechazo de credenciales inválidas, revocación de token y persistencia de sesión en frontend.
Mascotas	Creación, edición, listado, detalle, eliminación, mascotas propias, mascotas sin propietario y cambio de estado al reportar pérdida o encontrada.	Se debe comprobar que una persona usuaria no pueda modificar mascotas de otra persona y que los estados se reflejen en UI y API.
Reportes de peligros	Creación, listado, detalle, edición, eliminación, comentarios, categoría, nivel de peligro, radio de impacto y visualización en mapa.	Los endpoints usan nombres técnicos como `hazards`, pero la documentación QA siempre los nombra como reportes de peligros.
Reportes de mascotas perdidas	Creación desde mascota propia, bloqueo de duplicados activos, comentarios, detalle, edición, eliminación y retorno de estado al encontrar la mascota.	Se debe verificar ubicación, fecha, teléfono, foto y relación con mascota.
Reportes de mascotas encontradas	Creación sin mascota asociada, creación vinculada a un reporte de mascota pérdida, comentarios, detalle, edición y eliminación.	Se debe comprobar la tarjeta navegable hacia el reporte de mascota pérdida cuando exista vinculación.
Reportes genéricos de tipo otro	Creación, nivel de peligro, radio de impacto, comentarios, listado, detalle y visualización en mapa.	Se debe validar que el radio influya en la visualización de rutas cuando corresponda.
Rutas y planeación	CRUD de rutas, puntos de ruta, planificación con GraphHopper, polilínea, instrucciones, filtros de peligro y recálculo.	Se debe probar una ruta con dos puntos, una ruta multipunto y condiciónes de error cuando GraphHopper no esté disponible.

Módulo	Qué se prueba	Detalles que no deben quedar implícitos
Mapa	Marcadores, capas, selección manual de ubicación, ubicación del usuario, foco de cámara, polilíneas y paneles sobre el mapa.	Se debe validar desktop y viewport móvil, con permisos de ubicación aceptados y denegados.
Medios	Carga de imágenes, buckets permitidos, recuperación de archivo, tipo de contenido y errores por archivo inválido.	Se debe verificar perfil, mascota, reporte de peligro, reporte perdido, reporte encontrado y reporte genérico.

3.2. Elementos fuera de alcance

Elemento	Motivo	Decision QA
Auditoría formal OWASP completa	Requiere métodología, herramientas y tiempo de seguridad dedicado.	Solo se cubren validaciónes básicas cómo payload HTML, permisos y tokens.
Publicación en tiendas móviles	No forma parte del ciclo local de pruebas solicitado.	Se deja para una fase de release móvil.
Validación cartográfica exhaustiva de todas las calles	El mapa depende de datos externos y no se busca certificar cartografía.	Se usan coordenadas representativas en Cali.
Pruebas con datos productivos reales	Puede comprometer privacidad y repetibilidad.	Se usan usuarios y datos QA.
Certificación WCAG formal	Requiere checklist completo y herramientas específicas.	Se hace revisión funcional mínima de teclado, foco y legibilidad.

4. Tipos de pruebas

Familia	Tipo	Propósito	Herramientas	Criterio de aprobación
Automatizadas	Caja blanca unitaria backend	Validar unidades de Laravel como servicios, formateadores, requests y reglas de dominio con conocimiento del código.	PHPUnit, `php artisan test`.	La prueba termina en verde y cubre al menos ruta exitosa y una condición negativa relevante.
Automatizadas	Caja blanca integración backend	Validar controladores, middleware, modelos, base de datos de prueba y respuestas JSON.	PHPUnit Feature.	Cada endpoint probado responde con código, estructura y persistencia esperada.

Familia	Tipo	Propósito	Herramientas	Criterio de aprobación
Automatizadas	Caja blanca frontend	Validar servicios, providers, widgets y lógica de presentación de Flutter.	`flutter analyze`, `flutter test`.	No hay errores de analyzer y las pruebas pasan.
Automatizadas	Caja negra E2E navegador	Simular comportamiento de una persona usuaria real sin depender del código interno.	Playwright sobre Flutter web.	El flujo termina con la pantalla, datos y respuestas API esperadas.
Sistema	Integración local completa	Confirmar que frontend, backend y GraphHopper funcionan al mismo tiempo.	Scripts npm, Docker, navegador, logs.	Flujos P0 funcionan sin errores de consola ni HTTP 5xx inesperados.
Manuales	Funcionales exploratorias	Encontrar problemas de usabilidad, mensajes, estados vacíos y flujos no cubiertos por automatización.	Navegador, DevTools, capturas.	Cada observación queda documentada con pasos y evidencia.
No funcionales	Carga manual controlada	Observar comportamiento del backend local expuesto por Cloudflare ante muchas solicitudes repetidas.	Postman Runner, Cloudflare Tunnel.	Se registran latencias, errores, saturación y comportamiento de recuperación.
Regresión	Reejecución de P0 y P1	Confirmar que las correcciones no rompieron flujos críticos.	PHPUnit, Flutter test, Playwright, manual.	No quedan defectos críticos o altos sin decisión formal.

5. Estrategia por fases

La estrategia se divide en fases identificables para que el equipo pueda solicitar ejecuciones concretas por ID. Cada fase tiene entradas, actividades y salida esperada. Ningúna fase debe cerrarse sin evidencia.

Fase	Fecha	Nombre	Actividades detalladas	Salida esperada
F0	2026-05-05	Preparación y smoke de ambiente	Confirmar dependencias, scripts, health check, GraphHopper, Cloudflare Tunnel, datos QA, carpeta de evidencias y versión bajo prueba.	Ambiente listo, datos definidos y bloqueos iniciales documentados.
F1	2026-05-05 a 2026-05-06	Caja blanca backend unitaria	Ejecutar la suite PHPUnit `F1` sobre `backend/tests/F1`, revisando servicios, FormRequests, modelos, relaciónes, accessors, formateadores, reglas de dominio, planeación e instrucciones sin endpoints HTTP.	Resultado de `php artisan test --testsuite=F1`, cobertura `coverageF1` y lista de hallazgos unitarios.
F2	2026-05-06 a 2026-05-07	Caja blanca frontend	Ejecutar analyzer, pruebas de servicios, providers, widgets, audio, rutas y flujo de mapa.	Resultado completo de `flutter analyze` y `flutter test`.
F3	2026-05-07 a 2026-05-08	Integración backend API	Ejecutar la suite PHPUnit `F3` sobre `backend/tests/F3` para validar endpoints con tokens, payloads, permisos, errores 401/403/422, upload, health y planeación por API.	Evidencia de contratos API por módulo y cobertura `coverageF3`.
F4	2026-05-08 a 2026-05-09	Sistema integrado local	Levantar stack local, ejecutar flujos desde UI, validar mapa, rutas, reportes y GraphHopper.	Smoke funcional integrado con capturas.

Fase	Fecha	Nombre	Actividades detalladas	Salida esperada
F5	2026-05-09 a 2026-05-11	Caja negra E2E navegador	Ejecutar flujos de usuario en navegador: onboarding, mascotas, reportes, comentarios, rutas, permisos y responsive.	Reporte E2E o matriz manual equivalente si la automatización aún no existe.
F6	2026-05-11 a 2026-05-12	Pruebas manuales funcionales	Revisar mensajes, errores, datos límite, viewport móvil, permisos de ubicación, estados vacíos y recuperación visual.	Matriz exploratoria con observaciónes y defectos.
F7	2026-05-12 a 2026-05-13	No funcional y carga con Postman + Cloudflare	Ejecutar iteraciones sobre health, login, listados, creación de recursos, upload y planeación usando URL Cloudflare.	Reporte de latencia, errores y comportamiento del servidor.
F8	2026-05-14	Regresión y cierre QA	Reejecutar P0/P1, hacer retest, consolidar evidencias, clasificar incidencias abiertas y emitir decisión de salida.	Informe final con decisión Go/No-Go.

5.1. Regla de clasificación de casos por contexto

La clasificación de los casos no se hará solamente por pantalla o por endpoint. Cada funcionalidad debe analizarse desde el contexto donde realmente puede observarse con precisión. Por ejemplo, el login completo no debe tratarse como prueba unitaria del frontend, porque depende de formulario, API, token, persistencia de sesión y respuesta del backend. En frontend sí se pueden probar unitariamente validaciónes locales, transformación de estado, almacenamiento de sesión simulado, providers y widgets aislados. El login completo se prueba como integración API, sistema integrado y E2E de caja negra.

Contexto	Qué se válida	Qué no se fuerza en ese contexto	Evidencia aceptada
Unitarias backend	Servicios, reglas de dominio, FormRequests, modelos, relaciónes, normalización de payloads, autorización encapsulada, formateadores y transformaciones sin navegador ni endpoints HTTP.	No se validan rutas HTTP, controladores, middleware, respuestas JSON completas, experiencia visual ni navegación de usuario.	Salida `php artisan test --testsuite=F1`, método, clase, aserciones relevantes y cobertura `coverageF1`.
Integración backend API	Rutas HTTP, middleware `api.token`, controladores, modelos, base de datos de prueba, códigos 200/201/401/403/404/422/502 y forma JSON.	No se válida renderizado Flutter ni interacciones del mapa; no se mezcla con pruebas unitarias F1.	Respuesta JSON, código HTTP, registro persistido, salida `php artisan test --testsuite=F3` y cobertura `coverageF3`.
Unitarias frontend	Providers Riverpod, clases de estado, servicios Dart, validaciónes de formulario sin servidor, parsers, filtros, selección de capas, audio y preferencias.	No se válida login real contra Laravel ni subida real de archivos.	Salida `flutter test`, asserts de estado y widget tests aislados.
Integración frontend	Widgets conectados a providers, repositorios mockeados, estados de carga/error/vacío, navegación de paneles y lectura de datos de prueba.	No se asume disponibilidad del backend real.	Widget test, captura, log de test y descripción de mock utilizado.
Sistema integrado local	Frontend Flutter, backend Laravel, SQLite local, medios, GraphHopper y Cloudflare Tunnel funcionando juntos.	No se mide carga máxima ni se simulan usuarios masivos.	Capturas, logs de servicios, respuestas HTTP y estado de contenedores.
Caja negra E2E	Flujos completos desde navegador usando Playwright o ejecución manual equivalente mientras se crean los scripts.	No se inspecciona código interno ni se depende de clases específicas.	Reporte Playwright, screenshots, traces, consola y Network.
Contrato GraphHopper	Payload enviado por Laravel, respuesta de ruteo, polilínea, distancia, tiempo e instrucciones convertidas al formato de A4MPets.	No se prueba la lógica interna de GraphHopper cómo producto externo.	Request/response, logs del servicio Laravel, estado del contenedor y salida del endpoint `/routes/plan`.

5.2. Mapa detallado de cobertura Backend Laravel

La cobertura backend debe incluir pruebas unitarias, pruebas de integración API y validaciónes de sistema cuando el comportamiento dependa de frontend o GraphHopper. La cobertura de código debe revisarse por sentencia, rama y condición en servicios con decisiónes relevantes, especialmente validaciónes, permisos, eliminaciones en cascada, comentarios, filtros de peligro y manejo de respuestas externas.

Módulo backend	Unitarias de caja blanca	Integración API	Sistema / E2E relaciónado
Autenticación	Validar `AuthTokenService`, generacion/revocación de tokens, hash SHA-256, `formatUser` y FormRequests de auth.	`POST /auth/register`, `POST /auth/login`, `GET /auth/me`, `PUT /auth/profile`, `POST /auth/logout`, errores 401 y 422.	Registro desde UI, login por correo, login por username, persistencia de sesión, logout y bloqueo de acciones privadas sin token.
Perfil de usuario	Validar `formatUser`, ausencia de campos sensibles y reglas unitarias de `UpdateProfileRequest`.	Actualizar perfil con token válido, rechazar payload incompleto y confirmar que `me` devuelve datos actualizados.	Pantalla o panel que consume perfil local, comentarios mostrando autor y reportes asociados al usuario.
Mascotas	Validar `PetService`, `Pet` model, relaciónes, normalización de estado, `canMutate` y `markFound`.	`GET /pets`, `GET /pets/{pet}`, `POST /pets`, `PUT /pets/{pet}`, `DELETE /pets/{pet}`, `POST /pets/{pet}/mark-found`, 403 sobre mascotas ajenas.	Crear mascota desde UI, editar datos, eliminar mascota, marcar encontrada y usar la mascota en reportes de pérdida.
Reportes de mascota pérdida	Validar `LostPetReportService`, modelo, accessors, asociacion con mascota, bloqueo de duplicados, comentarios y sync documentado.	`GET /lost-pet-reports`, `GET /lost-pet-reports/{id}`, `POST`, `PUT`, `DELETE`, `comments`, permisos de autor y validaciónes 422.	Crear reporte en mapa, selecciónar ubicación manual, ver marcador, abrir detalle, comentar y verificar que aparece en capas.
Reportes de mascota encontrada	Validar `FoundPetReportService`, modelo, accessors y contexto con o sin reporte perdido vinculado.	`GET /found-pet-reports`, `GET /found-pet-reports/{id}`, `POST`, `PUT`, `DELETE`, comentarios y payloads con `matchedLostPetReportId`.	Crear reporte encontrado, vincularlo a reporte perdido, visualizar tarjeta con resumen de mascota y navegar a ubicación.
Reportes genéricos	Validar `GenericReportService`, modelo, normalización de riesgo, radio, autor, comentarios y sync documentado.	`GET /generic-reports`, `POST /generic-reports`, `GET /generic-reports/{id}`, `PUT`, `DELETE`, `comments`, validaciónes de coordenadas y radio.	Crear reporte genérico desde UI, verlo en mapa, abrir detalles, comentar y confirmar persistencia.

Módulo backend	Unitarias de caja blanca	Integración API	Sistema / E2E relaciónado
Reportes de peligros	Validar `HazardReportService`, `HazardReport`, catálogos, normalizadores, filtros de riesgo, scopes y helpers de estado.	`GET /hazards`, `POST /hazards`, `GET /hazards/{id}`, `PUT`, `DELETE`, `comments`, filtros por nivel/categoría y errores de payload.	Crear peligro desde mapa, alternar círculos de peligro, abrir detalle, confirmar/denegar con comentarios rápidos y usarlo en planeación.
Comentarios	Validar `ReportCommentService`, `HazardCommentSyncService`, hashes, deduplicación, autor, anónimo, fecha y relaciónes.	Endpoints de comentarios para peligros, perdidos, encontrados y genéricos; 401 sin token, 404 si el reporte no existe, 422 texto inválido.	Agregar comentario desde panel, ver actualización del contador/lista y comprobar que el autor se muestra correctamente.
Medios	No tiene servicio unitario dedicado en F1; reglas y contratos HTTP se cubren en F3.	`POST /media/upload`, `GET /media/{bucket}/{fileName}`, rechazo por tipo no permitido, payload corrupto o archivo inexistente.	Subir foto de mascota, foto de reporte y vista previa; confirmar que la URL queda visible desde frontend y Cloudflare.
Rutas guardadas	Validar `TravelRouteService`, `TravelRoute`, `RoutePoint`, orden de puntos, `points_json`, eliminación y pertenencia.	`GET /routes`, `GET /routes/{id}`, `POST /routes`, `PUT /routes/{id}`, `DELETE /routes/{id}`, permisos y payloads inválidos.	Crear ruta desde editor de mapa, editar nombre/puntos, eliminar ruta, selecciónarla para viaje y verificar polilínea.
Planificacion de ruta	Validar `GraphHopperRoutingService` y `NavigationInstructionFormatter` con HTTP fakeado, payload, perfiles, filtros, fallback y errores controlados.	`POST /routes/plan` con dos puntos, multipunto, filtros de evitar peligro, GraphHopper disponible y GraphHopper no disponible.	Planear ruta desde UI, observar polilínea, instrucciones, distancia, alertas de proximidad y recálculo si cambia la selección.
Health y configuración	No corresponde a F1 unitaria; se cubre en F0/F3/F4 según ambiente y contrato HTTP.	`GET /health`, arranque con Docker, arranque local y verificacion de respuesta estable.	Smoke F0/F4, Cloudflare Tunnel y monitoreo durante carga manual.
Seguridad y permisos	Validar ownership encapsulado en servicios (`canMutate`) y reglas FormRequest; middleware/token completo se válida en F3.	Repetir mutaciones sin token, con token inválido, con usuario no propietario y con recurso inexistente.	E2E: intentar acciones privadas después de logout y confirmar que la UI no deja guardar datos protegidos.

5.3. Mapa detallado de cobertura Frontend Flutter

La cobertura frontend se divide entre lógica aislada, widget tests, integración con repositorios mockeados y flujos completos en navegador. No se debe convertir cada flujo visual en una prueba unitaria. Formularios que dependen del backend, del mapa o de subida de archivos se cubren mejor con widget tests, integración

o E2E.

Módulo frontend	Unitarias / providers	Widget o integración Flutter	Sistema / E2E relaciónado
Sesion local	`session_providers`, `SessionStore`, lectura/escritura de token, usuario invitado y perfil local.	Estado autenticado, estado invitado, expiracion simulada y limpieza después de logout.	Login real, logout real, reinicio de app conservando sesión cuando el token existe.
Paneles de autenticación	Validaciónes locales de campos, habilitacion de botones, mensajes de error y estado de carga.	Render de `LoginPanelContent` y `RegisterPanelContent` con entradas válidas e inválidas.	Registro y login contra backend desde navegador; no se trata como unitaria pura.
Providers de mapa	`panelFlowProvider`, `routeEditorProvider`, `reportFlowProvider`, `tripFlowProvider`, `mapPresentationProvider` y transiciones principales.	Panel activo, cierre, minimizado, selección de ubicación manual, selección de reporte y reinicio de flujos.	Navegación real entre paneles, apertura desde marcadores y consistencia visual del mapa.
Mapa y capas	Filtros de peligros bajo/medio/alto, reportes perdidos, encontrados y genéricos, foco de cámara y selección de marcador.	`MapWidget` con datos simulados, marcadores visibles, polilínea de ruta y taps sobre mapa.	Usuario activa/desactiva capas, selecciónа marcadores y confirma que el detalle corresponde al item tocado.
Mascotas	Validaciónes de formulario, modelo `PetRecord`, foto local, estados vacíos y lista de mascotas.	Panel de gestión de mascotas con crear/editar/eliminar usando repositorio mockeado.	Crear mascota real, editarla, eliminarla, marcarla encontrada y usarla para reporte perdido.
Reporte de mascota pérdida	Estado de ubicación manual, mascota selecciónada, preview, fecha, teléfono, foto y descripción.	Formulario con mascota válida, sin mascota, ubicación ausente y foto opcional.	Crear reporte perdido desde mapa, ver marcador, abrir detalle y comentar.
Reporte de mascota encontrada	Vínculo opcional a reporte perdido, selección de mascota, coordenadas, foto, teléfono y descripción.	Panel con reporte vinculado, reporte no vinculado y datos incompletos.	Crear reporte encontrado, vincularlo, comprobar tarjeta de resumen y navegar a ubicación.
Reportes genéricos	Categoria, descripción, nivel/categoría de peligro cuando aplique, radio de impacto, foto y preview.	Formulario con datos válidos, faltantes y límites de coordenadas/radio.	Crear reporte genérico, verlo en mapa, abrir detalle y comentar.
Reportes de peligros	Categoria, duración, nivel bajo/medio/alto, radio, foto, ubicación manual, preview y círculos.	Formulario, panel de detalle, panel de comentarios rápidos y panel de capas.	Crear peligro, alternar capas, confirmar/denegar peligro, ver impacto en planeación de ruta.

Módulo frontend	Unitarias / providers	Widget o integración Flutter	Sistema / E2E relaciónado
Comentarios	Construcción de comentario, validación de texto vacío, selección de tipo de reporte y actualización optimista o recarga.	`ReportCommentsPanelContent` para peligro, perdido, encontrado y genérico con repositorios mockeados.	Publicar comentario real y confirmar que aparece después de refrescar.
Rutas y editor	`routeEditorProvider`, agregar punto, editar punto, selecciónar ruta, borrar borrador, polilínea y nombre.	Panel de rutas con crear/editar/eliminar usando rutas simuladas.	Crear ruta en mapa, editar puntos, guardar, selecciónar y eliminar.
Viaje y navegación	`tripFlowProvider`, filtros de niveles de peligro, mascota/ruta selecciónada, instrucciones, progreso y finalizacion.	Panel de inicio de viaje, soporte de viaje, instrucciones y cambios de filtro.	Iniciar viaje con ruta real, recibir instrucciones, ver alertas y finalizar viaje.
Audio y preferencias	Preferencia de transporte, configuración de audio, mensajes generados, progreso de reproducción y mute.	Widgets de configuración y clases de soporte con estados simulados.	Activar/desactivar audio durante viaje y confirmar mensajes esperados.
Estados visuales	Carga, error, vacío, datos offline simulados, formularios con texto largo, responsive y accesibilidad básica.	Widget tests con tamaños de viewport y datos límite.	E2E responsive en escritorio y móvil, consola sin errores visibles.

5.4. Mapa detallado de cobertura GraphHopper

GraphHopper se trata como servicio externo ejecutado localmente con Docker. No se prueba su algoritmo interno, sino la integración de A4MPets con su API, la estabilidad del contenedor, la calidad de los datos de entrada y la forma en que Laravel convierte la respuesta para Flutter.

Área GraphHopper	Contrato / integración	Sistema integrado	No funcional
Contenedor y configuración	Validar `docker-compose.yml`, puertos 8989/8990, volumen `map.osm.pbf`, cache de grafo y `backend/graphhopper/config.yml`.	`npm run graphhopper` deja el servicio disponible antes de planear rutas.	Registrar tiempo de arranque y condición cuando el mapa no existe o la caché está fría.
Endpoint de ruteo	Confirmar que Laravel envía puntos en orden, perfil esperado, coordenadas válidas y payload compatible.	`POST /routes/plan` devuelve distancia, duración, geometría e instrucciones consumibles por Flutter.	Medir latencia de rutas cortas, multipunto y rutas con filtros de peligro.
Transformación de respuesta	Validar conversión de polilínea, instrucciones, distancia, tiempo, waypoints y mensajes de navegación.	UI muestra polilínea, lista de instrucciones y resumen de viaje sin romper el mapa.	Documentar respuestas lentas, vacías o incompletas.

Área GraphHopper	Contrato / integración	Sistema integrado	No funcional
Filtros por peligros	Confirmar que los niveles bajo, medio y alto se consideran en el cálculo o en la decisión de advertencia según implementación.	Cambiar filtros en UI altera la planificación o las alertas esperadas.	Comparar tiempos con todos los filtros activos, filtros parciales y sin filtros.
Fallo controlado	Simular GraphHopper detenido, URL incorrecta, timeout, coordenadas fuera de cobertura y respuesta no JSON.	Backend devuelve error controlado y frontend muestra mensaje comprensible.	Confirmar que el backend no queda caído y que `/health` sigue respondiendo.

5.5. Mapa detallado de flujos E2E de caja negra

Los flujos E2E se ejecutarán preferiblemente con Playwright sobre Flutter web. Mientras no exista automatización completa, cada flujo puede ejecutarse manualmente siguiendo los mismos pasos y guardando evidencia equivalente. El objetivo es cubrir comportamiento real de una persona usuaria sin depender del conocimiento interno del código.

Flujo E2E	Backend implicado	Frontend implicado	GraphHopper implicado	Validaciónes principales
Registro, login y logout	Auth, tokens, perfil	Panel login/register, sesión local, paneles protegidos	No aplica	Usuario queda autenticado, puede ver datos privados, hace logout y pierde acceso a mutaciones protegidas.
Gestión de mascotas	Pets, media	Panel mascotas, formularios, foto, estados vacíos	No aplica	Crear, editar, listar, eliminar y marcar encontrada sin afectar mascotas de otro usuario.
Reporte de mascota pérdida	Pets, lost-pet-reports, media, comments	Seleccion de mascota, ubicación manual, mapa, detalle, comentarios	No aplica	Reporte aparece en mapa, detalle corresponde al marcador, comentario queda persistido.
Reporte de mascota encontrada	Found-pet-reports, lost-pet-reports, media, comments	Vínculo a reporte perdido, formulario, lista, detalle	No aplica	Reporte encontrado puede quedar vinculado o independiente, y se visualiza correctamente.
Reporte de peligro	Hazards, media, comments	Formulario de peligro, capas bajo/medio/alto, detalle y comentario rápido	Indirecto cuando se planea ruta	Peligro aparece con circulo correcto, filtros lo ocultan/muestran y comentarios funcionan.

Flujo E2E	Backend implicado	Frontend implicado	GraphHopper implicado	Validaciónes principales
Reporte genérico	Generic-reports, media, comments	Formulario genérico, detalle y capas	No aplica	Reporte se crea, se lista, se abre, se comenta y respeta coordena-das/radio.
Rutas guardadas	Routes, route points	Editor de rutas, polilínea, panel de gestión	No aplica	Ruta de dos puntos y ruta multipunto se guardan, editan, eliminan y mantienen orden.
Planeación de viaje	Routes, `/routes/plan`, hazards	Panel de viaje, filtros, instrucciones, audio, mapa	Sí aplica	Se calcula ruta, se muestran instrucciones, alertas y finalizacion de viaje.
Permisos y errores	Middleware, validaciónes, ownership	Mensajes de error, formularios, estados de carga	Puede aplicar si la ruta falla	Sin token, token inválido, datos incompletos, recurso ajeno y servicio externo caído se manejan sin pantalla rota.
Responsive y compatibilidad	Todos los módulos públicos y privados	Layout móvil/escritorio, paneles, mapa, formularios	Segun flujo	No hay overflow, texto ilegible, botones inaccesibles ni errores de consola.

5.6. Relación esperada entre esta estrategia y el documento de casos

El documento `casos_de_prueba_a4mpets_v2.md` contiene los casos ejecutables. Esta sección del plan sirve cómo mapa de cobertura para verificar que cada módulo importante aparece en al menos una fase. Cuando se agreguen nuevos casos, deben conservar el formato por bloque y deben indicar fase, prioridad, tipo de prueba, módulo, descripción detallada, objetivo, precondiciónes, datos, pasos, resultado esperado, evidencia y resultado real.

La distribución mínima esperada es la siguiente:

Sistema	Unitarias	Integración	Sistema	E2E caja negra	Manual / no funcional
Backend Laravel	Servicios, requests, permisos, formateadores, eliminaciones y transformacio-nes.	Endpoints API, middleware, base de datos, medios y comentarios.	Health, logs, SQLite, Cloudflare y relación con frontend.	Flujos completos que crean y consultan datos desde navegador.	Carga con Postman, recuperación y observabilidad.

Sistema	Unitarias	Integración	Sistema	E2E caja negra	Manual / no funcional
Frontend Flutter	Providers, servicios Dart, validaciónes locales, filtros, preferencias y audio.	Widgets con providers y repositorios mockeados.	Mapa, paneles, formularios y API real local.	Playwright sobre registro, mascotas, reportes, rutas y viaje.	Exploratoria visual, responsive, permisos de ubicación y mensajes.
GraphHopper	No aplica como unidad interna del proyecto.	Contrato desde Laravel y conversión de respuesta.	Contenedor, rutas reales, instrucciones y fallos controlados.	Planeación de viaje desde navegador.	Latencia, arranque, timeouts y estabilidad bajo repetición.

5.7. Flutter

Herramienta	Uso exacto	Evidencia
`flutter analyze`	Detectar errores estáticos, imports rotos, tipos inválidos y advertencias relevantes.	Salida completa del comando.
`flutter test`	Ejecutar pruebas unitarias, providers y widgets.	Salida completa del comando o reporte machine si se configura después.
Flutter web en Chrome	Ejecutar flujos visuales de sistema y manuales.	Capturas, consola y Network.
DevTools navegador	Revisar errores de JavaScript, solicitudes HTTP, CORS y tiempos de respuesta.	Captura de consola/Network o HAR.
Playwright	Automatizar flujos E2E de caja negra.	HTML report, screenshots y traces cuando existan.

5.8. Laravel y PHP

Herramienta	Uso exacto	Evidencia
`php artisan test --testsuite=F1`	Ejecutar pruebas unitarias backend aisladas en `backend/tests/F1`.	Salida de PHPUnit y reporte `coverageF1`.
`php artisan test --testsuite=F3`	Ejecutar pruebas de integración backend/API en `backend/tests/F3`.	Salida de PHPUnit y reporte `coverageF3`.
`php artisan test`	Ejecutar regresión completa del backend incluyendo F1 y F3.	Salida consolidada de PHPUnit.
PHPUnit Feature	Validar endpoints, middleware y base de datos de prueba dentro de F3.	Resultado por clase de prueba.
Laravel logs	Revisar excepciones, errores 500 y trazas no controladas.	Extracto de `storage/logs`.
Composer	Confirmar dependencias y paquetes instalados.	`composer install` o estado de `vendor`.

Herramienta	Uso exacto	Evidencia
Artisan	Ejecutar health, pruebas, migraciones de test si aplica y utilidades del proyecto.	Salida del comando.

5.9. Integración y no funcionales

Herramienta	Uso exacto	Evidencia
`npm run backend`	Levantar Laravel local desde scripts del repo.	Consola del proceso.
`npm run frontend`	Levantar Flutter web local.	URL y captura de app.
`npm run graphhopper`	Levantar GraphHopper con Docker Compose.	Estado del contenedor.
`npm run dev:cloudflare`	Levantar backend, frontend runtime y GraphHopper con túnel.	URL pública y logs Cloudflare.
Postman Runner	Ejecutar solicitudes repetidas para carga manual.	Reporte exportado con iteraciones.

6. Ambiente de pruebas

Componente	Configuración obligatoria	Validación antes de probar
Backend Laravel	Ejecutar desde `C:\projects\Web\A4MPets\backend` o mediante script npm del root.	`GET /api/health` responde JSON con `status=ok`.
Frontend Flutter	Ejecutar desde `C:\projects\Web\A4MPets\frontend` o mediante script npm del root.	La app carga el mapa y no muestra pantalla blanca.
GraphHopper	Ejecutar con Docker usando `npm run graphhopper`.	El endpoint de planeación no falla por conexión rechazada.
Cloudflare Tunnel	Usar `cloudflared` mediante `npm run dev:cloudflare`.	`.runtime/cloudflare-url.txt` contiene URL `trycloudflare.com`.
Base de datos	Ambiente local o test DB controlada.	Usuarios QA y datos de prueba no colisionan con ejecuciones previas.
Navegador	Chrome desktop y viewport móvil simulado.	DevTools disponible para consola y Network.

6.1. Datos QA controlados

ID	Dato	Valor exacto o formato
D-001	Usuario principal	Email `qa_a4m_01@example.com`, usuario `qa_a4m_01`, password `QAtest1234`.

ID	Dato	Valor exacto o formato
D-002	Usuario secundario	Email `qa_a4m_02@example.com`, usuario `qa_a4m_02`, password `QAtest1234`.
D-003	Mascota base	Nombre `Luna QA`, nacimiento `2024-01-15`, adopción `2024-05-20`.
D-004	Coordenadas base	`3.4516,-76.5320`, `3.4570,-76.5280`, `3.4620,-76.5240`.
D-005	Imagen válida	Archivo PNG o JPG pequeño, menor a 1 MB, legible por frontend y backend.
D-006	Imagen inválida	Texto literal `no-es-base64` enviado como si fuera contenido de imagen.
D-007	Texto de comentario	`Comentario QA generado para evidencia funcional`.
D-008	Descripción larga	Texto de al menos 500 caracteres para validar layout y persistencia.

7. Criterios de entrada y salida

7.1. Criterios de entrada

Criterio	Regla verificable
Versión identificada	Antes de ejecutar, registrar branch, commit o descripción exacta de build.
Ambiente levantado	Backend, frontend y GraphHopper deben estar disponibles para fases integradas.
Datos QA preparados	D-001 a D-008 deben existir o poder crearse durante la fase.
Evidencia definida	Cada fase debe tener lugar donde guardar capturas, logs y respuestas.
Matriz vigente	El documento de casos v2 debe ser la fuente de ejecución.

7.2. Criterios de salida

Criterio	Regla verificable
Casos P0	Todos los P0 ejecutados con resultado Aprobado, Fallido con defecto, o Bloqueado con causa documentada.
Defectos bloqueantes	No puede quedar defecto crítico o alto sin decisión formal del equipo.
Evidencia	Todo caso ejecutado debe tener evidencia suficiente para repetir o auditar el resultado.

Criterio	Regla verificable
Regresión	Los casos afectados por correcciones deben reejecutarse antes del cierre.
Informe final	Debe contener totales de aprobados, fallidos, bloqueados, no ejecutados e incidencias abiertas.

8. Amenazas del proceso y mitigaciones

Amenaza del proceso	Impacto posible	Mitigación operacional
Cloudflare Tunnel cambia la URL pública durante la sesión	Las pruebas externas o de Postman apuntan a una URL vencida.	Registrar URL al inicio de cada ejecución, validar health antes de cada bloque y actualizar variables de Postman.
GraphHopper no responde	Las pruebas de rutas y viaje quedan bloqueadas.	Separar casos de rutas del resto, registrar bloqueo y continuar con módulos no dependientes.
Datos QA duplicados	Registro, mascotas o reportes fallan por información previa.	Usar sufijos con timestamp cuando el caso lo permita y limpiar datos al cerrar fase.
Saturación del equipo local	La carga con Postman produce errores no atribuibles al código.	Documentar hardware, cerrar aplicaciones no necesarias y aumentar iteraciones de forma gradual.
Cambios de código durante la ejecución	Los resultados dejan de representar una versión estable.	Registrar commit antes de empezar y reiniciar fase si cambia la versión.
Evidencia incompleta	El resultado no se puede auditar ni repetir.	No cerrar casos sin captura, log, respuesta exportada o reporte de herramienta.
Diferencias desktop/móvil	Un flujo aprobado en desktop falla en viewport móvil.	Ejecutar smoke mínimo en 1366x768 y 390x844 para fases visuales.

9. Reglas de comunicación y cierre

1. Ningún caso se marca como aprobado sin evidencia.
2. Ningún fallo se resume solamente como “no funciona”; debe incluir pasos, resultado actual, resultado esperado y evidencia.
3. Toda incidencia debe indicar módulo, severidad, ambiente, datos usados y si bloquea una fase.
4. Las palabras del dominio deben ser consistentes: se habla de peligros, reportes de peligros y niveles de peligro.
5. Si una prueba no puede ejecutarse por ambiente, se marca Bloqueada y no Fallida.
6. Si una prueba no aplica a la versión evaluada, se marca No aplica con justificación.

Apéndice E

Casos de Prueba A4MPets

El presente apéndice contiene el documento consolidado de casos de prueba (CDP) definidos para el proyecto *A4MPets*. Este documento organiza de manera estructurada los escenarios de prueba diseñados para cada fase del proceso de aseguramiento de calidad, incluyendo pruebas de preparación de ambiente, pruebas unitarias, pruebas de integración, pruebas funcionales, pruebas de interfaz, pruebas de arquitectura y validaciones finales del sistema.

Cada caso de prueba incorpora un identificador único, descripción, precondiciones, pasos detallados de ejecución, datos de entrada, resultado esperado y resultado real, permitiendo garantizar trazabilidad, repetibilidad y cobertura sistemática sobre los distintos componentes del prototipo. Asimismo, los casos definidos se alinean con la estrategia de pruebas establecida previamente en el Plan de Pruebas, sirviendo como base para la ejecución técnica y la recolección de evidencias presentadas en los anexos correspondientes.

Casos de Prueba A4MPets v2

Documento de casos de prueba ejecutables por fase

Campo	Valor
Proyecto	A4MPets
Documento	Casos de prueba ejecutables por fase
Versión	2.0
Fecha de creación	2026-05-06
Responsable	Equipo QA
Referencia	plan_de_pruebas.md v2

Este documento contiene los casos de prueba organizados por fase según la estrategia definida en el plan de pruebas. Cada caso incluye identificador único, descripción, precondiciones, pasos detallados, datos de entrada, resultado esperado y resultado real.

Fase F0 - Preparación y smoke de ambiente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F0-001	Verificar que el repositorio del proyecto se encuentra en laruta esperada y contiene las carpetas principales backend, frontend, docs y scripts	Acceso al sistema de archivos del equipo de pruebas	1. Abrir una terminal en el equipo de pruebas. 2. Navegar a la ruta C:\projects\Web\A4MPets. 3. Listar el contenido del directorio raíz. 4. Confirmar que existen las carpetas backend, frontend, docs y scripts. 5. Confirmar que existe el archivo package.json en laraiz.	Ruta del repositorio: C:\projects\Web\A4 MPets	Se visualizan las carpetas backend, frontend, docs y scripts, asi como el archivo package.j son en laraiz del repositori o	Pendiente
F0-002	Verificar que las dependencias de Node.js del proyecto raíz están instaladas correctamente incluyendo concurr ently y puppeteer	Node.js instalado en el equipo de pruebas. Repositorio clonado en laruta esperada	1. Abrir terminal en C:\projects\Web\A4MPets. 2. Ejecutar npm install. 3. Verificar que el directorio node_modules se crea sin errores. 4. Confirmar que concurrently y puppeteer aparecen dentro de node_modules.	Comando: npm install	La instalacion finaliza sin errores críticos y las depende ncias concurrently y puppeteer quedan disponibles en node_mo dules	Pendiente
F0-003	Verificar que las dependencias de PHP del backend están instaladas mediante Composer y que la carpeta vendor existe	PHP y Composer instalados. Repositorio clonado	1. Abrir terminal en C:\projects\Web\A4MPets \backend. 2. Ejecutar composer install. 3. Verificar que la carpeta vendor se genera. 4. Confirmar que no hay errores de paquetes faltantes en la salida del comando.	Comando: composer install en el directorio backend	La carpeta vendor se genera correctamente y composer install finaliza sin errores de depende ncias	Pendiente
F0-004	Verificar que el backend de Laravel arranca correctamente mediante el scriptnpmrun backend	Dependencias de Node.js y Composer instaladas	1. Abrir terminal en C:\projects\Web\A4MPets. 2. Ejecutar npm run backend. 3. Esperar a que el servidor local inicie. 4. Observar la salida de la terminal para confirmar que el servidor arranca en el puerto 8000. 5. No cerrar la terminal.	Comando: npm run backend	El servidor de Laravel inicia sin errores y muestra la URL local de escucha en el puerto8000	Pendiente
F0-005	Verificar que el endpoint de health check del backend responde con estadook y un timestamp válido	Backend de Laravel ejecutandose en el puerto8000	1. Abrir un navegador o herramienta como Postman. 2. Enviar una solicitud GET a http://localhost:8000/api/ health. 3. Revisar el código de respuesta HTTP. 4. Revisar el cuerpo de la respuesta JSON. 5. Confirmar que el campo status tiene el valor ok y que el campo timestamp contiene una fecha válida.	URL: GET http://localhost:800 0/api/health	La respuest a tiene codigoHTTP200, el campo status esok y el campo timestamp contiene una marca de tiempo válida en formatoISO	Pendiente
F0-006	Verificar que el frontend de Flutterweb arranca correctamente mediante el scriptnpmrun frontend	Flutter SDK instalado. Dependencias del proyecto raíz instaladas	1. Abrir una nueva terminal en C:\projects\Web\A4MPets. 2. Ejecutar npm run frontend. 3. Esperar a que el proceso de compilacion y el servidor local de Flutter inicien. 4. Observar la salida para confirmar que el servidor local está disponible.	Comando: npm run frontend	El servidor de Flutterweb inicia correctamente y muestra la URL local donde la aplicación está disponibl e para acceso desde el navegador	Pendiente
F0-007	Verificar que la aplicación Flutterweb carga elmapa en el navegador sin mostrar una pantalla en blanco	Frontend de Flutter web ejecutandose	1. Abrir el navegador Chrome. 2. Navegar a laURL local del frontend de Flutter web. 3. Esperar a que la aplicación cargue completamente. 4. Verificar visualmente que el componente del mapa se renderiza correctamente. 5. Abrir DevTools del navegador y revisar la consola para confirmar que no hay errores críticos de JavaScript.	URL del frontend local	La aplicación muestra el mapa interactiv o de la ciudad, no se observa pantalla en blanco y la consola del navegador no muestra errores críticos de carga	Pendiente
F0-008	Verificar que el contenedor de GraphH opper arranca correctamente mediante el scriptnpmrun graphhopper	Docker Desktop instalado y ejecutandose. Archivo demapa map.osm.pbf presente en backend/graphhopper/data/	1. Abrir terminal en C:\projects\Web\A4MPets. 2. Ejecutar npm run graphhopper. 3. Esperar a que Docker descargue la imagen si es necesario y levante el contenedor. 4. Verificar con docker ps que el contenedor a4mpets-graphhopper está en estado running. 5. Confirmar que los puertos8989 y 8990 están mapeados.	Comando: npm run graphhopper	El contenedor a4mpets- graphhopper aparece en estado running con los puertos8989 y8990 mapeados correctamente alhost	Pendiente
F0-009	Verificar que el servicio de GraphH opper responde a solicitud es de ruteo básicas después de que el contenedorhaya completado la indexacion delmapa	Contenedor de GraphHopper en estado running con indexacion completada	1. Esperar al menos 60 segundos después de iniciar el contenedor para permitir la indexacion. 2. Abrir Postman o un navegador. 3. Enviar una solicitud GET a http://localhost:8989/route?point=3.4516,- 76.5320&point=3.4570,-76.5280&profile=car&type=json. 4. Revisar el código de respuesta y la estructura del JSON de respuesta.	URL: GET http://localhost:898 9/route con dos puntos de coordenadas en Cali	La respuest a tiene codigoHTTP 200 y contiene un objeto paths con al menos una ruta que incluye distancia, tiempo y geometri a	Pendiente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F0-010	Verificar que el archivo de configuración de GraphH opper existe y define los perfiles necesarios para el proyecto	Repositorio clonado	1. Navegar a C:\projects\Web\A4MPets \backend\graphhopper. 2. Abrir el archivo config.yml. 3. Revisar que define al menos los perfiles foot ycar. 4. Confirmar que el archivo referencia los datos del mapa correctamente.	Ruta: backend/graphhopper/config.yml	El archivo config.yml existe, define perfiles de ruteofoot ycar, y las referencias a los datos delmapa son coherent es con la estructur a del contenedor Docker	Pendiente
F0-011	Verificar que el archivo demapa de OpenStr eetMap utilizad o por GraphH opper existe y tiene un tamaño mayor acero bytes	Repositorio clonado	1. Navegar a C:\projects\Web\A4MPets \backend\graphhopper\data. 2. Verificar que existe el archivo map.osm.pbf. 3. Comprobar que el tamaño del archivo es mayor a 0 bytes.	Ruta: backend/graphhopper/data/map.osm.pbf	El archivo map.osm.pbf existe en la ruta indicada y su tamaño es superior a cero bytes, lo que confirma que contiene datos cartograficos	Pendiente
F0-012	Verificar que Cloudflare Tunnel se puede iniciar y genera una URL pública tempor al válida	Herramienta cloudflared instalada en el sistema. Backend ejecutandose	1. Abrir terminal en C:\projects\Web\A4MPets. 2. Ejecutar npm run dev:cloudflare. 3. Esperar a que el proceso genere laURL del tunel. 4. Verificar que el archivo.runtime/cloudflare-url.txt se crea con una URL que termine en trycloudflare.com. 5. Abrir la URL en el navegador para confirmar conectividad.	Comando: npm run dev:cloudflare	Se genera el archivo.runtime/ cloudflar e-url.txt con unaURL válida de trycloudfl are.com y la URL responde al ser abierta en el navegador	Pendiente
F0-013	Verificar que labase de datos SQLite del backend es accesibl e y que las tablas principales existen después de ejecutar las migraciones	Backend de Laravel ejecutandose o al menos las migraciones ejecutadas	1. Abrir terminal en C:\projects\Web\A4MPets \backend. 2. Ejecutar php artisan migrate:status. 3. Revisar la lista de migraciones para confirmar que todas están marcadas como ejecutadas. 4. Verificar que existen tablas para users, pets, hazard_reports, lost_pet_reports, found_pet_reports, generic_reports, travel_routes, route_points, report_comments y hazard_comments.	Comando: php artisan migrate:status	Todas las migraciones están marcadas como ejecutadas y las tablas principal es del sistema existen en labase de datos SQLite	Pendiente
F0-014	Verificar que es posible registrar al usuario principal de pruebasQA mediante el endpoint de registro de laAPI	Backend de Laravel ejecutandose. No existe un usuario previo con el correo qa_a4m_01 @example.com	1. Abrir Postman o herramienta equivalente. 2. Enviar una solicitudPOST a http://localhost:8000/api/ auth/register con el cuerpo JSON que incluyaname, email, password y password_confirmation. 3. Revisar el código de respuesta HTTP. 4. Confirmar que la respuesta contiene los campos success, data.user y data.access_token.	JSON: {"name": "qa_a4m_01", .""ail": "qa_a4m_01@example.com", "password": "QAtest1234", "password_confirmation": "QAtest1234"}	La respuest a tiene codigoHTTP201, el campo success es true, se recibe un objetouser conid, name y email correctos, y se genera un access_token detipo Bearer	Pendiente
F0-015	Verificar que es posible registrar al usuario secundario de pruebasQA mediante el endpoint de registro de laAPI	Backend de Laravel ejecutandose. No existe un usuario previo con el correo qa_a4m_02 @example.com	1. Abrir Postman o herramienta equivalente. 2. Enviar una solicitudPOST a http://localhost:8000/api/ auth/register con los datos del usuario secundario. 3. Revisar el código de respuesta HTTP. 4. Confirmar que la respuesta contiene access_token y los datos del usuario registrado.	JSON: {"name": "qa_a4m_02", .""ail": "qa_a4m_02@example.com", "password": "QAtest1234", "password_confirmation": "QAtest1234"}	La respuest a tiene codigoHTTP201, el campo success es true y se registra correctamente un segundo usuario independ iente del primero	Pendiente
F0-016	Verificar que el usuario principal QA puede iniciar sesión exitosa mente con su correo electronico después del registro	Usuario qa_a4m_01 @example.com registrado previamente. Backend ejecutandose	1. Enviar una solicitudPOST a http://localhost:8000/api/ auth/login con login igual al correo del usuario principal y password igual a la contraseña registrada. 2. Revisar el código de respuesta. 3. Confirmar que se recibe un token de acceso válido. 4. Guardar el token para las siguientes pruebas.	JSON: {"login": "qa_a4m_01@example.com", "password": "QAtest1234"}	La respuest a tiene codigoHTTP200, el campo success es true y se recibe un access_token detipo Bearer que se puedeusar en solicitudes autenticadas	Pendiente
F0-017	Verificar que la carpeta de evidencias para el ciclo de pruebas está creada y accesibl e para almacenar capturas, logs y reportes	Acceso al sistema de archivos del equipo de pruebas	1. Crear la carpeta docs/evidencias en el repositorio del proyecto si no existe. 2. Crear subcarpetas para cada fase: F0, F1, F2, F3, F4, F5, F6, F7 y F8. 3. Confirmar que se puede escribir un archivo de texto de prueba en cada subcarpeta. 4. Eliminar los archivos de prueba después de la verificacion.	Ruta base: C:\projects\Web\A4MPets\docs\evidencias\	La estructur a de carpetas de evidencias queda creada con subcarpetas porfase y se confirma que se pueden escribir archivos en cada una de ellas	Pendiente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F0-018	Verificar que el archivo docker-compose.yml del proyecto define correctamente los servicios de GraphH opper, backend y frontend con sus puertos y volumenes	Repositorio clonado	1. Abrir el archivo docker- compose.yml en la raíz del proyecto. 2. Verificar que define el servicio graphhopper con puertos8989 y 8990, volumen para map.osm.pbf y referencia al config.yml. 3. Verificar que define el servicio backend con puerto 8000, conexion SQLite y variable GRAPHHOP-PER_BASE_URL. 4. Verificar que define el servicio frontend con puerto 8080.	Archivo: docker- compose.yml	El archivo define lostres servicios con los puertos, volumen es, variables de entorno y depende ncias correctas según la arquitectura del proyecto	Pendiente
F0-019	Verificar que el scriptnpmrun dev levanta simulta neamente el backend, el frontend y GraphH opper utilizando concurr ently	Docker ejecutandose. Dependencias instaladas	1. Abrir terminal en C:\projects\Web\A4MPets. 2. Ejecutar npm run dev. 3. Observar que concurrently inicia tres procesos en paralelo: backend, frontend y graphhopper. 4. Esperar a que los tres servicios esten disponibles. 5. Verificar con GET /api/health que el backend responde. 6. Verificar que el frontend carga en el navegador.	Comando: npm rundev	Los tres servicios arrancan simultan eamente sin errores críticos, el health check del backend responde con status ok y el frontend carga elmapa en el navegador	Pendiente
F0-020	Verificar la Versión y el commit del codigobajo prueba y registrar está informacion como referencia paratodo el ciclo de pruebas	Git instalado. Repositorio clonado	1. Abrir terminal en C:\projects\Web\A4MPets. 2. Ejecutar git log -1 – format="%H %ai %s"para obtener el hash completo, fecha y mensaje del ultimo commit. 3. Ejecutar git branch –show-current para obtener la rama actual. 4. Registrar el hash del commit, la rama, la fecha y el mensaje como referencia oficial del ciclo de pruebas.	Comandos: git log -1, git branch –show- current	Se obtiene y documenta el hash del commit, la rama activa, la fecha y el mensaje del ultimo commit como identificador de la versionbajo prueba para todo el ciclo	Pendiente
F0-021	Verificar que los datos de coordena-dasbase de prueba corresp onden a ubicaciones válidas dentro del area de cobertu ra delmapa de Cali utilizad o por GraphH opper	Contenedor de GraphHop-per ejecutandose con mapa indexado	1. Tomar las coordenadasbase D-004: 3.4516,-76.5320 y 3.4570,- 76.5280. 2. Enviar una solicitud de ruteo a GraphHopper con estosdos puntos. 3. Verificar que GraphHopper devuelve una ruta válida entre ellos. 4. Repetir con el tercer punto 3.4620,- 76.5240 como destino adicional.	coordenadas: punto A 3.4516,-76.5320, punto B 3.4570,- 76.5280, punto C 3.4620,-76.5240	GraphHopper devuelve rutas válidas entre los puntos de prueba, confirmando que las coordenadas están dentro del area de cobertur a delmapa	Pendiente
F0-022	Verificar que el middleware de autentic acion por token del backend rechaza solicitud es sin token en endpoints protegidos	Backend ejecutandose	1. Enviar una solicitud GET a http://localhost:8000/api/ auth/me sin incluir ningun header de Authorization. 2. Revisar el código de respuesta HTTP. 3. Confirmar que el cuerpo de la respuesta indica que el usuario no está autenticado.	URL: GET http://localhost:800 0/api/auth/me sin header Authorization	La respuest a tiene codigoHTTP 401 y el cuerpo indica que la solicitud no está autenticada	Pendiente

Fase F1 - Caja blanca backend unitaria

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F1-001	Verificar que el servicio AuthTokenService genera un token de 64 caracteres aleatorios y lo almacena comohash SHA-256 en la tabla api_tokens	Base de datos de prueba migrada. ModeloUser disponible	1. Crear una instancia del servicio AuthTokenService. 2. Crear un usuario de prueba en labase de datos. 3. Invocar el método issueToken pasando el usuario y un nombre de token como mobile- register. 4. Verificar que el array retornado contiene la clave plain_token. 5. Verificar que en la tabla api_tokens existe un registro con el user_id del usuario y elhash SHA-256 del token plano.	Usuario de prueba con id generado automaticamente. Nombre del token: mobile-register	El método retorna un array con plain_token que tiene exactamente 64 caracteres, y la tabla api_tokens contiene un registro con el token_hash correspond iente alhash SHA- 256 del token plano	Pendiente
F1-002	Verificar que el servicio AuthTokenService revoca correctamente el token actual eliminandolo de la tabla api_tokens	Token emitido previament e para un usuario de prueba	1. Emitir un token para un usuario de prueba. 2. Simular un objeto Request que contenga el atributo api_token_mo del con la instancia del modelo ApiToken creado. 3. Invocar el método revokeCurrent Token. 4. Verificar que el registro del token ya no existe en la tabla api_tokens.	Instancia de ApiToken activa	El registro del token se elimina de la tabla api_tokens y la consulta posterior no encuentra tokens activos para ese usuario	Pendiente
F1-003	Verificar que el método formatUser del servicio AuthTokenService retorna unicamente los campos id, name, email y profile_photo_ url sin exponer datos sensibles como la contraseña	Usuario de prueba con todos los campos completados	1. Crear un usuario de prueba con name, email, password y profile_photo_url. 2. Invocar formatUser con el usuario. 3. Verificar que el array retornado contiene exactamente las claves id, name, email y profile_photo_url. 4. Verificar que no se incluyen campos sensibles como password o remember_token.	Usuario con name=qa_test, email=qa@test.com, profile_photo_url=http:/ /example.com/photo.jpg	El array retornado contiene los cuatro campos esperados con los valores correctos y no expone datos sensibles	Pendiente
F1-004	Verificar que los FormRequests de autenticación autorizan suuso y exponen reglas para login, registro y actualizacion de perfil	Clases de FormRequest disponibles	1. Instanciar cada request de autenticación. 2. Invocar authorize(). 3. Invocar rules(). 4. Verificar presencia de reglas obligatorias como login, password, email, name y profile_photo_url según corresponda.	LoginRequest, RegisterRequest, UpdateProfileRequest	Cada request autoriza y contiene las reglas esperadas sin ejecutar endpointsHTTP	Pendiente
F1-005	Verificar que los FormRequests de dominio autorizan suuso y exponen reglas para mascotas, peligros, reportes, comentarios y rutas	Clases de FormRequest disponibles	1. Instanciar requests dePets, Hazards, Reports y Routes. 2. Invocar authorize(). 3. Invocar rules(). 4. Verificar campos obligatorios y opcionales de cada modulo.	StorePetRequest, StoreHazardRequest, StoreLostPetReportRequest, StoreTravelRouteRequest y variantes update/comment	Cada request autoriza y declara reglas coherentes con su payload de dominio	Pendiente
F1-006	Verificar relaciones principales entre User, Pet, ApiToken, reportes de peligros y rutas	Base de datos de prueba migrada y factories disponibles	1. Crear usuario, mascota, token, peligro y ruta. 2. Invocar relaciones Eloquent. 3. Verificar belongsTo/hasMany esperados.	Usuario QA con mascota, token, peligro y ruta	Las relaciones de modelos retornan los registros esperados	Pendiente
F1-007	Verificar accessors de reportes y comentarios serializados como comments_json y reported_by_username	Reportes con comentarios asociados	1. Crear reportes de mascota pérdida, encontrada, genérico y peligro. 2. Asociar comentarios. 3. Leer accessorsJSON. 4. Verificar autor, anónimo, texto y timestamp.	Comentario QA con autor y fecha conocida	Los accessors retornanJSON estable y nombres de autor esperados	Pendiente
F1-008	Verificar que HazardReport resuelve catalogos, scopes y helpers de estado local/sincronizado	Catalogos de peligro migrados	1. Crear reportes de peligro con risk_level_code, category_code y estimated_dur ation_code. 2. Leer accessors. 3. Probar scopes nearby, byRiskLevel, synced y unsynced. 4. Probar isLocal, isSynced, markAsSynced y touchLastMod ified.	risk_level_code=high, category_code=pest_pre sence, duration=1_day	El modelo resuelve catalogos, scopes y helpers de estado correctamente	Pendiente
F1-009	Verificar que ReportComment y HazardComment generan hashes de contenido estables a precision de segundo y mantienen relaciones	Comentarios con fechafija y autor definido	1. Construir atributos de comentario. 2. Invocar makeContentHash. 3. Repetir con DateTime, milisegundos y string equivalente. 4. Verificar relaciones con usuario y reporte.	comment_text=Comentario QA, fecha fija	Los hashes son estables para el mismo contenido y las relaciones son correctas	Pendiente
F1-010	Verificar que TravelRoute decodifica points_json, normaliza puntos y sincroniza RoutePoint en orden secuencial	Modelo TravelRout e disponible	1. Crear ruta con points_json. 2. Guardar puntos con lat/lng y latitude/longitude. 3. Refrescar modelo. 4. Verificar RoutePoint con sequence0..n y accessor points_json.	Dos puntos en Cali	La ruta mantiene puntos válidos, descarta entradas invalidas y conserva el orden	Pendiente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F1-011	Verificar que PetService crea una mascota con campos obligatorios y opcionales correctamente persistidos	Base de datos de prueba con usuario creado	1. Invocar store con userId y array completo. 2. Verificar modelo retornado. 3. Verificar persistencia en tabla pets.	name=Luna QA, birth_date=2024-01-15, adoption_date=2024-05- 20, status=normal, is_ownerless=false	Mascota creada con campos correctos y registro persistido	Pendiente
F1-012	Verificar que PetService normaliza el estado de mascota convirtiendolost y pérdida al valor estandar pérdida	Servicio PetService instanciado	1. Invocar store con status=lost. 2. Invocar store con status=perdid a. 3. Invocar store con status=normal. 4. Verificar resultados.	Valores de status: lost, pérdida, normal	lost y pérdida quedan como pérdida; normal permanece sin cambio	Pendiente
F1-013	Verificar que PetService canMutate distingue propietario, no propietario y usuario nulo	Mascota creada por usuario A	1. Invocar canMutate con usuario B. 2. Invocar con usuario A. 3. Invocar connull. 4. Verificar booleanos.	Usuario A propietario, usuario B no propietario, null	Retorna false para no propietario/null y true para propietario	Pendiente
F1-014	Verificar que PetService markFound cambia estado a normal y elimina reportes de mascota pérdida asociados	Mascota pérdida con reporte activo	1. Crear mascota pérdida. 2. Crear reporte perdido asociado. 3. Invocar markFound. 4. Verificar estado y eliminacion del reporte.	Mascota status=pérdida	Estado cambia a normal y reportes perdidos asociados se eliminan	Pendiente
F1-015	Verificar que ReportCommen tService crea comentarios asociados, respeta anónimo y vincula autor cuando corresponde	Reporte existente y usuario disponible	1. Invocar storeComment con usuario autenticado. 2. Invocar storeComment anónimo. 3. Verificar texto, user_id, author_name e is_anonymous.	Texto Comentario QA	Comentarios creados con asociacion y anonimato correctos	Pendiente
F1-016	Verificar sincronización, deduplicacion y formateo de comentarios de reportes y peligros	Reportes con comentarios locales/rem otos	1. Invocar syncComments con duplicados. 2. Verificar deduplicacion por content_hash. 3. Formatear comentarios. 4. Verificar timestamps y campos de salida.	Comentarios con mismo texto/autor/fecha	La sincronización evita duplicados y los formatters retornan estructura estable	Pendiente
F1-017	Verificar que HazardValueNo rmalizer normaliza codigos de riesgo, categoría y duracion	Servicio HazardValu eNormalizer instanciado	1. Normalizar valores válidos. 2. Normalizar unknown/uns pecified/vacios. 3. Normalizar alias aggressive_animals. 4. Verificar fallback.	risk=high, category=aggressive_animals, duration=1_week	Codigos válidos permanecen, alias se mapean y valores desconocidos caen en fallback esperado	Pendiente
F1-018	Verificar que HazardRouting RiskFilter normaliza filtros seleccionados y decide que riesgos evitar	Servicio HazardRout ingRiskFilter instanciado	1. Invocar normalizeSele ctedFilters connull, duplicados e invalidos. 2. Evaluar shouldAvoidRisk para low, medium, high y undefined. 3. Evaluar shouldAvoidOt herReports.	filtros [high, other, invalid]	Filtros quedan unicos/válidos y las decisiones de avoidance son correctas	Pendiente
F1-019	Verificar que HazardReportS ervice gestiona store, update, paginate, delete y permisos de mutacion	Base de datos con usuario y catalogos	1. Crear peligro con store. 2. Listar con paginate. 3. Actualizar campos. 4. Validar canMutate. 5. Eliminar reporte y comentarios.	Peligro con coordenadas, radio y risk_level_code	El servicio persiste, actualiza, lista, válida permisos y elimina dependencias correctamente	Pendiente
F1-020	Verificar ramas de sincronización y consulta unsynced de HazardReportS ervice sin modificar código productivo	Peligros locales y payloads de sync	1. Crear peligro local. 2. Invocar unsynced. 3. Invocar sync con payload actual. 4. Documentar errores legacy si aparecen.	Payload local-first de peligro	unsynced retorna peligros locales ysync queda cubierto/d ocumentado según comportam iento actual	Pendiente
F1-021	Verificar que GenericReportS ervice gestiona store, update, delete, canMutate, paginate y sync	Usuario y payload de reporte genérico	1. Invocar store. 2. Actualizar. 3. Validar permisos. 4. Listar. 5. Ejecutar sync. 6. Eliminar.	latitude, longitude, description, risk_level_code	Servicio cubre ciclo de vida y sincronización documentada de reportes genéricos	Pendiente
F1-022	Verificar que LostPetReportS ervice gestiona store, update, bloqueo de duplicados, delete, permisos y sync	Mascota existente y usuario propietario	1. Crear reporte perdido. 2. Intentar duplicado activo. 3. Actualizar. 4. Validar permisos. 5. Ejecutar sync. 6. Eliminar.	pet_id, coordenadas, details, disappearance_date, phone	Servicio maneja ciclo devida, evita duplicados y documentasync actual	Pendiente
F1-023	Verificar que FoundPetRepor tService gestiona reportes encontrados con y sin vinculo a reporte perdido	Usuario, mascota o reporte perdido existente	1. Crear reporte encontrado vinculado. 2. Crear reporte sin vinculo. 3. Actualizar. 4. Validar permisos. 5. Ejecutar sync. 6. Eliminar.	matched_lost_pet_report_id opcional	Servicio maneja ambos contextos y sincronización documentada	Pendiente
F1-024	Verificar que TravelRouteService crea, válida permisos y elimina rutas de viaje	Usuario y puntos deruta válidos	1. Invocar store conname y points_json. 2. Verificar puntos. 3. Evaluar canMutate para propietario/no propietario. 4. Invocar delete.	name=Ruta QA, points_json con dos puntos	Ruta creada con puntos ordenados, permisos correctos y eliminacion persistida	Pendiente
F1-025	Verificar GraphHopperR outingService y NavigationInstr uctionFormatter como unidades aisladas de ruteo y soporte	Http fakeado y configuración de GraphHopper controlada	1. Invocar planRoute con walking y vehicle. 2. Verificar payload [lng,lat], perfiles, APIkey, fallback y custom_model. 3. Simular paths vacios/geometria vacia. 4. Normalizar instrucciones con campos completos y defaults.	Puntos de Cali, filtros high/other, instrucciones con interval	Servicio construye payload correcto, maneja errores controlados y soporte normaliza instruccion es	Pendiente

Fase F2 - Caja blanca frontend

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F2-001	Verificar que el comando flutter analyze no reporta errores críticos en el proyecto frontend completo	Flutter SDK instalado. Dependencias del frontend resueltas con flutter pub get	1. Abrir terminal en C:\projects\Web \A4MPets\frontend. 2. Ejecutar flutter pub get para resolver dependencias. 3. Ejecutar flutter analyze. 4. Revisar la salida línea por línea buscando errores de tipo error (no warnings niinfo). 5. Registrar la cantidad total de errores, warnings e infos reportados.	Comando: flutter analyze en el directorio frontend	El comando finaliza sin errores críticos de tipo error. Se permit e la presencia de warnings e infos que no bloque en la compil acion, pero se docum entan para revision posterior	Pendiente
F2-002	Verificar que el comando fluttertest ejecuta todas las pruebas unitarias existentes en el proyecto frontend sin fallos	Flutter SDK instalado. Dependencias resueltas	1. Abrir terminal en C:\projects\Web \A4MPets\frontend. 2. Ejecutar flutter test. 3. Esperar a que todas las suites de pruebas se ejecuten. 4. Revisar la salida para identificar pruebas pasadas y fallidas. 5. Registrar el total de pruebas ejecutadas, pasadas y fallidas.	Comando: flutter test	Todas las pruebas unitarias existentes pasan sin errores. Si hay fallos, se docum entan con el nombr e de la prueba y el mensaj e de error	Pendiente
F2-003	Verificar que el provider de sesión session_providers gestiona correctamente el estado autenticado almacenando el token y los datos del usuario local	código fuente del provider session_providers.dart disponible	1. Revisar el archivo providers/sessio n_providers.dart. 2. Crear un test unitario que instancie el provider de sesión. 3. Simular el almacenamiento de un token de acceso y un perfil de usuario local. 4. Verificar que el estado del provider refleja que el usuario está autenticado. 5. Verificar que los datos del usuario son accesibles a traves del provider.	Token de acceso simulado y datos de perfil de usuario con nombre, correo y foto	El provider almacena el token y los datos del usuario, el estado cambia a autenticado y los datos son accesib les correct ament e	Pendiente
F2-004	Verificar que el servicio SessionStore permite escribir y leer el token de sesión de forma persistente en el almacenamient o local	código fuente del servicio services/session_store. dart disponible	1. Revisar el archivo services/session _store.dart. 2. Crear un test que invoque el método de escritura del token. 3. Verificar que el método de lectura retorna el mismo token almacenado. 4. Invocar el método de limpieza o eliminacion del token. 5. Verificar que después de limpiar, la lectura retornanulo o un valor vacio.	Token de prueba: token_qa_simul ado_64_caracteres	El token se escribe correct ament e, selee sin alteraciones y después de la limpiez a la lectura retornanulo	Pendiente
F2-005	Verificar que el provider panelFlowProvider gestiona las transiciones entre paneles activos delmapa correctamente, incluyendo apertura, cierre y minimizado	código fuente de providers/panel_flow_ provider.dart disponible	1. Instanciar el provider panelFlowProvider en un contexto de prueba. 2. Verificar que el estado inicial no tiene ningun panel activo. 3. Activar un panel de reporte de peligro. 4. Verificar que el estado refleja el panel activo de reporte de peligro. 5. Solicitar el cierre del panel. 6. Verificar que el estado vuelve a no tener panel activo.	Tipo de panel: reporte de peligro	Las transiciones de estado del panel se ejecutan correct ament e: sin panel, panel activo y cierre de panel	Pendiente
F2-006	Verificar que el provider reportFlowProvider gestiona el flujo de creación de reportes incluyendo la seleccion de ubicación manual en elmapa	código fuente de providers/report_flow _provider.dart disponible	1. Instanciar el provider reportFlowProvider. 2. Iniciar un flujo de creación de reporte de peligro. 3. Simular la seleccion de una ubicación manual con coordenadas de prueba. 4. Verificar que el estado del provider almacena las coordenadas seleccionadas. 5. Verificar que el flujo puede avanzar al siguiente paso con los datos de ubicación.	coordenadas de seleccion manual: latitud 3.4516, longitud - 76.5320	El provider almacena las coorde nadas de la ubicación manual y permit e avanzar en el flujo de creación del reporte	Pendiente
F2-007	Verificar que el provider routeEditorProvider permite agregar puntos de ruta, editarlos y limpiar el borrador completo de laruta	código fuente de providers/route_editor _provider.dart disponible	1. Instanciar el provider routeEditorProvider. 2. Agregar un punto de ruta con coordenadas de prueba. 3. Verificar que el estado contiene un punto. 4. Agregar un segundo punto. 5. Verificar que el estado contiene dos puntos en orden. 6. Editar el primer punto con nuevas coordenadas. 7. Verificar la actualizacion. 8. Limpiar el borrador. 9. Verificar que el estado queda vacio.	Punto 1: latitud 3.4516 longitud -76.5320. Punto2: latitud 3.4570 longitud - 76.5280	Los puntos se agregan en orden, se pueden editar individ ualmente y la limpiez a del borrador deja el estado sin puntos	Pendiente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F2-008	Verificar que el provider tripFlow-Provider gestiona el inicio de un viaje con ruta seleccionada, filtros de nivel de peligro y mascota asociada	código fuente de providers/trip_flow_provider.dart disponible	1. Instanciar el provider tripFlowProvider. 2. Configurar una ruta seleccionada con al menosdos puntos. 3. Configurar filtros de nivel de peligro con los valores bajo, medio y alto. 4. Asociar una mascota al viaje. 5. Iniciar el viaje. 6. Verificar que el estado del provider refleja viaje activo conruta, filtros y mascota.	Ruta con dos puntos, filtros de peligro: bajo, medio, alto. Mascota: Luna QA	El viaje se inicia correct ament e con la ruta, los filtros de peligro y la mascot a configu rados en el estado del provider	Pendiente
F2-009	Verificar que el provider de preferencia de transporte trip_transport_ preference_provider almacena y recupera la preferencia demodo de viaje seleccionada por el usuario	código fuente de providers/trip_transport_preference_provider.dart disponible	1. Instanciar el provider de preferencia de transporte. 2. Verificar que el valor por defecto es walking o elmodo configurado. 3. Cambiar la preferencia a vehicle. 4. Verificar que el estado refleja vehicle. 5. Cambiar de vuelta a walking. 6. Verificar que el estado refleja walking.	Modos de viaje: walking, vehicle	La preferencia se almacena y recupera correct ament e al cambiar entre modos de viaje	Pendiente
F2-010	Verificar que el servicio de filtrado de riesgo de peligros hazard_risk_filt ering determina correctamente cuales niveles de peligro deben evitarse según los filtros seleccionados	código fuente de services/hazard_risk_filtering.dart disponible	1. Revisar el archivo services/hazard_risk_filtering.dart. 2. Probar con filtro que incluyasolo el nivel alto y verificar que los peligros de nivel alto se marcan como evitables. 3. Probar con filtro que incluya bajo, medio y alto. 4. Probar con filtro vacio y verificar que ningun peligro se marca como evitable.	Filtros: soloalto, bajo+medio+alt o, filtro vacio	El servicio identifica correct ament e los peligros evitabl es según los filtros activos	Pendiente
F2-011	Verificar que el servicio de construccion de mensajes de audio de navegación navigation_audio_message_builder genera mensajes coherentes a partir de las instrucciones deruta	código fuente de services/navigation_audio_message_builder.dart disponible	1. Revisar el archivo del servicio. 2. Crear un test con una lista de instrucciones de navegación simuladas que incluyan distancia, direccion y nombre de calle. 3. Invocar el método de construccion de mensajes. 4. Verificar que los mensajes generados son cadenas de texto legibles y coherentes con las instrucciones.	Lista de instrucciones: girar a la derecha en 100 metros, continuar recto200 metros	Los mensaj es generados son cadenas de texto legibles que describ en las instruc ciones de navegación de forma compr ensible	Pendiente
F2-012	Verificar que el servicio de progreso de audio de navegación navigation_audi o_progress rastrea correctamente el avance de reproduccion y el estado de silencio	código fuente de services/navigation_audio_progress.dart disponible	1. Instanciar el servicio de progreso de audio. 2. Simular el inicio de reproduccion de un mensaje de navegación. 3. Verificar que el estado refleja reproduccion activa. 4. Activar el modo silencio. 5. Verificar que el estado refleja silencio activado. 6. Desactivar silencio. 7. Verificar que el estado vuelve a reproduccion activa.	Mensaje de audio simulado, estados de silencio y reproduccion	El progres o se rastrea correct ament e y el estado de silencio se activa y desactiva sin afectar la posicion de reprod uccion	Pendiente
F2-013	Verificar que el provider mapPresentatio nProvider gestiona el foco de cámara, el nivel de zoom y la posicion central delmapa correctamente	código fuente de providers/map_presentation_provider.dart disponible	1. Instanciar el provider mapPresentatio nProvider. 2. Verificar el estado inicial delfoco de cámara. 3. Solicitar un cambio de centro de mapa a coordenadas de prueba. 4. Verificar que el estado refleja las nuevas coordenadas. 5. Solicitar un cambio de nivel de zoom. 6. Verificar que elzoom se actualiza.	coordenadas de centro: latitud 3.4516, longitud -76.5320. Nivel de zoom: 15	El provider actualiz a el centro delmapa y el nivel dezoom correct ament e, reflejando los cambios en el estado	Pendiente
F2-014	Verificar que el modelo PetRecord se serializa y deserializa correctamente desde un mapaJSON con todos los campos incluyendo nombre, fechas, foto y estado	código fuente de models/pet_record.dar t disponible	1. Crear unmapa JSON con todos los campos de una mascota: id, name, photo_path, total_walks, registered_at, adoption_date, birth_date y status. 2. Deserializar elmapa a una instancia de PetRecord. 3. Verificar que todos los campos tienen los valores correctos. 4. Serializar la instancia de vuelta a JSON. 5. Verificar que elmapa resultante contiene los mismos datos.	Mapa JSON con name=Luna QA, birth_date=202 4-01-15, status=normal	El modelo PetRec ord se deserializa con todos los campos correctos y la serializ acion produc e unmapaJSON equivalente al original	Pendiente
F2-015	Verificar que el modelo HazardReportRecord maneja correctamente los campos de coordenadas de incidente, radio de impacto, nivel de riesgo, categoría y duracion estimada	código fuente de models/hazard_report_record.dart disponible	1. Crear unmapa JSON con todos los campos de un reporte de peligro incluyendo incident_latitude, incident_longitu de, impact_radius_ meters, risk_level_code, category_code y estimated_durat ion_code. 2. Deserializar a HazardReportRecord. 3. Verificar cada campo.	Mapa JSON con coordenadas enCali, radio de100 metros, riesgo alto, categoría y duracion	El modelo deserializa correct ament e todos los campos del reporte de peligro incluyendo coorde nadas, radio y clasific acion	Pendiente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F2-016	Verificar que el servicio BackendApiClient construye correctamente las URL de los endpoints de laAPI incluyendo el prefijo base y los headers de autenticación	código fuente de services/backend_api_client.dart disponible	1. Revisar el servicio BackendApiClient. 2. Verificar que las solicitudes incluyen el header Authorization con el formato Bearer seguido del token. 3. Verificar que lasURL se construyen con el prefijo base correcto del backend. 4. Verificar que las solicitudes incluyen el header Content- Type application/json cuando corresponda.	Token de prueba y URLbase del backend	El servicio construye las URL correct ament e con el prefijobase y agrega los headers de autenti cacion y tipo de contenido apropiados	Pendiente
F2-017	Verificar que el provider de datos locales local_data_prov iders gestiona correctamente la inicializacion y la referencia al almacenamiento o de datos del usuario invitado	código fuente de providers/local_data_p roviders.dart disponible	1. Instanciar el provider de datos locales. 2. Verificar que el estado inicial está correctamente configurado. 3. Verificar que el provider permite acceder al servicio GuestDataStore para almacenamiento local de datos sin autenticación.	Ninguno, se usa el estado inicial del provider	El provider se inicializa a correct ament e y permit e acceder al almace namiento de datos del usuario invitad o	Pendiente
F2-018	Verificar que el servicio GuestDataStore permite almacenar y recuperar reportes de peligros, mascotas y rutas de forma local sin necesidad de conexion con el backend	código fuente de services/guest_data_store.dart disponible	1. Instanciar el servicio GuestDataStore. 2. Almacenar un reporte de peligro de prueba. 3. Recuperar el reporte almacenado y verificar que los datos son correctos. 4. Almacenar una mascota de prueba. 5. Recuperar y verificar. 6. Almacenar unaruta de prueba. 7. Recuperar y verificar.	Datos de peligro, mascota y ruta de prueba simulados	Los datos se almacenan y recuperan correct ament e desde el almace namiento local sin depender del backend	Pendiente
F2-019	Verificar que el provider audio_feedback _settings_provider permite configurar las preferencias de audio incluyendo volumen, velocidad y activacion del feedback de navegación	código fuente de providers/audio_feedback_settings_provider.dart disponible	1. Instanciar el provider de configuración de audio. 2. Verificar los valores por defecto de las preferencias de audio. 3. Modificar la configuración de activacion del feedback. 4. Verificar que el cambio se refleja en el estado. 5. Restaurar los valores por defecto.	Preferencias de audio: activar y desactivar feedback de navegación	Las prefere ncias de audio se configuran y persist en correct ament e en el estado del provider	Pendiente
F2-020	Verificar que el provider de reportes de peligros hazard_report_providers gestiona correctamente el listado asincrono de reportes con estados de carga, datos y error	código fuente de providers/hazard_report_providers.dart disponible	1. Instanciar el provider de reportes de peligros. 2. Simular una solicitud de listado con repositorio mockeado que retorna datos exitosamente. 3. Verificar que el estado pasa de carga a datos disponibles. 4. Simular una solicitud que falla. 5. Verificar que el estadopasa a error con mensaje descriptivo.	Repositorio mockeado con respuesta exitosa y respuesta de error	El provider maneja correct ament e los estados de carga, datos exitosos y error, mostrando la inform acion corresp ondiente en cadacaso	Pendiente
F2-021	Verificar que el servicio de visualizacion de peligros en ruta route_hazard_visualizer_service identifica correctamente los peligros cercanos a una polilínea de rutadada	código fuente de services/route_hazard _visualizer_service.dart disponible	1. Crear una polilínea de ruta simulada con coordenadas enCali. 2. Crear una lista de peligros con coordenadas dentro y fuera del rango de proximidad a laruta. 3. Invocar el servicio para identificar los peligros cercanos. 4. Verificar quesolo los peligros dentro del rango de proximidad son identificados.	polilínea deruta con 5 puntos y lista de 3 peligros, 2 cercanos y 1 lejano	El servicio identifica correct ament e los 2 peligros cercanos a laruta y excluye el peligro que está fuera del rango de proximidad	Pendiente
F2-022	Verificar que el modelo RouteRecord se serializa y deserializa correctamente con nombre deruta y lista de puntos de ruta ordenados	código fuente de models/route_record. dart disponible	1. Crear unmapa JSON con los campos de una ruta: id, name y points como lista de objetos con lat, lng y order_index. 2. Deserializar a RouteRecord. 3. Verificar nombre y cantidad de puntos. 4. Verificar que los puntos mantienen el orden correcto.	Mapa JSON con name=Ruta QA, points con 3 puntos ordenados	El modelo RouteR ecord se deseriaLiza con nombr e correct o y los puntos mantienen el orden original	Pendiente
F2-023	Verificar que el provider de mascotas pet_providers gestiona correctamente la creación, actualizacion y eliminacion de mascotas locales con sincronización al repositorio	código fuente de providers/pet_provide rs.dart disponible	1. Instanciar el provider de mascotas con repositorio mockeado. 2. Crear una mascota de prueba. 3. Verificar que aparece en la lista de mascotas del provider. 4. Actualizar el nombre de la mascota. 5. Verificar que la lista refleja el cambio. 6. Eliminar la mascota. 7. Verificar que desaparece de la lista.	Mascota de prueba: nombre LunaQA, estado normal	Las operaci ones de creación, actualiz acion y eliminacion se reflejan correct ament e en el estado del provider	Pendien te
F2-024	Verificar que el servicio OsmRoutingService construye correctamente la solicitud de planificación deruta con puntos, modo de viaje y filtros de peligro para enviar al backend	código fuente de services/osm_routing_ service.dart disponible	1. Revisar el servicio OsmRoutingService. 2. Preparar una solicitud de planificación condos puntos de coordenadas, modo de viaje walking y filtros de peligro activos para niveles alto y medio. 3. Verificar que la solicitud construida contiene todos los campos requeridos por el endpoint /routes/plan del backend.	Puntos de ruta en Cali, modo de viaje walking, filtros de peligro alto y medio	La solicitud contiene los puntos en format o correct o, elmodo de viaje y los filtros de peligro listos para enviar al backend	Pendiente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F2-025	Verificar que el provider theme_provider permite alternar entre tema claro y tema oscuro y que el estado se refleja correctamente	código fuente de providers/theme_provider.dart disponible	1. Instanciar el provider detema. 2. Verificar el tema por defecto. 3. Alternar al tema oscuro. 4. Verificar que el estado reflejatema oscuro. 5. Alternar al tema claro. 6. Verificar que el estado reflejatema claro.	Ninguno, se usa alternancia detema	El provider alterna correct ament e entre temas y el estado se actualiz a en cada cambio	Pendiente

Fase F3 - integración backend API

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F3-001	Verificar que el endpoint GET /api/auth/me retorna los datos del usuario autenticado cuando se envia un token de acceso válido en el header Authorization	Usuario registrado y con token de acceso vigente	1. Obtener un token de acceso válido mediante login. 2. Enviar solicitudGET a /api/auth/me con header Authorization: Bearer {token}. 3. Verificar que la respuesta tiene codigoHTTP 200. 4. Verificar que el campo success estrue. 5. Verificar quedata contiene id, name, email y profile_photo_url del usuario autenticado.	Token de acceso del usuario qa_a4m_01	RespuestaHTTP200 con success true y los datos completos del usuari o autenticado en el campodata	Pendiente
F3-002	Verificar que el endpoint GET /api/auth/me retorna error401 cuando no se incluye el header Authorization en la solicitud	Backend ejecutandose	1. Enviar solicitud GET a /api/auth/me sin ningun header de Authorization. 2. Verificar que la respuesta tiene codigoHTTP 401. 3. Verificar que el cuerpo indica que el usuario no está autenticado.	Solicitud sin header Authorization	RespuestaHTTP401 con mensaj e indican do que la solicitud no está autenti cada	Pendiente
F3-003	Verificar que el endpoint GET /api/auth/me retorna error401 cuando se envia un token de acceso queya fue revocado por un logout previo	Token de acceso que fue revocad o mediante el endpoint de logout	1. Iniciar sesión y obtener un token. 2. Enviar POST a /api/auth/logout conese token para revocarlo. 3. Intentar enviar GET a /api/auth/me con el token revocado. 4. Verificar que la respuesta tiene codigoHTTP 401.	Token revocado por logout	RespuestaHTTP401 indicando que el tokenya no es válido después deser revocado	Pendiente
F3-004	Verificar que el endpoint PUT /api/auth/profil e actualiza el nombre y lafoto de perfil del usuario autenticado y retorna los datos actualizados	Usuario autenticado con token válido	1. Enviar solicitud PUT a /api/auth/profile con header Authorization Bearer y un cuerpoJSON con name y profile_photo_url. 2. Verificar que la respuesta tiene codigoHTTP 200. 3. Verificar que data.user contiene los valores actualizados. 4. Enviar GET a /api/auth/me y confirmar que los datos se actualizaron en labase de datos.	name=QA Tester Actualizado, profile_photo_url=http://example.com/new-foto.jpg	RespuestaHTTP200 con datos actuali zados, y GET/me confirma la persistencia de los cambios	Pendiente
F3-005	Verificar que el endpoint POST /api/pets crea una mascota asociada al usuario autenticado y retorna los datos de la mascota con su identificador generado	Usuario autenticado con token válido	1. Enviar solicitud POST a /api/pets con token de autenticación y cuerpoJSON con name, status, birth_date y adoption_date. 2. Verificar respuesta HTTP201. 3. Verificar que los datos retornados incluyen un id generado y los campos enviados. 4. Verificar con GET /api/pets/{id} que la mascota existe.	name=Luna QA, status=normal, birth_date=2024-01- 15, adoption_date=2024- 05-20	Mascota creada con codigo201, id generado, campos correctos y verificable medianteGET	Pendiente
F3-006	Verificar que el endpoint GET /api/pets retorna el listado paginado de mascotas del usuario autenticado sin incluir mascotas de otros usuarios	Usuario A con mascota propia. UsuarioB con mascota propia. Token de usuario A	1. Crear una mascota con usuario A y otra con usuario B. 2. Enviar GET a /api/pets con token de usuario A. 3. Verificar que la respuesta solo contiene la mascota del usuario A. 4. Verificar la estructura de paginacion en la respuesta.	Token de usuario A	La respuesta incluyesolo las mascotas del usuari o autenticado y la estructura de paginacion es correct a	Pendiente
F3-007	Verificar que el endpoint PUT /api/pets/{pet} permite al propietario actualizar los datos de su mascota y retorna los valores actualizados	Mascota creada por el usuario autenticado	1. Crear una mascota con nombre Luna QA. 2. Enviar PUT a /api/pets/{id} con un nuevo nombre Max QA. 3. Verificar respuestaHTTP 200. 4. Verificar que el nombre retornado es Max QA. 5. Verificar con GET que el cambio persiste.	name=Max QA para actualizar la mascota existente	RespuestaHTTP200 con nombr e actualizado aMaxQA y verificacion de persist encia exitosa	Pendiente
F3-008	Verificar que el endpoint DELETE /api/pets/{pet} elimina una mascota del usuario autenticado y retorna confirmacion de eliminacion exitosa	Mascota creada por el usuario autenticado	1. Crear una mascota de prueba. 2. Enviar DELETE a /api/pets/{id} con token del propietario. 3. Verificar respuesta HTTP 200 con success true. 4. EnviarGET a /api/pets/{id} y verificar respuesta HTTP404.	Identificador de la mascota a eliminar	Mascota eliminada con respuesta200, GET posterior retorn a 404	Pendiente
F3-009	Verificar que el endpoint POST /api/pets/{pet} /mark-found cambia el estado de la mascota a normal y elimina los reportes de mascota pérdida asociados	Mascota con estado pérdida y reporte de mascota pérdida activo	1. Crear una mascota. 2. Crear un reporte de mascota pérdida asociado. 3. Enviar POST a /api/pets/{id}/mark- found con token del propietario. 4. Verificar que la respuesta muestra status=normal. 5. Verificar que GET a /api/lost-pet-reports ya no incluye el reporte de la mascota marcada como encontrada.	Identificador de la mascota pérdida	Estado cambia a normal, report e de perdid a eliminado, respuesta exitosa	Pendiente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F3-010	Verificar que el endpoint POST /api/hazardscrea un reporte de peligro con todos los campos obligatorios y retorna el reporte formateado	Usuario autenticado con token válido	1. Enviar POST a /api/hazards con token y cuerpo JSON con incident_latitude, incident_longitude, impact_radius_meters, risk_level_code, description y reported_at. 2. Verificar respuesta HTTP 201. 3. Verificar que el reporte retornado contiene todos los campos enviados y un id generado.	incident_latitude=3.4516, incident_longitude=-76.5320, impact_radius_meters=150, risk_level_code=medium, description=Zona de riesgo QA	Report e de peligro creado con codigo201, id generado y todos los campos format eados correct ament e	Pendiente
F3-011	Verificar que el endpoint GET /api/hazards retorna el listado paginado de reportes de peligros accesible sin autenticación	Backend ejecutandose con al menos un reporte de peligro creado	1. Enviar GET a /api/hazards sin header de Authorization. 2. Verificar respuesta HTTP200. 3. Verificar que la respuesta contiene un array de reportes de peligros en data. 4. Verificar que la estructura de paginacion incluye total, per_page, current_page y last_page.	Solicitud sin autenticación	RespuestaHTTP200 con lista de report es de peligros accesib le pública mente y paginacion correct a	Pendiente
F3-012	Verificar que el endpoint GET /api/hazards/{id} retorna el detalle de un reporte de peligro incluyendo sus comentarios y relaciones de catalogo	Reporte de peligro existent e con al menos un comentario	1. Crear un reporte de peligro y agregar un comentario. 2. EnviarGET a /api/hazards/{id}. 3. Verificar respuestaHTTP 200. 4. Verificar que data contiene los campos del reporte, los comentarios cargados y las relaciones con riskLevel, category y estimatedDuration.	Identificador de reporte de peligro existente	RespuestaHTTP200 con detalle completo del report e, comen tarios y relaciones de catalog o cargadas	Pendiente
F3-013	Verificar que el endpoint POST /api/hazards/{i d}/comments agrega un comentario al reporte de peligro y retorna el comentario creado con los datos del autor	Usuario autenticado y reporte de peligro existent e	1. Enviar POST a /api/hazards/{id}/comments con token y body con el texto del comentario. 2. Verificar respuesta HTTP 201. 3. Verificar que el comentario retornado tiene el body enviado y los datos del autor.	body=Comentario QA generado para evidencia funcional	Comen tario creado con codigo201, texto correct o y datos del autor incluidos en la respuesta	Pendiente
F3-014	Verificar que el endpoint POST /api/lost-pet- reports crea un reporte de mascota pérdida con todos los campos obligatorios asociado a una mascota existente	Usuario autenticado con mascota registrada	1. Crear una mascota de prueba. 2. Enviar POST a /api/lost-pet-reports con token y cuerpoJSON con pet_id, last_seen_latitude, last_seen_longitude, details, disappearance_date y phone. 3. Verificar respuesta HTTP 201. 4. Verificar que el reporte retornado tiene los datos correctos y la asociacion con la mascota.	pet_id de mascota existente, last_seen_latitude=3.4516, last_seen_longitude=-76.5320, details=Se perdio en el parque, disappearance_date=2 026-05-01, phone=3001234567	Report e creado con codigo201, asociado a la mascot a y con todos los campos correctos	Pendiente
F3-015	Verificar que el endpoint POST /api/found-pet- reports crea un reporte de mascota encontrada con vinculacion opcional a un reporte de mascota pérdida y retorna el reporte creado	Usuario autenticado y reporte de mascota pérdida existent e	1. Crear reporte de mascota pérdida. 2. Enviar POST a /api/found-pet-reports con token, coordenadas, detalles, fecha, teléfono y matched_lost_pet_repo rt_id apuntando al reporte de pérdida. 3. Verificar respuesta HTTP201. 4. Verificar que el campo matched_lost_pet_repo rt_id tiene el valor correcto en la respuesta.	latitude=3.4570, longitude=-76.5280, details=Mascota encontrada cerca delrio, found_date=2026- 05-02, phone=3009876543, matched_lost_pet_rep ort_id del reporte de pérdida	Report e encontrado creado con codigo201 y vinculacion al report e de perdid a correct ament e registrada	Pendiente
F3-016	Verificar que el endpoint POST /api/generic- reports crea un reporte genérico con nivel de riesgo, radio de impacto y coordenadas y retorna el reporte creado	Usuario autenticado con token válido	1. Enviar POST a /api/generic-reports con token y cuerpo JSON con latitude, longitude, description, reported_date, risk_level_code e impact_radius_meters. 2. Verificar respuestaHTTP 201. 3. Verificar que los datos retornados incluyen todos los campos enviados.	latitude=3.4620, longitude=-76.5240, description=Reporte genérico QA de prueba, reported_date=2026- 05-03, risk_level_code=low, impact_radius_meters=50	Report e generic o creado con codigo201 y todos los campos correct ament e almace nados	Pendiente
F3-017	Verificar que el endpoint POST /api/routescrea una ruta de viaje con nombre y puntos de ruta en formato JSON y retorna la ruta creada con su identificador	Usuario autenticado con token válido	1. Enviar POST a /api/routes con token y cuerpo JSON con name y points_json que contenga un array de al menos dos puntos conlat y lng. 2. Verificar respuesta HTTP 201. 3. Verificar que la ruta retornada tiene el nombre correcto y los puntos asociados.	name=Ruta de paseo-QA, points_json con puntos [{lat:3.4516,lng:-76.5320},{lat:3.4570,ln g:-76.5280}]	Ruta creada con codigo201, nombr e correct o, puntos de ruta almace nados en orden	Pendiente
F3-018	Verificar que el endpoint POST /api/routes/plan calcula una ruta entre dos puntos usando GraphHopper y retorna las coordenadas de la polilínea, distancia, duracion e instrucciones de navegación	Usuario autenticado. GraphH opper ejecutandose	1. Enviar POST a /api/routes/plan con token y cuerpo JSON con points (dos puntos con lat y lng), travel_mode=walking. 2. Verificar respuestaHTTP 200. 3. Verificar que data contiene coordinates como array no vacio, distance_meters mayor a cero, duration_seconds mayor a cero y navigation_instructions como array.	points=[{lat:3.4516,lng:- 76.5320},{lat:3.4570,ln g:-76.5280}], travel_mode=walking	RespuestaHTTP200 con coorde nadas de poliline a, distancia y duracion positivas, perfilfoot e instruc ciones de navegación	Pendiente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F3-019	Verificar que el endpoint POST /api/routes/plan retorna errorHTTP 502 con mensaje descriptivo cuando GraphHopper no está disponible	Usuario autenticado. GraphH opper detenid o intencio nalment e	1. Detener el contenedor de GraphHopper. 2. EnviarPOST a /api/routes/plan con puntos válidos. 3. Verificar que la respuesta tiene codigoHTTP 502. 4. Verificar que el campo success es false y que message contiene un texto descriptivo del error. 5. Verificar que el backend sigue respondiendo a /api/health después del error.	points válidos con GraphHopper no disponible	RespuestaHTTP502 con success false y mensaj e descriptivo, backend sigue operativo	Pendiente
F3-020	Verificar que el endpoint GET /api/media/{bu cket}/{fileName } retorna correctamente un archivo de imagen previamente subido con eltipo de contenido apropiado	imagen subida previamente al bucketpets	1. Subir una imagen al bucket pets mediantePOST /api/media/upload. 2. Extraer la URL retornada en data.url. 3. EnviarGET a esa URL. 4. Verificar respuesta HTTP200. 5. Verificar que el header Content-Type es image/png o image/jpeg según la imagen subida. 6. Verificar que el cuerpo de la respuesta contiene datos binarios de imagen.	URL de imagen previamente subida	RespuestaHTTP200 con Conten t-Type correct o y datos binarios de imagen válidos	Pendiente
F3-021	Verificar que el endpoint POST /api/hazards/{i d}/comments retorna errorHTTP 401 cuando se intenta agregar un comentario sin token de autenticación	Reporte de peligro exis- tent e	1. Enviar POST a /api/hazards/{id}/comments sin header Authorization. 2. Verificar respuesta HTTP401. 3. Verificar que no se crea ningun comentario.	body=Comentario sin token, sin header Authorization	RespuestaHTTP401, ningun comen tario creado en labase de datos	Pendiente
F3-022	Verificar que el endpoint DELE- TE /api/hazards/{id} retorna errorHTTP 403 cuando un usuario intenta eliminar un reporte de peligro creado por otro usuario	Reporte de peligro creado por usuario A, token de usuarioB	1. Crear reporte de peligro con usuario A. 2. Enviar DELETE a /api/hazards/{id} con token de usuario B. 3. Verificar respuesta HTTP403. 4. Verificar que el reporte sigue existiendo con GET.	Token de usuario B, identificador de reporte de usuario A	RespuestaHTTP403, report e no eliminado, verificable conGET	Pendiente
F3-023	Verificar que el endpoint POST /api/lost-pet- reports retorna error HTTP 422 cuando se omiten campos obligatorios como pet_id o coordenadas de ultima ubicación conocida	Usuario autenticado con token válido	1. Enviar POST a /api/lost-pet-reports con token pero sin el campo pet_id. 2. Verificar respuesta HTTP422. 3. Verificar que los errores de validación mencionan los campos faltantes. 4. Repetir omitiendo last_seen_latitude y verificar error.	Cuerpo JSON incompleto sin pet_idni coordenadas	RespuestaHTTP422 con errores de validación que indican los campo s obligat orios faltant es	Pendiente
F3-024	Verificar que el endpoint GET /api/media/{bu cket}/{fileName} retorna errorHTTP 404 cuando se solicita un archivo que no existe en el almacenamient o	Backend ejecutandose	1. Enviar GET a /api/media/pets/archiv o_inexistente_12345.png. 2. Verificar respuestaHTTP 404. 3. Verificar que no se produce un error interno del servidor.	Nombre de archivo inexistente: archivo_inexistente_12345.png	RespuestaHTTP404 indicando que el archivo no fue encontrado	Pendiente
F3-025	Verificar que el endpoint POST /api/routes/plan válida correctamente que se envien al menos dos puntos de coordenadas y rechaza solicitudes con un solo punto	Usuario autenticado. GraphH opper ejecutandose	1. Enviar POST a /api/routes/plan con un array de points que contenga solo un punto. 2. Verificar respuestaHTTP 422. 3. Verificar que el error de validación menciona que se requieren al menos 2 puntos.	points con un solo punto: [{lat:3.4516,lng:- 76.5320}]	RespuestaHTTP422 con error de validación indicando que el array de puntos requier e minim o 2 elementos	Pendiente

Fase F4 - Sistema integrado local

Objetivo: Levantar el stack local completo con backend Laravel, frontend Flutter web y GraphHopper funcionando simultaneamente para validar flujos de integración desde la interfaz de usuario, verificando mapa, rutas, reportes, marcadores y la comunicacion entre todos los componentes del sistema.

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F4-001	Verificar que lostres servicios del stack local arrancan simultanea mente sin conflictos de puertos y que cada uno responde a su verificacion de disponibilidad	Docker ejecutandose. De- pende ncias instaladas	1. Ejecutar npm rundev desde la raíz del proyecto. 2. Esperar a que los tres servicios inicien. 3. En- viar GET a http://localhost:800 0/api/health y verificar respuesta con status ok. 4. Abrir http://localhost del frontend en Chrome y veri- ficar que elmapa carga. 5. Verificar con dockerps que el contenedor a4mpets- graphhopper está en estado running.	Comando: npm run dev	Los tres servicios arrancan sin erro- res de puertos, el health check res- pondeok, elmapa carga en el navega- dor y GraphHopper está activo	Pendiente
F4-002	Verificar que un usuario puede registrarse desde la interfaz de Flutterweb y que los datos se per- sisten correctamente en el backend	Stack local completo ejecutandose	1. Abrir la aplicación Flutter web en Chrome. 2. Localizar y abrir el panel de registro de usuario. 3. Completar el formulario con nombre, correo electronico, contraseña y confirmacion de contra- seña. 4. Enviar el formulario. 5. Verificar que la interfaz muestra un mensaje de registro exitoso o navega al estado autenticado. 6. Verificar con GET /api/auth/me que el usuario existe en el backend.	nombre=qa_sistema_01, email=qa_sistema_01@ example.com, pass- word=QAtest1234, confir- macion=QAtest12 34	El registro se completa exitosamente desde la interfaz, el usuario queda autenticad o y los datos son verifica- bles desde laAPI del backend	Pendiente
F4-003	Verificar que un usuario puede iniciar sesión desde la interfaz de Flutterweb usando su correo elec- tronico y que la sesión se mantiene activa para acceder a funciones privadas	Usuario registrad o previa- mente en el sistema	1. Abrir la aplicación Flutter web. 2. Localizar y abrir el panel de inicio de sesión. 3. Ingresar el correo electronico y la contraseña del usuario registrado. 4. Enviar el formulario de login. 5. Ve- rificar que la interfaz cambia al estado autenticado mostrando el nombre del usuario o un indicador visual de sesión activa. 6. Verificar que se pue- de acceder a la seccion de mascotas que requiere autenticación.	email=qa_a4m_01@exa mple.com, pass- word=QAtest1234	El login se completa exitosamente, la interfaz muestra el estado auten- ticad o y las funciones privadas son accesibles	Pendiente
F4-004	Verificar que un usuario autenticad o puede crear una mascota desde el panel de gestion de mascotas de la interfaz y que la mascota aparece en su listado personal	Usuario autenticado en la interfaz	1. Abrir el panel de gestion de mascotas desde la interfaz. 2. Seleccionar la opcion de crear nueva mascota. 3. Completar el formulario con nombre, fecha de nacimiento y foto opcional. 4. Guardar la mascota. 5. Verificar que la mascota aparece en el listado de mascotas del usuario. 6. Verificar con GET /api/pets que la mascota existe en el backend.	nombre=Luna QA, fecha de nacimiento=2024-01- 15	La mascota se crea exitosamente, aparece en el listado de la interfaz y se verifica su existencia en el backend	Pendiente
F4-005	Verificar que elmapa muestra mar- cadores de reportes de peligros existentes y que al tocar un marca- dor se abre el panel de detalle con la informacion correcta del peligro	Al menos un reporte de peligro creado previamen- te con coordenadas dentro del area visible del mapa	1. Abrir la aplicación Flutter web con elmapa visible. 2. Verificar que aparecen marcadores de peligros en las posiciones correspondientes. 3. Tocar o hacer clic en un marcador de peligro. 4. Verificar que se abre el panel de detalle del peligro con descripción, nivel de riesgo, categoría y radio de impacto. 5. Verificar que los datos del panel corresponden al reporte seleccionado.	Reporte de peligro existente con coordenadas 3.4516, -76.5320	El marcador aparece en la posicion correcta del mapa y el panel de detalle muestratoda la informacion del peligro	Pendiente
F4-006	Verificar que un usuario autenti- cad o puede crear un reporte de peligro desde elmapa seleccionan- do manualmente la ubicación del incidente y completando todos los campos del formulario	Usuario autenticado con- mapa visible	1. Abrir el panel de creación de reportes desde la interfaz. 2. Seleccionar el tipo reporte de peligro. 3. Usar la seleccion manual de ubicación para marcar un punto en el mapa. 4. Completar los campos de nivel de riesgo, categoría, radio de impacto y descripción. 5. Enviar el formulario. 6. Verificar que el reporte aparece como marcador en elmapa. 7. Verificar que el reporte existe en el backend conGET /api/hazards.	ubicación manual en elma- pa, risk_level_code=high, impact_radius_meters= 200, description=Peligro reportado desde la interfaz QA	El reporte se crea exitosamente, aparece como marcador en elmapa y es verificable desde laAPI del backend	Pendiente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F4-007	Verificar que las capas de peligros del mapa permiten filtrar la visibilidad de los marcadores por nivel de riesgobajo, medio yalto	Reportes de peligros existentes con diferentes niveles de riesgo	1. Abrir la aplicación con el mapa visible. 2. Abrir el panel de capas de peligros. 3. Desactivar la capa de nivel de riesgo bajo. 4. Verificar que los marcadores de peligros de nivel bajo desaparecen delmapa. 5. Reactivar lacapa de nivel bajo. 6. Desactivar la capa de nivel alto. 7. Verificar que los marcadores de nivel alto desaparecen. 8. Reactivar todas las capas.	Capas de peligro: bajo, medio, alto	Las capas controlan la visibilidad de los marcador es, desactivar una capa oculta los marcador es del nivel correspon diente y reactivarla los muestra de nuevo	Pendiente
F4-008	Verificar que un usuario puede crear un reporte de mascota pérdida desde elmapa seleccionando una mascota propia y la ubicación donde fue vista por ultima vez	Usuario autenticado con al menos una mascota registrad a	1. Abrir el panel de creación de reportes. 2. Seleccionar el tipo reporte de mascota pérdida. 3. Seleccionar la mascota propia de la lista. 4. Seleccionar la ubicación de ultima vez vista en elmapa. 5. Completar los campos de detalles, fecha de desaparicion y teléfono de contacto. 6. Enviar el formulario. 7. Verificar que el reporte aparece como marcador en elmapa.	Mascota: Luna QA, ubicación en el mapa, detalles de desaparicion, fecha=2026-05-01, teléfono=3001234567	El reporte de mascota pérdida se crea exitosamente, aparece en elmapa y la mascota seleccionada queda asociada al reporte	Pendiente
F4-009	Verificar que un usuario puede crear un reporte de mascota encontrad a desde elmapa vinculando lo opcionalmente a un reporte de mascota pérdida existente	Usuario autenticado. Reporte de mascota pérdida existente	1. Abrir el panel de creación de reportes. 2. Seleccionar el tipo reporte de mascota encontrada. 3. Seleccionar la ubicación donde se encontro la mascota. 4. Completar los campos de detalles, fecha y teléfono. 5. Vincular al reporte de mascota pérdida existente si el formulario lo permite. 6. Enviar el formulario. 7. Verificar que el reporte aparece en el mapa.	ubicación en el mapa, detalles de la mascota encontrada, fecha=2026-05-02, teléfono=3009876543, vinculo a reporte de pérdida	El reporte de mascota encontrad a se crea y aparece en elmapa, con la vinculacion al reporte de pérdida sifue seleccionada	Pendiente
F4-010	Verificar que la planificación de ruta con GraphHopper funciona desde la interfaz del mapa mostrando la polilínea del recorrido, la distancia y la duracion estimada	Usuario autenticado. GraphHopper ejecutandose. Ruta guardada con al menosdos puntos	1. Abrir el panel de rutas en la interfaz. 2. Seleccionar unaruta guardada o crear una nueva con al menos dos puntos. 3. Solicitar la planificación de laruta. 4. Verificar que la polilínea de la ruta se dibuja sobre elmapa. 5. Verificar que se muestra la distancia total del recorrido. 6. Verificar que se muestra la duracion estimada del viaje.	Ruta con dos puntos enCali: punto A 3.4516,- 76.5320 y punto B 3.4570,-76.5280	La polilínea se dibuja correctamente sobre elmapa, se muestran la distancia y la duracion del recorrido calculadas por GraphHopper	Pendiente
F4-011	Verificar que un usuario puede crear unaruta de viaje desde el editor de rutas del mapa agregando puntos y guardando la ruta con un nombre descriptivo	Usuario autenticado conmapa visible	1. Abrir el panel de editor de rutas. 2. Agregar un primer punto de ruta seleccionando una ubicación en elmapa. 3. Agregar un segundo punto deruta. 4. Asignar un nombre a la ruta. 5. Guardar la ruta. 6. Verificar que la ruta aparece en el listado de rutas guardadas. 7. Verificar con GET /api/routes que laruta existe en el backend.	Nombre de ruta: Paseo matutino QA. Puntos deruta en el mapa de Cali	La ruta se guarda exitosamente con nombre y puntos, aparece en el listado y se verifica en el backend	Pendiente
F4-012	Verificar que un usuario puede agregar un comentari o a un reporte de peligro desde el panel de detalles del reporte y que el comentari o aparece después de guardarlo	Usuario autenticado. Reporte de peligro existente	1. Abrir el detalle de un reporte de peligro desde elmapa. 2. Localizar la seccion de comentarios. 3. Escribir un texto de comentario. 4. Enviar el comentario. 5. Verificar que el comentario aparece en la lista de comentarios del reporte. 6. Verificar que el autor del comentario se muestra correctamente.	Texto del comentario: Comentario QA generado para evidencia funcional	El comentari o se guarda y aparece en la lista con el texto correcto y el nombre del autor visible	Pendiente
F4-013	Verificar que el usuario puede hacer logout desde la interfaz y que después de cerrar sesión no puede acceder a las funciones privadas del sistema	Usuario autenticado en la interfaz	1. Verificar que el usuario está autenticado con acceso a funciones privadas. 2. Localizar y usar la opcion de cerrar sesión. 3. Verificar que la interfaz vuelve al estado no autenticado. 4. Intentar acceder al panel de mascotas o crear un reporte. 5. Verificar que el sistema solicita iniciar sesión o bloquea la accion.	Ninguno, se usa el estado de sesión activa	El logout se ejecuta correctamente, la interfaz vuelve al estado no autenticad o y las funciones privadas quedan bloqueadas	Pendiente
F4-014	Verificar que el reporte de mascota pérdida aparece como marcador en la capa de mascotas perdidas del mapa y que al tocarlo se muestra el detalle con la foto y datos de la mascota	Reporte de mascota pérdida creado previamente confoto y datos completos	1. Abrir la aplicación con el mapa visible. 2. Verificar que lacapa de mascotas perdidas está activa. 3. Localizar el marcador del reporte de mascota pérdida. 4. Hacer clic en el marcador. 5. Verificar que se abre el panel de detalle con la foto de la mascota, la descripción, la fecha de desaparicion y el teléfono de contacto.	Reporte de mascota pérdida existente confoto y datos	El marcador aparece en la posicion correcta y el panel de detalle muestratoda la informacion del reporte incluyend o foto y datos de contacto	Pendiente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F4-015	Verificar que la seleccion manual de ubicación en el mapa permite al usuario marcar un punto preciso y que las coordenadas seleccionadas se reflejan correctamente en el formulario de reporte	Usuario autenticado con flujo de creación de reporte activo	1. Iniciar la creación de un reporte que requiera seleccion de ubicación. 2. Activar el modo de seleccion manual de ubicación en elmapa. 3. Hacer clic en una ubicación especifica del mapa. 4. Verificar que las coordenadas del punto seleccionado se muestran en el formulario. 5. Verificar que las coordenadas son coherentes con la posicion visual seleccionada en elmapa.	Clic en una ubicación visible del mapa de Cali	Las coordenadas del punto seleccionado se capturan correctamente y se reflejan en el formulario del reporte	Pendiente
F4-016	Verificar que la planificación de ruta con filtros de peligro activos altera el recorrido calculado para evitar las zonas de peligro marcadas en el mapa	GraphHopper ejecutandose. Al menos un reporte de peligro con radio de impacto dentro del recorrido entre dos puntos	1. Crear un reporte de peligro entre losdos puntos de laruta. 2. Planificar laruta sin filtros de peligro y registrar la distancia y el recorrido. 3. Activar los filtros de peligro para el nivel del reporte creado. 4. Replanificar la ruta con los filtros activos. 5. Comparar el recorrido y la distancia con los valores anteriores.	Ruta entre dos puntos con peligro de nivel alto en medio del recorrido	La ruta con filtros de peligro activos difiere de la ruta sin filtros, mostrand o un recorrido alternativ o que evita lazona de peligro o la distancia cambia	Pendiente
F4-017	Verificar que la consola del navegador no muestra errores críticos de JavaScript durante la navegación normal por los flujos principales del sistema	Stack local completo ejecutandose	1. Abrir Chrome DevTools en la pestaÃ±a de consola. 2. Navegar por los flujos principales: login, mapa, mascotas, reportes, rutas y logout. 3. Revisar la consola después de cada flujo. 4. Documentar cualquier error de nivel error que aparezca, excluyendo warnings informativos.	navegación por flujos principales del sistema	La consola del navegador no muestra errores críticos durante la navegación normal, se permiten warnings informativos que no afecten la funcionalidad	Pendiente
F4-018	Verificar que el endpointGET /api/health sigue respondiendo correctam ente después de ejecutar multiples operaciónes de creación de reportes y mascotas desde la interfaz	Stack local ejecutandose con multiples operaciónes realizadas previamente	1. Crear al menos 3 mascotas, 2 reportes de peligros y 1 reporte de mascota pérdida desde la interfaz. 2. EnviarGET a /api/health. 3. Verificar que la respuesta es HTTP 200 con status ok. 4. Verificar que el timestamp es reciente.	Solicitud GET a /api/health después de operaciónes intensivas	El health check respondeHTTP 200 con statusok, confirmando que el backend permanec e estable después de las operación es	Pendiente
F4-019	Verificar que la URL pública de Cloudflare Tunnel permite acceder a la aplicación completa desde un navegador externo con la misma funcionalidad que el acceso local	Cloudflar e Tunnel activo con URL generada	1. Obtener la URL pública de.runtime/cloudflare-url.txt. 2. Abrir laURL en un navegador. 3. Verificar que elmapa carga correctamente. 4. Verificar que el login funciona desde laURL pública. 5. Verificar que la creación de un reporte funciona a traves del tunel.	URL de Cloudflare Tunnel	La aplicación es accesible desde laURL pública con mapa, login y creación de reportes funcionando correctamente	Pendiente
F4-020	Verificar que al crear un reporte genérico desde la interfaz delmapa con radio de impacto, el circulo de impacto se visualiza correctamente alrededor del marcador del reporte	Usuario autenticado conmapa visible	1. Crear un reporte genérico con coordenadas y radio de impacto de 200 metros. 2. Verificar que el marcador aparece en el mapa. 3. Verificar que se dibuja un circulo alrededor del marcador que represente visualmente el radio de impacto. 4. Comparar el tamaño visual del circulo con la escala del mapa.	Reporte genérico con latitud=3.4516, longitud=-76.5320, radio de impacto=200 metros	El marcador y el circulo de impacto se visualizan correctamente en elmapa con un tamaño coherente con el radio especificado	Pendiente
F4-021	Verificar que las instruccion es de navegación se muestran correctamente en la interfaz cuando se planifica un viaje con ruta ymodo de transporte a pie	GraphHopper ejecutandose. Ruta guardada con al menosdos puntos	1. Seleccionar unaruta guardada. 2. Iniciar la planificación de viaje con modo de transporte a pie. 3. Verificar que las instrucciones de navegación se muestran en una lista ordenada. 4. Verificar que cada instruccion incluye la accion a realizar y la distancia. 5. Verificar que las instrucciones son coherentes con el recorrido mostrado en elmapa.	Ruta con dos puntos, modo de transporte: apie	Las instrucciones de navegación se muestran ordenada mente con acciones y distancias coherentes con laruta planificad a	Pendiente
F4-022	Verificar que la subida de imagen funciona correctamente al crear un reporte de peligro confoto de evidencia desde la interfaz y que la imagen se visualiza en el detalle del reporte	Usuario autenticado. Stack local completo	1. Crear un reporte de peligro con una imagen de evidencia adjunta. 2. Verificar que el formulario acepta la imagen y muestra una vista previa. 3. Guardar el reporte. 4. Abrir el detalle del reporte recien creado. 5. Verificar que la imagen de evidencia se visualiza correctamente en el panel de detalle.	imagen PNG válida menor a 1 megabyte	La imagen se sube correctamente, se muestra en la vista previa del formulario y se visualiza en el detalle del reporte guardado	Pendiente

Fase F5 - Caja negra E2E navegador

Objetivo: Ejecutar flujos completos de usuario en el navegador simulando el comportamiento de una persona usuaria real sin depender del conocimiento interno del código, cubriendo registro, gestion de mascotas, reportes de todo tipo, comentarios, rutas, viajes, permisos, responsive y compatibilidad visual.

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F5-001	Verificar el flujo completo de registro de un usuario nuevo desde el formulario de la interfaz hasta la confirmacion visual de sesión activa con acceso a funciones privadas	Stack local completo ejecutando se. No existe el usuario de prueba	1. Abrir la aplicación en Chrome. 2. Navegar al panel de registro. 3. Completar nombre, correo, contraseña y confirmacion. 4. Enviar formulario. 5. Verificar que la interfaz cambia a estado autenticado. 6. Verificar que el nombre del usuario aparece visible en la interfaz. 7. Verificar que el panel de mascotas es accesible.	nombre=qa_e2e_01, email=qa_e2e_01@example.com, password=QAtest1234	El usuario se registra, la interfaz muestra sesión activa, el nombre del usuario es visible y las funciones privadas son accesibles	Pendiente
F5-002	Verificar el flujo completo de inicio de sesión por correo electronico, navegación a mascotas y cierre de sesión con pérdida de acceso a funciones privadas	Usuario registrado previament e	1. Abrir la aplicación. 2. Abrir panel de login. 3. Ingresar correo y contraseña. 4. Enviar formulario. 5. Verificar estado autenticado. 6. Navegar a mascotas. 7. Verificar que el listado de mascotas es accesible. 8. Hacer logout. 9. Verificar estado no autenticado. 10. Intentar acceder a mascotas y verificar bloqueo.	email=qa_a4m_01@example.com, password=QAtest1234	Login exitoso, acceso a mascotas confirmado, logout correcto y funciones privadas bloqueadas después de cerrar sesión	Pendiente
F5-003	Verificar el flujo completo de creación de una mascota nueva con nombre, fecha de nacimiento y foto, incluyendo la visualizacion en el listado personal del usuario	Usuario autenticad o en la interfaz	1. Navegar al panel de mascotas. 2. Seleccionar crear nueva mascota. 3. Ingresar nombre de la mascota. 4. Ingresar fecha de nacimiento. 5. Adjuntar una foto opcional. 6. Guardar la mascota. 7. Verificar que la mascota aparece en el listado. 8. Verificar que el nombre y la foto se muestran correctamente.	nombre=Luna E2E, fecha de nacimiento=2024-01- 15, foto PNG válida	La mascota se crea exitosament e, aparece en el listado con nombre y foto visibles	Pendiente
F5-004	Verificar el flujo completo de edicion de una mascota existente cambiando el nombre y verificando que el cambio se refleja inmediatamente en el listado	Usuario autenticad o con al menos una mascota registrada	1. Navegar al panel de mascotas. 2. Seleccionar la mascota a editar. 3. Cambiar el nombre de la mascota. 4. Guardar los cambios. 5. Verificar que el listado muestra el nuevo nombre. 6. Abrir el detalle de la mascota y verificar que el nombre actualizado se muestra correctamente.	Nuevo nombre: Max E2E Actualizado	El nombre de la mascota se actualiza y se refleja tanto en el listado como en el detalle	Pendiente
F5-005	Verificar el flujo completo de eliminacion de una mascota desde el panel de gestion confirmand o que desaparece del listado personal del usuario	Usuario autenticad o con mascota de prueba	1. Navegar al panel de mascotas. 2. Seleccionar la mascota a eliminar. 3. Confirmar la eliminacion. 4. Verificar que la mascota desaparece del listado. 5. Verificar que la mascota no es accesible desde ningun otro lugar de la interfaz.	Mascota a eliminar identificada por su nombre	La mascota se elimina yya no aparece en el listado ni en ningunaotra seccion de la interfaz	Pendiente
F5-006	Verificar el flujo completo de creación de un reporte de mascota pérdida desde elmapa incluyendo seleccion de mascota, ubicación manual, detalles y visualizacion del marcador resultante	Usuario autenticad o con al menos una mascota registrada	1. Abrir elmapa. 2. Iniciar creación de reporte de mascota pérdida. 3. Seleccionar mascota de la lista. 4. Seleccionar ubicación en el mapa. 5. Completar detalles, fecha y teléfono. 6. Enviar formulario. 7. Verificar que el marcador aparece en el mapa. 8. Abrir el detalle desde el marcador y verificar datos.	Mascota: Luna E2E, ubicación en mapa, detalles=Se perdio en el parque central, fecha=2026-05-01, teléfono=3001234567	El reporte se crea, el marcador aparece en la ubicación correcta y el detalle muestra todos los datos ingresados	Pendiente
F5-007	Verificar el flujo completo de creación de un reporte de mascota encontrada con vinculacion a un reporte de mascota pérdida existente y visualizacion de la tarjeta de resumen	Usuario autenticad o. Reporte de mascota pérdida existente	1. Abrir elmapa. 2. Iniciar creación de reporte de mascota encontrada. 3. Seleccionar ubicación donde se encontro la mascota. 4. Completar detalles, fecha y teléfono. 5. Vincular al reporte de mascota pérdida existente. 6. Enviar formulario. 7. Verificar marcador enmapa. 8. Abrir detalle y verificar que aparece la tarjeta con el resumen del reporte de pérdida vinculado.	ubicación en mapa, detalles=Mascota encontrada cerca del rio, fecha=2026-05-02, vinculo a reporte de pérdida	El reporte se crea con vinculacion, el marcador aparece y el detalle muestra la tarjeta con resumen del reporte de pérdida	Pendiente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F5-008	Verificar el flujo completo de creación de un reporte de peligro desde elmapa con nivel de riesgo alto, radio de impacto yfoto de evidencia, verificando el circulo de impacto en el mapa	Usuario autenticad o con mapa visible	1. Iniciar creación de reporte de peligro. 2. Seleccionar ubicación en el mapa. 3. Seleccionar nivel de riesgo alto. 4. Configurar radio de impacto de150 metros. 5. Agregar descripción. 6. Adjuntar-foto de evidencia. 7. Enviar formulario. 8. Verificar marcador con circulo de impacto en el mapa. 9. Abrir detalle y verificar foto y datos.	ubicación en mapa, nivel=alto, radio=150 metros, descripción=Zona de riesgo E2E, foto PNG	El reporte se crea, el circulo de impacto se visualiza con el radio correcto y lafoto de evidencia se muestra en el detalle	Pendiente
F5-009	Verificar el flujo completo de creación de un reporte genérico desde elmapa con nivel de riesgo, radio de impacto y visualizacion correcta en las capas delmapa	Usuario autenticad o	1. Iniciar creación de reporte genérico. 2. Seleccionar ubicación. 3. Completar descripción y fecha. 4. Seleccionar nivel de riesgo. 5. Configurar radio de impacto. 6. Enviar formulario. 7. Verificar marcador enmapa. 8. Verificar que el reporte aparece en la capa correspondiente.	ubicación en mapa, descripción=Reporte genérico E2E, fecha=2026-05-03, risk_level_code=medium, radio=100 metros	El reporte genérico secrea y se visualiza correctamente en elmapa dentro de lacapa correspondiente	Pendiente
F5-010	Verificar el flujo completo de agregar un comentario a un reporte de peligro y verificar que el comentario aparece inmediatamente en la lista de comentarios del panel	Usuario autenticad o. Reporte de peligro existente	1. Abrir el detalle de un reporte de peligro. 2. Navegar a la seccion de comentarios. 3. Escribir el texto del comentario. 4. Enviar el comentario. 5. Verificar que el comentario aparece en la lista con el texto correcto. 6. Verificar que el nombre del autor es el del usuario autenticado.	Texto: Comentario E2E generado para evidencia funcional	El comentario se pública y aparece en la lista con texto y autor correctos	Pendiente
F5-011	Verificar el flujo completo de creación de una ruta de viaje condos puntos desde el editor de rutas, incluyendo guardar laruta y seleccionarl a para planificación de viaje	Usuario autenticad o. GraphHopper ejecutando se	1. Abrir editor de rutas. 2. Agregar primer punto en elmapa. 3. Agregar segundo punto en elmapa. 4. Asignar nombre a laruta. 5. Guardar laruta. 6. Verificar que aparece en el listado de rutas. 7. Seleccionar la ruta para planificación . 8. Verificar que la polilínea se dibuja en elmapa con distancia y duracion.	Nombre: Ruta E2E de prueba, dos puntos en elmapa de Cali	La ruta se guarda, aparece en el listado, y al planificar se dibuja la polilínea con distancia y duracion	Pendiente
F5-012	Verificar el flujo completo de planificación de viaje conruta seleccionad a, filtros de nivel de peligro activos y visualizacion de instrucciones de navegación	Usuario autenticad o. Ruta guardada. Reportes de peligros existentes. GraphHopper ejecutando se	1. Seleccionarruta para viaje. 2. Activar filtros de nivel de peligro alto y medio. 3. Iniciar planificación. 4. Verificar que la polilínea se dibuja en elmapa. 5. Verificar que las instrucciones de navegación se muestran. 6. Verificar que las alertas de proximidad a peligros aparecen si corresponde. 7. Verificar la distancia y duracion del recorrido.	Ruta guardada, filtros de peligro: alto y medio	El viaje se planifica con la ruta ajustada por filtros, instrucciones visibles, alertas de peligro y metricas de recorrido	Pendiente
F5-013	Verificar que un usuario no autenticado no puede crear reportes, mascotas ni rutas desde la interfaz, y que el sistema muestra un mensaje o redirige al panel de login	aplicación abierta sin sesión activa	1. Abrir la aplicación sin iniciar sesión. 2. Intentar crear un reporte de peligro. 3. Verificar que se muestra un mensaje de autenticación requerida o se redirige al login. 4. Intentar crear una mascota. 5. Verificar el mismo bloqueo. 6. Intentar crear unaruta. 7. Verificar el mismo bloqueo.	Ninguno, usuario sin sesión activa	Todas las acciones de creación son bloqueadas para usuarios no autenticados con mensaje apropiado o redireccion al login	Pendiente
F5-014	Verificar que un usuario no puede modificar ni eliminar mascotas que pertenecen a otro usuario desde la interfaz del sistema	Dos usuarios registrados, cada uno con sus mascotas	1. Iniciar sesión con usuario A. 2. Verificar quesolo aparecen las mascotas del usuario A en el listado. 3. Cerrar sesión. 4. Iniciar sesión con usuarioB. 5. Verificar quesolo aparecen las mascotas del usuario B. 6. Verificar que no hay opcion de editar o eliminar mascotas del usuario A desde la sesión del usuario B.	Mascotas de usuario A y usuario B	Cada usuario solove y puede gestionar sus propias mascotas, sin acceso a las mascotas del otro usuario	Pendiente
F5-015	Verificar que la aplicación se visualiza correctamente en un viewport de escritorio con resolucion 1366x768 sin overflow, texto ilegible ni botones inaccesibles	Stack local ejecutando se	1. Abrir Chrome y configurar el viewport a 1366x768 usando DevTools. 2. Navegar por los flujos principales: mapa, login, mascotas, reportes y rutas. 3. Verificar que no hay overflow horizontal ni vertical en ningun panel. 4. Verificar que todos los textos son legibles. 5. Verificar que todos los botones son accesibles y clicables.	Viewport: 1366x768	La aplicación se visualiza correctamente sin problemas de overflow, legibilidad o accesibilidad de botones en resolucion de escritorio	Pendiente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F5-016	Verificar que la aplicación se visualiza correctamente en un viewport móvil simulado con resolucion 390x844 sin overflow, texto ilegible ni elementos superpuestos	Stack local ejecutando se	1. Abrir Chrome DevTools y activar el simulador de dispositivo móvil con resolucion 390x844. 2. Navegar por los flujos principales. 3. Verificar que los paneles se adaptan al ancho móvil. 4. Verificar que el mapa ocupa el espacio disponible. 5. Verificar que los formularios son utilizables con teclado virtual simulado. 6. Verificar que no hay elementos superpuestos que bloqueen la interaccion.	Viewport: 390x844 simulado	La aplicación se adapta correctamente al viewport móvil sin overflow, superposiciones ni problemas de usabilidad	Pendiente
F5-017	Verificar que la edicion de una ruta existente permite modificar el nombre y los puntos de ruta y que los cambios se reflejan correctamente después de guardar	Usuario autenticad o con ruta guardada	1. Abrir el listado de rutas. 2. Seleccionar la ruta a editar. 3. Cambiar el nombre de la ruta. 4. Modificar la posicion de un punto deruta. 5. Guardar los cambios. 6. Verificar que el listado muestra el nuevo nombre. 7. Seleccionar la ruta y verificar que los puntos actualizados se muestran correctamente en elmapa.	Nuevo nombre: Ruta E2E actualizada, punto de ruta modificado	La ruta se actualiza con nombre y puntos nuevos, verificables en el listado y en elmapa	Pendiente
F5-018	Verificar que la eliminacion de una ruta desde el panel de gestion de rutas remueve laruta del listado y que ya no puede ser seleccionad a para planificación	Usuario autenticad o con ruta guardada de prueba	1. Abrir el listado de rutas. 2. Seleccionar la ruta a eliminar. 3. Confirmar la eliminacion. 4. Verificar que la ruta desaparece del listado. 5. Verificar que no aparece como opcion en la seleccion de rutas para viaje.	Ruta de prueba identificada por nombre	La ruta se elimina del listado y no aparece como opcion en ningun selector de rutas del sistema	Pendiente
F5-019	Verificar que al marcar una mascota como encontrada desde la interfaz, el estado de la mascota cambia a normal y los reportes de mascota pérdida asociados desaparecen del mapa	Usuario autenticad o con mascota en estado pérdida y reporte de pérdida activo	1. Navegar al panel de mascotas. 2. Localizar la mascota en estado pérdida. 3. Usar la opcion de marcar como encontrada. 4. Verificar que el estado de la mascota cambia a normal en la interfaz. 5. Verificar que el marcador del reporte de mascota pérdida ya no aparece en el mapa.	Mascota en estado pérdida con reporte activo	El estado cambia a normal, el reporte de pérdida desaparece del mapa y la mascota se muestra con estado normal en el listado	Pendiente
F5-020	Verificar que los datos ingresados con texto de longitud extendida de al menos500 caracteres se manejan correctame nte en los formularios de reporte sin romper la interfaz ni truncar la informacion	Usuario autenticad o	1. Crear un reporte de peligro con una descripción de al menos500 caracteres. 2. Verificar que el formulario acepta el texto completo sin errores. 3. Guardar el reporte. 4. Abrir el detalle del reporte. 5. Verificar que la descripción completa se muestra sin truncamient o. 6. Verificar que el layout del panel de detalle no se rompe con el texto largo.	Texto de descripción de500 caracteres: cadena repetida de texto QA para prueba de longitud	El texto de500 caracteres se guarda y se muestra completamente en el detalle sin romper el layout de la interfaz	Pendiente
F5-021	Verificar que el flujo de agregar comentarios funciona correctamente en los cuatro tipos de reportes: peligros, mascotas perdidas, mascotas encontradas y reportes genéricos	Usuario autenticad o. Un reporte de cada tipo existente	1. Abrir detalle de un reporte de peligro y agregar un comentario. 2. Verificar que el comentario aparece. 3. Abrir detalle de un reporte de mascota pérdida y agregar un comentario. 4. Verificar. 5. Repetir para reporte de mascota encontrada. 6. Repetir para reporte genérico.	Texto de comentario para cada tipo de reporte	Los comentarios se agregan y se muestran correctamente en los cuatro tipos de reportes	Pendiente
F5-022	Verificar que la consola del navegador no muestra errores críticos de JavaScript ni errores dered HTTP500 durante la ejecución completa de los flujosE2E de las pruebas anteriores	Stack local ejecutando se con DevTools abierto	1. Mantener la consola de DevTools abierta durante la ejecución de los flujosE2E. 2. Al finalizar todos los flujos, revisar la consola buscando errores detipo error. 3. Revisar la pestaÃ±a Network buscando respuestasHTTP 500. 4. Documentar cualquier error encontrado con contexto y captura.	Revision de consola y network durante todos los flujos E2E	No se encuentran errores críticos de JavaScript ni respuestasHTTP 500 durante la ejecución de los flujos	Pendiente

Fase F6 - Pruebas manuales funcionales

Objetivo: Revisar manualmente mensajes de la interfaz, comportamiento ante errores, datos límite, viewport móvil, permisos de ubicación, estados vacios y recuperación visual para encontrar problemas de usabilidad y flujos no cubiertos por la automatizacion.

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F6-001	Verificar que los mensajes de error de validación del formulario de registro son claros, descriptivos y se muestran junto al campo correspondiente cuando se envian datos invalidos	Stack local ejecutandose	1. Abrir el panel de registro. 2. Dejar todos los campos vacios y enviar el formulario. 3. Verificar que cada campo obligatorio muestra un mensaje de error claro. 4. Ingresar un correo con formato invalido y verificar el mensaje. 5. Ingresar una contraseña menor a 8 caracteres y verificar el mensaje. 6. Ingresar contraseñas que no coinciden y verificar el mensaje.	Campos vacios, correo invalido, contraseña corta, contraseñas no coincidentes	Cada campo muestra un mensaje de error descriptivo y visible junto al campo correspondiente, sin mensajes genéricos ni tecnicismos	Pendiente
F6-002	Verificar que los mensajes de error del formulario de login son claros cuando se ingresan credenciales incorrectas sin revelar informacion sobre la existencia del usuario	Stack local ejecutandose	1. Abrir el panel de login. 2. Ingresar un correo que no existe en el sistema. 3. Enviar el formulario y verificar el mensaje de error. 4. Ingresar un correo existente con contraseña incorrecta. 5. Enviar y verificar que el mensaje de error no diferencia entre usuario inexistente y contraseña incorrecta.	Correo inexistente, correo existente con contraseña incorrecta	Los mensajes de error no revelan si el correo existe o no en el sistema, mostrando un mensaje genérico de credenciales invalidas	Pendiente
F6-003	Verificar que el estado vacio de la lista de mascotas muestra un mensaje orientativo cuando el usuario no tiene mascotas registradas indicando como crear una	Usuario autenticado sin mascotas registradas	1. Iniciar sesión con un usuario que no tenga mascotas. 2. Navegar al panel de mascotas. 3. Verificar que se muestra un mensaje indicando que nohay mascotas registradas. 4. Verificar que se ofrece una opcion o indicacion para crear la primera mascota.	Ninguno, usuario sin mascotas	Se muestra un mensaje orientativo en el estado vacio con indicacion de como crear la primera mascota	Pendiente
F6-004	Verificar que el estado vacio de la lista de rutas muestra un mensaje adecuado cuando el usuario no tiene rutas guardadas	Usuario autenticado sin rutas guardadas	1. Iniciar sesión con un usuario sin rutas. 2. Navegar al panel de rutas. 3. Verificar que se muestra un mensaje indicando que nohay rutas guardadas. 4. Verificar que se sugiere como crear una ruta.	Ninguno, usuario sin rutas	Se muestra un mensaje adecuado en el estado vacio de rutas con orientacion para el usuario	Pendiente
F6-005	Verificar que los formularios de creación de reportes muestran estados de carga visibles durante el envio de datos al backend para que el usuario sepa que la operación está en proceso	Usuario autenticado	1. Iniciar la creación de un reporte de peligro. 2. Completar todos los campos. 3. Enviar el formulario. 4. Observar si se muestra un indicador de carga durante el envio. 5. Verificar que el boton de envio se deshabilita o muestra estado de procesamiento para evitar envios duplicados.	Datos completos de un reporte de peligro	Se muestra un indicador visual de carga durante el envio y el boton queda deshabilitado para evitar duplicados	Pendiente
F6-006	Verificar que la aplicación maneja correctamente la denegacion de permisos de ubicación del navegador mostrando un mensaje informativo sin bloquear la funcionalidad del mapa	Stack local ejecutandose. permisos de ubicación del navegador configurables	1. Abrir la aplicación en Chrome. 2. Cuando el navegador solicite permisos de ubicación, denegar el acceso. 3. Verificar que elmapa carga correctamente a pesar de la denegacion. 4. Verificar que se muestra un mensaje informativo sobre la falta de acceso a la ubicación. 5. Verificar que la seleccion manual de ubicación sigue funcionando.	Permiso de ubicación: denegado	El mapa carga correctamente sin acceso a ubicación, se muestra mensaje informativo y la seleccion manual sigue operativa	Pendiente
F6-007	Verificar que la aplicación maneja correctamente la aceptacion de permisos de ubicación centrando elmapa en la posición del usuario	Stack local ejecutandose. permisos de ubicación del navegador aceptados	1. Abrir la aplicación en Chrome. 2. Aceptar los permisos de ubicación cuando el navegador los solicite. 3. Verificar que elmapa centra la cámara en la posicion del usuario o en una ubicación cercana. 4. Verificar que la posicion del usuario se muestra como un marcador distinguible.	Permiso de ubicación: aceptado	El mapa centra la cámara en la posicion del usuario y muestra un marcador de posicion actual	Pendiente
F6-008	Verificar que los campos de texto de los formularios de reportes aceptan y muestran correctamente caracteres especiales, acentos, la letra enie y emojis sin corrupcion de datos	Usuario autenticado	1. Crear un reporte de peligro con una descripción que incluya acentos, la letra enie, signos de puntuacion especiales y emojis. 2. Guardar el reporte. 3. Abrir el detalle del reporte. 4. Verificar que todos los caracteres especiales se muestran correctamente sin corrupcion, caracteres de reemplazo ni codificacion rota.	descripción: Peligro en la caneria, atencion alnino. Hay danos y riesgo alto	Los caracteres especiales, acentos, enie y emojis se guardan y se muestran correctamente sin corrupcion	Pendiente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F6-009	Verificar que el panel de comentarios de un reporte muestra el autor correcto de cada comentario con su nombre de usuario visible	Reporte con comentarios de al menosdos usuarios diferentes	1. Crear comentarios en un reporte desdedos sesiones de usuarios diferentes. 2. Abrir el detalle del reporte con los comentarios. 3. Verificar que cada comentario muestra el nombre del autor correspondiente. 4. Verificar que los comentarios están ordenados cronologicament e.	Comentarios de usuario qa_a4m_01 y qa_a4m_02	Cada comentario muestra el nombre del autor correcto y están ordenados por fecha	Pendient e
F6-010	Verificar que el formulario de creación de mascota válida localmente que el nombre no este vacio antes de permitir el envio y muestra un mensaje de error visible	Usuario autenticado en el panel de mascotas	1. Abrir el formulario de creación de mascota. 2. Dejar el campo nombre vacio. 3. Intentar guardar la mascota. 4. Verificar que se muestra un mensaje de error indicando que el nombre es obligatorio. 5. Verificar que el formulario no se envia al backend con el campo vacio.	Campo nombre vacio	El formulario muestra un mensaje de error para el campo nombre vacio y no permite el envio sin datos obligatorios	Pendient e
F6-011	Verificar que los paneles de la aplicación se pueden abrir y cerrar correctamente sin dejar residuos visuales ni estados inconsistentes en la interfaz	Stack local ejecutandose con usuario autenticado	1. Abrir el panel de mascotas y cerrarlo. 2. Verificar que no queda ningun elemento visual del panel en la pantalla. 3. Abrir el panel de rutas y cerrarlo. 4. Abrir el panel de creación de reportes y cerrarlo. 5. Abrir el panel de capas de peligro y cerrarlo. 6. Verificar que elmapa se muestra correctamente después de cada cierre.	Apertura y cierre secuencial de todos los paneles principales	Cada panel seabre y cierra correctamente sin residuos visuales ni estados inconsistentes	Pendient e
F6-012	Verificar que los circulos de impacto de los reportes de peligros de diferentes niveles de riesgo se diferencian visualmente en el mapa mediante colores o estilos distintos	Reportes de peligros existentes con niveles de riesgobajo, medio y alto	1. Crear reportes de peligro con nivel bajo, medio y alto. 2. Verificar que los circulos de impacto en elmapa tienen colores o estilos visualmente distintos para cada nivel. 3. Verificar que la leyenda o el panel de capas permite identificar el significado de cada color.	Reportes con nivel bajo, medio y alto	Los circulos de impacto se diferencian visualmente por nivel de riesgo con colores o estilos distinguibles	Pendient e
F6-013	Verificar que el campo de teléfono en los formularios de reportes de mascotas perdidas y encontradas acepta formatos de teléfono colombiano y no permite texto no numerico	Usuario autenticado creando un reporte de mascota pérdida	1. Abrir el formulario de reporte de mascota pérdida. 2. Ingresar un numero de teléfono colombiano válido en el campo de teléfono. 3. Verificar que el campo acepta el formato. 4. Intentar ingresar texto no numerico como letras y verificar el comportamiento del campo.	teléfono válido: 3001234567. Texto invalido: abc	El campo acepta numeros de teléfono válidos y rechaza o no permite la entrada de texto no numerico	Pendient e
F6-014	Verificar que la aplicación muestra correctamente las fechas en los reportes y las mascotas con un formato legible y coherente para el usuario hispanohablante	Reportes y mascotas existentes con fechas variadas	1. Revisar las fechas mostradas en el detalle de una mascota. 2. Revisar las fechas en el detalle de un reporte de peligro. 3. Revisar las fechas en los reportes de mascotas perdidas y encontradas. 4. Verificar que el formato de fecha es consistente entoda la aplicación. 5. Verificar que las fechas son legibles y no se muestran en formato tecnico como timestamps crudos.	Fechas de mascotas y reportes existentes	Las fechas se muestran en un formato legible y coherente entoda la aplicación	Pendient e
F6-015	Verificar que el panel de detalle de un reporte de peligro muestra correctamente todos los campos incluyendo descripción, nivel de riesgo, categoría, duracion estimada, radio de impacto, foto y ubicación	Reporte de peligro completo con todos los campos opcionales completados	1. Abrir el detalle de un reporte de peligro con todos los campos completados. 2. Verificar que la descripción se muestra completa. 3. Verificar que el nivel de riesgo es legible. 4. Verificar categoría y duracion estimada. 5. Verificar el radio de impacto. 6. Verificar que lafoto de evidencia se carga y se muestra. 7. Verificar las coordenadas o la representacion de ubicación.	Reporte de peligro con todos los campos	Todos los campos del reporte se muestran correctamente en el panel de detalle sin informacion faltante	Pendient e
F6-016	Verificar que la aplicación recupera su estado visual correctamente después de un error de red simulado al intentar guardar un reporte, sin dejar formularios bloqueados o pantallas rotas	Usuario autenticado con formulario de reporte completado	1. Completar un formulario de reporte de peligro. 2. Desconectar lared o detener el backend momentaneamente antes de enviar. 3. Intentar enviar el formulario. 4. Verificar que se muestra un mensaje de error de conexion. 5. Reconectar la red o reiniciar el backend. 6. Verificar que el formulario sigue disponible con los datos ingresados o que se puede reintentar la operación.	Desconexion de red durante el envio	Se muestra un mensaje de error claro, el formulario no queda bloqueado y es posible reintentar o corregir la operación	Pendient e

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F6-017	Verificar que el mapa mantiene la posicion de la cámara y el nivel de zoom cuando seabre y cierra un panel lateral sin resetear la vista del usuario	Mapa visible con posicion y zoom personalizados	1. Navegar elmapa hasta una posicion y nivel de zoom especificos. 2. Abrir un panel lateral como el de mascotas o capas. 3. Cerrar el panel. 4. Verificar que elmapa mantiene la misma posicion de cámara y nivel dezoom que tenia antes de abrir el panel.	Posicion yzoom personalizados en elmapa	El mapa mantiene su posicion yzoom después de abrir y cerrar paneles laterales	Pendiente
F6-018	Verificar que los botones de accion principales de la interfaz son accesibles mediante navegación por teclado y que el orden de tabulacion es logico y predecible	Stack local ejecutandose	1. Abrir la aplicación en Chrome. 2. Navegar usandosolo la tecla Tab para recorrer los elementos interactivos. 3. Verificar que los botones principales reciben foco visible. 4. Verificar que el orden de tabulacion sigue una secuencia logica de arriba hacia abajo y de izquierda a derecha. 5. Verificar que los elementos focalizados tienen indicador visual de foco.	navegación por teclado con tecla Tab	Los botones principales son accesibles por teclado, el orden de tabulacion es logico y el foco visual es distinguible	Pendiente
F6-019	Verificar que los textos de la interfaz son legibles con suficiente contraste entre el color del texto y el fondo en los paneles principales del sistema	Stack local ejecutandose	1. Revisar los paneles principales de la aplicación: login, registro, mascotas, reportes, rutas y detalles. 2. Verificar que el texto tiene suficiente contraste con el fondo para ser legible. 3. Verificar enmodo claro y enmodo oscuro si la aplicación soporta ambos temas. 4. Identificar textos con contraste insuficiente y documentarlos.	Revision visual de todos los paneles principales	Los textos son legibles con contraste suficiente en todos los paneles principales	Pendiente
F6-020	Verificar que al intentar subir un archivo que no es una imagen válida como un archivo de texto o un PDF el sistema muestra un mensaje de error claro sin romper la interfaz	Usuario autenticado creando un reporte confoto	1. Iniciar la creación de un reporte que permita adjuntarfoto. 2. Intentar seleccionar un archivo de texto o PDF en lugar de una imagen. 3. Verificar que el sistema muestra un mensaje indicando que el formato no es válido. 4. Verificar que el formulario sigue funcional y no se rompe. 5. Seleccionar una imagen válida y verificar que se acepta correctamente.	Archivo de texto o PDF no válido como imagen	El sistema rechaza el archivo no válido con mensaje claro, el formulario sigue funcional y una imagen válida posterior es aceptada	Pendiente
F6-021	Verificar que la aplicación muestra correctamente el estado de carga mientras se obtienen los datos del listado de reportes de peligros desde el backend	Stack local ejecutandose	1. Abrir la aplicación y navegar al listado de reportes de peligros. 2. Observar si se muestra un indicador de carga mientras se obtienen los datos. 3. Verificar que los datos aparecen después de la carga. 4. Si la carga es instantanea, simular latencia desde DevTools y verificar el comportamiento del indicador de carga.	Ninguno, observación del estado de carga	Se muestra un indicador de carga visible durante la obtencion de datos y los datos aparecen correctamente después	Pendiente
F6-022	Verificar que el panel de perfil del usuario autenticado muestra correctamente el nombre, el correo electronico y la foto de perfil cuando está disponible	Usuario autenticado con perfil completado y foto de perfil subida	1. Iniciar sesión con un usuario que tenga foto de perfil. 2. Abrir el panel de perfil o menu del usuario. 3. Verificar que el nombre del usuario se muestra correctamente. 4. Verificar que el correo electronico se muestra. 5. Verificar que lafoto de perfil se carga y se visualiza.	Usuario con nombre, correo y foto de perfil	El panel de perfil muestra nombre, correo y foto correctamente sin datos faltantes ni imágenes rotas	Pendiente

Fase F7 - Performance API con Postman Performance Testing y

Objetivo: Ejecutar escenarios no funcionales de performance contra el backend expuesto mediante Cloudflare Tunnel usando Postman Desktop Performance Testing. La fase debe medir disponibilidad, latencia promedio, p90, p95, p99, throughput, requests por segundo, porcentaje de error, assertions fallidas bajo usuarios virtuales y ausencia de HTTP 5xx en condiciones de carga controlada. Entorno de ejecución usado como referencia para interpretar resultados: Ubuntu 24.04.4 LTS, kernel Linux 6.8.0-111-generic x86_64, laptop Toshiba Satellite C55-B, CPU Intel Core i3-4005U de 2 nucleos / 4 hilos hasta 1.70 GHz, 6 GiB RAM, disco local 465.76 GiB, red Wi-Fi Qualcomm Atheros AR9485, Docker/GraphHopper local, Postman Desktop y Cloudflare Tunnel. Estos resultados representan performance local controlada y no equivalen a infraestructura productiva.

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F7-001	Medir baseline de disponibi lidad del endpointGET /api/health sin carga concurre nte	Postman Desktop configu- rad o, Cloudflare Tunnel activo, stack local disponi- ble	1. Ejecutar la carpeta F7- 001/F7-002 - Baseline en la pestana Performance. 2. Usar perfil fixed con 1 VU durante 1 a 2 minutos. 3. Registraravg, p90, p95, p99, RPS, error % y 5xx. 4. Verificar assertions de status ok.	GET /api/health, perfil fixed, 1 VU	HTTP 200 sostenido, status=ok, p95 <= 500 ms, error <= 1 % y 0 respuestas5xx	Aprobado. 352 requests, 5.81 RPS, avg 100 ms, p95 146 ms, p99 326 ms, error0.00 %, failure0.00 %.
F7-002	Medir baseline autenticado de login y emisión de token sin carga concurrente	Usuario QA registrado. Cloudflare activo	1. Ejecutar login desde la carpeta Baseline con perfil fixed de1 VU. 2. ConfirmarHTTP 200 y ac- cess_token en cada ciclo. 3. Registrar percentiles y error %.	POST /api/auth/login con credenciales QA	Login responde con token válido, p95 <= 1500 ms, error <= 1 % y 0 respuestas5xx	Aprobado. 74 requests, 1.19 RPS, avg 739 ms, p95 834 ms, p99 1201 ms, error0.00 %, failure0.00 %.
F7-003	Evaluar carga fija sobre endpoints publicos de lectura	Datos de reportes dis- ponibles o ambiente con datosQA. Cloudflare acti- vo	1. Ejecutar carpeta FixedRead Load. 2. Usar perfil fixed con 5 a10 VU durante 3 a 5 minutos. 3. Incluir health, hazards, lost- pet-reports, found- pet- reports y generic- reports. 4. Revisar Testsvs VU y percentiles.	Perfil fixed, 5-10 VU, end- points publicos GET	Lecturas publicas mantienenp95 <= 1500 ms, error <= 1 % y 0 respuestas5xx	Aprobado. Hazards, lost, found y generic entre1699 y 1703 requests por endpoint; p95 max 154 ms, p99 max 248 ms, error0.00 %, failure0.00 %.

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F7-004	Evaluar carga fija sobre endpoints autenticados de lectura	Setup ejecutado, access_tok en disponible, rutas/pets del usuarioQA existentes o vacias	1. Ejecutar carpeta de lecturas autenticadas con Authorization Bearer. 2. Usar perfil fixed con 5 a10 VU durante 3 a 5 minutos. 3. Verificar que /api/routes y /api/pets responden para el usuario autenticado.	Token QA, GET /api/routes y GET /api/pets	Endpoints autenticados responden200, p95 <= 1500 ms, error <= 1 % y0 respuestas5xx	Aprobado con observaciones. Routes: 2706 requests, p95 255 ms, failure0.00 %. Pets: 2703 requests, p95 252 ms, failure 0.04 %, max aislado8781 ms.
F7-005	Evaluar carga fija de creación de mascotas con datos unicos	Setup ejecutado, token válido, datafile o variables dinamicas disponibles	1. Ejecutar escenario Fixed WriteLoad. 2. Usar perfil fixed con 3 a 5 VU durante 3 a 5 minutos. 3. Crear mascotas con nombre único por VU/iteracion. 4. ConfirmarHTTP 201 eid.	POST /api/pets con nombre MascotaPerf_{{unique_suffix}}	Creaciones exitosas, p95 <= 3500 ms, error <= 2 % y0 respuestas5xx	Aprobado. 930 requests, 5.13 RPS, avg 514 ms, p95 890 ms, p99 1190 ms, error0.00 %, failure0.00 %.
F7-006	Evaluar carga controlada de creación de reportes de peligro	Setup ejecutado, token válido, coordenadas variables	1. EjecutarPOST /api/hazards dentro de Fixed WriteLoad. 2. Usar coordenadas variables y risk_level_co de=low. 3. Registrar percentiles y fallos por assertion.	POST /api/hazards con coordenadas deCali	Reportes creados conHTTP 201, p95 <= 3500 ms, error <= 2 % y 0 respuestas5xx	Aprobado. 760 requests, 4.20 RPS, avg 640 ms, p95 1221 ms, p99 1601 ms, error0.00 %, failure0.00 %.
F7-007	Evaluar carga controlada de media upload	imagen base64 válida preparada, Cloudflare activo	1. EjecutarPOST /api/media/u pload en Fixed WriteLoad. 2. Usar perfil fixed con 3 a 5 VU. 3. Confirmar que cada respuesta devuelveURL.	bucket=pets, image_data PNG base64	Upload responde conURL válida, p95 <= 3500 ms, error <= 2 % y 0 respuestas5xx	Aprobado. 1508 requests, 8.32 RPS, avg305 ms, p95 700 ms, p99 1383 ms, error0.00 %, failure0.00 %.

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F7-008	Medir performance de planeación deruta simple con GraphHopper	GraphHopper activo, token QA válido, puntos compactos dentro deCali	1. Ejecutar escenario GraphHopper Performance. 2. Enviar ruta de dos puntos enmodo car. 3. Registrar percentiles, timeouts y errores GraphHopper.	POST /api/routes/plan con 2 puntos	Ruta contiene coordenadas, distancia y duracion; p95 <= 15000 ms, error <= 5 %, sin timeouts sistematicos	Aprobado. 1693 requests, 9.34 RPS, avg264 ms, p95 403 ms, p99 1625 ms, error0.00 %, failure0.00 %.
F7-009	Comparar performance deruta simple contraruta multipunto	GraphHopper activo, datos deruta simple y 5 puntos compactos disponibles	1. Ejecutar requests deruta simple y multipunto en el mismo escenario. 2. Compararavg, p95 yp99 por request. 3. Registrar diferencia porcentual.	Rutas de 2 puntos y5 puntos	Se documenta diferencia de latencia y ambas variantes cumplen p95 <= 15000 ms sin 5xx	Aprobado. Multipunto: 1621 requests, 8.90 RPS, avg276 ms, p95 476 ms, p99 710 ms, error0.00 %, failure0.00 %.
F7-010	Comparar planeación sin filtros contra planeación con filtros de riesgo	GraphHopper activo, reportes de peligro existentes	1. Ejecutar rutas sin filtros y con avoid_hazard s=true. 2. Usar hazard_risk_f ilters high y medium. 3. Comparar percentiles y errores.	Route plan sin filtros y con filtros high/medium	Filtros no generan timeout sistematico, error <= 5 % y0 respuestas5xx	Aprobado. 1580 requests, 8.74 RPS, avg284 ms, p95 506 ms, p99 856 ms, error0.00 %, failure0.00 %.
F7-011	Evaluar degradacion progresiva con escenari o mixto ramp-up	Setup ejecutado, Cloudflare activo	1. Ejecutar carpeta Ramp-Up Mixed API. 2. Usar perfil ramp-up con máximo 10 VU durante 5 minutos. 3. Revisar en que nivel de VU fallan assertions si ocurre.	Perfil ramp-up, max10 VU	La API mantiene assertions verdes o fallos <= umbrales, sin degradacion abrupta ni 5xx	Aprobado con observaciones. Ramp-up 3 a10 VU por 3m; health/hazards/routes tuvieron failure 2.65 %, 0.94 % y0.31 %, con error HTTP 0.00 % y sin5xx reportados.

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F7-012	Evaluar recuperación ante spike de lecturas	Cloudflare activo, endpoints publicos disponibles	1. Ejecutar carpeta SpikeRead Load. 2. Usar perfil spike con máximo 20 VU durante 3 minutos. 3. Ejecutar health posterior al spike.	Perfil spike, max 20 VU, endpoints GET	El backend nocae, health posterior responde 200 y no se registran 5xx	Aprobado con observaciones. Spike max 20 VU por 3m; hazards failure3.54 % y health failure 3.59 %, error HTTP0.00 %, healthp95 481 ms.
F7-013	Evaluarpico sostenid o corto de flujo mixto autenticado	Setup ejecutado, token válido	1. Ejecutar carpeta Peak Authenticated Mix. 2. Usar perfil peak con maximo10 VU durante 5 minutos. 3. Registrar RPS, p95, p99 y error %.	Perfil peak, endpoints autenticados y publicos	Flujo mixto se mantiene dentro de umbrales por endpoint y sin5xx	Aprobado con observaciones. Pets/routes tuvieron failure 0.81 % con p95 1023/1087 ms; comentarios tuvieron failure 0.00 %, p95 1115 ms; error HTTP0.00 %.
F7-014	Ejecutar endurance corta para detectar degradacion por sesión prolongada	Stack estable, Cloudflare activo, Postman Desktop disponible	1. Ejecutar carpeta Short Endurance. 2. Usar perfil fixed con 5 VU durante15 a 30 minutos. 3. Observar tendencia dep95, error %, recursos delhost y estabilidad del tunel.	Perfil fixed, 5 VU, 15-30 min	No hay aumento sostenido de errores, nohay 5xx y Cloudflare permanece operativo o el cambio queda documentado	Aprobado. Endurance15m; health 17390 requests y hazards 17387 requests, 19.30 RPS por endpoint, p95 163/185 ms, error 0.00 %, failure 0.00 %.

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F7-015	Verificar seguridad bajo carga en endpoint s protegidos sin token	Cloudflare activo, token omitido intencional mente	1. Ejecutar carpeta Security Under Load. 2. Usar perfil fixed con 5 VU durante 3 minutos. 3. Enviar requests protegidos sin Authorization.	POST /api/pets yGET /api/routes sin token	100 % respuestasHTTP 401 esperadas, 0 respuestas 5xx	Fallido. POST /api/pets sin token yPOST /api/routes/plan sin token tuvieron 100.00 % failure de assertion; GET /api/routes sin tokentuvo 0.00 % failure. Sin error HTTP ni5xx reportados.
F7-016	Verificar recuperación posterior a spike opeak	Haber ejecutado F7-012 o F7-013	1. Esperar 10 segundos tras el escenario de mayor carga. 2. Enviar GET /api/health. 3. Enviar lectura pública y autenticada. 4. Registrar latencias post-carga.	Health, hazards y routes posterior a carga	El backend responde normalmente después de la carga, sin errores persistentes	Aprobado con observaciones. Recovery con6699 requests, 37.15 RPS, avg93 ms, p95 205 ms, p99 293 ms, error0.00 %, failure0.21 %.
F7-017	Verificar estabilidad de Cloudflar e Tunnel durante la sesionF7	URL Cloudflare registrada al inicio	1. RegistrarURL inicial. 2. Ejecutar escenariosF7. 3. Validar la misma URL al cierre con /api/health. 4. Documentar cambios si loshay.	URL trycloudflare.com	La URL permanece funcional o cualquier cambio se documenta como condición de ambiente	No aplica / justificado. Todos los escenarios F7 reportados se ejecutaron mediante Cloudflare; laURL exacta no quedo registrada en el entorno actual.

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F7-018	Analizar assertions vs usuarios virtuales en Postman	Al menos un performance run ejecutado	1. Abrir Runs >Performance >Tests. 2. Revisar Testsvs VU. 3. Identificar assertions que cambian de verde a amarillo, naranja orojo. 4. Registrar VU donde ocurre degradacion.	Vista Tests vs VU de Postman	Se documentan puntos de degradacion por assertion o se confirma estabilidadbajo VU usados	Completado. Tabla de runs consolida 71514 requests, 0 % error HTTP, throughput variable por mezcla de endpoints y degradaciones de assertions en ramp-up/spike/peak y F7-015.
F7-019	Consolidar 5xx, error budget y endpoints lentos	Todos los escenarios ejecutados con resultados disponibles	1. Revisar cada performancerun. 2. Consolidar requests totales, RPS, error %, 5xx, p95 y p99. 3. Identificar endpoints con mayor latencia o fallos.	Resultados de runsF7	Se obtiene tabla consolidada con 0 errores5xx o detalle de cada 5xx encontrado	Completado. Tabla granular: 76228 requests, errorHTTP 0.00 %, avg ponderado 151.48 ms, failure ponderado2.58 % inchuyendo F7-015 y 0.32 % excluyendo F7-015; 0 HTTP 5xx reportados.

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F7-020	Emitir reporte final de performance y conclusion de estabilidad	F7-001 a F7-019 ejecutados o justificados	1. Completar reporte F7. 2. Adjuntar capturas, links o exportaciones de Postman. 3. Registrar entorno hardware/SO. 4. Emitir conclusion de estabilidad local y riesgos.	Evidencias F7 completas	Reporte final incluye metricas por escenario, umbrales, hallazgos y conclusion Go/observaci ones/no aprobada	Completado. Reporte F7 emitido como aprobado con observaciones por hallazgo de seguridad F7-015 y degradaciones menores de assertions en ramp-up/spike/peak.

Fase F8 - Regresión y cierre QA

Objetivo: Reejecutar los casos de prueba de prioridad P0 y P1 para confirmar que las correcciones aplicadas durante el ciclo no rompieron flujos críticos, consolidar las evidencias de todas las fases, clasificar las incidencias abiertas y emitir la decisión formal de salida del ciclo de pruebas.

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F8-001	Reejecutar la verificacion de health check del backend para confirmar que el endpointGET /api/health sigue respondiendo correctamente después de todo el ciclo de pruebas	Backend ejecutandose al final del ciclo de pruebas	1. Enviar solicitudGET a /api/health. 2. Verificar respuesta HTTP200 con status ok. 3. Verificar que el timestamp es reciente. 4. Documentar el resultado como prueba de Regresión.	GET /api/health	Respuest a HTTP200 con status ok, confirmando estabilidad del backend al cierre del ciclo	Pendiente
F8-002	Reejecutar el flujo completo de registro de usuario para confirmar que el registro sigue funcionand o correctamente después de las correcciones del ciclo	Backend ejecutandose. No existe el usuario de Regresión	1. Registrar un usuario nuevo con datos unicos de Regresión. 2. Verificar respuestaHTTP 201. 3. Verificar que se recibe token de acceso. 4. Verificar que el usuario puede iniciar sesión inmediatamente después.	nombre=qa_regresion_01, email=qa_regresion_01@ example.com, pass- word=QAtest1234	El registro funciona correctamente y el usuario puede iniciar sesión después del registro	Pendiente
F8-003	Reejecutar el flujo completo de inicio de sesión por correo electronico y por nombre de usuario para confirmar que ambos métodos siguen funcionand o	UsuariosQA registrados	1. Iniciar sesión con correo electronico y verificar token. 2. Cerrar sesión. 3. Iniciar sesión con nombre de usuario y verificar token. 4. Documentar ambos resultados.	Login por correo y por nombre de usuario con credencia- les de QA	Ambos métodos de inicio de sesión funcionan correctamente y retornan tokens válidos	Pendiente
F8-004	Reejecutar el flujo completo de creación, edicion y eliminacion de mascota para confirmar que la gestion de mascotas sigue estable	Usuario autenticado	1. Crear una mascota de Regresión. 2. Verificar que aparece en el listado. 3. Editar el nombre de la mascota. 4. Verificar que el cambio se refleja. 5. Eliminar la mascota. 6. Verificar que desaparece del listado.	nombre=Mascota Regresión QA	El ciclo completo de creación, edi- cion y eliminacion funciona sin errores	Pendiente
F8-005	Reejecutar el flujo completo de creación de un reporte de peligro y verificar que aparece en el ma- pa con el marcador y circulo de impacto correctos	Usuario autenticado. Stack local completo	1. Crear un reporte de peligro de Regresión con coordenadas, nivel de riesgo y radio de impacto. 2. Verificar marcador en mapa. 3. Verificar circulo de impacto. 4. Abrir detalle y verificar datos.	coordenadas en Cali, risk_level_code=high, im- pact_radius_meters=1 00	El reporte se crea, aparece en elma- pa con marcador y circulo correctos y el detalle muestra los datos espera- dos	Pendiente
F8-006	Reejecutar el flujo completo de creación de reporte de mascota pérdida y verificar que el marcador aparece en la capa correspondiente del mapa	Usuario autenticado con mascota registrad a	1. Crear reporte de mascota pérdida de Regresión. 2. Verificar marcador en mapa. 3. Abrir detalle y verificar datos. 4. Verificar que la mascota queda asociada al reporte.	Mascota existente, coor- denadas, detalles, fecha y teléfono	El reporte se crea, el marcador aparece y los datos son correctos en el detalle	Pendiente
F8-007	Reejecutar el flujo completo de pla- nificación de ruta con GraphHop- per para confirmar que la polilínea, distancia, duracion e instrucciones se generan correctamente	GraphHopper ejecutando- se. Ruta guardada con dos puntos	1. Planificar unaruta de Regresión con dos puntos. 2. Verificar que se dibuja la polilínea. 3. Verificar distancia y duracion positivas. 4. Verificar que las instrucciones de navegación se generan.	Ruta con dos puntos enCali, modo walking	La planificación genera polilínea, distancia, duracion e instrucciones válidas	Pendiente
F8-008	Reejecutar la verificacion de que los endpoints protegidos rechazan solicitudes sin token de autentica- ción con código HTTP401	Backend ejecutandose	1. Enviar GET a /api/auth/me sin token y veri- ficar401. 2. Enviar POST a /api/pets sin token y verificar401. 3. Enviar POST a /api/hazards sin token y verificar401. 4. Enviar DELETE a /api/hazards/1 sin token y verificar401.	Solicitudes sin header Authorization a endpoints protegidos	Todos los endpoints protegidos retornanHTTP 401 sin token	Pendiente
F8-009	Reejecutar la verificacion de que un usuario no puede modificar mas- cotas de otro usuario confirmand o el codigoHTTP 403	Mascota de usuario A, token de usuario B	1. Enviar PUT a /api/pets/{id_masc ota_de_A} con token de B. 2. Verificar respuestaHTTP 403. 3. Verificar que la mascota no se modifico.	Token de usuario B, mascota de usuario A	Respuest a HTTP403 y mascota sin cambios	Pendiente
F8-010	Reejecutar la verificacion de que el endpointPOST /api/media/ upload rechaza datos de imagen invalidos con codigoHTTP 422	Backend ejecutand ose	1. Enviar POST a /api/media/upload con image_data=no-es- base64. 2. Verificar respuesta HTTP422. 3. Verificar mensaje de formato invali- do.	bucket=pets, image_data=no-es- base64	Respuest a HTTP 422 con mensaje de formato de imagen no válido	Pendi ente

ID	Descripción	Precondiciones	Pasos detallados	Datos de entrada	Resultado esperado	Resultado real
F8-011	Reejecutar la verificacion de que flutter analyze no reporta errores críticos en el proyecto frontend al cierre del ciclo	FlutterSDK instalado. código fuente al final del ciclo	1. Ejecutar flutter analyze en el directorio frontend. 2. Registrar errores, warnings e infos. 3. Comparar con los resultados de lafase F2.	Comando: flutter analyze	No hay errores críticos nuevos respecto a la faseF2	Pendiente
F8-012	Reejecutar la verificacion de que todas las pruebas unitarias de PHPUnit del backend pasan correctamente al cierre del ciclo	Backend con depende ncias instaladas	1. Ejecutar php artisan test en el directorio backend. 2. Registrar pruebas pasadas y fallidas. 3. Comparar con los resultados de lafase F1.	Comando: php artisantest	Todas las pruebas del backend pasan sin fallos nuevos respecto a la faseF1	Pendiente
F8-013	Reejecutar la verificacion de que la aplicación se visualiza correctamente en viewport de escritorio 1366x768 al cierre del ciclo	Stack local ejecutandose	1. Configurar viewport a 1366x768. 2. Navegar por flujos principales. 3. Verificar que nohay overflow ni elementos rotos.	Viewport 1366x768	La aplicación se visualiza correctamente sin problemas de layout	Pendiente
F8-014	Reejecutar la verificacion de que la aplicación se visualiza correctamente en viewport móvil 390x844 al cierre del ciclo	Stack local ejecutandose	1. Configurar viewport móvil 390x844. 2. Navegar por flujos principales. 3. Verificar que los paneles se adaptan sin overflow.	Viewport 390x844	La aplicación se adapta correctamente al viewport móvil	Pendiente
F8-015	Verificar que todas las incidencias encontradas durante el ciclo de pruebas están documentadas con modulo, severidad, ambiente, datos usados y estado actual	Resultados de todas las fases completadas	1. Recopilar todas las incidencias de las fases F0 a F7. 2. Verificar que cada incidencia tiene modulo, severidad, ambiente y datos usados. 3. Clasificar cada incidencia como resuelta, pendiente o aceptada. 4. Documentar las incidencias pendientes con justificacion.	Incidencias recopiladas de todas las fases	Todas las incidencias están documen tadas con la informacion requerid a y clasificadas correctamente	Pendiente
F8-016	Verificar que no quedan defectos críticos o de severidadalta sin decisión formal del equipo antes de emitir la decisión de salida	Lista de incidencias clasificadas	1. Filtrar las incidencias por severidad critica yalta. 2. Verificar que cada defecto crítico tiene una decisión formal: corregido, aceptado con riesgo o pospuesto con justificacion. 3. Verificar que nohay defectos críticos sin decisión.	Lista de defectos críticos y altos	No quedan defectos críticos sin decisión formal documentada	Pendiente
F8-017	Verificar que los casos de prueba afectados por correcciones realizadas durante el ciclo fueron reejecutados con resultado documentado	Lista de correccio nes aplicadas y casos afectados	1. Identificar los casos de prueba que fueron afectados por correcciones de código. 2. Verificar que cada caso afectado fue reejecutado después de la correccion. 3. Verificar que el resultado de la reejecucion está documentado con evidencia.	Casos afectados por correcciones	Todos los casos afectados por correcciones tienen reejecucion documentada con resultado y evidencia	Pendi ente
F8-018	Consolidar las evidencias de todas las fases verificando que cada carpeta defase contiene las capturas, logs y reportes correspondi entes	Carpetas de evidencias creadas en F0	1. Revisar la carpeta docs/evidencias/F0 y verificar que contiene evidencias de lafase F0. 2. Repetir para cada fase deF1 a F7. 3. Verificar que no hay fases sin evidencia. 4. Documentar la cantidad de archivos de evidencia por fase.	Estructura de carpetas docs/evidencias/	Cada carpeta de fase contiene al menos las evidencias minimas requeridas por elplan de pruebas	Pendiente
F8-019	Generar el conteo final de casos de prueba por resultado clasificando cada caso como Aprobado, Fallido, Bloqueado o No ejecutado	Todos los casos de prueba del documento con resultadoreal registrad o	1. Recorrer todas las tablas de casos de prueba del documento. 2. Contar los casos con resultado Aprobado. 3. Contar los casos con resultado Fallido. 4. Contar los casos con resultado Bloqueado. 5. Contar los casos con resultado No ejecutado. 6. Calcular el porcentaje de aprobacion.	Resultados reales de todos los casos	Se genera el conteo final con totales y porcentaj e de aprobacion para el informe de cierre	Pendiente
F8-020	Emitir la decisión formal de salida del ciclo de pruebas como Go o No-Go basada en los criterios de salida definidos en el plan de pruebas	Todos los criterios de salida evaluados	1. Verificar que todos los casos P0 fueron ejecutados con resultado documentado. 2. Verificar que nohay defectos críticos sin decisión. 3. Verificar que la evidencia es suficiente para auditar cada caso. 4. Verificar que los casos de Regresión fueron ejecutados. 5. Emitir la decisión Go si todos los criterios se cumplen, o No- Go con la lista de criterios incumplidos y acciones requeridas.	Criterios de salida delplan de pruebas evaluados	Se emite la decisionGo si todos los criterios se cumplen, o No-Go con justificacion detallada y acciones pendient es	Pendi ente

Apéndice F

Reporte F0

El presente anexo corresponde al reporte F0, asociado a la fase de preparación y validación inicial del ambiente de pruebas de la plataforma A4MPets. Esta fase tuvo como objetivo verificar que todos los componentes necesarios para la ejecución de las pruebas posteriores se encontraran correctamente instalados, configurados y operativos dentro del entorno de desarrollo definido para el proyecto.

Durante esta etapa se validó el funcionamiento general de los servicios principales de la arquitectura, incluyendo el backend desarrollado en Laravel, el frontend implementado en Flutter Web, la integración con GraphHopper mediante contenedores Docker y los mecanismos de exposición local utilizando Cloudflare Tunnel. Asimismo, se comprobó la disponibilidad de dependencias, rutas de ejecución, scripts automatizados, acceso a la base de datos y conectividad entre los distintos componentes del sistema.

Aunque esta fase no estuvo orientada a evaluar funcionalidades específicas del producto, su ejecución permitió garantizar la estabilidad mínima requerida para iniciar las pruebas unitarias, de integración, funcionales y end-to-end descritas en las fases posteriores del proceso de validación.

El reporte incluido a continuación documenta el estado del entorno de pruebas, los procedimientos de verificación realizados y las evidencias asociadas a la preparación del ambiente de ejecución.

1. Reporte de Evidencias - Fase F0: Preparación y Smoke de Ambiente

Campo	Valor
Proyecto	A4MPets
Fecha de ejecución	2026-05-08
Fase	F0 - Preparación y smoke de ambiente
Herramientas	PowerShell, npm, Composer, Artisan, Docker, GraphHopper y Cloudflared
Entorno	Windows, repositorio local `C:\projects\Web\A4MPets`

Este reporte consolida la evidencia de la fase F0, orientada a validar que el entorno base del proyecto está listo para continuar con las fases automatizadas y de integración del plan de pruebas.

2. Resumen Ejecutivo

La fase F0 valida que el entorno base del proyecto esté listo para continuar con las fases automatizadas y de integración del plan de pruebas. Durante la ejecución se verificaron los elementos principales del repositorio, las dependencias técnicas, el arranque de servicios, la base de datos, la autenticación inicial, GraphHopper, Docker y Cloudflare Tunnel.

2.1. Elementos verificados

1. Estructura del repositorio y carpetas de evidencias.
2. Dependencias de Node.js, PHP/Composer y Flutter.
3. Arranque temporal del backend Laravel.
4. Health check del backend.
5. Migraciones y tablas principales.
6. Registro y login de usuarios QA.
7. Middleware de autenticación sin token.
8. Configuración y arranque de GraphHopper.
9. Ruta básica y ruta con tres puntos en GraphHopper.
10. Generación temporal de URL pública con Cloudflare Tunnel.
11. Versión bajo prueba en Git.

2.2. Resultado general

Estado	Cantidad
PASS	20
PASS parcial	1
No ejecutado visualmente	1
FAIL	0

3. Versión bajo prueba

3.1. Comandos ejecutados

```
git log -1 --format="%H %ai %s"
git branch --show-current
```

3.2. Resultado

Dato	Valor
Commit	`6ba3ff8584167eb30ca8ed902c524ffa5d5f23a1`
Fecha	2026-05-08 13:26:31 -0500
Mensaje	`pre test creation`
Rama	`feature/fixes`

4. Verificación de dependencias

4.1. Comandos ejecutados

```
node --version
npm --version
php --version
composer --version
flutter --version
npm ls concurrently puppeteer --depth=0
composer check-platform-reqs
```

4.2. Resultados principales

Componente	Resultado
Node.js	v24.15.0
npm	11.12.1
PHP	8.4.19
Composer	2.9.7
Flutter	3.41.7
Dart	3.11.5
`concurrently`	Instalado: 9.2.1
`puppeteer`	Instalado: 24.42.0
Composer platform reqs	Todas las dependencias requeridas en estado `success`.

4.3. Validaciones de archivos y carpetas

Elemento	Estado
`backend/`	Existe
`frontend/`	Existe
`docs/`	Existe
`scripts/`	Existe
`package.json` raíz	Existe
`node_modules/` raíz	Existe
`backend/vendor/`	Existe
`docs/evidencias/F0` a `docs/evidencias/F8`	Existen

5. Backend Laravel

5.1. Arranque temporal

Comando ejecutado:

```
npm run backend -- --dry-run
```

Resultado:

```
Server running on [http://0.0.0.0:8000]
Backend listo en: http://127.0.0.1:8000/
Exit code: 0
```

5.2. Health check

Solicitud ejecutada:

```
GET http://127.0.0.1:8000/api/health
```

Resultado:

```
HEALTH status=ok timestamp=2026-05-08T19:04:54.045707Z
```

5.3. Registro y login QA

Solicitudes ejecutadas:

```
POST /api/auth/register
POST /api/auth/login
GET /api/auth/me sin Authorization
```

Validación	Resultado
Registro `qa_a4m_01@example.com`	PASS, `success=true`, token de 64 caracteres
Registro `qa_a4m_02@example.com`	PASS, `success=true`, token de 64 caracteres
Login `qa_a4m_01@example.com`	PASS, `success=true`, token de 64 caracteres
`/api/auth/me` sin token	PASS, HTTP 401

6. Base de datos

Comando ejecutado:

```
php artisan migrate:status
```

Resultado: todas las migraciones listadas aparecen en estado `Ran`. La última migración aplicada fue `2026_05_01_000021_remove_domain_sync_columns`.

6.1. Tablas principales verificadas

Tabla	Estado
users	Existe
pets	Existe
hazard_reports	Existe
lost_pet_reports	Existe
found_pet_reports	Existe
generic_reports	Existe
routes	Existe
route_points	Existe
report_comments	Existe
hazard_comments	Existe

7. Frontend Flutter

Comando ejecutado:

```
npm run frontend -- --dry-run
```

Resultado:

```
URLs API configuradas:
http://localhost:8000/api |
http://127.0.0.1:8000/api |
http://192.168.1.3:8000/api
```

Archivo generado: `frontend\env\runtime.local.json`.

Validación	Estado
Flutter SDK disponible	PASS
Runner frontend genera configuración runtime	PASS
Arranque visual en Chrome y validación de mapa	No ejecutado visualmente en esta corrida

8. GraphHopper

8.1. Configuración

Elemento	Resultado
backend/graphhopper/config.yml	Existe
Perfil car	Definido
Perfil foot	Definido
Puertos configurados	8989 HTTP, 8990 admin
backend/graphhopper/data/map.osm.pbf	Existe
Tamaño map.osm.pbf	311,442,414 bytes

8.2. Arranque del contenedor

Comando ejecutado:

```
npm run graphhopper
```

Resultado:

```
Container a4mpets-graphhopper Started
```

Estado posterior:

```
a4mpets-graphhopper Up About a minute (healthy)
0.0.0.0:8989-8990->8989-8990/tcp
```

8.3. Rutas de prueba

Prueba	Solicitud	Resultado
Ruta con dos puntos	`http://127.0.0.1:8989/route?point=3.4516,-76.5320&point=3.4570,-76.5280&profile=car&type=json`	`ROUTE status=200 paths=1 distance=1376.638`
Ruta con tres puntos	`http://127.0.0.1:8989/route?point=3.4516,-76.5320&point=3.4570,-76.5280&point=3.4620,-76.5240&profile=car&type=json`	`ROUTE_3POINTS status=200 paths=1 distance=2279.574`

9. Docker y Docker Compose

9.1. Comandos ejecutados

```
docker --version
docker compose version
docker ps --filter name=a4mpets-graphhopper
```

9.2. Resultado

Componente	Resultado
Docker CLI	29.4.0
Docker Compose	v5.1.1
Contenedor GraphHopper	Up ... (healthy)
Puertos	8989 y 8990 expuestos

10. Cloudflare Tunnel

10.1. Comandos ejecutados

```
cloudflared --version
npm run backend:cloudflare -- --dry-run
```

10.2. Resultado

```
cloudflared version 2025.8.1
Tunnel detectado: https://steven-strange-ensure-satin.trycloudflare.com
Tunnel listo en: https://steven-strange-ensure-satin.trycloudflare.com/
URL pública guardada en: .runtime\cloudflare-url.txt
Exit code: 0
```

El comando fue ejecutado en modo `-dry-run`; el runner crea la URL temporal, valida conectividad y luego limpia procesos y archivos runtime al cerrarse.

11. Docker Compose y scripts

11.1. Validación estática de docker-compose.yml

Servicio	Validación	Estado
`graphhopper`	Puertos 8989/8990, volumen de mapa y `config.yml`.	PASS
`backend`	Puerto 8000, SQLite y `GRAPHHOPPER_BASE_URL=http://graphhopper:8989`.	PASS
`frontend`	Puerto 8080 y dependencia de backend.	PASS

11.2. Validación de scripts en package.json

Script	Estado
`npm run backend`	Definido y validado con `-dry-run`.
`npm run frontend`	Definido y validado con `-dry-run`.
`npm run graphhopper`	Definido y ejecutado.
`npm run dev`	Definido con backend, frontend y GraphHopper vía `concurrently`.
`npm run dev:cloudflare`	Definido con backend Cloudflare, frontend runtime y GraphHopper.

12. Mapeo de casos F0

Caso	Resultado	Evidencia
F0-001	PASS	Estructura raíz verificada.
F0-002	PASS	`node_modules`, `concurrently` y `puppeteer` instalados.
F0-003	PASS	`backend/vendor` existe y `composer check-platform-reqs` está en `success`.
F0-004	PASS	`npm run backend - -dry-run` inicia Laravel en puerto 8000.
F0-005	PASS	`/api/health` responde `status=ok`.
F0-006	PASS parcial	`npm run frontend - -dry-run` genera runtime y Flutter existe.
F0-007	No ejecutado visualmente	No se abrió Chrome para validar el mapa en esta corrida.
F0-008	PASS	`npm run graphhopper` inicia contenedor.
F0-009	PASS	GraphHopper responde ruta con `paths=1`.

Caso	Resultado	Evidencia
F0-010	PASS	`config.yml` define perfiles `car` y `foot`.
F0-011	PASS	`map.osm.pbf` existe y pesa 311,442,414 bytes.
F0-012	PASS	Cloudflare Tunnel genera URL `trycloudflare.com` en modo `dry-run`.
F0-013	PASS	Migraciones en `Ran` y tablas principales existen.
F0-014	PASS	Registro QA principal exitoso.
F0-015	PASS	Registro QA secundario exitoso.
F0-016	PASS	Login QA principal exitoso.
F0-017	PASS	Carpetas `docs/evidencias/F0` a `docs/evidencias/F8` existen.
F0-018	PASS	`docker-compose.yml` define servicios, puertos y volúmenes esperados.
F0-019	PASS parcial	Scripts `dev` definidos; componentes validados por separado. No se dejó stack persistente levantado con `npm run dev`.
F0-020	PASS	Commit y rama registrados.
F0-021	PASS	Rutas con dos y tres puntos de Cali devuelven `paths=1`.
F0-022	PASS	`/api/auth/me` sin token responde HTTP 401.

13. Hallazgos F0

#	Hallazgo	Impacto	Tratamiento
1	La validación visual del mapa en Chrome no se ejecutó en esta corrida.	F0-007 queda pendiente de evidencia visual.	Cubrir en F4 o en una corrida F0 visual con captura del navegador.
2	La primera consulta a GraphHopper falló por conexión mientras el contenedor terminaba de inicializar.	Riesgo normal de arranque frío/indexación.	Se esperó 60 segundos y la ruta respondió HTTP 200.
3	`npm run backend:cloudflare -- -dry-run` crea y valida una URL temporal, pero limpia `.runtime\cloudflare-url.txt` al terminar.	La URL no queda persistida después del `dry-run`.	Para pruebas manuales largas usar `npm run dev:cloudflare` sin `-dry-run`.

14. Conclusiones

El ambiente queda apto para continuar el ciclo de pruebas. En particular, se concluye lo siguiente:

1. Backend Laravel arranca y responde health.
2. Base de datos migrada y tablas principales disponibles.
3. Usuarios QA pueden registrarse e iniciar sesión.
4. GraphHopper arranca, queda healthy y responde rutas sobre coordenadas de Cali.
5. Cloudflare Tunnel puede generar una URL pública temporal.
6. Dependencias principales de Node.js, PHP/Composer y Flutter están instaladas.

Pendiente específico: generar evidencia visual de F0-007 abriendo el frontend en Chrome y verificando el mapa renderizado.

Apéndice G

Reporte F1

El presente anexo corresponde al reporte F1, asociado a la ejecución de pruebas unitarias de caja blanca sobre el backend de la plataforma A4MPets. Esta fase tuvo como propósito validar el comportamiento interno de los componentes desarrollados en Laravel, evaluando de manera aislada servicios, modelos, reglas de negocio, relaciones entre entidades, validaciones y mecanismos de transformación de datos.

Las pruebas ejecutadas durante esta etapa se enfocaron en verificar la consistencia lógica de los módulos backend sin depender directamente de la interfaz gráfica ni de la integración completa con servicios externos. En este contexto, se evaluaron aspectos relacionados con autenticación, gestión de mascotas, manejo de reportes, comentarios, permisos de acceso, validaciones de formularios y procesamiento de información geoespacial.

Adicionalmente, esta fase permitió identificar comportamientos internos susceptibles de mejora antes de realizar pruebas de integración y sistema, contribuyendo a incrementar la estabilidad general de la aplicación y reducir la propagación de errores hacia etapas posteriores del proceso de validación.

El reporte incluye la descripción de las pruebas realizadas, la estrategia utilizada para su ejecución, los resultados obtenidos, las métricas de cobertura generadas y las evidencias correspondientes al análisis del backend.

1. Reporte de Evidencias - Fase F1: Pruebas Unitarias Backend Laravel

Campo	Valor
Proyecto	A4MPets
Fecha de ejecución	2026-05-08
Fase	F1 - Caja blanca backend unitaria
Herramienta	PHPUnit / `php artisan test`
Entorno	Laravel con SQLite in-memory (`DB_CONNECTION=sqlite`, `DB_DATABASE=:memory:`)

2. Resumen Ejecutivo

La fase F1 queda separada formalmente como suite de pruebas unitarias backend. Las pruebas que antes estaban en `tests/Feature` fueron reclasificadas como F3, por lo que esta evidencia ya no mezcla cobertura de controladores/API con pruebas unitarias.

No se modificó código productivo. Los cambios de esta reorganización fueron:

- Pruebas unitarias movidas a `backend/tests/F1`.
- Suite PHPUnit F1 apuntando a `tests/F1`.
- Cobertura unitaria regenerada en `backend/coverageF1`.
- Pruebas Feature/API movidas fuera de F1 hacia `backend/tests/F3`.

3. Resultado Final de PHPUnit F1

Elemento	Resultado
Comando ejecutado	`php artisan test --testsuite=F1`
Pruebas	67 passed
Aserciones	327 assertions
Duración	9.45 s
Exit code	0

4. Resultado Final de Cobertura F1

Elemento	Resultado
Comando ejecutado	`php artisan test --testsuite=F1 --coverage-html coverageF1`
Pruebas	67 passed
Aserciones	327 assertions
Duración	14.44 s
Exit code	0
Reporte generado en	`C:\projects\Web\A4MPets\backend\coverageF1\index.html`

4.1. Cobertura real por carpeta principal

La siguiente tabla resume la cobertura real por carpeta principal, considerando únicamente la suite F1.

Carpeta	Cobertura de líneas	Líneas cubiertas	Observación
Http	20.55 %	163 / 793	F1 cubre FormRequests; los controladores pasan a F3.
Models	86.08 %	235 / 273	Cumple como cobertura unitaria de modelos.
Providers	100.00 %	2 / 2	Mantiene 100 %.
Services	67.60 %	822 / 1216	Muy cercano al 70 %; las ramas legacy/sync quedan documentadas.
Support	100.00 %	16 / 16	Cumple.
Total app	53.83 %	1238 / 2300	Referencia general de F1 aislado.

5. Pruebas F1

Carpeta	Propósito
`tests/F1/Http/Requests`	Reglas y autorización de FormRequests.
`tests/F1/Models`	Relaciones, accessors, hashes, scopes y `points_json`.
`tests/F1/Services`	Servicios de autenticación, mascotas, reportes, peligros, rutas, GraphHopper y comentarios.
`tests/F1/Support`	Normalización de instrucciones de navegación.

6. Hallazgos F1

#	Hallazgo	Tratamiento
1	Algunos métodos legacy de sincronización local-first referencian columnas removidas por migraciones recientes.	Documentado en pruebas unitarias como comportamiento/error actual. No se modifica código productivo en F1.
2	`api.token` resuelve usuario si el token es válido, pero no rechaza por sí solo tokens ausentes o inválidos.	La autorización final se valida en controladores y queda cubierta en F3.
3	La cobertura Http baja en F1 es esperada después de mover controladores/API a F3.	La cobertura de API queda reportada separadamente en `coverageF3`.

7. Conclusiones

F1 queda cerrada como suite unitaria backend separada y reproducible:

- Suite: `php artisan test --testsuite=F1`.
- Cobertura: `php artisan test --testsuite=F1 --coverage-html coverageF1`.
- Resultado: 67 pruebas aprobadas y 327 aserciones.

La cobertura de controladores/API ya no se informa en F1; corresponde a la fase F3.

Apéndice H

Reporte F1 - Cobertura

El reporte completo de cobertura correspondiente a la fase F1 fue generado automáticamente mediante PHPUnit en formato HTML.

Debido a la naturaleza interactiva y navegable de este tipo de reportes, el archivo se adjunta en formato pdf como evidencia digital complementaria:

- `coverageF1.pdf`

El archivo contiene el reporte HTML completo, incluyendo métricas de cobertura por clases, métodos, líneas y componentes del backend.

Apéndice I

Reporte F2

El presente anexo corresponde al reporte F2, asociado a la ejecución de pruebas unitarias y de validación funcional sobre el frontend de la plataforma A4MPets. Esta fase tuvo como propósito evaluar el comportamiento de los componentes desarrollados en Flutter Web, verificando tanto la estabilidad de la lógica interna de la interfaz como la correcta representación visual de los distintos módulos del sistema.

Las pruebas realizadas durante esta etapa se enfocaron en validar proveedores de estado, servicios, formularios, componentes visuales, paneles interactivos y mecanismos de navegación, garantizando que las funcionalidades implementadas respondieran de manera consistente ante diferentes escenarios de interacción del usuario.

Asimismo, se evaluaron aspectos relacionados con la representación geográfica, el manejo de estados de carga y error, la visualización de reportes, la administración de rutas y mascotas, así como la adaptación de la interfaz a distintos tamaños de pantalla y configuraciones visuales. Estas pruebas permitieron identificar inconsistencias de comportamiento, problemas de renderizado y posibles fallos en la interacción entre componentes del frontend.

El reporte incluido a continuación documenta la estrategia de pruebas aplicada sobre el cliente Flutter, las herramientas utilizadas durante la ejecución, los resultados obtenidos y las evidencias generadas durante el proceso de validación.

1. Reporte de Evidencias - Fase F2: Pruebas Frontend Flutter

Campo	Valor
Proyecto	A4MPets
Fecha de ejecución	2026-05-08
Fase	F2 - Caja blanca frontend
Herramienta	Flutter Analyzer, `flutter test` y LCOV `genhtml`
Entorno	Flutter web/unit tests en `C:\projects\Web\A4MPets\frontend`

2. Resumen ejecutivo

La fase F2 fue completada con pruebas unitarias y *widget tests* enfocados en cubrir lógica aislada del frontend Flutter: modelos, *providers* Riverpod, servicios Dart, tema y widgets compartidos.

No se modificó código productivo. Los cambios se limitaron a:

- Corrección de un *fixture* de prueba existente que usaba `is_anonymous` como entero en lugar de booleano.
- Nuevas pruebas en `frontend/test`.
- Regeneración de cobertura LCOV y reporte HTML en `frontend/coverage/html`.
- Creación de esta evidencia con datos reales de ejecución y cobertura.

3. Resultado final de Flutter Analyze

Elemento	Resultado
Comando ejecutado	`flutter analyze`
Salida	`Analyzing frontend...` `No issues found! (ran in 6.1s)`
Exit code	`0`

4. Resultado final de Flutter Test

Elemento	Resultado
Comando ejecutado	`flutter test`
Resultado	`Tests: 102 passed`
Salida	`All tests passed!`
Exit code	`0`

4.1. Ejecuciones focalizadas durante la implementación

Suite	Resultado
`flutter test test/models`	51 passed

Suite	Resultado
`flutter test test/services`	36 passed
`flutter test test/providers`	8 passed
`flutter test test/theme`	1 passed
`flutter test test/widgets`	6 passed
Total	102 passed

5. Resultado final de cobertura

Elemento	Resultado
Comandos ejecutados	`flutter test --coverage` `perl Ç:\ProgramData\chocolatey\lib\lcov\tools\bin\genhtml" coverage/lcov.info --output-directory coverage/html`
Resultado	`Tests: 102 passed`
Salida	`All tests passed!`
LCOV	Overall coverage rate lines: 90.5 % (1615 / 1785)
Exit code	`0`
Reporte generado	`C:\projects\Web\A4MPets\frontend\coverage\html\index.html`

5.1. Cobertura final por carpeta principal

Carpeta	Cobertura de líneas	Líneas cubiertas	Estado
models	98.40 %	430 / 437	Cumple $\geq$ 70 %
providers	92.86 %	247 / 266	Cumple $\geq$ 70 %
services	94.43 %	390 / 413	Cumple $\geq$ 70 %
theme	88.89 %	8 / 9	Cumple $\geq$ 70 %
widgets	81.82 %	540 / 660	Cumple $\geq$ 70 %
Total LCOV	90.48 %	1615 / 1785	Referencia general

5.2. Detalle relevante de cobertura

Área	Cobertura de líneas	Observación
Models/report records	100.00 %	Lost, found, generic, comments, rutas e instrucciones cubiertos.
Providers/`trip_flow_provider.dart`	100.00 %	Mascotas, transporte, filtros, ruta, inicio y cierre de viaje.
Providers/`theme_provider.dart`	100.00 %	Alternancia y reemplazo de modo.
Providers/`trip_transport_preference_provider.dart`	100.00 %	Persistencia y normalización de modo.
Services/`backend_api_client.dart`	100.00 %	Headers y fallback `GET/POST/PUT/DELETE` cubiertos con `MockClient`.

Área	Cobertura de líneas	Observación
Services/guest_data_store.dart	100.00 %	Pets, rutas, actualización, borrado, walks, clear y JSON inválido.
Services/navigation_audio_progress.dart	98.65 %	Cues far/approach/immediate, off-route, no audible y polilínea corta.
Widgets/found_pet_details_panel.dart	84.72 %	Render de detalle, reporte vinculado, comentarios y delete debug.
Widgets/reporting_form_widgets.dart	86.44 %	Inputs, dropdown, foto, checkbox, fecha y botones.
Widgets/panel_modal.dart	78.14 %	Título, loading, cierre, minimizar y layout.

6. Pruebas agregadas o ampliadas

6.1. Pruebas nuevas

Archivo	Propósito
test/models/report_records_extra_test.dart	Modelos de reportes, comentarios, rutas e instrucciones.
test/services/backend_api_client_extra_test.dart	Headers, URL base, fallbacks y errores HTTP.
test/services/guest_data_store_extra_test.dart	Persistencia local de mascotas y rutas de invitado.
test/services/navigation_services_extra_test.dart	Audio de navegación, progreso y visualización de peligros en ruta.
test/providers/flow_providers_extra_test.dart	Providers de paneles, reportes, rutas, viaje, mapa, tema y preferencias.
test/theme/app_palette_test.dart	Helpers de paleta por modo claro/oscuro.
test/widgets/shared_widgets_extra_test.dart	Widget tests de paneles, formularios, imágenes, perfil y detalle de mascota encontrada.

6.2. Pruebas ajustadas

Archivo	Ajuste
test/models/hazard_report_record_test.dart	Fixture de comentarios usa `is_anonymous: false`, alineado con `ReportCommentRecord.fromMap`.

7. Mapeo general F2

Caso	Cobertura actual
F2-001 a F2-002	`pet_record_test.dart` y propiedades/serialización de `PetRecord`.
F2-003 a F2-004	`session_store_test.dart`, `local_user_profile_test.dart` y persistencia con `SharedPreferences`.
F2-005	`flow_providers_extra_test.dart` cubre `panelFlowProvider`.
F2-006	`flow_providers_extra_test.dart` cubre `reportFlowProvider` y ubicaciones manuales.

Caso	Cobertura actual
F2-007	`flow_providers_extra_test.dart` cubre `routeEditorProvider`.
F2-008	`flow_providers_extra_test.dart` cubre `tripFlowProvider`.
F2-009	`flow_providers_extra_test.dart` cubre preferencia de transporte.
F2-010	`hazard_risk_filtering_test.dart` y `navigation_services_extra_test.dart`.
F2-011	`navigation_services_extra_test.dart` cubre `NavigationAudioMessageBuilder`.
F2-012	`navigation_services_extra_test.dart` cubre `NavigationAudioProgress`.
F2-013	`flow_providers_extra_test.dart` cubre `mapPresentationProvider`.
F2-014 a F2-015	`report_records_extra_test.dart`, `pet_record_test.dart` y `hazard_report_record_test.dart`.
F2-016	`backend_api_client_extra_test.dart`.
F2-017	Providers base y acceso local cubiertos por pruebas de `GuestDataStore` y providers relacionados.
F2-018	`guest_data_store_extra_test.dart` y `report_records_extra_test.dart`.
F2-019	`flow_providers_extra_test.dart` cubre `audioFeedbackEnabledProvider`.
F2-020	Estados asíncronos base no se prueban contra backend real en F2; se cubren providers de flujo local y se deja integración para fases posteriores.
F2-021	`navigation_services_extra_test.dart` cubre `RouteHazardVisualizerService`.
F2-022	`report_records_extra_test.dart` cubre `RouteRecord` y `RoutePoint`.
F2-023	Persistencia local de mascotas cubierta por `guest_data_store_extra_test.dart`; sincronización real queda fuera de F2.
F2-024	La construcción real contra `/routes/plan` depende de `http.post` estático y backend; se valida indirectamente en servicios de ruteo/visualización y queda integración para F3/F4.
F2-025	`app_palette_test.dart` y `flow_providers_extra_test.dart` cubren paleta y `themeModeProvider`.

8. Hallazgos F2

#	Hallazgo	Tratamiento
1	Un fixture existente enviaba `is_anonymous` como `0`, pero el modelo frontend actual espera `bool?`.	Corregido en prueba, sin modificar código productivo.
2	`local_image_widget_io.dart` queda con cobertura individual baja porque las ramas de imagen real/HTTP dependen de filesystem o red.	No bloquea F2: la carpeta widgets supera 70 % y se cubren fallback, data flow visual y diálogos.
3	`OsmRoutingService` no se prueba con backend real en F2 porque esta fase no ejecuta integración.	Se deja para F3/F4; en F2 se cubren servicios puros de ruta, visualización y cliente HTTP con mocks.

9. Conclusiones

La fase F2 queda completada con analyzer limpio, pruebas Flutter en verde y cobertura real por carpeta superior al mínimo esperado:

- Models: 98.40 %.
- Providers: 92.86 %.
- Services: 94.43 %.
- Theme: 88.89 %.
- Widgets: 81.82 %.

El reporte HTML de cobertura fue regenerado y esta evidencia refleja los resultados reales de `flutter analyze`, `flutter test`, `flutter test --coverage` y `genhtml`.

Apéndice J

Reporte F2 - HTML de ejecución

J.1. Reporte HTML de ejecución F2

El presente anexo corresponde al reporte HTML generado durante la ejecución de la fase F2, asociada a las pruebas realizadas sobre el frontend de la plataforma A4MPets.

El archivo fue generado automáticamente mediante las herramientas de testing utilizadas en Flutter, incluyendo `flutter test` y los mecanismos de generación de reportes descritos dentro de la estrategia de pruebas del proyecto. Debido a que este tipo de evidencia conserva información estructurada, navegación interna, métricas visuales y resultados detallados de ejecución, el reporte fue preservado en su formato HTML original.

Con el fin de mantener la integridad de los archivos generados y facilitar su consulta, el reporte completo se adjunta como archivo pdf bajo el nombre:

- `htmlF2.pdf`

El archivo contiene la página web completa generada automáticamente por las herramientas de pruebas, incluyendo resultados de ejecución, métricas asociadas y evidencias complementarias del proceso de validación del frontend.

Apéndice K

Reporte F3

El presente anexo corresponde al reporte F3, asociado a la ejecución de pruebas de integración sobre el backend de la plataforma A4MPets. Esta fase tuvo como objetivo validar la interacción entre los diferentes componentes internos del sistema desarrollados en Laravel, verificando el comportamiento conjunto de controladores, middleware, modelos, servicios, persistencia de datos y endpoints de la API REST.

A diferencia de las pruebas unitarias realizadas en la fase F1, esta etapa se enfocó en evaluar flujos completos de comunicación entre módulos, permitiendo comprobar la correcta integración entre autenticación, manejo de reportes, comentarios, gestión de mascotas, planificación de rutas y procesamiento de solicitudes HTTP.

Durante la ejecución se validaron respuestas JSON, códigos de estado HTTP, permisos de acceso, validaciones de entrada, sincronización de información y persistencia de datos dentro del entorno de pruebas. Asimismo, se analizaron escenarios asociados a errores controlados, acceso no autorizado y manejo de solicitudes inválidas, fortaleciendo la confiabilidad general del backend antes de las pruebas de sistema y pruebas end-to-end.

El reporte presentado a continuación documenta la estrategia utilizada, las pruebas ejecutadas, los resultados obtenidos y las evidencias generadas durante el proceso de integración del backend.

1. Reporte de Evidencias - Fase F3: Pruebas de Integración Backend/API

Campo	Valor
Proyecto	A4MPets
Fecha de ejecución	2026-05-08
Fase	F3 - Integración backend/API Laravel
Herramienta	PHPUnit / `php artisan test` / cobertura HTML
Entorno	Laravel con SQLite in-memory (`DB_CONNECTION=sqlite`, `DB_DATABASE=:memory:`)

2. Resumen Ejecutivo

La fase F3 fue implementada como suite separada para pruebas Feature/API del backend Laravel. Las pruebas de `tests/Feature` fueron reclasificadas como pruebas de integración y movidas a `backend/tests/F3`; las pruebas unitarias quedaron en `backend/tests/F1`.

No se modificó código productivo. Los cambios se limitaron a:

- Reorganización de carpetas de pruebas backend.
- Ajuste de suites en `backend/phpunit.xml`.
- Nuevas variantes F3-extra sobre endpoints API.
- Cobertura separada en `backend/coverageF3`.
- Actualización de evidencia F1 para reflejar `coverageF1`.

3. Estructura de Suites

Suite	Carpeta	Propósito
F1	`backend/tests/F1`	Pruebas unitarias backend.
F3	`backend/tests/F3`	Pruebas de integración backend/API.

Comandos disponibles:

```
php artisan test --testsuite=F1
php artisan test --testsuite=F3
php artisan test --testsuite=F1 --coverage-html coverageF1
php artisan test --testsuite=F3 --coverage-html coverageF3
```

4. Resultado de Validación

4.1. Autoload de Composer

Comando ejecutado:

```
composer dump-autoload
```

Resultado:

```
Generated optimized autoload files containing 6812 classes
Exit code: 0
```

4.2. Suite F3

Comando ejecutado:

```
php artisan test --testsuite=F3
```

Resultado:

```
Tests: 46 passed (235 assertions)
Duration: 9.50s
Exit code: 0
```

4.3. Regresión completa

Comando ejecutado:

```
php artisan test
```

Resultado:

```
Tests: 113 passed (562 assertions)
Duration: 17.67s
Exit code: 0
```

5. Resultado Final de Cobertura F3

Comando ejecutado:

```
php artisan test --testsuite=F3 --coverage-html coverageF3
```

Resultado:

```
Tests: 46 passed (235 assertions)
Duration: 15.14s
Exit code: 0
```

Reporte generado en:

```
coverageF3\index.html
```

Cobertura real por carpeta principal:

Carpeta	Cobertura de líneas	Líneas cubiertas	Estado
Http	90.29 %	716 / 793	Cumple $\geq$ 70 %
Models	70.33 %	192 / 273	Cumple $\geq$ 70 %
Providers	100.00 %	2 / 2	Mantiene 100 %
Services	47.86 %	582 / 1216	No cumple; ver hallazgo F3-1
Support	100.00 %	16 / 16	Cumple $\geq$ 70 %
Total app	65.48 %	1506 / 2300	Referencia general

6. Detalle Relevante de Services

Área	Cobertura de líneas	Líneas cubiertas	Observación
Services/Auth	100.00 %	19 / 19	Login, registro, perfil y logout cubiertos por API.
`GraphHopper`	89.10 %	188 / 211	Endpoint `/api/routes/plan` con `Http::fake`, fallback y avoidance.
Services/Comments	34.19 %	40 / 117	Store/list vía API cubierto; sync/formatters internos quedan para F1.
Services/Hazards	37.12 %	121 / 326	CRUD/comentarios vía API cubiertos; sync interno sin endpoint.
Services/Pets	32.74 %	37 / 113	CRUD, listado propio y `mark-found` cubiertos; sync interno sin endpoint.
Services/Reports	43.94 %	156 / 355	Reportes principales cubiertos; sync interno sin endpoint.
Services/Routes	28.00 %	21 / 75	CRUD de rutas cubierto; sync interno sin endpoint.

7. Pruebas Agregadas o Ampliadas

Archivo	Propósito
`Health/HealthEndpointTest.php`	Contrato de `/api/health`.
`TravelRouteAndPlanningControllerTest.php`	Planificación real por endpoint con `Http::fake`, avoidance, instrucciones, fallback y 502.
`Pets/PetCrudTest.php`	`scope=mine`, show, `mark-found`, 403 y relaciones persistidas.
`Hazards/HazardControllerExtraTest.php`	Validación adicional de relaciones owner/report tras flujos API.

8. Mapeo General F3

Caso	Cobertura actual
F3-001	`/api/health` y regresión base de disponibilidad API.
F3-002 a F3-004	Registro, login por email/username, errores de credenciales y validaciones.
F3-005	`/api/auth/me` con token válido y sin token.
F3-006	`/api/auth/profile` con update de usuario resuelto por token.
F3-007	`/api/auth/logout` y token revocado.
F3-008 a F3-010	CRUD/listado de mascotas, ownership, guest behavior documentado.
F3-011	`/api/pets/{pet}/mark-found` y borrado de reportes perdidos activos.
F3-012 a F3-014	Peligros: crear, listar, detalle, update, delete, 403 y validaciones.
F3-015	Comentarios de peligros por API.
F3-016 a F3-018	Reportes de mascota perdida: CRUD, listado, detalle, comentarios, 401/403.
F3-019 a F3-021	Reportes genéricos y normalización de riesgo `invalid`/`undefined`.

Caso	Cobertura actual
F3-022	Reportes de mascota encontrada con y sin reporte perdido vinculado.
F3-023	Media upload base64, bucket inválido, data URI, show y 404.
F3-024	Rutas guardadas: index/show/store/update/delete, validación y ownership.
F3-025	`/api/routes/plan` con GraphHopper fakeado, fallback, errores 422/502 y avoidance.

9. Hallazgos F3

#	Hallazgo	Tratamiento
1	`Services` queda en 47.86 % en F3 porque una parte grande de sus líneas corresponde a métodos internos de sincronización (sync) que no tienen endpoint en `routes/api.php`.	Se documenta como shortfall no factible dentro de F3 sin mezclar pruebas unitarias o crear endpoints productivos. Esos métodos quedan cubiertos/documentados en F1.
2	La integración real con GraphHopper/Docker no se ejecuta en F3.	Se usa `Http::fake` sobre el cliente HTTP para validar contrato API y payload sin dependencia externa. La integración real queda para fases posteriores.
3	El middleware `api.token` resuelve usuario cuando el token es válido, pero no rechaza por sí solo token ausente/inválido.	Se valida el comportamiento final en controladores con respuestas 401/403 según corresponda.

10. Conclusiones

F3 queda implementada y separada de F1:

- Suite F3: 46 pruebas aprobadas y 235 aserciones.
- Regresión backend completa: 113 pruebas aprobadas y 562 aserciones.
- Cobertura F3 separada en `backend/coverageF3`.
- Http, Models, Providers y Support cumplen el mínimo esperado.
- Services no alcanza 70 % por ausencia de endpoints para ramas internas de sincronización; queda documentado con detalle y sin alterar código productivo.

Apéndice L

Reporte F3 - Cobertura

El presente anexo corresponde al reporte de cobertura generado durante la ejecución de la fase F3, relacionada con las pruebas de integración del backend de la plataforma A4MPets.

El reporte fue generado automáticamente mediante PHPUnit y las herramientas de análisis de cobertura utilizadas durante la ejecución de las pruebas. Debido a que la evidencia producida consiste en un conjunto de archivos HTML interconectados, con navegación interna y métricas detalladas por clases, métodos y líneas de código.

Con el fin de preservar la estructura completa del reporte y facilitar su consulta, la evidencia se adjunta como archivo pdf bajo el nombre:

- `coverageF3.pdf`

El archivo contiene el reporte HTML completo de cobertura correspondiente a las pruebas de integración ejecutadas sobre el backend del sistema.

Apéndice M

Reporte F4

El presente anexo corresponde al reporte F4, asociado a las pruebas orientadas a la validación del sistema integrado de la plataforma A4MPets. En esta fase, el objetivo principal no estuvo centrado en la medición de cobertura de código, sino en la evaluación del comportamiento conjunto de la arquitectura completa bajo condiciones de ejecución reales dentro del entorno de desarrollo.

Estas pruebas se desarrollaron desde un enfoque de caja blanca a nivel arquitectónico, permitiendo analizar la interacción entre frontend, backend, servicios de rutas, almacenamiento local y mecanismos de sincronización, considerando el funcionamiento global de la plataforma como un sistema unificado.

A diferencia de las pruebas de integración tradicionales realizadas en la fase F3, esta etapa se enfocó en observar el comportamiento operacional de los componentes desplegados simultáneamente, evaluando flujos completos de interacción, estabilidad de servicios, sincronización entre módulos y consistencia general de la aplicación durante escenarios funcionales más cercanos al uso real del sistema.

Durante esta fase se verificaron procesos relacionados con autenticación, visualización geoespacial, manejo de reportes, rutas, sincronización de datos, consumo de servicios externos y funcionamiento integrado de GraphHopper dentro de la arquitectura propuesta. Asimismo, se analizaron comportamientos asociados a disponibilidad de servicios, manejo de errores y coordinación entre componentes distribuidos.

El reporte incluido a continuación documenta la estrategia utilizada para validar la arquitectura integrada del sistema, las pruebas ejecutadas, los resultados obtenidos y las evidencias generadas durante esta fase del proceso de evaluación.

1. Reporte de Evidencias - Fase F4: Sistema Integrado Local

Campo	Valor
Proyecto	A4MPets
Fecha de ejecución	2026-05-08
Fase	F4 - Sistema integrado local de caja blanca
Herramienta	Node.js QA harness, Laravel API, Flutter web-server, Docker GraphHopper y Playwright Chromium
Entorno	Stack local en `C:\projects\Web\A4MPets`

2. Resumen Ejecutivo

La fase F4 fue ejecutada como validación técnica del sistema integrado local. Se agregó un *harness* de pruebas en `scripts/f4-system-check.js` para levantar o reutilizar los servicios locales, consumir el backend Laravel con HTTP real, persistir datos en la base de datos local, consultar GraphHopper local y validar el arranque del frontend Flutter web.

No se modificó código productivo. Los cambios se limitaron a:

- Script de verificación técnica F4 en `scripts/f4-system-check.js`.
- Evidencia JSON de ejecución en `docs/evidencias/F4/f4_system_check_results.json`.
- Captura del *smoke* frontend en `docs/evidencias/F4/f4_frontend_smoke.png`.
- Creación del reporte de evidencias F4.

2.1. Resultado general

Área	Resultado
Backend Laravel/API	PASS
Base de datos local y persistencia	PASS
GraphHopper local	PASS
Media upload	PASS
Frontend web-server disponible por HTTP	PASS
Renderizado Flutter en Chromium headless	FAIL

La integración entre backend, base de datos y GraphHopper quedó validada. La fase también reveló un hallazgo F4: el servidor Flutter web responde, pero la aplicación renderiza una página en blanco en Chromium durante el *smoke* técnico.

3. Diferencia entre F3 y F4

F3 y F4 pueden tocar endpoints similares, pero no validan el mismo riesgo.

F3 corresponde a integración backend/API. Su objetivo es comprobar que Laravel cumple sus contratos HTTP desde una suite controlada de PHPUnit: rutas, controladores, middleware, permisos, códigos de respuesta, estructura JSON y persistencia de prueba. En F3 no se levanta el frontend Flutter, no se valida renderizado en navegador y los servicios externos pueden estar simulados o controlados para aislar el contrato de la API.

F4 corresponde a sistema integrado local. Su objetivo es comprobar que los componentes reales del stack local trabajan juntos: backend Laravel, frontend Flutter web, base de datos local, GraphHopper

Docker, media storage y variables de entorno. Por eso F4 usa peticiones HTTP reales contra el backend local, consulta GraphHopper real y verifica que el frontend web responda y renderice en navegador.

Fase	Pregunta principal	Evidencia esperada
F3	¿La API backend cumple sus contratos si se prueba de forma controlada?	Salida PHPUnit, respuestas JSON, códigos HTTP, base de datos de prueba y cobertura `coverageF3`.
F4	¿El sistema completo local funciona cuando sus componentes reales están conectados?	Health checks, estado de contenedores, peticiones HTTP reales, persistencia local, respuesta GraphHopper y captura del frontend.

Esta diferencia explica el hallazgo F4-017: las pruebas F3 pueden pasar correctamente aunque el frontend Flutter web renderice una página en blanco, porque F3 no abre ni valida el frontend en navegador.

4. Procedimiento de Ejecución

El procedimiento se automatizó en `scripts/f4-system-check.js` para que la validación pueda repetirse con el mismo criterio técnico:

1. Crear la carpeta de evidencia `docs/evidencias/F4`.
2. Levantar o reutilizar GraphHopper con `docker compose up -d graphhopper`.
3. Esperar disponibilidad de GraphHopper en `http://127.0.0.1:8989`.
4. Levantar o reutilizar Laravel en `http://127.0.0.1:8000` y validar `GET /api/health`.
5. Ejecutar migraciones si el backend es iniciado por el *harness*.
6. Generar `frontend/env/runtime.local.json` con las URLs locales del backend.
7. Levantar Flutter web-server en `http://127.0.0.1:7357` usando `--dart-define-from-file=env/runtime.local.json`.
8. Ejecutar peticiones HTTP reales contra Laravel para registro, login, mascotas, peligros, reportes, comentarios, rutas, planeación, logout, health y media upload.
9. Planear rutas mediante Laravel usando GraphHopper local real.
10. Abrir el frontend con Playwright Chromium, capturar consola, errores de página, request failures, superficie Flutter y screenshot.
11. Guardar resultados estructurados en `f4_system_check_results.json`.
12. Guardar captura del frontend en `f4_frontend_smoke.png`.
13. Cerrar los procesos locales iniciados por el *harness*.

El *harness* no reemplaza F5, porque no ejecuta flujos completos de usuario final con interacciones visuales profundas. F4 se enfoca en comprobar comunicación técnica real entre componentes y dejar defectos de integración listos para investigar antes de la fase E2E.

5. Comandos Ejecutados

5.1. Validación de sintaxis del harness

```
node --check scripts\f4-system-check.js
```

Resultado: Exit code: 0.

5.2. Ejecución F4

```
node scripts\f4-system-check.js
```

Caso	Resultado	Descripción
F4-001	PASS	Backend, frontend web-server y GraphHopper disponibles.
F4-002	PASS	Registro de usuario persistido mediante API Laravel real.
F4-003	PASS	Token de login válido contra endpoint backend protegido.
F4-004	PASS	Mascota creada y devuelta desde listado autenticado del propietario.
F4-005	PASS	Datos fuente de marcadores de peligro disponibles por API pública.
F4-006	PASS	Peligro creado con coordenadas manuales y persistido.
F4-007	PASS	Backend suministra datos de peligros filtrados por riesgo para capas del mapa.
F4-008	PASS	Reporte de mascota perdida creado con asociación persistida a mascota.
F4-009	PASS	Reporte de mascota encontrada creado y vinculado al reporte de pérdida.
F4-010	PASS	Backend planeó una ruta real mediante GraphHopper local.
F4-011	PASS	Ruta de viaje creada y persistida por backend.
F4-012	PASS	Comentario de peligro persistido y retornado con datos de autor.
F4-013	PASS	Logout revoca token y la API privada rechaza el token anterior.
F4-014	PASS	Datos fuente de marcadores de mascotas perdidas disponibles por API pública.
F4-015	PASS	Coordenadas manuales persistidas en payloads de reportes.
F4-016	PASS	Planeación considera peligros persistidos de alto riesgo.
F4-018	PASS	Health permanece OK después de operaciones integradas.
F4-020	PASS	Radio de reporte genérico persistido para círculo de impacto en mapa.
F4-021	PASS	Payload de instrucciones de navegación retornado para la ruta planeada.
F4-022	PASS	Endpoint de imagen almacena media utilizable por reportes de peligro.
F4-017	FAIL	Frontend responde por HTTP, pero la aplicación renderiza una página en blanco en Chromium.
F4-000	FAIL	Frontend no expuso texto visible ni superficie de renderizado Flutter. Failed requests: [].

El exit code final fue 1 debido al hallazgo de renderizado frontend F4-017. Los casos técnicos previos quedaron ejecutados y registrados como PASS.

6. Evidencias Generadas

Evidencia	Ruta
Resultado estructurado F4	`docs/evidencias/F4/f4_system_check_results.json`
Captura del frontend web	`docs/evidencias/F4/f4_frontend_smoke.png`
Script de ejecución	`scripts/f4-system-check.js`

6.1. Datos relevantes registrados

Dato	Valor
GraphHopper	`a4mpets-graphhopper Up 5 hours (healthy)`
Backend base URL	`http://127.0.0.1:8000/api`
Frontend URL	`http://127.0.0.1:7357`
GraphHopper URL	`http://127.0.0.1:8989`
Ruta planeada F4-010	967.585 m / 683.695 s / 19 coordenadas
Instrucciones F4-021	17 instrucciones, perfil `foot`
Peligros considerados F4-016	6
Imagen subida F4-022	URL `/api/media/hazards/...png`

7. Matriz de Casos F4

Caso	Resultado	Evidencia técnica
F4-001	PASS	Backend, frontend web-server y GraphHopper disponibles; contenedor `a4mpets-graphhopper` healthy.
F4-002	PASS	Registro por API Laravel real con usuario persistido.
F4-003	PASS	Login real y token válido contra `/api/auth/me`.
F4-004	PASS	Mascota creada y devuelta por listado autenticado.
F4-005	PASS	Datos de peligro disponibles como fuente de marcadores.
F4-006	PASS	Reporte de peligro creado con coordenadas manuales y persistido.
F4-007	PASS	Filtro de peligros por riesgo devuelve datos compatibles con capas del mapa.
F4-008	PASS	Reporte de mascota perdida creado con asociación a mascota.
F4-009	PASS	Reporte de mascota encontrada creado y vinculado al reporte de pérdida.
F4-010	PASS	Laravel planea ruta usando GraphHopper local real.
F4-011	PASS	Ruta de viaje creada y persistida.
F4-012	PASS	Comentario de peligro persistido y devuelto con datos de autor.
F4-013	PASS	Logout revoca token y API privada rechaza el token anterior.
F4-014	PASS	Reporte de mascota perdida disponible como fuente de marcador.
F4-015	PASS	Coordenadas manuales persistidas en reportes.
F4-016	PASS	Planeación considera peligros persistidos de alto riesgo.
F4-017	FAIL	Frontend responde por HTTP, pero Chromium muestra página en blanco: `bodyTextLength=0`, `flutterSurfaceCount=0`, sin request failures.
F4-018	PASS	`/api/health` permanece estable después de operaciones integradas.
F4-019	NO EJECUTADO	Cloudflare Tunnel no estaba dentro del stack local ejecutado por este *harness*. Queda para F7/no funcional o ejecución manual con URL pública.
F4-020	PASS	Reporte genérico guarda radio de impacto para círculo del mapa.
F4-021	PASS	Payload de instrucciones de navegación devuelto por la planeación de ruta.
F4-022	PASS	Upload de imagen base64 almacena media utilizable por reportes.

8. Cobertura Técnica por Componente

Componente	Validación F4
Laravel API	Health, autenticación, mascotas, peligros, comentarios, reportes, rutas, planeación y media.
Base de datos local	Persistencia verificada mediante creación y lectura posterior de recursos.
GraphHopper	Contenedor activo y cálculo real de ruta a través de `/api/routes/plan`.
Flutter web	Servidor local responde, pero el renderizado falla en Chromium headless.
Variables de entorno frontend	Se generó `frontend/env/runtime.local.json` con URLs locales antes de levantar Flutter.
Archivos/media	Upload real de imagen y URL devuelta por backend.

9. Hallazgos F4

#	Hallazgo	Evidencia	Impacto	Tratamiento
F4-1	El frontend Flutter web queda en página blanca durante el smoke técnico en Chromium headless.	`f4_frontend_smoke.png`; `_check_results.json` con F4-017 FAIL, `bodyTextLength=0`, `flutterSurfaceCount=0`, `requestFailures=[]`.	Bloquea la validación visual integrada de mapa, paneles y flujos desde navegador dentro de F4.	Registrar defecto para corrección antes de F5; backend, DB y GraphHopper quedan validados por F4.
F4-2	F4-019 no se ejecutó porque Cloudflare Tunnel no fue levantado por el *harness* local.	No existe URL pública en la evidencia F4.	No afecta la validación local principal, pero impide confirmar acceso externo.	Ejecutar en F7/no funcional o en una corrida manual con tunnel activo.

10. Conclusiones

La fase F4 fue ejecutada con el stack local y generó evidencia reproducible. La comunicación backend/base de datos/GraphHopper funciona correctamente para autenticación, persistencia, reportes, comentarios, rutas, planeación e imágenes.

La fase no puede declararse completamente aprobada porque F4-017 detectó que el frontend Flutter web responde por HTTP, pero renderiza una página en blanco en Chromium. Este hallazgo debe corregirse o investigarse antes de cerrar los flujos visuales de F5.

Apéndice N

Reporte F5

El presente anexo corresponde al reporte F5, asociado a la ejecución de pruebas end-to-end (E2E) y pruebas funcionales de caja negra sobre la plataforma A4MPets. Esta fase tuvo como propósito validar el comportamiento integral del sistema desde la perspectiva de una persona usuaria final, evaluando flujos completos de interacción sin depender del conocimiento interno del código fuente o de la estructura interna de los componentes.

Las pruebas ejecutadas durante esta etapa se enfocaron en reproducir escenarios reales de uso dentro de la aplicación, incluyendo procesos de autenticación, gestión de mascotas, creación de reportes, interacción con el mapa, planificación de rutas, inicio de viajes y visualización de información geoespacial. De esta manera, se verificó que las funcionalidades principales de la plataforma operaran correctamente bajo condiciones normales de uso.

Para la automatización de estas pruebas se utilizaron herramientas especializadas de navegación y simulación de interacción en navegador, permitiendo validar comportamiento visual, navegación entre paneles, persistencia de información y correcta integración entre frontend y backend desde una perspectiva funcional.

A diferencia de las fases anteriores, esta etapa priorizó la experiencia observable del usuario y la estabilidad funcional de la plataforma completa, evaluando la respuesta del sistema frente a diferentes escenarios de interacción y condiciones de ejecución.

El reporte incluido a continuación documenta la estrategia utilizada, los flujos funcionales evaluados, las herramientas empleadas y los resultados obtenidos durante el proceso de pruebas end-to-end del sistema.

1. Reporte de Evidencias - Fase F5: Pruebas E2E de Caja Negra en Navegador

Campo	Valor
Proyecto	A4MPets
Fecha de ejecución	2026-05-09, hora local America/Bogotá
Fase	F5 - Caja negra E2E navegador
Herramienta	Playwright / Chromium / Flutter Web / Laravel / GraphHopper / base de datos local
Entorno	Stack local en `C:\projects\Web\A4MPets`

2. Resumen Ejecutivo

La fase F5 fue implementada y ejecutada como una suite E2E de caja negra en navegador usando Playwright sobre Chromium. A diferencia de F3 y F4, estas pruebas no llaman servicios internos ni controladores directamente como atajo funcional: interactúan con la aplicación desde la interfaz final de Flutter Web, simulando acciones de una persona usuaria.

La suite valida flujos completos sobre el stack local:

- Frontend Flutter Web servido en navegador.
- Backend Laravel real.
- Base de datos local real.
- GraphHopper local levantado por Docker.
- Comunicación HTTP real entre frontend y backend.
- Render, formularios, paneles, mapa, marcadores, rutas, reportes y comentarios desde la UI.

Área	Resultado
Stack local F5	PASS
GraphHopper local	PASS
Backend Laravel `/api/health`	PASS
Flutter Web en Chromium	PASS
Activación de `flt-semantics`	PASS
Suite Playwright F5	23 passed
Estado final F5	Aprobada

3. Diferencia Frente a F3 y F4

F3 valida integración backend/API desde Laravel con pruebas Feature, por lo que comprueba contratos HTTP, validaciones, permisos y persistencia desde el lado del backend.

F4 valida técnicamente que el sistema integrado local pueda levantarse y comunicarse: backend, frontend, base de datos y GraphHopper. Su objetivo es confirmar el estado del stack completo.

F5 valida el sistema desde el navegador como caja negra. La prueba no conoce controladores, modelos ni servicios internos; observa botones, formularios, mensajes visibles, paneles, mapa y respuestas reales provocadas por acciones de usuario. Por eso F5 complementa a F3/F4 en lugar de repetirlas.

4. Problemas Encontrados y Soluciones

Problema	Impacto	Solución aplicada
Flutter Web no siempre estaba listo cuando Playwright iniciaba.	Chromium podía abrir una página aun sin superficie Flutter renderizada.	Se implementó `scripts/f5-stack.js` con readiness de Laravel, GraphHopper, Flutter, `main.dart.js` y detección de `flutter-view/canvas`.
La UI de Flutter Web se renderiza principalmente en canvas.	Selectores DOM como `getByText()` no eran confiables para botones y formularios.	Se adaptaron los helpers E2E para activar y usar `flt-semantics`.
Algunos controles no tenían etiquetas accesibles estables.	Playwright no podía encontrar botones flotantes o acciones específicas sin usar coordenadas frágiles.	Se agregaron labels `Semantics/Tooltip` mínimos en controles útiles también para accesibilidad real.
Varias pruebas iniciales no eran concluyentes.	Algunos casos solo hacían clic en un panel sin confirmar edición, borrado o envío real.	Se reforzaron las pruebas para llenar formularios, esperar requests reales y verificar cambios visibles.
La sesión se perdía al recargar la app durante los tests.	Los flujos autenticados fallaban después del login.	Se dejó de limpiar `localStorage` en cada apertura y se mantuvo la sesión durante la suite serial.
Flutter Web debug es lento para flujos largos.	Algunos casos complejos excedían el timeout base.	Se ajustaron timeouts por caso y se mantuvo una suite serial por dependencia de estado de usuario.

5. Implementación Realizada

5.1. Infraestructura E2E

Se agregó un proyecto Playwright para F5:

Archivo	Propósito
`playwright.f5.config.js`	Configura Chromium, timeouts, reportes, `webServer` y salida de artefactos F5.
`scripts/f5-stack.js`	Levanta/verifica GraphHopper, Laravel y Flutter Web; expone endpoint `/ready` para Playwright.
`tests/e2e/f5/helpers.js`	Centraliza helpers para abrir la app, activar semantics, llenar campos, hacer clicks y esperar respuestas API provocadas por UI.
`tests/e2e/f5/f5-e2e.spec.js`	Contiene los flujos funcionales principales F5-001 a F5-014 y F5-017 a F5-022.
`tests/e2e/f5/f5-responsive.spec.js`	Contiene validación responsive F5-015/F5-016.

También se agregó el script:

```
"test:e2e:f5": "playwright test --config=playwright.f5.config.js"
```

5.2. Ajustes Mínimos de Accesibilidad para E2E

Se realizaron cambios puntuales en frontend para exponer acciones reales a la capa de accesibilidad de Flutter Web:

Archivo	Ajuste
`widgets/map_widget.dart`	Labels para abrir marcadores de peligro, mascota perdida, mascota encontrada y reporte genérico.
`pages/map_page_components.dart`	Labels para seleccionar mascotas, seleccionar rutas, agregar mascota para viaje e iniciar viaje.
`widgets/pet_management_panels.dart`	Labels para seleccionar y editar una mascota específica.
`widgets/route_management_panels.dart`	Labels para seleccionar una ruta específica.
`widgets/panel_modal.dart`	Labels para expandir y minimizar paneles.

Estos cambios no reemplazan la validación visual; permiten que Playwright encuentre controles reales de la app sin depender de coordenadas absolutas.

5.3. Ajustes Funcionales Necesarios para E2E

Durante la validación se identificaron detalles que impedían que el flujo navegador-backend fuera estable:

Archivo	Ajuste
`models/pet_record.dart`	Lectura robusta de JSON de mascotas devuelto por backend.
`repositories/user_local_repository.dart`	Peticiones autenticadas de mascotas y rutas incluyen token bearer.
`providers/pet_providers.dart`	Recarga de datos tras mutaciones de mascotas.
`providers/route_providers.dart`	Recarga de datos tras mutaciones de rutas.
`providers/async_list_controller.dart`	Flag para desactivar refrescos de fondo durante F5.
`services/backend_api_client.dart`	Timeout de fallback ampliado para stack local lento.

6. Procedimiento de Ejecución

1. Se revisó que Playwright pudiera iniciar Chromium y abrir Flutter Web.
2. Se implementó `scripts/f5-stack.js` como servidor de preparación de entorno.
3. El stack verificó GraphHopper local en `http://127.0.0.1:8989/health`.
4. El stack verificó Laravel en `http://127.0.0.1:8000/api/health`.
5. Se generó `frontend/env/runtime.local.json` apuntando al backend local.
6. Se levantó Flutter Web en `http://127.0.0.1:7357`.
7. Playwright esperó el endpoint `http://127.0.0.1:7358/ready`.
8. Cada test abrió la app, activó la capa `flt-semantics` y ejecutó acciones de usuario.
9. Para acciones críticas, los tests esperaron la request HTTP real producida por la UI.
10. Al finalizar, Playwright generó reporte HTML, JSON y artefactos de diagnóstico.

7. Comandos Ejecutados

Validación sintáctica de scripts y specs:

```
node --check scripts\f5-stack.js
node --check playwright.f5.config.js
node --check tests\e2e\f5\helpers.js
node --check tests\e2e\f5\f5-e2e.spec.js
node --check tests\e2e\f5\f5-responsive.spec.js
```

Validación del frontend tras ajustes mínimos:

```
flutter analyze
```

Ejecuciones focalizadas usadas durante corrección:

```
npx playwright test --config=playwright.f5.config.js --project=F5-desktop-chromium -g "F5-001 registı
npx playwright test --config=playwright.f5.config.js --project=F5-desktop-chromium -g "F5-001 registı
npx playwright test --config=playwright.f5.config.js --project=F5-desktop-chromium -g "F5-001 registı
npx playwright test --config=playwright.f5.config.js --project=F5-desktop-chromium -g "F5-001 registı
npx playwright test --config=playwright.f5.config.js --project=F5-desktop-chromium tests/e2e/f5/f5-r
```

Ejecución final:

```
npx playwright test --config=playwright.f5.config.js --project=F5-desktop-chromium
```

Resultado final:

23 passed (28.3m)
Exit code: 0

8. Ejecuciones Focalizadas Relevantes

Comando focalizado	Resultado
`F5-001 registro | F5-002 login`	Aprobado
`F5-001 registro | F5-004 | F5-005`	Aprobado
`F5-001 registro | F5-006 | F5-007`	Aprobado
`F5-001 registro | F5-011`	Aprobado
`F5-001 registro | F5-019`	Aprobado
`F5-001 registro | F5-020`	Aprobado
`f5-responsive.spec.js`	1 passed

9. Evidencias Generadas

Evidencia	Ruta
Reporte principal F5	`docs/evidencias/F5/reporte_F5_e2e_navegador.md`
Reporte HTML Playwright	`docs/evidencias/F5/playwright-report/index.html`
Log del stack F5	`docs/evidencias/F5/f5-stack.log`
Artefactos temporales Playwright	`test-results/f5`
Diagnósticos por caso	`test-results/f5-fixtures`
Configuración Playwright	`playwright.f5.config.js`
Specs E2E	`tests/e2e/f5`

Nota: `test-results` es una carpeta temporal administrada por Playwright y puede limpiarse al iniciar nuevas corridas. La evidencia formal queda centralizada en `docs/evidencias/F5`, y el resultado final concluyente se documenta en este reporte como **23 passed (28.3m)**.

10. Resultado por Caso

Caso	Resultado	Validación concluyente realizada
F5-001	PASS	Registro real de usuario único desde UI y sesión activa posterior.
F5-001-extra	PASS	Registro con campos obligatorios vacíos y validación visible.
F5-002	PASS	Login real, acceso a gestión de mascotas y logout desde la interfaz.
F5-002-extra	PASS	Login inválido y mensaje de error visible.
F5-003	PASS	Creación real de mascota con nombre, fecha y foto desde panel de UI.
F5-004	PASS	Edición real de mascota existente, espera de `PUT /api/pets/{id}` y verificación del nuevo nombre en UI.
F5-005	PASS	Eliminación real de mascota, confirmación, espera de `DELETE /api/pets/{id}` y ausencia en UI.
F5-006	PASS	Creación real de reporte de mascota perdida desde mapa, formulario lleno y request `POST` validada.
F5-007	PASS	Creación real de reporte de mascota encontrada vinculado a reporte perdido.
F5-008	PASS	Creación real de reporte de peligro con ubicación, descripción, radio, categoría y foto.
F5-009	PASS	Creación real de reporte genérico desde la capa de reportes del mapa.
F5-010	PASS	Apertura de marcador de peligro y comentario real sobre el reporte.
F5-011	PASS	Creación de ruta desde UI, planificación real y validación de coordenadas de ruta.
F5-012	PASS	Planificación de viaje con filtros de peligro y confirmación de inicio de viaje.
F5-013	PASS	Usuario no autenticado bloqueado en acciones privadas desde UI.
F5-014	PASS	Aislamiento entre usuarios: datos de otro usuario no aparecen en gestión propia.
F5-015	PASS	Viewport escritorio sin overflow horizontal crítico.
F5-016	PASS	Viewport móvil validado tras redimensionar navegador, sin overflow crítico.
F5-017	PASS	Edición real de ruta existente y verificación de actualización visible.
F5-018	PASS	Eliminación real de ruta, confirmación y espera de `DELETE /api/routes/{id}`.
F5-019	PASS	Mascota marcada como encontrada con `POST /api/pets/{id}/mark-found`; se verifica que el marcador perdido desaparece.
F5-020	PASS	Formulario de reporte acepta texto extendido y lo conserva en el flujo.
F5-021	PASS	Comentarios reales en peligro, mascota perdida, mascota encontrada y reporte genérico.
F5-022	PASS	Smoke final sin errores críticos de consola ni respuestas HTTP 500.

11. Corrección de Pruebas No Concluyentes

Durante la revisión se detectó que algunas pruebas iniciales solo navegaban o hacían clic sin comprobar el efecto real. Se reforzaron así:

Caso	Antes	Después
F5-004	Abría el flujo de editar mascota.	Edita el nombre, espera `PUT /api/pets/{id}` y verifica el nombre actualizado.
F5-005	Abría el flujo de eliminación.	Crea una mascota temporal, confirma borrado, espera `DELETE /api/pets/{id}` y valida ausencia.
F5-006	No garantizaba que el formulario de pérdida estuviera completo.	Llena mascota, descripción, ubicación, contacto, envía y valida request `POST`.
F5-011	No garantizaba ruta planificada.	Espera `/api/routes/plan`, valida respuesta exitosa y coordenadas.
F5-012	No verificaba el camino con filtros.	Valida request de planificación con evitación/filtros antes de iniciar viaje.
F5-018	Solo accedía al flujo de ruta.	Confirma eliminación y espera `DELETE /api/routes/{id}`.
F5-019	Requería más tiempo en suite completa.	Se amplió timeout del caso y se valida el `POST mark-found` real.
F5-021	Era demasiado general.	Crea comentarios reales sobre los cuatro tipos principales de reportes.

12. Consideraciones de Caja Negra

Las pruebas F5 interactúan por UI:

- Clicks sobre botones y paneles visibles.
- Escritura en campos de formulario.
- Selección de ubicaciones en el mapa.
- Confirmaciones de diálogos.
- Navegación responsive en Chromium.

Las esperas de red se usan como comprobación del efecto producido por la UI, no como atajo para crear datos saltándose la interfaz. Las únicas verificaciones directas de backend previas corresponden al readiness técnico del stack: `/api/health`, GraphHopper health y carga de Flutter Web.

13. Hallazgos F5

#	Hallazgo	Tratamiento
F5-1	Flutter Web con canvas no es fiable con selectores DOM tradicionales.	Se migró la suite a helpers basados en `flt-semantics`.
F5-2	Algunas acciones de mapa requieren coordenadas porque son interacciones espaciales.	Se limitaron coordenadas a selección de ubicación/puntos de ruta, donde es parte natural del flujo.
F5-3	La suite completa es lenta en Flutter debug web-server.	Se documentó el tiempo final y se mantiene ejecución serial por flujos encadenados.
F5-4	`test-results` puede limpiarse entre corridas.	La evidencia formal se almacena en `docs/evidencias/F5`.

14. Conclusiones

La fase F5 queda aprobada. Playwright ejecuta los flujos E2E en Chromium contra el stack local completo y valida desde la interfaz final los caminos principales de usuario: autenticación, mascotas,

reportes, peligros, rutas, viaje, comentarios, permisos, aislamiento de datos, responsive y ausencia de errores críticos.

La corrida final confirmada fue:

23 passed (28.3m)

Con esto F5 queda separada conceptualmente de F3/F4 y aporta evidencia real de comportamiento de usuario en navegador.

Apéndice Ñ

Reporte F5 - Playwright

El presente anexo corresponde al reporte automatizado generado durante la ejecución de las pruebas end-to-end de la fase F5 mediante la herramienta Playwright.

El reporte fue generado automáticamente a partir de los flujos de interacción ejecutados sobre la versión web de la plataforma A4MPets, incluyendo validaciones relacionadas con navegación, formularios, autenticación, visualización geográfica, reportes y rutas.

Debido a que la herramienta produce una interfaz HTML interactiva con navegación interna, capturas, trazas de ejecución y métricas detalladas del comportamiento observado durante las pruebas, el reporte fue conservado en su formato original.

Con el fin de preservar la integridad de la evidencia generada y facilitar su consulta posterior, el reporte completo se adjunta como archivo pdf bajo el nombre:

- `playwright-report.pdf`

El archivo contiene el reporte HTML completo generado por Playwright, incluyendo resultados de ejecución, capturas automáticas, trazas y evidencias asociadas al proceso de validación funcional del sistema.

Apéndice O

Reporte F6

La fase F6 correspondió a la ejecución de pruebas manuales funcionales y exploratorias sobre el sistema A4MPets, enfocadas principalmente en la validación del comportamiento visual, la interacción del usuario con la interfaz y la robustez funcional de los distintos módulos integrados.

A diferencia de las fases anteriores orientadas a cobertura de código, pruebas unitarias o integración automatizada, esta etapa estuvo centrada en la experiencia de uso real del sistema, evaluando manualmente formularios, navegación, permisos, consistencia visual, mensajes de error, manejo de estados vacíos, responsive design y validaciones de entrada.

Durante la ejecución se realizaron pruebas sobre los módulos de autenticación, mascotas, rutas, reportes de peligros, comentarios, geolocalización y visualización cartográfica, verificando tanto escenarios exitosos como condiciones de fallo y recuperación.

Asimismo, esta fase permitió identificar defectos relacionados con visualización de información, persistencia de estados y control de acceso visual entre usuarios, los cuales fueron documentados dentro del informe técnico correspondiente.

Las evidencias gráficas obtenidas durante la ejecución manual de esta fase se presentan en el Apéndice P, mientras que el presente documento contiene el detalle técnico de los resultados obtenidos, hallazgos registrados y análisis funcional realizado durante las pruebas.

width=!,height=!, pages=-, pagecommand=, width=

Apéndice P

Reporte F6 - Evidencias

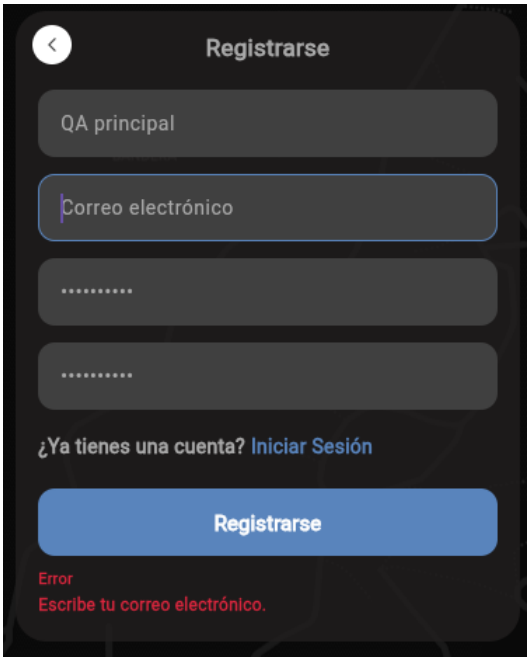
P.1. Evidencias de ejecución de F6

Este anexo presenta las principales evidencias gráficas obtenidas durante la ejecución de la fase F6 correspondiente a pruebas manuales funcionales y exploratorias del sistema A4MPets.

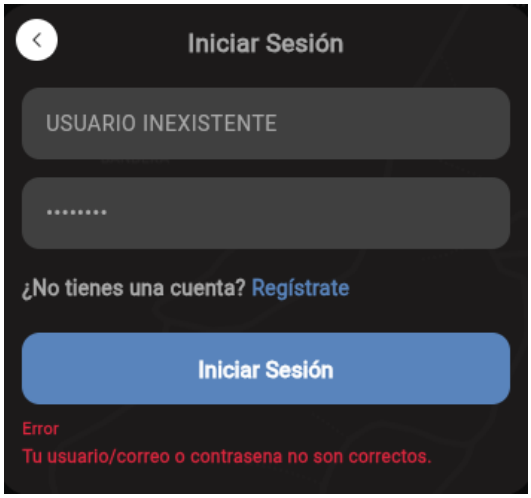
Las capturas fueron obtenidas durante la validación manual de formularios, permisos, estados visuales, manejo de errores, consistencia de interfaz, comportamiento responsive, visualización de peligros, comentarios, mascotas, rutas y funcionalidades asociadas al mapa geoespacial.

Las evidencias complementan el informe técnico presentado en el Apéndice O y permiten verificar visualmente los resultados documentados en la matriz de ejecución de pruebas manuales funcionales.

P.1.1. Validación de mensajes de error en autenticación



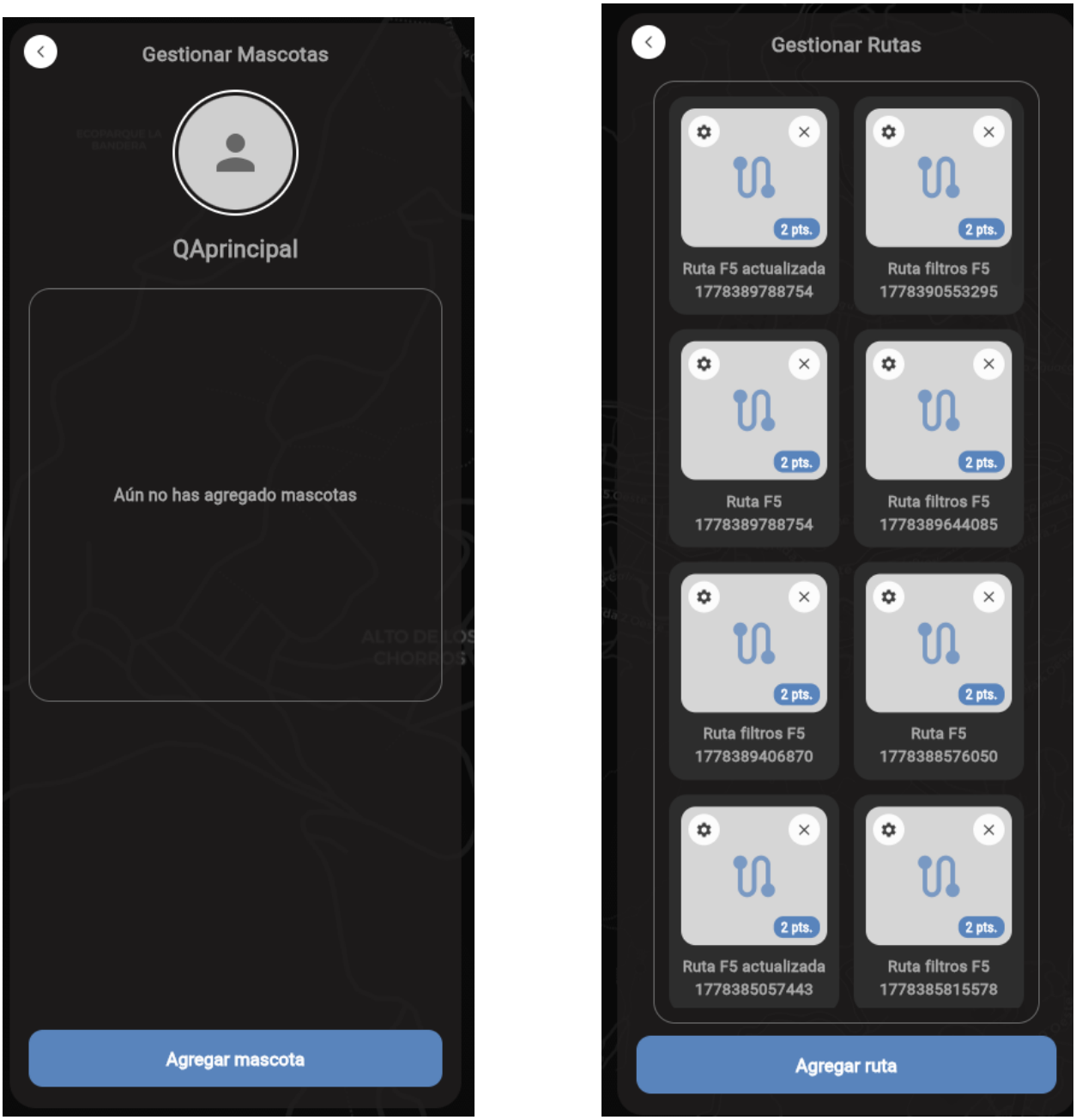
(a) Mensajes de validación durante el registro de usuario.



(b) Mensajes de error durante autenticación inválida.

Figura P.1: Evidencias correspondientes a validaciones de autenticación manual ejecutadas durante F6.

P.1.2. Validación de estados vacíos



(a) Estado vacío correcto para mascotas.

(b) Defecto detectado en visualización de rutas.

Figura P.2: Estados visuales asociados a mascotas y rutas durante pruebas manuales funcionales.

La Figura P.2 evidencia el defecto funcional DEF-F6-001, donde una cuenta nueva visualiza rutas que no le pertenecen. Este comportamiento fue clasificado con severidad media durante la fase F6.

P.1.3. Creación y visualización de peligros

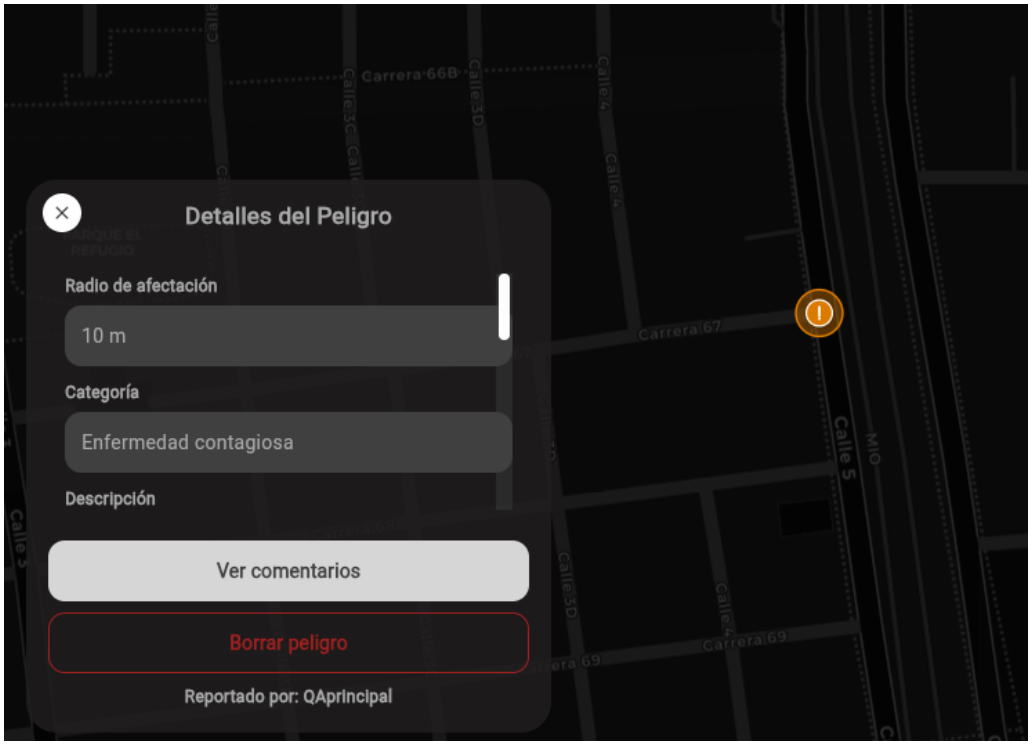
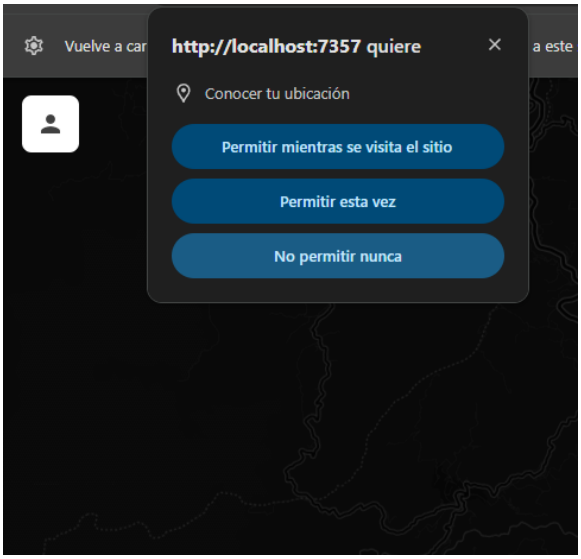


Figura P.3: Visualización del detalle de un peligro y su radio de afectación en el mapa.

La evidencia muestra la correcta representación visual de peligros geoespaciales y su integración con el mapa interactivo del sistema.

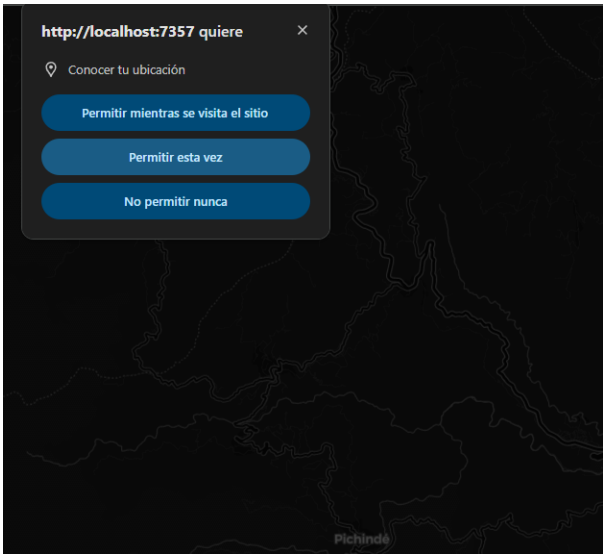
P.1.4. Validación de permisos de ubicación



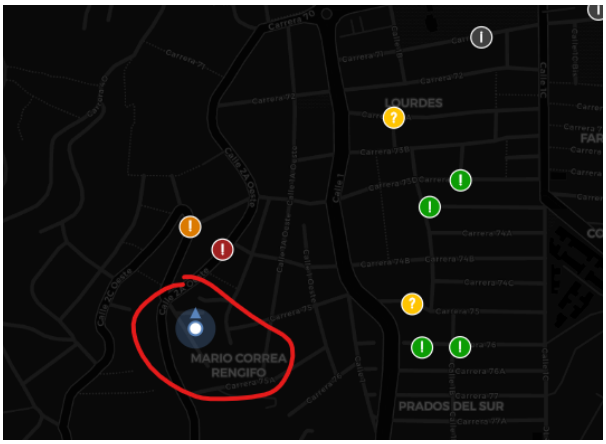
(a) Solicitud de permisos de ubicación desde el navegador.

(b) Comportamiento del sistema tras denegar permisos.

Figura P.4: Validación funcional de permisos de geolocalización denegados.



(a) Aceptación de permisos de ubicación.

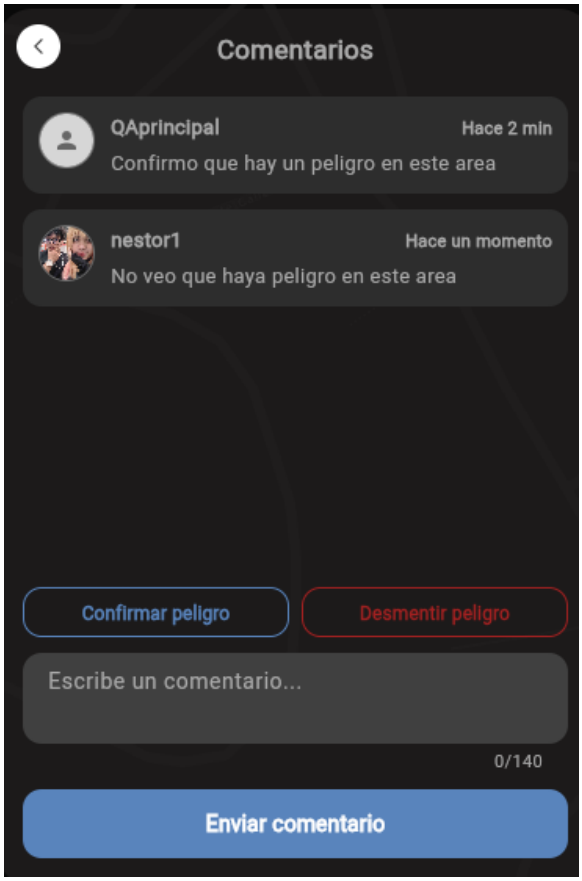


(b) Marcador de ubicación mostrado correctamente.

Figura P.5: Validación funcional de permisos de ubicación aceptados.

P.1.5. Caracteres especiales y comentarios

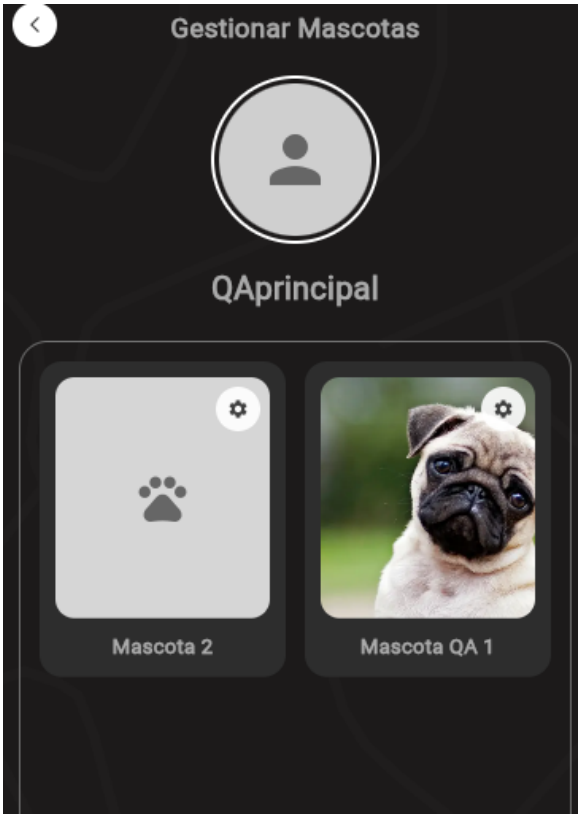
(a) Persistencia correcta de caracteres especiales.



(b) Visualización cronológica de comentarios.

Figura P.6: Validaciones manuales relacionadas con comentarios y caracteres especiales.

P.1.6. Gestión de mascotas



(a) Panel de mascotas con registros existentes.

(b) Validación del campo telefónico en reportes.

Figura P.7: Evidencias relacionadas con gestión de mascotas y validación de datos.

P.1.7. Diferenciación visual de niveles de peligro

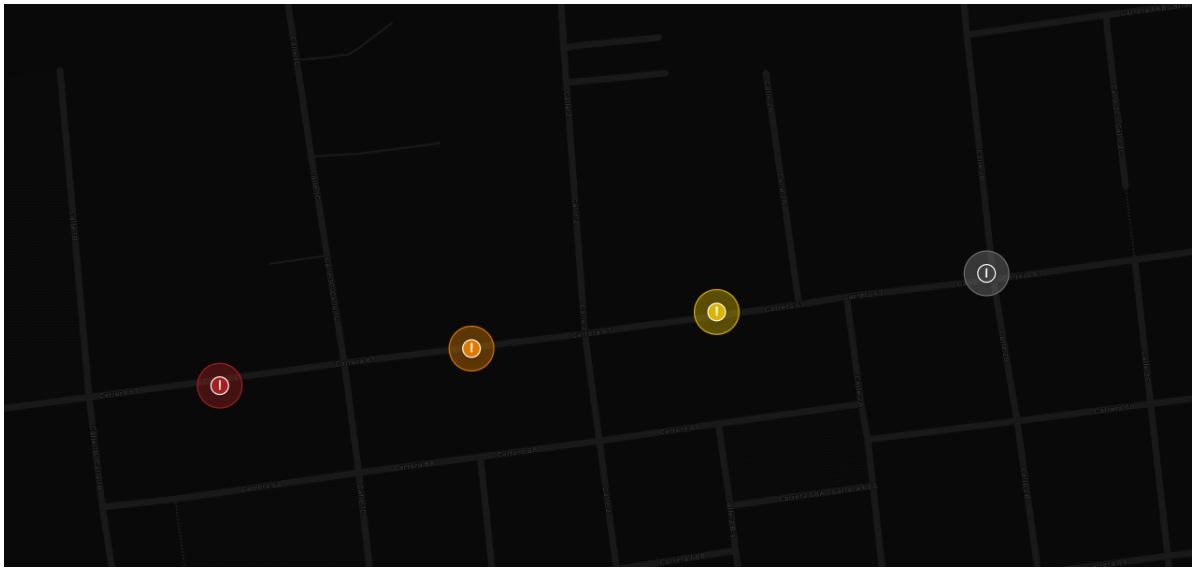


Figura P.8: Representación visual diferenciada de peligros según nivel de riesgo.

La representación visual permitió distinguir correctamente los niveles bajo, medio y alto mediante colores y estilos diferentes sobre el mapa geoespacial.

P.1.8. Validación de fechas

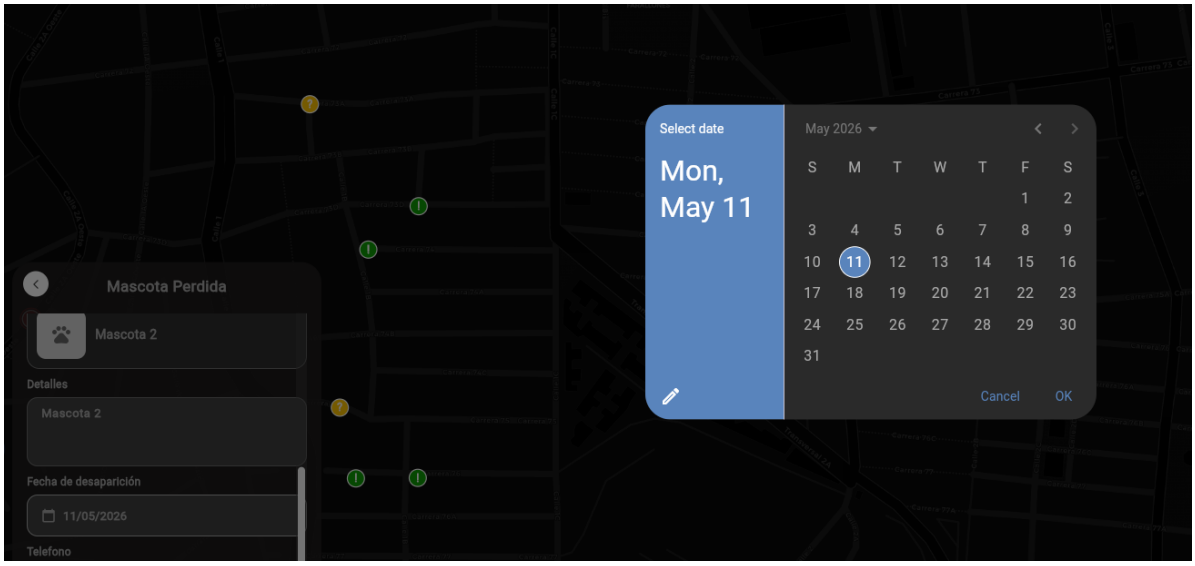
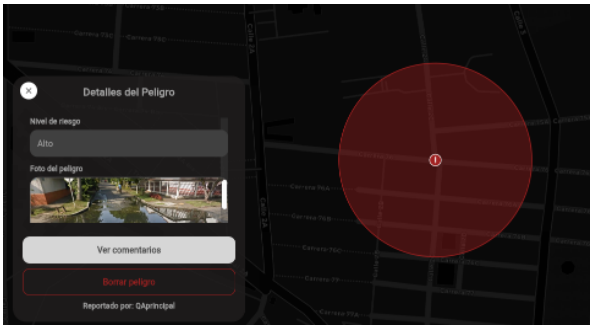
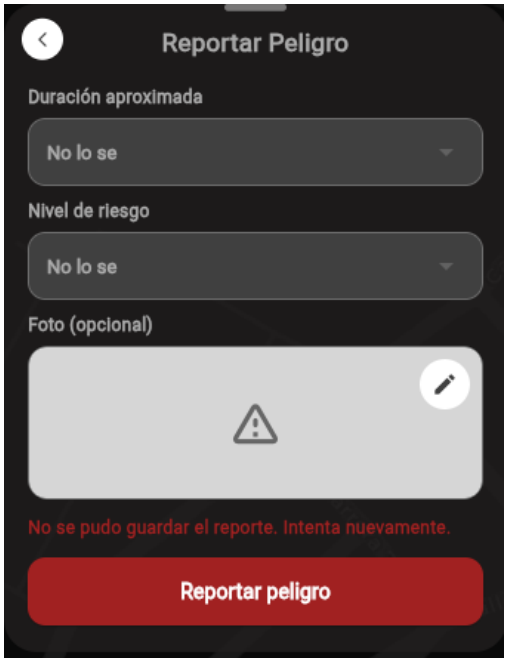


Figura P.9: Selector de fechas utilizado durante el reporte de mascotas perdidas.

P.1.9. Manejo de errores y datos completos



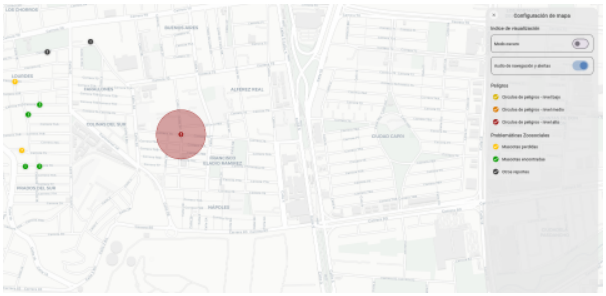
(a) Detalle completo de peligro con imagen asociada.



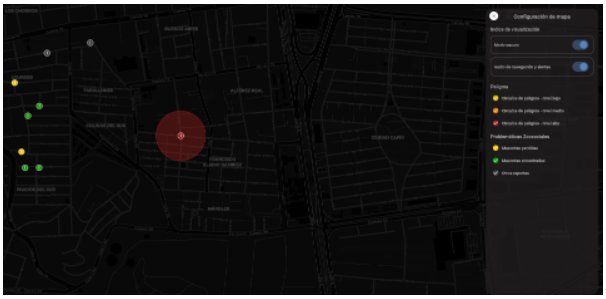
(b) Recuperación visual ante error de red.

Figura P.10: Validaciones funcionales relacionadas con manejo de errores y persistencia visual.

P.1.10. Legibilidad y contraste visual



(a) Modo claro del sistema.

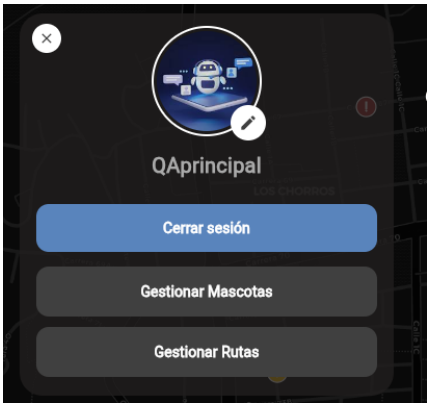


(b) Modo oscuro del sistema.

Figura P.11: Comparación visual entre modos claro y oscuro durante pruebas de legibilidad.

P.1.11. Carga de archivos y panel de perfil

(a) Manejo de reportes sin fotografía válida.



(b) Panel de perfil del usuario autenticado.

Figura P.12: Evidencias relacionadas con carga de archivos y visualización de perfil.

Apéndice Q

Reporte F7

La fase F7 estuvo orientada a la ejecución de pruebas no funcionales y pruebas de carga controlada sobre la arquitectura integrada de A4MPets, con el propósito de evaluar el comportamiento general del sistema bajo múltiples solicitudes consecutivas y validar la estabilidad de los servicios desplegados localmente.

A diferencia de las fases centradas en validaciones funcionales o de experiencia de usuario, esta etapa se enfocó principalmente en aspectos relacionados con rendimiento básico, disponibilidad de servicios, estabilidad de endpoints, tiempos de respuesta y tolerancia frente a cargas repetitivas ejecutadas desde herramientas externas.

Las pruebas fueron realizadas utilizando Postman Runner y Cloudflare Tunnel, permitiendo exponer temporalmente los servicios locales y ejecutar múltiples iteraciones sobre endpoints críticos relacionados con autenticación, reportes, comentarios, carga de archivos y planeación de rutas.

Durante esta fase se verificó el comportamiento coordinado entre el frontend Flutter, el backend Laravel, el servicio GraphHopper y los mecanismos de persistencia utilizados por el sistema, observando además posibles degradaciones de rendimiento, errores HTTP y comportamientos inesperados bajo condiciones de uso repetitivo.

El presente informe documenta los resultados obtenidos durante dichas ejecuciones, incluyendo observaciones relacionadas con estabilidad, recuperación ante fallos, latencia y consistencia general de la arquitectura durante escenarios de carga controlada.

Las evidencias complementarias y reportes generados por las herramientas utilizadas durante esta fase se presentan en los anexos correspondientes.

1. Reporte de Evidencias - Fase F7: Performance API con Postman y Cloudflare

Campo	Valor
Proyecto	A4MPets
Fecha de ejecución	2026-05-11 a 2026-05-12
Fase	F7 - Performance API / No funcional
Herramienta	Postman Desktop Performance Testing
Entorno	Stack local A4MPets publicado mediante Cloudflare Tunnel
URL Cloudflare usada	No registrada en el entorno actual; los escenarios fueron ejecutados mediante Cloudflare según la evidencia reportada.

1.1. Resumen Ejecutivo

La fase F7 validó estabilidad, latencia, throughput y comportamiento bajo carga controlada del backend cuando el sistema local se expuso mediante Cloudflare Tunnel. Se ejecutaron escenarios baseline, carga fija, ramp-up, spike, peak, endurance corta, seguridad sin token, rutas con GraphHopper y recuperación posterior a carga.

La ejecución queda aprobada con observaciones. No se reportaron errores HTTP ni respuestas 5xx en las métricas entregadas, y los percentiles principales se mantuvieron dentro de los umbrales definidos. Las observaciones relevantes son: fallos de assertions en escenarios ramp-up, spike y peak; un máximo aislado alto en `GET /api/pets` autenticado; y un hallazgo funcional de seguridad en F7-015 porque `POST /api/pets` sin token y `POST /api/routes/plan` sin token no cumplieron la assertion esperada de rechazo sin token.

1.2. Entorno de Ejecución F7

Categoría	Valor
Sistema operativo	Ubuntu 24.04.4 LTS Noble Numbat
Kernel	Linux 6.8.0-111-generic x86_64
Equipo	Laptop Toshiba Satellite C55-B
CPU	Intel Core i3-4005U, 2 núcleos / 4 hilos, hasta 1.70 GHz
Memoria RAM	6 GiB total, 5.7 GiB disponible aproximadamente
Almacenamiento	Disco Toshiba 465.76 GiB, ext4, 170.94 GiB usados aproximadamente
Red activa	Wi-Fi Qualcomm Atheros AR9485
Docker/red local	Interfaces Docker activas durante la prueba
Pantalla	1366x768, aunque F7 se ejecuta principalmente por API
Entorno de ejecución	Sesión SSH/terminal, Postman Desktop, Cloudflare Tunnel, Docker/GraphHopper y backend local
Limitación relevante	Equipo local de recursos moderados; resultados no equivalen a infraestructura productiva

Las métricas deben interpretarse como performance local controlada. Los resultados no equivalen a un benchmark productivo porque dependen del host local, Wi-Fi, Docker/GraphHopper, Postman Desktop y Cloudflare Tunnel.

1.3. Diferencia Frente a F3, F4 y F5

Fase	Enfoque	Diferencia con F7
F3	Integración backend/API automatizada en PHPUnit	F3 valida contratos funcionales; F7 mide comportamiento bajo carga externa.
F4	Sistema integrado local de caja blanca	F4 comprueba comunicación entre componentes; F7 mide latencia, throughput y estabilidad.
F5	Caja negra E2E en navegador	F5 simula usuario desde UI; F7 usa API HTTP y Postman Performance.

1.4. Preparación

1. Levantar backend Laravel, base de datos, frontend y GraphHopper.
2. Ejecutar Cloudflare Tunnel: `npm run dev:cloudflare`.
3. Copiar la URL generada desde `.runtime/cloudflare-url.txt`.
4. Importar en Postman Desktop la colección y el environment F7.
5. Configurar variables de entorno.

Variable	Valor usado
`base_url`	URL Cloudflare temporal usada durante la ejecución; no registrada en el entorno actual.
`qa_login`	Usuario QA configurado en Postman para F7.
`qa_password`	Oculto.
`access_token`	Generado por `00 - Setup`.
`hazard_id`	Generado por `00 - Setup` o asignado manualmente.

1.5. Resultado de Setup

Paso	Resultado	Observaciones
Health inicial	Aprobado	Los escenarios baseline y de cierre confirmaron HTTP 200 y `status=ok`.
Login QA	Aprobado	F7-002 obtuvo 0.00 % error y 0.00 % failure con token válido.
Crear peligro base	Aprobado	Se usó `hazard_id` para comentarios y escenarios autenticados.

1.6. Umbrales de Aceptación

Tipo	p95 max	Error max	HTTP 5xx
Health	500 ms	1 %	0
Login y lecturas	1500 ms	1 %	0
Escrituras y media	3500 ms	2 %	0
GraphHopper/rutas	15000 ms	5 %	0
Endpoints protegidos sin token	100 % HTTP 401	N/A	0

Error % se interpreta como errores HTTP reportados por Postman. Failure % se interpreta como assertions fallidas. En la evidencia entregada no se reportaron respuestas 5xx.

1.7. Resultados por Request

Caso	Request	Perfil	VU	Dur.	Req.	Req/s	Err. %	Fail. %	Avg	Max	P95	P99
F7-001	GET /api/health	fixed	3	3m	352	5.81	0.00	0.00	100	799	146	326
F7-002	POST /api/auth/login	fixed	3	3m	74	1.19	0.00	0.00	739	1201	834	1201
F7-003	GET /api/hazards	fixed	3	3m	1703	9.35	0.00	0.00	101	743	154	224
F7-003	GET /api/lost-pet-reports	fixed	3	3m	1702	9.34	0.00	0.00	91	436	143	199
F7-003	GET /api/found-pet-reports	fixed	3	3m	1701	9.34	0.00	0.00	91	306	140	194
F7-003	GET /api/generic-reports	fixed	3	3m	1699	9.33	0.00	0.00	92	585	143	248
F7-004	GET /api/routes autenticado	fixed	3	3m	2706	14.85	0.00	0.00	126	1215	255	391
F7-004	GET /api/pets autenticado	fixed	3	3m	2703	14.83	0.00	0.04	128	8781	252	381
F7-005	POST /api/pets	fixed	3	3m	930	5.13	0.00	0.00	514	1502	890	1190
F7-006	POST /api/hazards	fixed	3	3m	760	4.20	0.00	0.00	640	2066	1221	1601
F7-007	POST /api/media/upload	fixed	3	3m	1508	8.32	0.00	0.00	305	3438	700	1383
F7-008	POST /api/routes/plan simple	fixed	3	3m	1693	9.34	0.00	0.00	264	3499	403	1625
F7-009	POST /api/routes/plan multipunto	fixed	3	3m	1621	8.90	0.00	0.00	276	1227	476	710
F7-010	POST /api/routes/plan filtros riesgo	fixed	3	3m	1580	8.74	0.00	0.00	284	1081	506	856
F7-011	GET /api/health ramp-up	3 a 10	-	3m	1927	10.58	0.00	2.65	186	2376	398	736
F7-011	GET /api/hazards ramp-up	3 a 10	-	3m	1925	10.57	0.00	0.94	229	5577	471	1318
F7-011	GET /api/routes autenticado ramp-up	3 a 10	-	3m	1922	10.55	0.00	0.31	199	2587	395	711
F7-012	GET /api/hazards spike	3 a 20	-	3m	1893	10.44	0.00	3.54	205	5031	569	844
F7-012	GET /api/health post-spike incluido	3 a 20	-	3m	1892	10.43	0.00	3.59	169	5032	481	755
F7-013	GET /api/pets autenticado peak	max 20	-	3m	621	3.43	0.00	0.81	454	1739	1023	1378
F7-013	GET /api/routes autenticado peak	max 20	-	3m	620	3.42	0.00	0.81	484	1899	1087	1423
F7-013	POST /api/hazards/id/comments peak	max 20	-	3m	620	3.42	0.00	0.00	565	1635	1115	1381
F7-014	GET /api/health endurance	fixed	5	15m	17390	19.30	0.00	0.00	81	799	163	266
F7-014	GET /api/hazards endurance	fixed	5	15m	17387	19.30	0.00	0.00	101	820	185	304
F7-015	POST /api/pets sin token	fixed	3	3m	868	4.82	0.00	100.00	372	937	592	717
F7-015	GET /api/routes sin token	fixed	3	3m	868	4.82	0.00	0.00	228	682	445	571
F7-015	POST /api/routes/plan sin token	fixed	3	3m	864	4.80	0.00	100.00	271	725	512	619
F7-016	Recovery GET /api/health	3 a 20	-	3m	6699	37.15	0.00	0.21	93	853	205	293
F7-017	Cloudflare stability	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

1.7.1. Estado y observaciones por request

Caso	Request	Estado	Observación
F7-001	GET /api/health	Aprobado	P95 dentro del umbral health.
F7-002	POST /api/auth/login	Aprobado	Login estable, P95 dentro del umbral.
F7-003	GET /api/hazards	Aprobado	Lectura pública estable.
F7-003	GET /api/lost-pet-reports	Aprobado	Lectura pública estable.
F7-003	GET /api/found-pet-reports	Aprobado	Lectura pública estable.
F7-003	GET /api/generic-reports	Aprobado	Lectura pública estable.
F7-004	GET /api/routes autenticado	Aprobado	Lectura autenticada estable.
F7-004	GET /api/pets autenticado	Aprobado con observaciones	Assertion aislada y máximo alto no reflejado en P95/P99.
F7-005	POST /api/pets	Aprobado	Escritura dentro del umbral.
F7-006	POST /api/hazards	Aprobado	Escritura dentro del umbral.
F7-007	POST /api/media/upload	Aprobado	Upload dentro del umbral.
F7-008	POST /api/routes/plan simple	Aprobado	GraphHopper responde dentro del umbral.
F7-009	POST /api/routes/plan multipunto	Aprobado	Multipunto estable y cercano a ruta simple.
F7-010	POST /api/routes/plan filtros riesgo	Aprobado	Filtros de riesgo sin timeout sistemático.
F7-011	GET /api/health ramp-up	Aprobado con observaciones	Failure por assertions bajo ramp-up; sin error HTTP.
F7-011	GET /api/hazards ramp-up	Aprobado con observaciones	Failure menor y máximo alto aislado.
F7-011	GET /api/routes autenticado ramp-up	Aprobado con observaciones	Failure menor bajo ramp-up.
F7-012	GET /api/hazards spike	Aprobado con observaciones	Failure durante spike, sin error HTTP.
F7-012	GET /api/health post-spike incluido	Aprobado con observaciones	Health posterior cumple P95, pero con assertions fallidas.
F7-013	GET /api/pets autenticado peak	Aprobado con observaciones	Failure menor en pico autenticado.
F7-013	GET /api/routes autenticado peak	Aprobado con observaciones	Failure menor en pico autenticado.
F7-013	POST /api/hazards/id/comments peak	Aprobado	Comentarios dentro del umbral de escritura.
F7-014	GET /api/health endurance	Aprobado	Endurance sin degradación sostenida.
F7-014	GET /api/hazards endurance	Aprobado	Endurance sin degradación sostenida.
F7-015	POST /api/pets sin token	Fallido	Hallazgo de seguridad: assertion de rechazo sin token falla.
F7-015	GET /api/routes sin token	Aprobado	Rechazo sin token cumple assertion.
F7-015	POST /api/routes/plan sin token	Fallido	Hallazgo de seguridad: assertion de rechazo sin token falla.
F7-016	Recovery GET /api/health	Aprobado con observaciones	Recuperación estable; failure residual bajo.
F7-017	Cloudflare stability	No aplica / justificado	Todos los escenarios reportados se ejecutaron mediante Cloudflare.

1.8. Análisis Tests vs VU

Escenario	Assertion degradada	VU donde inicia degradación	Failure rate	Observación
F7-011 ramp-up	Health, hazards y routes bajo ramp-up	Entre 3 y 10 VU	0.31 % a 2.65 %	No hubo error HTTP; la degradación fue de assertions bajo aumento progresivo.
F7-012 spike	Hazards y health durante/post spike	Hasta 20 VU	3.54 % a 3.59 %	El backend no cae y health mantiene P95 $\leq$ 500 ms; quedan failures de assertion.
F7-013 peak	Pets/routes autenticados	Máx. 20 VU	0.81 %	Fallo menor de assertion en lecturas autenticadas; comentarios sin failure.
F7-015 security	Pets y route plan sin token	3 VU	100.00 %	Caso funcional fallido: endpoints protegidos no cumplieron assertion esperada de rechazo sin token.
F7-018 runs consolidados	Análisis general de runs	3, 5, 10 y 20 VU	0 % error HTTP	71514 requests en tabla de runs, throughput variable por mezcla de endpoints y 0 % error reportado.

1.9. Comparativas Requeridas

1.9.1. F7-009 - Ruta Simple vs Multipunto

Variante	Perfil	VU	P95	P99	Error %	Observación
2 puntos	fixed	3	403 ms	1625 ms	0.00 %	Ruta simple estable; mayor P99 por picos aislados.
5 puntos	fixed	3	476 ms	710 ms	0.00 %	Multipunto aumenta P95 frente a simple, pero reduce P99 observado y cumple umbral.

1.9.2. F7-010 - Rutas con Filtros de Riesgo

Variante	Perfil	VU	P95	P99	Error %	Observación
Sin filtros	fixed	3	403 ms	1625 ms	0.00 %	Se toma F7-008 como referencia sin filtros.
Filtros `high`, `medium`	fixed	3	506 ms	856 ms	0.00 %	Los filtros agregan latencia moderada en P95, sin error HTTP ni timeout sistemático.

1.10. Análisis F7-018 - Runs Performance

Run	Start Time	VUs	Duración	Total Requests	Requests/s	Response Time (ms)	Error %	CPU	Memoria (GB)
29	2026-05-12 00:43:54	20	3m	6699	37.26	93	0	6	15
28	2026-05-12 00:42:40	10	3m	560	15.94	64	0	6	15
27	2026-05-12 00:28:45	5	3m	2600	14.43	290	0	6	15
26	2026-05-12 00:09:18	5	15m	34777	38.62	90	0	6	15
25	2026-05-11 23:55:24	10	3m	1861	10.30	500	0	6	15
24	2026-05-11 23:49:30	20	3m	3785	20.96	186	0	6	15
23	2026-05-11 23:44:22	10	3m	5774	31.92	204	0	6	15
22	2026-05-11 23:37:59	10	3m	3305	20.66	322	0	6	15
21	2026-05-11 23:28:28	3	3m	1580	8.75	283	0	6	15
20	2026-05-11 23:23:54	3	3m	1621	8.98	276	0	6	15
19	2026-05-11 23:18:52	3	3m	1693	9.39	263	0	6	15
18	2026-05-11 23:18:08	3	3m	60	3.66	515	0	6	15
17	2026-05-11 22:05:47	3	3m	950	5.25	515	0	6	15
16	2026-05-11 22:01:29	3	3m	1508	8.35	305	0	6	15
15	2026-05-11 22:00:54	3	3m	170	6.22	391	0	6	15
14	2026-05-11 21:55:45	3	3m	826	4.57	587	0	6	15
13	2026-05-11 20:11:34	3	3m	290	6.53	388	0	6	15
12	2026-05-11 20:10:27	3	3m	410	9.53	253	0	6	15
10	2026-05-11 20:04:15	3	3m	1565	8.65	291	0	6	15
9	2026-05-11 20:00:13	3	3m	1480	8.19	313	0	6	15

La tabla de runs consolida 71514 requests con 0 % error HTTP. El throughput varía por mezcla de endpoints y perfil: rutas/GraphHopper y escrituras bajan RPS, mientras endurance y recovery sobre lecturas/health alcanzan los mayores RPS. No hay evidencia de HTTP 5xx en los datos entregados. Los campos CPU Cores y Memory (GB) se conservan tal como fueron reportados en el resumen de runs.

1.11. F7-019 - Consolidación 5xx, Error Budget y Endpoints Lentos

Indicador	Resultado
Requests totales en tabla granular	76228
Error HTTP consolidado	0.00 %
Avg ponderado	151.48 ms
Failure ponderado incluyendo F7-015	2.58 %
Failure ponderado excluyendo F7-015	0.32 %
Max observado	8781 ms en `GET /api/pets` autenticado
Mayor P95 observado	1221 ms en `POST /api/hazards`
Mayor P99 observado	1625 ms en `POST /api/routes/plan` simple
HTTP 5xx reportados	0

Los endpoints más lentos por promedio fueron `POST /api/auth/login` (739 ms), `POST /api/hazards` (640 ms), `POST /api/hazards/id/comments` (565 ms) y `POST /api/pets` (514 ms). Todos quedaron dentro de los umbrales de su categoría. El mayor riesgo no es latencia sino seguridad funcional en F7-015.

1.12. Hallazgos F7

#	Hallazgo	Impacto	Evidencia	Tratamiento
1	`POST /api/pets` sin token no cumple la assertion esperada de rechazo sin token.	Alto para seguridad funcional si el endpoint permite crear recursos sin autenticación.	F7-015: 868 requests, 0.00 % error HTTP, 100.00 % failure.	Registrar defecto y revisar middlewa-re/autorización de creación de mascotas.
2	`POST /api/routes/plan` sin token no cumple la assertion esperada de rechazo sin token.	Medio/alto según decisión de producto; el caso espera 401 en endpoints protegidos.	F7-015: 864 requests, 0.00 % error HTTP, 100.00 % failure.	Confirmar si planificación debe ser privada; si aplica, proteger endpoint con token.
3	Assertions fallidas menores durante ramp-up, spike y peak.	Bajo/medio; no se observaron errores HTTP ni 5xx, pero hay degradación bajo concurrencia.	F7-011, F7-012, F7-013 con failures entre 0.31 % y 3.59 %.	Mantener como observación y reejecutar después de corregir F7-015 o al mover a infraestructura más robusta.
4	Máximo aislado de 8781 ms en `GET /api/pets` autenticado.	Bajo si es outlier; no afecta P95/P99 de forma relevante.	F7-004 pets: P95 252 ms, P99 381 ms, max 8781 ms.	Monitorear en siguientes corridas; revisar logs si se repite.

1.13. Criterios de Aceptación

Criterio	Resultado
Escenarios F7-001 a F7-020 ejecutados o justificados	Cumplido: F7-001 a F7-016 ejecutados, F7-017 justificado, F7-018 a F7-020 completados por análisis.
No se presentan respuestas HTTP 5xx sin documentar	Cumplido: no se reportan 5xx en los datos entregados.
Error % queda dentro del umbral por tipo de endpoint o se justifica	Cumplido: Error % reportado en 0.00 % para la tabla granular.
P95 queda dentro del umbral por tipo de endpoint o se justifica	Cumplido: todos los P95 reportados quedan dentro de umbrales.
GraphHopper responde para escenarios de rutas	Cumplido: F7-008, F7-009 y F7-010 completados sin error HTTP.
Cloudflare Tunnel permanece operativo o se documenta cambio de URL	Cumplido con observación: se documenta que la URL exacta no está registrada en el entorno actual.
Tests vs VU revisado para detectar degradación	Cumplido: se documentan failures en ramp-up, spike, peak y seguridad sin token.

1.14. Conclusión

F7 queda aprobada con observaciones para performance local controlada. El backend sostuvo las cargas ejecutadas mediante Cloudflare sin errores HTTP reportados ni 5xx, con percentiles dentro de umbrales incluso en rutas con GraphHopper. La endurance corta mostró estabilidad sostenida con 34777 requests en el run de 15 minutos y 0 % error HTTP.

La aprobación queda condicionada por el hallazgo F7-015: `POST /api/pets` sin token y `POST /api/routes/plan` sin token fallaron el criterio esperado de rechazo sin autenticación. También se recomienda reejecutar ramp-up, spike y peak después de atender ese hallazgo para confirmar que las assertions fallidas menores no esconden degradación funcional.

Apéndice R

Reporte F8 - Regresión y Cierre del Proceso de Pruebas

La fase F8 correspondió a la etapa final del proceso de pruebas del sistema A4MPets, enfocada en actividades de regresión, verificación de estabilidad global y consolidación de resultados obtenidos durante las fases anteriores.

En esta etapa se realizó la reevaluación de funcionalidades críticas previamente validadas, con el objetivo de comprobar que las correcciones, ajustes y mejoras implementadas durante el ciclo de pruebas no introdujeran nuevos defectos sobre módulos ya estabilizados.

Las actividades de regresión incluyeron nuevamente validaciones relacionadas con autenticación, gestión de mascotas, reportes geoespaciales, comentarios, visualización cartográfica, planeación de rutas y navegación dentro de la interfaz, verificando la consistencia funcional del sistema tras las modificaciones realizadas durante las fases F6 y F7.

Adicionalmente, esta fase permitió consolidar las evidencias generadas durante todo el proceso QA, clasificar los defectos encontrados según severidad e impacto funcional, y emitir una evaluación general sobre el estado de estabilidad alcanzado por el prototipo desarrollado.

El presente informe resume los resultados finales del proceso de pruebas, incluyendo observaciones de regresión, defectos persistentes, validaciones exitosas y consideraciones relacionadas con la preparación del sistema para escenarios de demostración, evaluación académica y futuras iteraciones de desarrollo.

Las evidencias complementarias asociadas a la ejecución de esta fase se presentan en los anexos correspondientes.

1. Reporte de Evidencias - Fase F8: Regresión y Cierre QA

Campo	Valor
Proyecto	A4MPets
Fecha de ejecución	2026-05-12
Fase	F8 - Regresión y cierre QA
Versión bajo prueba	`feature/fixes` - `4677558c24f87f014d6bae82ee0d6e28995024e0` - 2026-05-11 23:09:27 -0500 - GraphHopper latest version
Responsable	Equipo QA
Decisión	Go con observaciones

1.1. Resumen Ejecutivo

F8 cierra la regresión final de A4MPets consolidando la evidencia de F0 a F7 sin modificar los reportes históricos de esas fases. La evidencia nueva de F8 queda centralizada en archivos Markdown compactos, evitando logs extensos, capturas sueltas o resultados Playwright pesados dentro de esta carpeta.

La regresión F0-F5 queda aprobada. F6 se conserva sin variación frente a su evidencia previa y `DEF-F6-001` queda cerrado por validación QA. F7 conserva sus métricas históricas de performance; el único cambio de cierre es `F7-015`, documentado como solucionado y validado por QA.

1.2. Evidencia F8

Evidencia	Archivo	Contenido
Consolidado F0-F5	`evidencia_F0_F5_regresion_2026-05-12.md`	Resumen por fase, comandos principales, métricas, resultados y conclusión de regresión.
Cierre F6-F7	`evidencia_F6_F7_cierre_2026-05-12.md`	Trazabilidad de F6/F7, cierre de `DEF-F6-001` y solución de `F7-015`.

Los reportes F0-F7 quedan como fuentes históricas inmutables:

Fase	Fuente histórica
F0	`docs/evidencias/F0/reporte_F0_ambiente.md`
F1	`docs/evidencias/F1/reporte_F1_backend.md`
F2	`docs/evidencias/F2/reporte_F2_frontend.md`
F3	`docs/evidencias/F3/reporte_F3_integracion_backend.md`
F4	`docs/evidencias/F4/reporte_F4_sistema_integrado_local.md`
F5	`docs/evidencias/F5/reporte_F5_e2e_navegador.md`
F6	`docs/evidencias/F6/reporte_F6_pruebas_manuales_funcionales.md`
F7	`docs/evidencias/F7/reporte_F7_postman_cloudflare.md`

1.3. Ambiente

Elemento	Valor
Backend	Laravel local
Frontend	Flutter web
GraphHopper	Docker local activo durante regresión integrada F4/F5
Navegador	Chrome / Chromium para F4-F5
Base de datos	Local/test
Cloudflare	Validado previamente en F7; F8 no recalcula métricas de performance

1.4. Consolidado F0-F8

Fase	Enfoque	Resultado de cierre	Evidencia F8
F0	Preparación y smoke de ambiente	Aprobada con observaciones cerradas	`evidencia_F0_F5_regresion_2026-05-12.md`
F1	Unitarias backend Laravel	Aprobada: 67 pruebas, 327 aserciones	`evidencia_F0_F5_regresion_2026-05-12.md`
F2	Unitarias/widget frontend Flutter	Aprobada: `flutter analyze` sin issues y 102 pruebas aprobadas	`evidencia_F0_F5_regresion_2026-05-12.md`
F3	Integración backend/API	Aprobada: 48 pruebas F3 y backend completo 115 pruebas	`evidencia_F0_F5_regresion_2026-05-12.md`
F4	Sistema integrado local	Aprobada: 22 checks PASS	`evidencia_F0_F5_regresion_2026-05-12.md`
F5	E2E navegador	Aprobada: 23 pruebas Playwright PASS	`evidencia_F0_F5_regresion_2026-05-12.md`
F6	Manual funcional	Aprobada; resultados sin variación frente a evidencia previa	`evidencia_F6_F7_cierre_2026-05-12.md`
F7	Performance Postman + Cloudflare	Aprobada con observación cerrada; `F7-015` solucionado	`evidencia_F6_F7_cierre_2026-05-12.md`
F8	Regresión y cierre	Go con observaciones	Este reporte

1.5. Automatizadas Reejecutadas

Suite	Comando	Resultado	Evidencia	Observaciones
F0 backend	`npm run backend - -dry-run`	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Backend local listo.
F0 frontend	`npm run frontend - -dry-run`	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Runtime frontend generado.
F0/F4 GraphHopper	`npm run graphhopper` / validación integrada F4-F5	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Docker/GraphHopper operativo para cierre.
F0 Cloudflare	`npm run backend:cloudflare - -dry-run`	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Tunnel validado previamente.
Backend F1	`php artisan test -testsuite=F1`	Aprobado: 67 / 327	`evidencia_F0_F5_regresion_2026-05-12.md`	Sin regresión.
Backend F1 coverage	`php artisan test -testsuite=F1 -coverage-html coverageF1`	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Cobertura regenerada.
Frontend analyze	`flutter analyze`	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Sin issues.
Frontend test	`flutter test`	Aprobado: 102 tests	`evidencia_F0_F5_regresion_2026-05-12.md`	Sin fallos.
Frontend coverage	`flutter test -coverage`	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Coverage generado.
Backend F3	`php artisan test -testsuite=F3`	Aprobado: 48 / 248	`evidencia_F0_F5_regresion_2026-05-12.md`	Incluye seguridad de endpoints protegidos.
Backend completa	`php artisan test`	Aprobado: 115 / 575	`evidencia_F0_F5_regresion_2026-05-12.md`	Sin regresión backend.
Backend F3 coverage	`php artisan test -testsuite=F3 -coverage-html coverageF3`	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Cobertura regenerada.
F4 integrado	`node scripts/f4-system-check.js`	Aprobado: 22 checks	`evidencia_F0_F5_regresion_2026-05-12.md`	Sistema integrado local validado.
F5 E2E	`npm run test:e2e:f5`	Aprobado: 23/23	`evidencia_F0_F5_regresion_2026-05-12.md`	Flujos E2E principales aprobados.

1.6. Retest de Hallazgos F0-F7

ID / Origen	Descripción	Estado F8	Evidencia	Decisión
`DEF-F6-001`	Rutas visibles en cuenta nueva	Corregido / retestado por QA	`evidencia_F6_F7_cierre_2026-05-12.md`	Cerrado para F8; se conserva evidencia histórica F6.
`F7-015 pets sin token`	Operación protegida de mascotas sin token	Solucionado y validado por QA	`evidencia_F6_F7_cierre_2026-05-12.md`	Cerrado; respuesta esperada HTTP 401.
`F7-015 route plan sin token`	`POST /api/routes/plan` sin Authorization	Solucionado y validado por QA	`evidencia_F6_F7_cierre_2026-05-12.md`	Cerrado; respuesta esperada HTTP 401.
`F7-015 routes sin token`	`GET /api/routes` sin Authorization	Validado	`evidencia_F6_F7_cierre_2026-05-12.md`	Cerrado; respuesta esperada HTTP 401.
F7 performance histórica	Ramp-up, spike, peak y endurance	Sin variación de métricas	Reporte histórico F7 y `evidencia_F6_F7_cierre_2026-05-12.md`	Se mantienen métricas previas; no se recalculan en F8.
F0/F4/F5 GraphHopper	Disponibilidad Docker/GraphHopper	Corregido	`evidencia_F0_F5_regresion_2026-05-12.md`	Cerrado por regresión integrada F4/F5.
F5 readiness Flutter Web	Mayor tolerancia para render inicial en Playwright	Corregido	`evidencia_F0_F5_regresion_2026-05-12.md`	Cerrado; F5 aprobó 23/23.

1.7. Matriz F8

Caso	Resultado	Evidencia	Observaciones
F8-001	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Health final del backend validado en F0/F4/F5.
F8-002	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Registro cubierto por F0, F4 y F5.
F8-003	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Login cubierto por F0, F4 y F5.
F8-004	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	CRUD de mascotas cubierto por F3/F4/F5.
F8-005	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Reporte de peligro y mapa cubiertos por F4/F5.
F8-006	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Reporte de mascota perdida cubierto por F4/F5.
F8-007	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Planificación con GraphHopper validada en F4/F5.
F8-008	Aprobado	`evidencia_F6_F7_cierre_2026-05-12.md`	Endpoints protegidos sin token validados; F7-015 cerrado.
F8-009	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Ownership de mascotas cubierto por F3/F5.
F8-010	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Upload inválido y media cubiertos por F3/F4/F5.
F8-011	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	`flutter analyze` sin issues.
F8-012	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	`php artisan test` aprobado.
F8-013	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Viewport desktop validado por F5.
F8-014	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Viewport móvil validado por F5.
F8-015	Aprobado con observación	`evidencia_F6_F7_cierre_2026-05-12.md`	Incidencias clasificadas; F6/F7 cerradas por trazabilidad.
F8-016	Aprobado	`evidencia_F6_F7_cierre_2026-05-12.md`	Sin defectos críticos o altos sin decisión.
F8-017	Aprobado	`evidencia_F0_F5_regresion_2026-05-12.md`	Casos afectados por correcciones revalidados.
F8-018	Aprobado	Este reporte	Evidencias F0-F7 consolidadas y referenciadas.
F8-019	Aprobado	Este reporte	Conteo final generado.
F8-020	Aprobado con observación	Este reporte	Decisión final: Go con observaciones.

1.8. Conteo Final

Resultado	Total
Aprobados	18
Aprobados con observación	2
Fallidos	0
Bloqueados	0
No ejecutados	0
No aplica	0
Total	20

1.9. Criterios de Salida

Criterio	Resultado
Casos P0/P1 reejecutados	Cumple: F0-F5 consolidados y F6/F7 cerrados por trazabilidad.
Sin defectos críticos abiertos	Cumple.
Sin defectos altos sin decisión	Cumple.
Evidencia suficiente por fase	Cumple: F0-F7 referenciados y F8 contiene evidencia consolidada.
Regresión automatizada ejecutada	Cumple: F1-F5 aprobados y F0 validado como ambiente base.
Incidencias clasificadas	Cumple: `DEF-F6-001` y `F7-015` cerrados.

1.10. Observaciones de Cierre

- F6 no cambia su evidencia histórica; se registra en F8 que `DEF-F6-001` fue corregido y retestado por QA.
- F7 no cambia sus métricas históricas de performance; se registra en F8 que `F7-015` fue solucionado y validado por QA.
- La decisión se emite como Go con observaciones porque F7 conserva observaciones históricas de performance, aunque no quedan defectos críticos o altos abiertos sin decisión.

1.11. Conclusión

F8 queda cerrada con decisión Go con observaciones. La regresión F0-F5 queda aprobada, F6 queda cerrada por validación QA sin variación de evidencia, y F7 queda aprobada con observación cerrada por la solución de `F7-015`. No quedan fallos, bloqueos ni casos sin decisión formal en la matriz F8.

Apéndice S

Consolidado de evidencias QA F0-F7

Con el propósito de centralizar las evidencias históricas generadas durante las fases F0 a F7, se elaboró un documento consolidado que resume los resultados obtenidos durante el proceso de pruebas del sistema A4MPets.

Este consolidado integra las actividades de regresión, validaciones funcionales, pruebas automatizadas, pruebas de integración, ejecución E2E, verificaciones manuales y actividades de cierre QA desarrolladas a lo largo del ciclo de pruebas. Asimismo, preserva la trazabilidad hacia los reportes individuales de cada fase, manteniendo los documentos históricos como fuentes inmutables de evidencia técnica.

El documento incluye métricas generales, defectos corregidos, criterios de aceptación, validaciones de seguridad relacionadas con autenticación y control de acceso, así como el estado final de cierre de observaciones reportadas durante las fases F6 y F7.

Particularmente, el consolidado documenta el cierre formal de los hallazgos DEF-F6-001 y F7-015, además de registrar la aprobación global de las fases F0 a F7 como evidencia de estabilidad funcional y técnica previa al cierre final del proceso QA.

1. Reporte de Evidencia Completa F0 a F7

Campo	Valor
Proyecto	A4MPets
Documento	Reporte de evidencia completa de F0 a F7
Fecha de consolidación	2026-05-12
Responsable	Equipo QA
Tipo de evidencia	Consolidado de regresión y cierre por trazabilidad
Alcance	Fases F0, F1, F2, F3, F4, F5, F6 y F7
Decisión general	Evidencia F0-F7 consolidada para cierre de regresión QA

Este documento unifica las dos evidencias consolidadas generadas para el cierre de QA: la regresión F0-F5 y el cierre por trazabilidad F6-F7. Su propósito es presentar en un único archivo LaTeX el estado completo de las fases F0 a F7, sin modificar los reportes históricos de cada fase.

2. Objetivo

Consolidar la evidencia de regresión y cierre de las fases F0 a F7 del proyecto A4MPets, manteniendo la trazabilidad hacia los reportes históricos de cada fase y dejando documentados los resultados finales, métricas principales, observaciones, defectos cerrados y criterios de aceptación relacionados con el ciclo de pruebas.

3. Fuentes históricas

Los reportes F0-F7 se mantienen como fuentes históricas inmutables. Este documento no reemplaza sus contenidos detallados; los resume y los referencia para efectos de cierre.

Fase	Fuente histórica
F0	`docs/evidencias/F0/reporte_F0_ambiente.md`
F1	`docs/evidencias/F1/reporte_F1_backend.md`
F2	`docs/evidencias/F2/reporte_F2_frontend.md`
F3	`docs/evidencias/F3/reporte_F3_integracion_backend.md`
F4	`docs/evidencias/F4/reporte_F4_sistema_integrado_local.md`
F5	`docs/evidencias/F5/reporte_F5_e2e_navegador.md`
F6	`docs/evidencias/F6/reporte_F6_pruebas_manuales_funcionales.md`
F7	`docs/evidencias/F7/reporte_F7_postman_cloudflare.md`

4. Evidencias consolidadas utilizadas

Evidencia	Archivo	Contenido
Consolidado F0-F5	`evidencia_F0_F5_regresion_2026-05-12.md`	Resumen por fase, comandos principales, métricas, resultados y conclusión de regresión.
Cierre F6-F7	`evidencia_F6_F7_cierre_2026-05-12.md`	Trazabilidad de F6 y F7, cierre de `DEF-F6-001` y solución de `F7-015`.

5. Resumen ejecutivo

La regresión automatizada e integrada de F0 a F5 queda aprobada para el cierre QA. No se identifican defectos críticos ni altos abiertos en estas fases al cierre. F6 y F7 quedan cerradas por trazabilidad: F6 conserva sus resultados funcionales sin variación frente a la evidencia previa, mientras que F7 conserva sus métricas históricas de performance y registra `F7-015` como solucionado y validado por QA.

En conjunto, las fases F0-F7 quedan consolidadas como evidencia completa del ciclo de pruebas previo al cierre F8. La decisión global queda documentada con observaciones controladas, sin fallos críticos o altos abiertos y con trazabilidad suficiente hacia las fuentes históricas.

6. Resumen consolidado por fase

Fase	Resultado	Comandos / validaciones	Métricas	Observaciones
F0 - Ambiente	Aprobado con observaciones	`npm run backend -- --dry-run`; `npm run frontend -- --dry-run`; `npm run graphhopper`; `npm run backend:cloudflare -- --dry-run`.	0 fallos HTTP; 0 bloqueos críticos.	Ambiente base validado. GraphHopper queda confirmado por la reejecución integrada F4/F5 con Docker activo.
F1 - Backend unitario	Aprobado	`php artisan test --testsuite=F1`; cobertura F1.	67 pruebas; 327 aserciones; 0 fallos.	Suite F1 aprobada sin regresiones.
F2 - Frontend unitario/widget	Aprobado	`flutter analyze`; `flutter test`; `flutter test --coverage`.	0 issues de analyze; 102 pruebas; 0 fallos.	Análisis estático y pruebas Flutter aprobadas.
F3 - Integración backend/API	Aprobado	`php artisan test --testsuite=F3`; `php artisan test`; cobertura F3.	F3: 48 pruebas y 248 aserciones; backend completo: 115 pruebas y 575 aserciones.	Incluye validación automatizada de endpoints protegidos sin token relacionados con `F7-015`.
F4 - Sistema integrado local	Aprobado	`node --check scripts/f4-system-check.js`; `node scripts/f4-system-check.js`.	22 checks PASS; exit code 0.	Backend, frontend, GraphHopper, persistencia, media upload y smoke Chromium aprobados.
F5 - E2E navegador	Aprobado	`node --check scripts/f5-stack.js`; `node --check playwright.f5.config.js`; `npm run test:e2e:f5`.	23 pruebas Playwright; 0 fallos; 0 bloqueos.	Suite Playwright F5 aprobada después de ajustar readiness/timeouts y validar autenticación en planificación de rutas.
F6 - Pruebas manuales funcionales	Aprobado	Validación manual desde navegador; revisión funcional, visual, permisos, formularios, reportes, mascotas, perfil y evidencias fotográficas.	Sin defectos críticos abiertos; sin defectos altos sin decisión.	Los resultados funcionales no varían frente a la evidencia previa. `DEF-F6-001` queda corregido y retestado por QA.
F7 - Performance API Postman + Cloudflare	Aprobado con observación cerrada	Postman Desktop Performance Testing; Cloudflare Tunnel; escenarios baseline, carga fija, ramp-up, spike, peak, endurance, seguridad sin token y rutas con GraphHopper.	Sin respuestas HTTP 5xx reportadas; métricas históricas conservadas; `F7-015` solucionado.	No se recalculan métricas históricas. El único cambio de cierre es `F7-015`, validado por QA.

7. Detalle de regresión F0-F5

7.1. F0 - Preparación y smoke de ambiente

F0 queda aprobada con observaciones. El ambiente base fue validado mediante comandos de arranque del backend, frontend, GraphHopper y Cloudflare en modo de verificación. La evidencia confirma ausencia de fallos HTTP y bloqueos críticos. La disponibilidad de GraphHopper queda reforzada por la reejecución integrada en F4/F5 con Docker activo.

7.2. F1 - Pruebas unitarias backend Laravel

F1 queda aprobada como suite unitaria backend. La ejecución de `php artisan test --testsuite=F1` reporta 67 pruebas, 327 aserciones y 0 fallos. La cobertura F1 se conserva como evidencia técnica de la regresión unitaria del backend.

7.3. F2 - Pruebas frontend Flutter

F2 queda aprobada con `flutter analyze` sin issues y `flutter test` con 102 pruebas aprobadas. También se generó cobertura mediante `flutter test --coverage`. La fase valida lógica aislada del frontend, widgets, providers, modelos y servicios Dart.

7.4. F3 - Integración backend/API

F3 queda aprobada como suite de integración backend/API. La ejecución incluye `php artisan test --testsuite=F3`, regresión backend completa y cobertura F3. El cierre reporta 48 pruebas F3, 248 aserciones, y backend completo con 115 pruebas y 575 aserciones. Esta fase también incluye validación automatizada de endpoints protegidos sin token asociados al hallazgo `F7-015`.

7.5. F4 - Sistema integrado local

F4 queda aprobada con 22 checks PASS y exit code 0. La validación confirma comunicación técnica entre backend, frontend, GraphHopper, persistencia, media upload y smoke Chromium. Esto cierra los riesgos de integración local que no se cubren únicamente desde las pruebas unitarias o de API.

7.6. F5 - Pruebas E2E de caja negra en navegador

F5 queda aprobada con 23 pruebas Playwright, 0 fallos y 0 bloqueos. La suite valida flujos desde la interfaz de usuario en navegador, incluyendo autenticación, mascotas, reportes, peligros, rutas, viaje, comentarios, permisos, aislamiento de datos y responsive.

8. Cierre por trazabilidad F6-F7

Fase	Resultado	Variación frente a evidencia histórica	Defectos críticos abiertos	Defectos altos sin decisión	Nota de cierre
F6 - Pruebas manuales funcionales	Aprobado	No	0	0	Los resultados funcionales no varían frente a la evidencia previa. `DEF-F6-001` queda corregido y retestado por QA según validación del usuario.
F7 - Performance API Postman + Cloudflare	Aprobado con observación cerrada	No se recalculan métricas históricas	0	0	Las métricas históricas de performance no se modifican. El único cambio de cierre es `F7-015`, solucionado y validado por QA.

9. Validación específica de F7-015

El hallazgo `F7-015` queda solucionado y validado por QA. La expectativa confirmada es que las operaciones protegidas sin token rechacen la solicitud con HTTP 401.

Elemento	Resultado
Estado	Solucionado y validado por QA.
Expectativa validada	Las operaciones protegidas sin token rechazan la solicitud con HTTP 401.
`POST /api/pets` sin token	Validado como rechazado.
`POST /api/routes/plan` sin token	Validado como rechazado.
`GET /api/routes` sin token	Validado como rechazado.
Recálculo de métricas Postman	No aplica en F8; se conservan las métricas históricas de F7.

10. Trazabilidad de defectos y observaciones

ID / origen	Descripción	Estado de cierre	Decisión
`DEF-F6-001`	Rutas visibles en cuenta nueva.	Corregido y retestado por QA.	Cerrado para F8; se conserva evidencia histórica F6.
`F7-015 pets sin token`	Operación protegida de mascotas sin token.	Solucionado y validado por QA.	Cerrado; respuesta esperada HTTP 401.
`F7-015 route plan sin token`	`POST /api/routes/plan` sin Authorization.	Solucionado y validado por QA.	Cerrado; respuesta esperada HTTP 401.
`F7-015 routes sin token`	`GET /api/routes` sin Authorization.	Validado.	Cerrado; respuesta esperada HTTP 401.
F7 performance histórica	Ramp-up, spike, peak y endurance.	Sin variación de métricas.	Se mantienen métricas previas; no se recalculan en F8.

11. Criterios de aceptación consolidados

Criterio	Resultado
F0-F5 reejecutadas o consolidadas por evidencia de regresión.	Cumplido.
F6 y F7 cerradas por trazabilidad.	Cumplido.
No existen defectos críticos abiertos.	Cumplido.
No existen defectos altos sin decisión.	Cumplido.
Los reportes históricos F0-F7 se conservan como fuentes inmutables.	Cumplido.
`DEF-F6-001` queda corregido y retestado.	Cumplido.
`F7-015` queda solucionado y validado con rechazo HTTP 401 para operaciones protegidas sin token.	Cumplido.
Las métricas históricas de F7 se conservan sin recálculo.	Cumplido.

12. Conclusión

La evidencia completa F0-F7 queda consolidada. La regresión automatizada e integrada de F0 a F5 queda aprobada; F6 queda cerrada por validación QA sin variación frente a su evidencia histórica; y F7 queda aprobada con observación cerrada por la solución de `F7-015`. No quedan defectos críticos o altos abiertos sin decisión formal.

Este documento puede anexarse como reporte consolidado de evidencia F0-F7 dentro del cierre QA de A4MPets.