

UNIVERSIDAD DE LAS FUERZAS ARMADAS – ESPE

DEPARTAMENTO DE CIENCIAS COMPUTACIONALES



MATERIA: MODELOS DE PROCESOS DE DESARROLLO DE SOFTWARE

DESARROLLO DEL SISTEMA LOCALPET

CUARTO SEMESTRE

INTEGRANTES:

LUIS EDUARDO CARRERA CHISAG
JORGE MATEO GUTIERREZ MORALES
JOSTIN VLADIMIR MEDINA QUINGA
DILAN ALEJANDRO MOYA OCAÑA
DUVAL EDUARDO MUÑOZ NUÑEZ
EMILY ANABEL PANCHI PANCHI
JORDY DAVID QUIMBITA MOLINA
DANILO JOSUE TAPIA CONDORCANA
ANGELI YOMAIRA TELLO REATQUI
OMAR ALEXANDER TOAPANTA TAPIA
FRANCISCO DAVID TORO CORDOVA
LISBETH AMARILES YUPANGUI AGUIRRE

FECHA DE ENTREGA: 05 DE MARZO DEL 2025

Contenido

Introducción.....	3
Propósito.....	3
Ámbito del Sistema.....	3
1. Gestión de reportes de mascotas:.....	4
2. Búsqueda y filtrado:	4
3. Interacción del usuario:	4
Beneficios	4
Beneficios generales:	4
Beneficios para los usuarios:.....	4
Beneficios para el equipo de desarrollo:.....	4
Objetivos	5
Actores	5
Descripción General	5
Perspectiva del Producto	5
Arquitectura del Sistema.....	6
2. Internet:	6
3. Servidor:.....	6
4. Base de datos:.....	6
Tecnologías y Protocolos.....	6
Funciones del Producto	7
Gestión de reportes de mascotas.....	7
2. Publicación de reportes de mascotas encontradas:	8
3. Edición y eliminación de reportes:	8
4. Cambio de estado de los reportes:	8
Búsqueda y filtrado de reportes.....	8
2. Visualización de reportes:.....	8
Gestión de usuarios	8
2. Gestión de información personal:	9
Características de los Usuarios Finales	9
1. Personas que buscan mascotas extraviadas.....	9
• Interacciones con el sistema:	9
• Características relevantes:.....	9
2. Personas que encuentran mascotas	10
• Interacciones con el sistema:	10
• Características relevantes:.....	10
Requisitos comunes de los usuarios finales.....	10

Restricciones	11
Suposiciones y Dependencias	11
Requerimientos Futuros	11
Requerimientos Específicos.....	12
Requerimientos funcionales	12
Requerimientos no funcionales.....	15
Requerimientos de Rendimiento	16
Diagrama de casos de uso	17
Casos de Uso de Alto Nivel.....	18
Casos de Uso Expandido.....	20
Diagrama de secuencia	28
Diagrama de clases	29
Pruebas unitarias.....	30
Se procede a realizar las pruebas unitarias referentes a el código y su funcionalidad.	30
Encuesta para pruebas con usuarios.....	59
Anexo 1 Pruebas con usuarios	65
Anexo 2 Manual de instalación.....	68
Anexo 3 Manual de usuario	69
Anexo 4 Repositorio del Proyecto	74

Introducción

El sistema LocalPet surge como respuesta a la necesidad de facilitar la localización de mascotas extraviadas, un problema común que afecta a muchas personas. Actualmente, los métodos tradicionales para encontrar mascotas, como carteles o publicaciones en redes sociales, no siempre son efectivos. Este proyecto tiene como objetivo proporcionar una solución más centralizada y eficiente mediante una aplicación web que permita a los usuarios reportar mascotas perdidas o encontradas y buscar activamente coincidencias.

Este documento describe los requisitos generales y específicos del sistema LocalPet, detallando las funcionalidades clave, el público objetivo, y los beneficios esperados. Servirá como guía para el diseño, desarrollo y pruebas del sistema.

Propósito

El propósito de este documento es establecer de manera clara y concisa los requisitos funcionales y no funcionales del sistema LocalPet, con el fin de garantizar que todas las partes interesadas (usuarios, desarrolladores y evaluadores) tengan una comprensión común del alcance y las expectativas del sistema. Este documento también busca proporcionar una base para las actividades posteriores de diseño, desarrollo, y pruebas.

El sistema LocalPet tiene como objetivo principal permitir la publicación y búsqueda de reportes de mascotas extraviadas o encontradas, asegurando una experiencia de usuario intuitiva y funcional. Este documento está dirigido a los desarrolladores, analistas, evaluadores y gestores del proyecto.

Ámbito del Sistema

El sistema **LocalPet** es una aplicación web que permitirá a los usuarios publicar reportes de mascotas perdidas o encontradas y buscar activamente coincidencias con base en criterios como ubicación, tipo de mascota y características físicas. La aplicación será accesible desde cualquier dispositivo con conexión a Internet y tendrá los siguientes módulos principales:

1. Gestión de reportes de mascotas:

- Registro de mascotas perdidas y encontradas.
- Edición y eliminación de reportes.
- Incorporación de fotos, videos y descripciones.

2. Búsqueda y filtrado:

- Filtrado por ubicación y tipo de mascota.
- Visualización de reportes detallados.

3. Interacción del usuario:

- Registro y autenticación de usuarios.
- Calificación de la aplicación y retroalimentación.

El sistema estará diseñado para operar en un entorno multiplataforma, utilizando una arquitectura orientada a servicios y tecnologías web modernas.

Beneficios

Beneficios generales:

1. Proporcionar una solución centralizada para la localización de mascotas.
2. Mejorar la efectividad y rapidez en la búsqueda de mascotas extraviadas.
3. Fomentar la interacción y colaboración entre la comunidad.

Beneficios para los usuarios:

1. Herramientas avanzadas para publicar y buscar reportes de mascotas.
2. Acceso fácil e intuitivo desde cualquier dispositivo conectado a Internet.
3. Mayor posibilidad de reunir a las mascotas con sus dueños gracias a filtros efectivos.

Beneficios para el equipo de desarrollo:

1. Uso de metodologías ágiles (Scrum) para garantizar la entrega de un producto de calidad.

2. Base tecnológica escalable que permitirá futuras integraciones y mejoras.

Objetivos

OBJ-001	Permitir reportes de mascotas
Descripción	El sistema debe permitir crear reportes de mascotas perdidas y reportes de los lugares por donde se la ha visto
Importancia	Alta
Comentarios	La creación del reporte de la mascota debe incluir la información pertinente de la mascota y de manera opcional la información del usuario. Además debe permitir reportar una mascota como encontrada

OBJ-002	Buscar entre los reportes de mascotas
Descripción	El sistema debe dar acceso a un sistema para búsquedas entre los registros disponibles
Importancia	Alta
Comentarios	La búsqueda debe permitir filtrar por tipo de mascota, ubicación y cercanía

Actores

ACT-01	Usuario Registrado
Descripción	Tiene acceso a la página web del aplicativo.
Comentarios	Es el usuario que reporta mascotas y/o realiza búsquedas

ACT-02	Usuario No Registrado
Descripción	No tiene acceso a la página web del aplicativo.
Comentarios	Es el usuario realiza búsquedas

Descripción General

Perspectiva del Producto

El sistema **LocalPet** es una aplicación web diseñada para facilitar la búsqueda de mascotas extraviadas y fomentar la interacción entre los usuarios de una comunidad. Basándose en una arquitectura cliente-servidor, el sistema utiliza Internet para conectar múltiples dispositivos de usuarios con un servidor central que gestiona la aplicación y su base de datos.

Arquitectura del Sistema

1. Cliente:

- Los usuarios acceden al sistema desde dispositivos conectados a Internet, como computadoras o teléfonos móviles, utilizando navegadores web. Estos dispositivos actúan como clientes que envían solicitudes al servidor.
- Los clientes pueden realizar diversas acciones, como registrar mascotas perdidas/encontradas, buscar reportes, y gestionar su información personal.

2. Internet:

- El sistema aprovecha la conectividad a través de la red para garantizar acceso remoto, posibilitando que los usuarios interactúen con el sistema desde cualquier lugar.

3. Servidor:

- Es el núcleo del sistema y se encarga de procesar las solicitudes de los clientes, ejecutar la lógica del negocio y mantener la comunicación con la base de datos.
- El servidor también controla la autenticación de usuarios, la seguridad de los datos y la ejecución de algoritmos de búsqueda y filtrado.

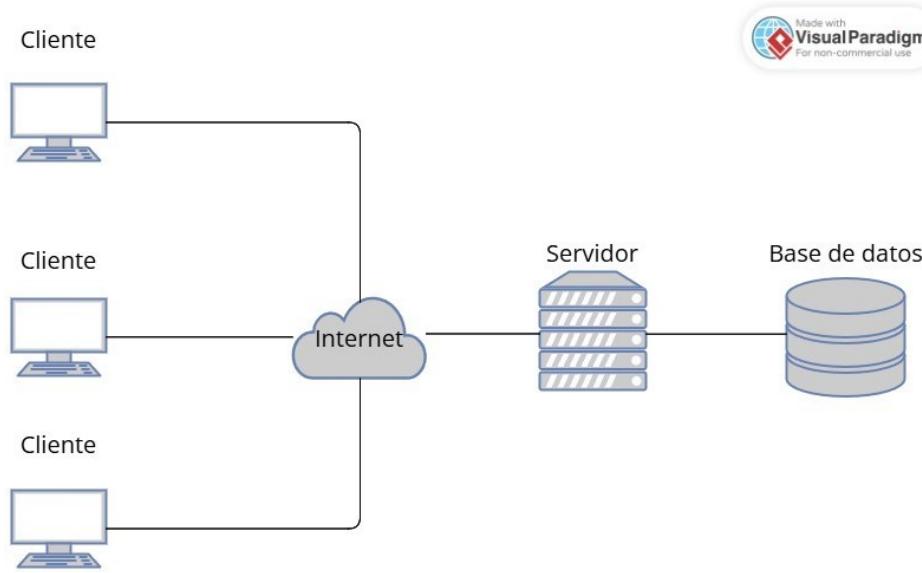
4. Base de datos:

- La base de datos almacena toda la información del sistema, incluyendo los reportes de mascotas, las credenciales de usuarios, las ubicaciones geográficas, y las calificaciones de la aplicación.
- Está diseñada para ser eficiente y escalable, soportando múltiples consultas concurrentes de los usuarios.

Tecnologías y Protocolos

- **Conexión Cliente-Servidor:** Utiliza el protocolo HTTP/HTTPS para garantizar una comunicación segura entre los clientes y el servidor.

- **Base de datos:** Implementada en SQLite, garantizando una estructura ligera pero confiable para almacenar datos.
- **Interfaces de usuario:** Desarrolladas en HTML, CSS y JavaScript, brindando una experiencia de usuario interactiva y sencilla.



Funciones del Producto

El sistema **LocalPet** está diseñado para cumplir con funciones específicas que permitan la publicación y búsqueda eficiente de reportes de mascotas extraviadas o encontradas. Las funcionalidades descritas se basan exclusivamente en lo especificado en el documento de desarrollo del proyecto.

Gestión de reportes de mascotas

1. Publicación de reportes de mascotas perdidas:

- Permitir a los usuarios registrar información sobre mascotas extraviadas, incluyendo:
 - Datos descriptivos (tipo de mascota, características físicas).
 - Ubicación de pérdida.

- Fotografías y videos.

- Generar un estado inicial de "Perdida" para el reporte.

2. Publicación de reportes de mascotas encontradas:

- Los usuarios pueden registrar información de mascotas halladas, similar a los reportes de pérdida:

- Ubicación donde fue encontrada.
- Fotografías y descripciones.

3. Edición y eliminación de reportes:

- Los usuarios pueden modificar o eliminar sus reportes si es necesario.

4. Cambio de estado de los reportes:

- Los reportes pueden ser actualizados por el usuario para indicar que la mascota ha sido encontrada o que ya está con su dueño.

Búsqueda y filtrado de reportes

1. Búsqueda avanzada:

- Permitir a los usuarios buscar reportes de mascotas mediante filtros específicos:

- Ubicación (con visualización en mapa interactivo).
- Tipo de mascota.
- Características físicas (color, tamaño, etc.).

2. Visualización de reportes:

- Mostrar reportes con detalles relevantes como descripción, ubicación y fotos, tanto de mascotas perdidas como encontradas.

Gestión de usuarios

1. Registro y autenticación de usuarios:

- Los usuarios deben registrarse con datos básicos (nombre, correo, etc.) y autenticarse para acceder a las funcionalidades del sistema.

2. Gestión de información personal:

- Los usuarios podrán editar su perfil y administrar su información personal.

Características de los Usuarios Finales

El sistema LocalPet está diseñado para ser utilizado por dos tipos principales de usuarios finales: **personas que buscan mascotas extraviadas** y **personas que encuentran mascotas y desean reportarlas**. Cada tipo de usuario interactúa con el sistema de manera distinta, pero ambos comparten el objetivo de facilitar la localización de mascotas.

1. Personas que buscan mascotas extraviadas

- **Descripción:** Este grupo incluye a propietarios de mascotas que han perdido a sus animales y desean localizarlos mediante el sistema.
- **Objetivo principal:** Encontrar reportes de mascotas que coincidan con la descripción de su animal perdido y contactar con las personas que los hayan encontrado.
- **Interacciones con el sistema:**
 - Registrar información sobre su mascota extraviada, incluyendo fotos, descripción, y última ubicación conocida.
 - Realizar búsquedas de reportes existentes utilizando filtros como tipo de mascota, ubicación, o características físicas.
 - Visualizar reportes coincidentes en un mapa interactivo y contactar al usuario que registró la mascota.
- **Características relevantes:**
 - No se requiere un conocimiento técnico avanzado.

- La interfaz debe ser intuitiva y accesible desde cualquier dispositivo conectado a Internet.
- Necesidad de recibir notificaciones automáticas sobre posibles coincidencias.

2. Personas que encuentran mascotas

- **Descripción:** Usuarios que han encontrado una mascota y desean reportarla para ayudar a localizar a su dueño.
- **Objetivo principal:** Publicar reportes detallados sobre las mascotas encontradas y facilitar su reunión con los propietarios.
- **Interacciones con el sistema:**
 - Crear reportes de mascotas encontradas, añadiendo fotos, descripciones, y la ubicación donde fueron halladas.
 - Consultar reportes de mascotas extraviadas para identificar posibles coincidencias.
 - Actualizar el estado del reporte si el dueño es localizado.
- **Características relevantes:**
 - La facilidad de uso es crucial para maximizar la cantidad de reportes registrados.
 - Se espera que la aplicación proporcione recordatorios o notificaciones para mantener los reportes actualizados.

Requisitos comunes de los usuarios finales

- Ambos tipos de usuarios requieren una plataforma segura para proteger la información personal compartida.
- La experiencia de usuario debe ser consistente y accesible, independientemente del tipo de dispositivo utilizado.
- La comunicación entre usuarios (por ejemplo, mediante mensajes o notificaciones) debe ser efectiva y privada.

Restricciones

RST-001	Lenguaje de Programación
Descripción	Se utilizara Python en su versión 3.13
Importancia	Alta
Comentarios	NA

RST-002	Base de datos
Descripción	Se utilizara SQLite por medio de Python en la versión integrada dentro de Python 3.13
Importancia	Alta
Comentarios	NA

RST-003	Arquitectura simple
Descripción	Se utilizará una arquitectura simple de cliente servidor y base de datos
Importancia	Media
Comentarios	NA

Suposiciones y Dependencias

SUP-001	Sistema operativo
Descripción	El sistema operativo para el servidor será Windows 10
Importancia	Alta
Comentarios	Se podría cambiar el sistema operativo a Linux por mayor eficiencia pero puede generar más dependencias

SUP-002	Python 3.13
Descripción	Python 3.13 ya instalado y disponible en el PATH
Importancia	Alta
Comentarios	NA

Requerimientos Futuros

El sistema LocalPet está diseñado para ser escalable y adaptable, lo que permitirá la integración de nuevas funcionalidades en el futuro según las necesidades de los usuarios y el crecimiento de la plataforma. Entre los requerimientos futuros, se considera la implementación de un sistema de notificaciones automatizadas que informe a los usuarios sobre reportes que

coincidan con sus búsquedas o publicaciones, ya sea mediante correos electrónicos o notificaciones en la misma aplicación.

Otra posible funcionalidad es la integración con redes sociales para permitir a los usuarios compartir reportes directamente en plataformas populares, aumentando el alcance y las probabilidades de encontrar o reunir mascotas. También se planea incluir un módulo de estadísticas que proporcione información útil sobre los reportes registrados, como tendencias de pérdida por región o tipo de mascota, para mejorar la toma de decisiones y fomentar campañas preventivas.

La norma ISO/IEC 830 es una referencia fundamental en la gestión de la documentación técnica y de calidad en proyectos de software y sistemas informáticos. En su implementación, se utiliza para estructurar de manera clara y precisa los requisitos, especificaciones y toda la documentación necesaria a lo largo del ciclo de vida del producto y se presenta a continuación.

Requerimientos Específicos

Requerimientos funcionales

RQF-001	Permitir registro, inicio de sesión y navegación como invitado
Descripción	El sistema debe permitir a los usuarios registrarse, iniciar sesión y navegar como invitados para visualizar mascotas reportadas.
Objetivo	OBJ-001 y OBJ-002
Importancia	Alta
Estado	Pendiente
Estabilidad	Alta
Comentario	Para usuarios registrados

RQF-002	Subir fotos y agregar descripciones detalladas
Descripción	Permitir a los usuarios subir fotos (máx. 5 MB, formatos JPEG o PNG) y agregar descripciones claras de las mascotas.
Objetivo	OBJ-001
Importancia	Alta
Estado	Pendiente
Estabilidad	Alta

Comentario	Garantizar compatibilidad con los principales navegadores y dispositivos.
-------------------	---

RQF-003	Registrar ubicación y filtrar reportes por criterios específicos
Descripción	Integrar un sistema de geolocalización para registrar la última ubicación conocida y permitir filtrar reportes por ubicación, fecha o tipo de mascota.
Objetivo	OBJ-001
Importancia	Alta
Estado	Aprobado
Estabilidad	Media
Comentario	Utilizar una API de mapas como Google Maps para facilitar la geolocalización.

RQF-004	Gestión de reportes
Descripción	Permitir a los usuarios actualizar el estado de los reportes (de "perdido" a "encontrado").
Objetivo	OBJ-001
Importancia	Alta
Estado	Aprobado
Estabilidad	Alta
Comentario	Restringir permisos de edición y eliminación únicamente al creador del reporte.

RQF-005	Mostrar y buscar reportes
Descripción	Visualizar listas de mascotas perdidas y encontradas con opciones avanzadas de búsqueda y filtros aplicables. También debe permitir búsqueda por proximidad geográfica y mostrar los reportes en formato de lista.
Objetivo	OBJ-002
Importancia	Alta
Estado	Aprobado
Estabilidad	Alta
Comentario	Ofrecer información clara y ordenada en las listas. La búsqueda debe ser rápida y eficiente.

RQF-006	Facilitar la comunicación entre usuarios
----------------	--

Descripción	Permitir a los usuarios agregar un contacto opcional (teléfono o correo), asegurando la privacidad de los datos personales.
Objetivo	OBJ-001
Importancia	Media
Estado	Aprobado
Estabilidad	Alta
Comentario	No compartir información de contacto públicamente sin consentimiento del usuario.

Requerimientos no funcionales

RQN-001	Interfaz intuitiva
Descripción	El sistema debe contar con una interfaz fácil de usar para garantizar que los usuarios puedan navegar y utilizar todas las funcionalidades sin dificultad.
Objetivo	NA
Importancia	Alta
Estado	Aprobado
Estabilidad	Alta
Comentario	Asegurarse de probar en múltiples dispositivos.

RQN-002	Compatibilidad multiplataforma
Descripción	Asegurar que el sistema funcione correctamente en navegadores modernos (Chrome, Firefox, Edge, Safari) y dispositivos móviles.
Objetivo	NA
Importancia	Alta
Estado	Aprobado
Estabilidad	Alta
Comentario	Probar en diversos navegadores y tamaños de pantalla.

RQN-003	Restricción de tamaño de imágenes
Descripción	Limitar el tamaño de imágenes a un máximo de 5 MB en formatos JPEG o PNG, optimizando la carga de recursos y almacenamiento.
Objetivo	NA
Importancia	Media
Estado	Aprobado
Estabilidad	Media
Comentario	Implementar validaciones automáticas en la carga.

RQN-004	Rendimiento óptimo
Descripción	El tiempo de respuesta del sistema para cargar páginas, listas de reportes o mapas interactivos.
Objetivo	NA
Importancia	Alta
Estado	Aprobado
Estabilidad	Media
Comentario	Optimizar consultas a la base de datos y servicios.

RQN-005	Seguridad de la información
Descripción	Garantizar la protección de datos personales mediante cifrado, evitando compartir información sin consentimiento explícito.
Objetivo	NA
Importancia	Alta
Estado	Aprobado
Estabilidad	Alta
Comentario	Cumplir con estándares de privacidad de datos.

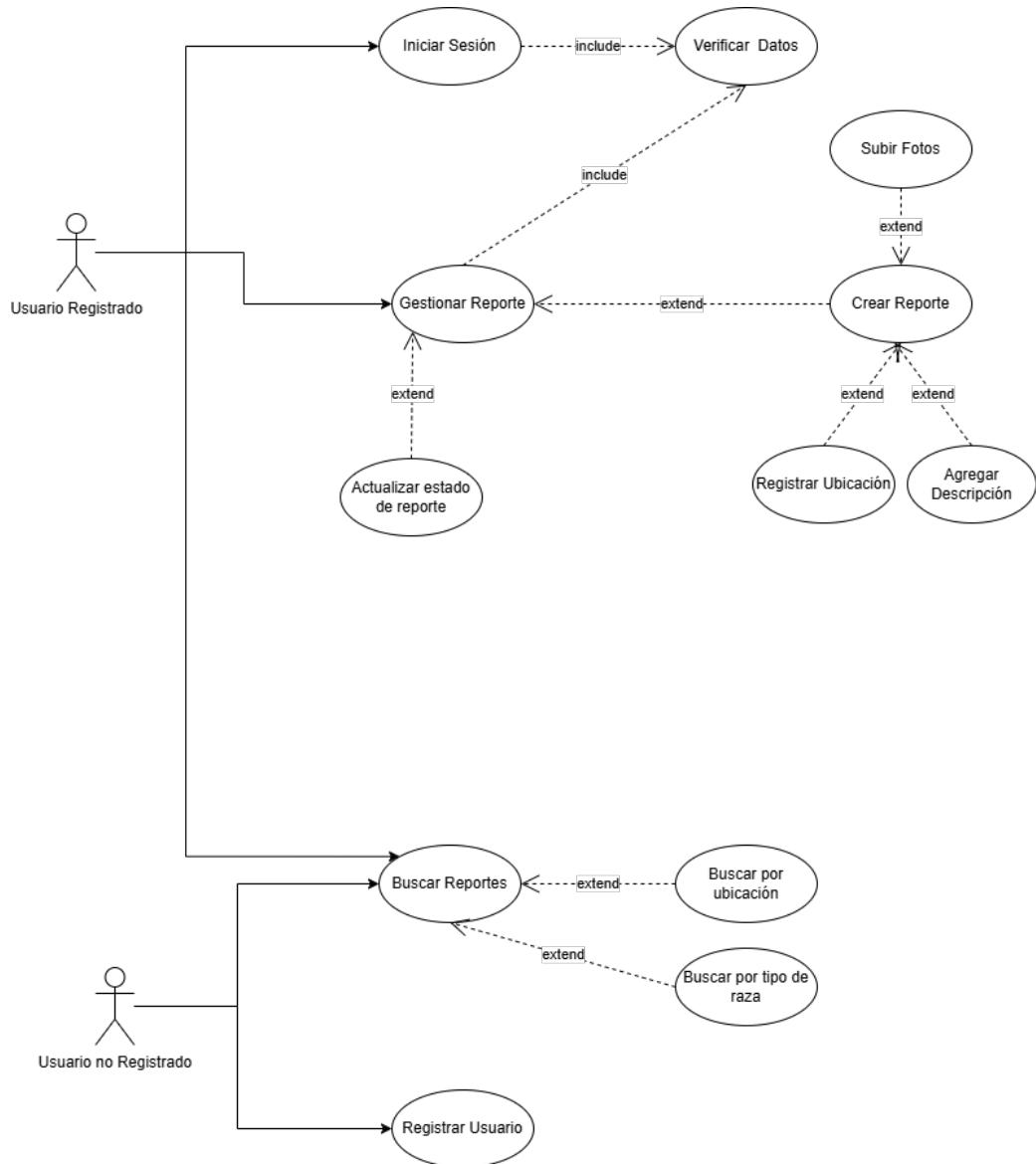
RQN-006	Documentación técnica
Descripción	Proporcionar documentación completa del sistema para facilitar el mantenimiento, actualizaciones y resolución de problemas.
Objetivo	NA
Importancia	Media
Estado	Aprobado
Estabilidad	Media
Comentario	Usar manuales técnicos claros y herramientas estándar.

Requerimientos de Rendimiento

RQR-001	Cantidad de usuarios simultaneo
Descripción	Se espera por lo menos 5 usuarios simultáneos en el sistema sin causar problemas
Objetivo	NA
Importancia	Alta
Estado	Aprobado
Estabilidad	Media
Comentario	NA

RQR-002	Velocidad de carga
Descripción	Se espera que la interface web se demore menos de 30s para cargar desde LocalHost o en la misma red privada
Objetivo	NA
Importancia	Media
Estado	Aprobado
Estabilidad	Media
Comentario	NA

Diagrama de casos de uso



Casos de Uso de Alto Nivel

CU-001	Iniciar sesión
Requerimientos	RQF-001, RQF-002
Casos de Uso relacionados	NA
Actores	ACT-01
Tipo	Primario
Descripción	El usuario registrado accede al sistema proporcionando credenciales válidas.

CU-002	Verificar Datos
Requerimientos	RQF-002, RQF-003
Casos de Uso relacionados	NA
Actores	ACT-01
Tipo	Secundario
Descripción	El sistema valida los datos proporcionados por el usuario durante el inicio de sesión.

CU-003	Subir Fotos
Requerimientos	RQF-005
Casos de Uso relacionados	NA
Actores	ACT-01
Tipo	Secundario
Descripción	El usuario registrado agrega imágenes a su perfil o reportes dentro del sistema.

CU-004	Gestionar reporte
Requerimientos	RQF-006, RQF-007
Casos de Uso relacionados	NA
Actores	ACT-01
Tipo	Primario
Descripción	El usuario registrado crea, edita, actualiza reportes relacionados con mascotas perdidas o encontradas.

CU-005	Crear reporte
Requerimientos	RQF-006, RQF-007

Casos de Uso relacionados	CU-004
Actores	ACT-01
Tipo	Secundario
Descripción	El usuario registrado crea un nuevo reporte de mascota perdida.

CU-006	Registrar Ubicación
Requerimientos	RQF-009
Casos de Uso relacionados	CU-005
Actores	ACT-01
Tipo	Secundario
Descripción	El usuario registrado agrega la ubicación geográfica al crear un reporte.

CU-007	Agregar descripción
Requerimientos	RQF-010
Casos de Uso relacionados	CU-005
Actores	ACT-01
Tipo	Secundario
Descripción	El usuario registrado proporciona detalles descriptivos sobre el reporte, incluyendo características de la mascota.

CU-008	Actualizar estado del reporte
Requerimientos	RQF-011
Casos de Uso relacionados	CU-004
Actores	ACT-01
Tipo	Secundario
Descripción	El usuario registrado cambia el estado del reporte (por ejemplo, de "perdido" a "encontrado").

CU-009	Buscar reportes
Requerimientos	RQF-014, RQF-015
Casos de Uso relacionados	NA
Actores	ACT-01, ACT-02

Tipo	Primario
Descripción	El usuario busca reportes de mascotas extraviadas o encontradas mediante diferentes criterios de filtrado.

CU-010	Buscar por ubicación
Requerimientos	RQF-017
Casos de Uso relacionados	CU-009
Actores	ACT-01, ACT-02
Tipo	Secundario
Descripción	El usuario busca reportes mediante la ubicación geográfica proporcionada en el sistema.

CU-011	Buscar por tipo de raza
Requerimientos	RQF-018
Casos de Uso relacionados	CU-009
Actores	ACT-01,
Tipo	Secundario
Descripción	El usuario filtra los reportes utilizando el tipo de raza como criterio de búsqueda.

CU-012	Registrar usuario
Requerimientos	RQF-019
Casos de Uso relacionados	NA
Actores	ACT-02
Tipo	Primario
Descripción	El usuario no registrado crea una cuenta en el sistema proporcionando sus datos personales y configurando credenciales de acceso.

Casos de Uso Expandido

CU-001	Iniciar sesión
Requerimientos	RQF-001, RQF-002
Casos de Uso relacionados	NA
Actores	ACT-01
Tipo	Primario

Descripción	El usuario registrado accede al sistema proporcionando credenciales válidas.	
Precondiciones	1. El usuario debe estar registrado en el sistema. 2. El usuario debe tener credenciales válidas.	
Postcondiciones	1. El sistema activa la sesión del usuario. 2. El usuario accede al panel principal.	
	ACCIÓN DEL ACTOR	RESPUESTA DEL SISTEMA
Curso típico de eventos	1. El usuario accede al formulario de inicio de sesión. 2. Ingresa sus credenciales (usuario y contraseña). 3. Oprime el botón "Iniciar sesión".	1. El sistema verifica las credenciales ingresadas. 2. Si son válidas, redirige al usuario al panel principal.
	CURSO ALTERNATIVO	
	Acción: Si las credenciales son incorrectas, el sistema muestra un mensaje de error y permite reintentar.	
Importancia	Alta	
Comentarios	Este caso de uso es fundamental para garantizar el acceso seguro al sistema.	

CU-002	Verificar Datos	
Requerimientos	RQF-002, RQF-003	
Casos de Uso relacionados	NA	
Actores	ACT-01	
Tipo	Secundario	
Descripción	El sistema valida los datos proporcionados por el usuario durante el inicio de sesión.	
Precondiciones	1. El usuario debe haber ingresado sus credenciales.	
Postcondiciones	1. El sistema valida que los datos coinciden con los registrados en la base de datos.	
Curso típico de eventos	ACCIÓN DEL ACTOR	RESPUESTA DEL SISTEMA
	1. El usuario envía los datos de inicio de sesión.	1. El sistema valida las credenciales ingresadas. 2. Notifica al usuario si los datos son válidos.
	CURSO ALTERNATIVO	
	Acción: Si los datos no coinciden, el sistema muestra un mensaje de error y finaliza la operación.	
Importancia	Media	
Comentarios	Este caso de uso garantiza la seguridad de los accesos al sistema.	

CU-003	Subir Fotos	
Requerimientos	RQF-005	
Casos de Uso relacionados	NA	
Actores	ACT-01	
Tipo	Secundario	
Descripción	El usuario registrado agrega imágenes a su perfil o reportes dentro del sistema.	
Precondiciones	1. El usuario debe estar autenticado en el sistema.	
Postcondiciones	1. Las fotos subidas se almacenan correctamente en el sistema.	
Curso típico de eventos	ACCIÓN DEL ACTOR	RESPUESTA DEL SISTEMA
	1. El usuario selecciona la opción “Subir fotos”.	1. El sistema carga la foto seleccionada.
	2. Escoge una imagen desde su dispositivo.	2. Notifica al usuario que la foto se ha subido exitosamente.
	3. Oprime el botón “Cargar”.	
	CURSO ALTERNATIVO	
	Acción: Si la imagen no cumple con los requisitos, el sistema notifica el error y cancela la operación.	
Importancia	Baja	
Comentarios	Útil para complementar los reportes y personalizar perfiles.	

CU-004	Gestionar reporte	
Requerimientos	RQF-006, RQF-007	
Casos de Uso relacionados	NA	
Actores	ACT-01	
Tipo	Primario	
Descripción	El usuario registrado crea, edita, actualiza o elimina reportes relacionados con mascotas perdidas o encontradas.	
Precondiciones	1. El usuario debe haber iniciado sesión.	
Postcondiciones	1. Los cambios realizados en el reporte se reflejan correctamente en el sistema.	
Curso típico de eventos	ACCIÓN DEL ACTOR	RESPUESTA DEL SISTEMA
	1. El usuario selecciona la opción de gestionar reportes.	1. El sistema ejecuta la acción seleccionada por el usuario.
	2. Elige una acción específica (crear, editar, eliminar o actualizar estado).	2. Notifica si la operación fue exitosa.
	CURSO ALTERNATIVO	
	Acción: Si ocurre un error en el sistema, este muestra un mensaje indicando el problema.	
Importancia	Alta	

Comentarios	Este caso de uso centraliza todas las operaciones relacionadas con los reportes.
--------------------	--

CU-005	Crear reporte	
Requerimientos	RQF-006, RQF-007	
Casos de Uso relacionados	CU-004	
Actores	ACT-01	
Tipo	Secundario	
Descripción	El usuario registrado crea un nuevo reporte de mascota perdida o encontrada.	
Precondiciones	1. El usuario debe haber iniciado sesión.	
Postcondiciones	1. El nuevo reporte se almacena correctamente en la base de datos.	
Curso típico de eventos	ACCIÓN DEL ACTOR	RESPUESTA DEL SISTEMA
	1. El usuario selecciona la opción "Crear reporte".	1. El sistema valida los datos ingresados.
	2. Ingrera los datos del reporte.	2. Almacena el reporte en la base de datos.
	3. Oprime el botón "Guardar".	3. Notifica que el reporte ha sido creado exitosamente.
	CURSO ALTERNATIVO	
	Acción: Si los datos no son válidos, el sistema muestra un mensaje de error.	
Importancia	Alta	
Comentarios	Este caso de uso es clave para la funcionalidad principal del sistema.	

CU-006	Registrar Ubicación	
Requerimientos	RQF-009	
Casos de Uso relacionados	CU-005	
Actores	ACT-01	
Tipo	Secundario	
Descripción	El usuario registrado agrega la ubicación geográfica al crear un reporte.	
Precondiciones	1. El usuario debe estar creando un reporte. 2. La ubicación debe ser válida.	
Postcondiciones	1. La ubicación queda registrada en el reporte.	
Curso típico de	ACCIÓN DEL ACTOR	RESPUESTA DEL SISTEMA

eventos	1. El usuario selecciona la opción "Aregar ubicación". 2. Introduce la ubicación manualmente o la selecciona desde un mapa.	1. El sistema valida la ubicación. 2. Muestra un mensaje de éxito.
CURSO ALTERNATIVO		
Acción: Si no se proporciona una ubicación válida, el sistema solicita corregir el dato.		
Importancia	Media	
Comentarios	Importante para identificar la zona relacionada con el reporte.	

CU-007	Agregar descripción	
Requerimientos	RQF-010	
Casos de Uso relacionados	CU-005	
Actores	ACT-01	
Tipo	Secundario	
Descripción	El usuario registrado proporciona detalles descriptivos sobre el reporte, incluyendo características de la mascota.	
Precondiciones	1. El usuario debe estar creando un reporte.	
Postcondiciones	1. La descripción queda guardada en el reporte.	
Curso típico de eventos	ACCIÓN DEL ACTOR	RESPUESTA DEL SISTEMA
	1. El usuario selecciona el campo de descripción.	1. El sistema guarda la información ingresada.
	2. Introduce los detalles necesarios.	2. Muestra un mensaje de éxito.
	3. Confirma la operación.	
CURSO ALTERNATIVO		
Acción: Si la descripción es insuficiente, el sistema solicita completar más detalles.		
Importancia	Media	
Comentarios	La descripción ayuda a identificar y diferenciar los reportes de mascotas.	

CU-008	Actualizar estado del reporte	
Requerimientos	RQF-011	
Casos de Uso relacionados	CU-004	
Actores	ACT-01	
Tipo	Secundario	
Descripción	El usuario registrado cambia el estado del reporte (por ejemplo, de "perdido" a "encontrado").	

Precondiciones	1. El reporte debe existir en el sistema. 2. El usuario debe tener permisos para modificarlo.	
Postcondiciones	1. El estado del reporte se actualiza correctamente.	
Curso típico de eventos	ACCIÓN DEL ACTOR	RESPUESTA DEL SISTEMA
	1. El usuario selecciona un reporte existente.	1. El sistema guarda el nuevo estado.
	2. Cambia el estado.	2. Muestra un mensaje de éxito.
	3. Confirma la acción.	
	CURSO ALTERNATIVO	
	Acción: Si ocurre un error, el sistema informa al usuario sobre el problema.	
Importancia	Media	
Comentarios	Permite mantener los reportes actualizados según el progreso de los casos.	

CU-009	Buscar reportes
Requerimientos	RQF-014, RQF-015
Casos de Uso relacionados	NA

Actores	ACT-01, ACT-02	
Tipo	Primario	
Descripción	El usuario busca reportes de mascotas extraviadas o encontradas mediante diferentes criterios de filtrado.	
Precondiciones	1. Debe haber reportes existentes en el sistema.	
Postcondiciones	1. Se muestran los resultados que cumplen con los criterios de búsqueda.	
Curso típico de eventos	ACCIÓN DEL ACTOR	RESPUESTA DEL SISTEMA
	1. El usuario accede a la función de búsqueda.	1. El sistema realiza la búsqueda.
	2. Ingresa los criterios de búsqueda.	2. Muestra los reportes correspondientes.
	3. Solicita los resultados.	
CURSO ALTERNATIVO		
	Acción: Si no se encuentran resultados, el sistema muestra un mensaje indicando la falta de coincidencias.	
Importancia	Alta	
Comentarios	Es una funcionalidad clave para conectar a usuarios con reportes relevantes.	

CU-010	Buscar por ubicación	
Requerimientos	RQF-017	
Casos de Uso relacionados	CU-009	
Actores	ACT-01, ACT-02	
Tipo	Secundario	
Descripción	El usuario busca reportes mediante la ubicación geográfica proporcionada en el sistema.	
Precondiciones	1. Debe haber reportes con ubicaciones registradas.	
Postcondiciones	1. Se muestran resultados relacionados con la ubicación especificada.	
Curso típico de eventos	ACCIÓN DEL ACTOR	RESPUESTA DEL SISTEMA
	1. El usuario selecciona el filtro de ubicación.	1. El sistema localiza los reportes relacionados.
	2. Ingresa una dirección o área.	2. Muestra los resultados encontrados.
	3. Solicita los resultados.	
CURSO ALTERNATIVO		
	Acción: Si no se encuentran reportes, el sistema informa al usuario.	
Importancia	Media	
Comentarios	Este filtro facilita búsquedas precisas por localización.	

CU-011	Buscar por tipo de raza	
Requerimientos	RQF-018	

Casos de Uso relacionados	CU-009	
Actores	ACT-01	
Tipo	Secundario	
Descripción	El usuario filtra los reportes utilizando el tipo de raza como criterio de búsqueda.	
Precondiciones	1. Debe haber reportes con razas registradas.	
Postcondiciones	1. Se muestran resultados relacionados con la raza especificada.	
Curso típico de eventos	ACCIÓN DEL ACTOR	RESPUESTA DEL SISTEMA
	1. El usuario selecciona el filtro de raza.	1. El sistema filtra los reportes.
	2. Indica la raza deseada.	2. Muestra los resultados encontrados.
	3. Solicita los resultados.	
	CURSO ALTERNATIVO	
	Acción: Si no hay reportes para la raza especificada, el sistema informa al usuario.	
Importancia	Media	
Comentarios	Útil para búsquedas específicas según características de las mascotas.	

CU-012	Registrar usuario	
Requerimientos	RQF-019	
Casos de Uso relacionados	NA	
Actores	ACT-02	
Tipo	Primario	
Descripción	El usuario no registrado crea una cuenta en el sistema proporcionando sus datos personales y configurando credenciales de acceso.	
Precondiciones	1. El usuario no debe tener una cuenta existente.	
Postcondiciones	1. La cuenta se registra exitosamente en el sistema.	
Curso típico de eventos	ACCIÓN DEL ACTOR	RESPUESTA DEL SISTEMA
	1. El usuario accede a la opción de registro.	1. El sistema valida los datos ingresados.
	2. Completa el formulario con sus datos.	2. Crea la cuenta y envía un mensaje de confirmación.
	3. Envía la solicitud de registro.	
	CURSO ALTERNATIVO	
	Acción: Si los datos ingresados son incorrectos, el sistema solicita corregirlos.	
Importancia	Alta	
Comentarios	Es esencial para permitir que nuevos usuarios accedan a todas las funcionalidades del sistema.	

Diagrama de secuencia

DIAGRAMA DE SECUENCIA GENERAL DEL SISTEMA

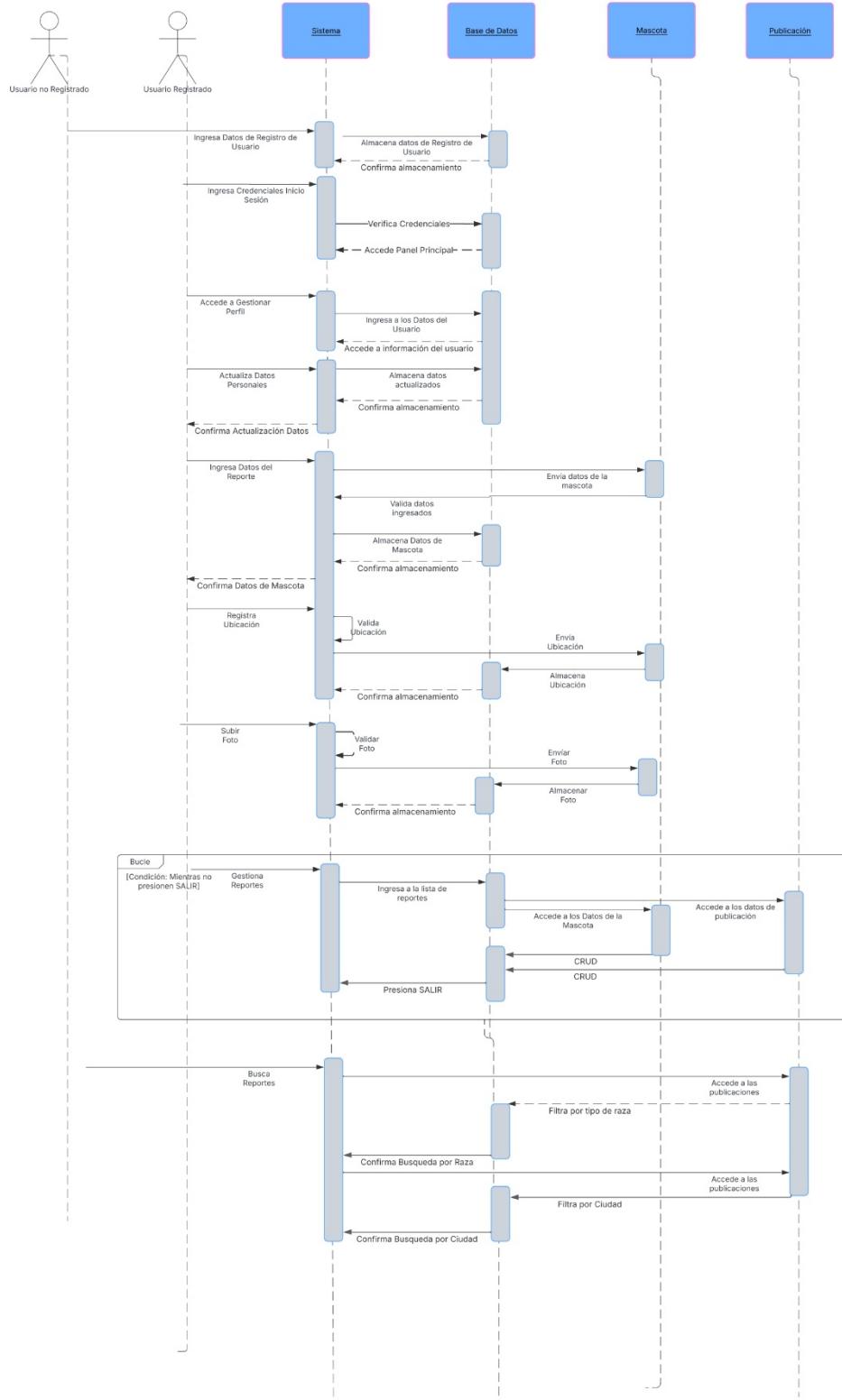
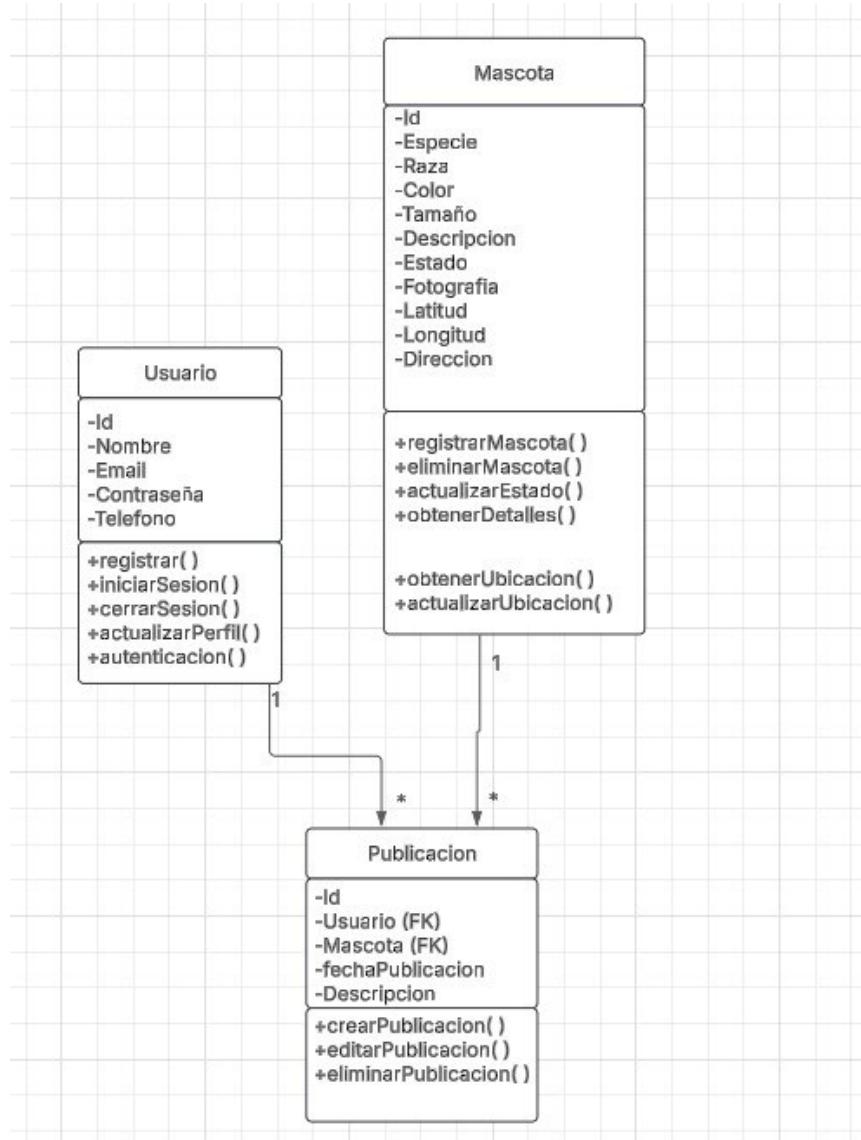


Diagrama de clases



Pruebas unitarias

Se procede a realizar las pruebas unitarias referentes a el código y su funcionalidad.

Para realizar las pruebas unitarias estamos usando UNITTEST es un módulo estándar de Python para escribir y ejecutar pruebas unitarias.

¿Qué hace unittest?

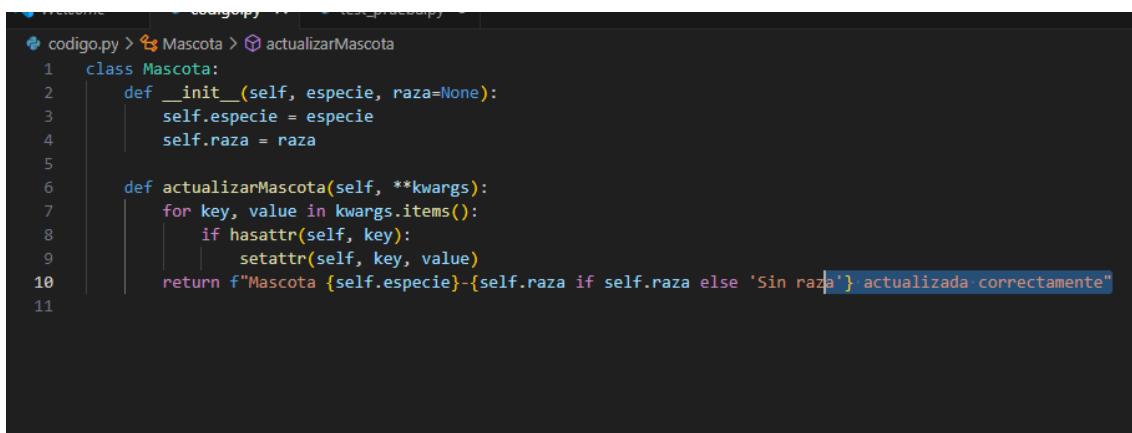
- Permite crear **pruebas automáticas** para funciones y clases.
- Verifica si el código funciona como se espera.
- Genera reportes de pruebas con **OK, FAIL o ERROR**.
- Facilita la detección de errores antes de que el código llegue a producción.

¿Cómo usar unittest?

1. Importar unittest.
2. Crear una clase que herede de unittest.TestCase.
3. Definir métodos que comiencen con test_ para cada caso de prueba.
4. Ejecutar las pruebas con python -m unittest estas las ejecutamos en cmd
5. Recordatorio el archivo clase.py y test_prueba.py deben estar en la misma carpeta

Pasos.

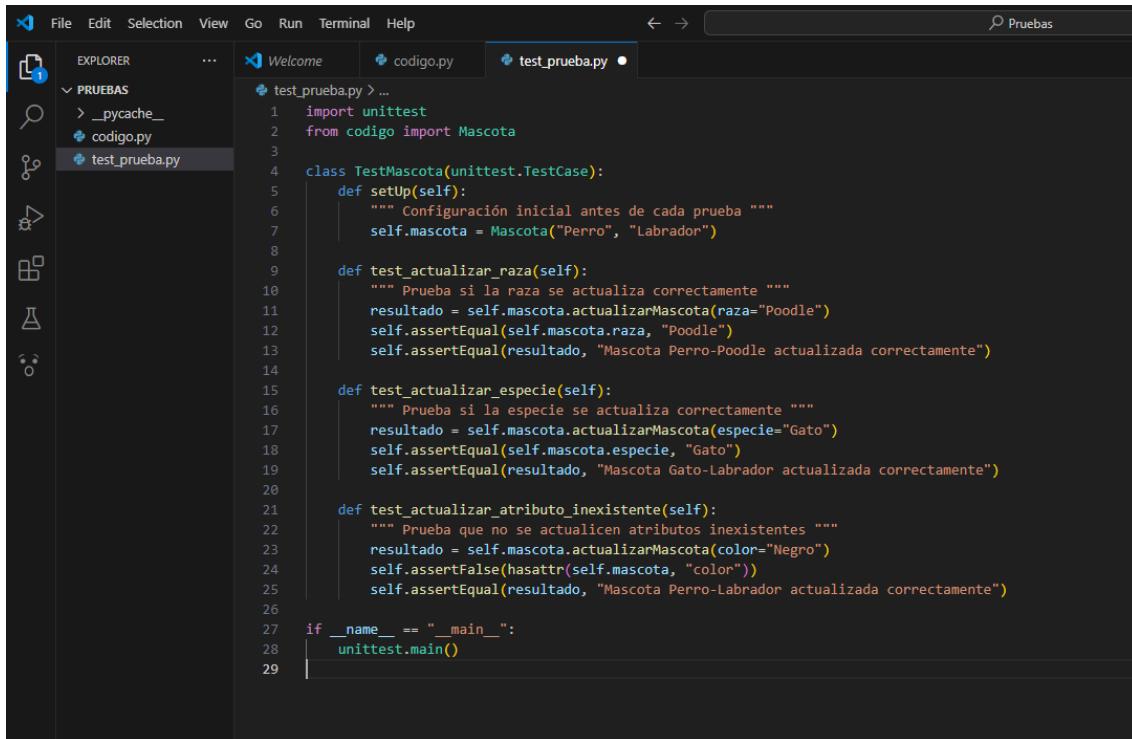
1. Código o función que queremos verificar lo colocamos en código.py



```
1  class Mascota:
2      def __init__(self, especie, raza=None):
3          self.especie = especie
4          self.raza = raza
5
6      def actualizarMascota(self, **kwargs):
7          for key, value in kwargs.items():
8              if hasattr(self, key):
9                  setattr(self, key, value)
10         return f"{'Mascota {self.especie}-{self.raza if self.raza else 'Sin raza'} actualizada correctamente'}
```

2. Creamos otro archivo esta ves llamo test_prueba.py donde vamos a realizar las pruebas si es que la raza se actualiza correctamente, si es que la especie se actualiza correctamente y también una prueba para que no se

actualicen atributos inexistentes



```
import unittest
from codigo import Mascota

class TestMascota(unittest.TestCase):
    def setUp(self):
        """ Configuración inicial antes de cada prueba """
        self.mascota = Mascota("Perro", "Labrador")

    def test_actualizar_raza(self):
        """ Prueba si la raza se actualiza correctamente """
        resultado = self.mascota.actualizarMascota(raza="Poodle")
        self.assertEqual(self.mascota.raza, "Poodle")
        self.assertEqual(resultado, "Mascota Perro-Poodle actualizada correctamente")

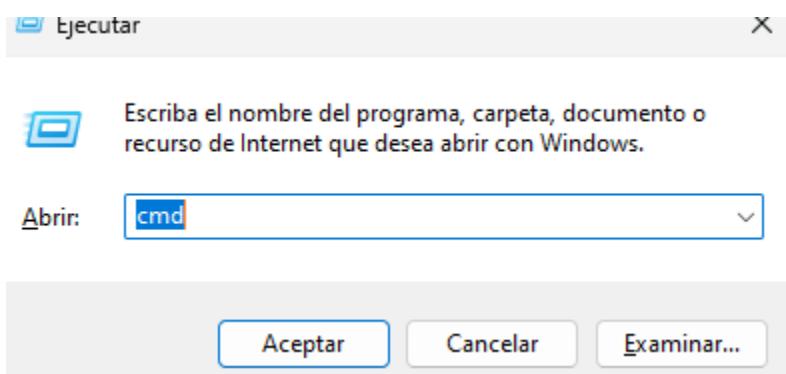
    def test_actualizar_especie(self):
        """ Prueba si la especie se actualiza correctamente """
        resultado = self.mascota.actualizarMascota(especie="Gato")
        self.assertEqual(self.mascota.especie, "Gato")
        self.assertEqual(resultado, "Mascota Gato-Labrador actualizada correctamente")

    def test_actualizar_atributo_inexistente(self):
        """ Prueba que no se actualicen atributos inexistentes """
        resultado = self.mascota.actualizarMascota(color="Negro")
        self.assertFalse(hasattr(self.mascota, "color"))
        self.assertEqual(resultado, "Mascota Perro-Labrador actualizada correctamente")

if __name__ == "__main__":
    unittest.main()
```

Muy importante que importemos unittest

Ahora como paso final vamos a abrir CMD escribiendo Windows R en el buscador así se abrirá el símbolo de sistema, ahora se utiliza el comando cd para ubicar la carpeta donde están ubicados los 2 archivos en este caso pruebas



```
Microsoft Windows [Versión 10.0.22631.4460]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Usuario PC>cd C:\Users\Usuario PC\Desktop\Pruebas
```

Ahora para ejecutar el archivo test_prueba.py tenemos que usar el siguiente comando py -m unittest test_prueba, antes verificar tener instalado en el PATH para poder ejecutar el comando py

Si es que nuestra función está bien realizada nos va a salir un ok de esta manera y como hicimos 3 pruebas en el código de arriba comprobamos que corre los 3 test de manera correcta

```
cmd C:\WINDOWS\system32\cmd. X + v

C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba
...
-----
Ran 3 tests in 0.000s
OK

C:\Users\Usuario PC\Desktop\Pruebas>
```

Si es que no está bien hecha la función nos saldría FAIL O ERROR (error a propósito)

```

C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba
E
=====
ERROR: test_prueba (unittest.loader._FailedTest.test_prueba)
-----
ImportError: Failed to import test module: test_prueba
Traceback (most recent call last):
  File "C:\Python312\Lib\unittest\loader.py", line 137, in loadTestsFromName
    module = __import__(module_name)
               ^^^^^^^^^^^^^^^^^^^^
  File "C:\Users\Usuario PC\Desktop\Pruebas\test_prueba.py", line 2, in <module>
    from prueba import Mascota
ModuleNotFoundError: No module named 'prueba'

-----
Ran 1 test in 0.000s

FAILED (errors=1)

C:\Users\Usuario PC\Desktop\Pruebas>

```

De esta manera vamos haciendo pruebas con cada función de LOCALPET para asegurarnos su correcto funcionamiento

PRUEBAS UNITARIAS:

Prueba MODELS-Eliminar mascota

Código original

```

Welcome prueba.py test_prueba.py
prueba.py > ...
1  class Mascota:
2      def __init__(self, especie=None, raza=None):
3          self.especie = especie
4          self.raza = raza
5
6      def eliminarMascota(self):
7          # Simula la eliminación de una mascota
8          self.especie = None
9          self.raza = None
10         return f"Mascota {self.especie}-{self.raza} eliminada con éxito."
11
12     def actualizarMascota(self, **kwargs):
13         for key, value in kwargs.items():
14             if hasattr(self, key):
15                 setattr(self, key, value)
16         self.save() # Simula el guardar en la base de datos
17         return f"Mascota {self.especie}-{self.raza if self.raza else 'Sin raza'} actualizada correctamente"
18
19     def save(self):
20         # Simula la acción de guardar la mascota, como si fuera a la base de datos
21         pass
22

```

Codigo _ test

```
test_prueba.py > ...
1  import unittest
2  from prueba import Mascota
3
4  class TestMascota(unittest.TestCase):
5
6      def setUp(self):
7          """Configuración inicial antes de cada prueba"""
8          # Crea un objeto de la clase Mascota para cada prueba
9          self.mascota = Mascota(especie="Perro", raza="Labrador")
10
11     def test_actualizar_mascota(self):
12         """Prueba si la actualización de la mascota funciona correctamente"""
13         resultado = self.mascota.actualizarMascota(especie="Gato", raza="Siames")
14         self.assertEqual(self.mascota.especie, "Gato")
15         self.assertEqual(self.mascota.raza, "Siames")
16         self.assertEqual(resultado, "Mascota Gato-Siames actualizada correctamente")
17
18     def test_eliminar_mascota(self):
19         """Prueba si la mascota se elimina correctamente"""
20         resultado = self.mascota.eliminarMascota()
21         self.assertIsNone(self.mascota.especie)
22         self.assertIsNone(self.mascota.raza)
23         self.assertEqual(resultado, "Mascota None-None eliminada con éxito.")
24         print("Mascota eliminada correctamente")
25
26 if __name__ == "__main__":
27     unittest.main()
28
```

Captura cmd

```
C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba
.Mascota eliminada correctamente
.
-----
Ran 2 tests in 0.000s

OK

C:\Users\Usuario PC\Desktop\Pruebas>
```

Prueba Obtener Detalles

Código Original

```
❸ prueba.py > ...
1  class Mascota:
2      def __init__(self, id=None, especie=None, raza=None, color=None, tamaño=None, descripcion=None, estado=None, fotografía=None, dirección=None):
3          self.id = id
4          self.especie = especie
5          self.raza = raza
6          self.color = color
7          self.tamaño = tamaño
8          self.descripcion = descripcion
9          self.estado = estado
10         self.fotografía = fotografía
11         self.dirección = dirección
12
13     def obtenerDetalles(self):
14         return {
15             "ID": self.id,
16             "Especie": self.especie,
17             "Raza": self.raza if self.raza else "Sin raza",
18             "Color": self.color,
19             "Tamaño": self.tamaño,
20             "Descripción": self.descripcion,
21             "Estado": self.estado,
22             "Fotografía": getattr(self.fotografía, 'url', "No disponible"),
23             "Dirección": self.dirección if self.dirección else "No especificada"
24         }
25
```

Código Test

```

prueba.py > testmascota > setup
import unittest
from prueba import Mascota

class TestMascota(unittest.TestCase):

    def setUp(self):
        """Configuración inicial antes de cada prueba"""
        # Crea un objeto de la clase Mascota con algunos detalles
        self.mascota = Mascota([
            {
                id=1,
                especie="Perro",
                raza="Labrador",
                color="Negro",
                tamaño="Grande",
                descripción="Activo y juguetón",
                estado="Saludable",
                fotografía=None, # Si no hay URL de fotografía
                dirección="123 Calle Ficticia"
            }
        ])

    def test_obtener_detalles(self):
        """Prueba si la función obtenerDetalles devuelve los detalles correctamente"""
        detalles = self.mascota.obtenerDetalles()

        self.assertEqual(detalles["ID"], 1)
        self.assertEqual(detalles["Especie"], "Perro")
        self.assertEqual(detalles["Raza"], "Labrador")
        self.assertEqual(detalles["Color"], "Negro")
        self.assertEqual(detalles["Tamaño"], "Grande")
        self.assertEqual(detalles["Descripción"], "Activo y juguetón")
        self.assertEqual(detalles["Estado"], "Saludable")
        self.assertEqual(detalles["Fotografía"], "No disponible") # Porque no se pasó una URL
        self.assertEqual(detalles["Dirección"], "123 Calle Ficticia")

        print("\nTest 'test_obtener_detalles' PASADO: Detalles de la mascota obtenidos correctamente.")

    def test_obtener_detalles_sin_raza(self):
        """Prueba si la función obtenerDetalles maneja el caso cuando no hay raza"""
        mascota_sin_raza = Mascota([
            {
                id=2,
                especie="Gato",
                raza=None, # Sin raza
                color="Blanco",
                tamaño="Pequeño",
                descripción="Tímido",
                estado="Saludable",
                fotografía=None,
                dirección="456 Calle Real"
            }
        ])

```

```

def test_obtener_detalles_sin_raza(self):
    """Prueba si la función obtenerDetalles maneja el caso cuando no hay raza"""
    mascota_sin_raza = Mascota(
        id=2,
        especie="Gato",
        raza=None, # Sin raza
        color="Blanco",
        tamaño="Pequeño",
        descripción="Timido",
        estado="Saludable",
        fotografía=None,
        dirección="456 Calle Real"
    )

    detalles = mascota_sin_raza.obtenerDetalles()

    self.assertEqual(detalles["Raza"], "Sin raza") # Debería devolver "Sin raza"

    print("\nTest 'test_obtener_detalles_sin_raza' PASADO: Mascota sin raza manejada correctamente.")

def test_obtener_detalles_sin_dirección(self):
    """Prueba si la función obtenerDetalles maneja el caso cuando no hay dirección"""
    mascota_sin_dirección = Mascota(
        id=3,
        especie="Conejo",
        raza="Miniatura",
        color="Marrón",
        tamaño="Pequeño",
        descripción="Saltador",
        estado="Saludable",
        fotografía=None,
        dirección=None # Sin dirección
    )

    detalles = mascota_sin_dirección.obtenerDetalles()

    self.assertEqual(detalles["Dirección"], "No especificada") # Debería devolver "No especificada"

    print("\nTest 'test_obtener_detalles_sin_dirección' PASADO: Mascota sin dirección manejada correctamente.")

if __name__ == "__main__":
    unittest.main()

```

Captura

```

Símbolo del sistema
C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba
Test 'test_obtener_detalles' PASADO: Detalles de la mascota obtenidos correctamente.
.
Test 'test_obtener_detalles_sin_direccion' PASADO: Mascota sin dirección manejada correctamente.
.
Test 'test_obtener_detalles_sin_raza' PASADO: Mascota sin raza manejada correctamente.
-----
Ran 3 tests in 0.001s
OK
C:\Users\Usuario PC\Desktop\Pruebas>

```

Prueba Obtener Ubicación y Actualizar ubicación

Código Original

```

prueba.py > Mascota > actualizarUbicacion
1  class Mascota:
2      def __init__(self, id, especie, raza=None, color=None, tamaño=None, descripcion=None, estado=None, fotografía=None, dirección=None):
3          self.id = id
4          self.especie = especie
5          self.raza = raza
6          self.color = color
7          self.tamaño = tamaño
8          self.descripcion = descripcion
9          self.estado = estado
10         self.fotografía = fotografía
11         self.dirección = dirección
12
13     def obtenerUbicacion(self):
14         return self.dirección if self.dirección else "Ubicación no disponible"
15
16     def actualizarUbicacion(self, nueva_dirección):
17         self.dirección = nueva_dirección
18         self.save()
19         return f"Ubicación actualizada a: {self.dirección}"
20
21     def save(self):
22         """Simulación de un método de guardado"""
23         print(f"Guardando cambios para la mascota {self.id}")
24

```

Código Test

```

Welcome | prueba.py | test_prueba.py x
test_prueba.py > ...
1  import unittest
2  from prueba import Mascota
3
4  class TestMascota(unittest.TestCase):
5
6      def setUp(self):
7          """Configuración inicial antes de cada prueba"""
8          # Crea un objeto de la clase Mascota con algunos detalles
9          self.mascota = Mascota(
10              id=1,
11              especie="Perro",
12              raza="Labrador",
13              color="Negro",
14              tamaño="Grande",
15              descripción="Activo y juguetón",
16              estado="Saludable",
17              fotografía=None, # Si no hay URL de fotografía
18              dirección="123 Calle Ficticia"
19          )
20
21      def test_obtener_ubicacion(self):
22          """Prueba si la función obtenerUbicacion devuelve la ubicación correctamente"""
23          ubicación = self.mascota.obtenerUbicacion()
24
25          self.assertEqual(ubicación, "123 Calle Ficticia")
26          print("\nTest 'test_obtener_ubicacion' PASADO: Ubicación obtenida correctamente.")
27
28      def test_actualizar_ubicacion(self):
29          """Prueba si la función actualizarUbicacion actualiza la ubicación correctamente"""
30          nueva_ubicación = "456 Calle Nueva"
31          mensaje = self.mascota.actualizarUbicacion(nueva_ubicación)
32
33          self.assertEqual(self.mascota.dirección, nueva_ubicación)
34          self.assertEqual(mensaje, f"Ubicación actualizada a: {nueva_ubicación}")
35          print("\nTest 'test_actualizar_ubicacion' PASADO: Ubicación actualizada correctamente.")
36
37  if __name__ == "__main__":
38      unittest.main()
39

```

Captura

```
C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba
Guardando cambios para la mascota 1

Test 'test_actualizar_ubicacion' PASADO: Ubicación actualizada correctamente.
.
Test 'test_obtener_ubicacion' PASADO: Ubicación obtenida correctamente.
.

Ran 2 tests in 0.000s

OK

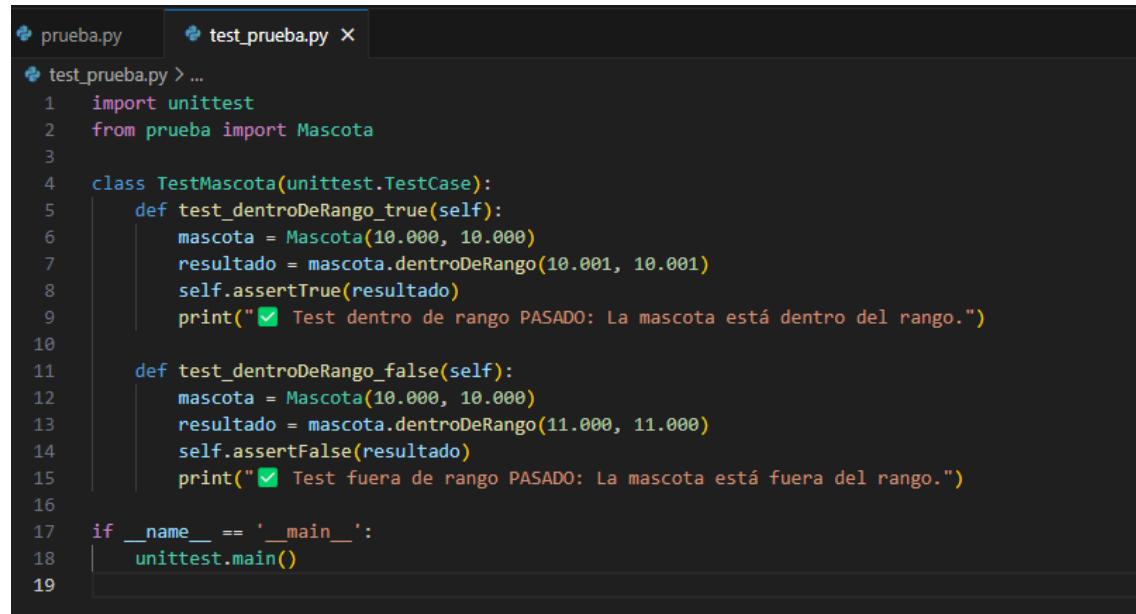
C:\Users\Usuario PC\Desktop\Pruebas>
```

Prueba Función-Ubicación dentro de un rango

Código Original

```
def dentroDeRango(self,latitud,longitud):
    #calcula la distancia entre la ubicacion de la mascota y las coordenadas de
    rango=1
    cordenada1=(self.latitud,self.longitud)
    cordenada2=(latitud,longitud)
    distancia=geodesic(cordenada1,cordenada2).kilometers
    return distancia<=rango
```

Código Test



```
prueba.py test_prueba.py x

# test_prueba.py > ...
1 import unittest
2 from prueba import Mascota
3
4 class TestMascota(unittest.TestCase):
5     def test_dentroDeRango_true(self):
6         mascota = Mascota(10.000, 10.000)
7         resultado = mascota.dentroDeRango(10.001, 10.001)
8         self.assertTrue(resultado)
9         print("✓ Test dentro de rango PASADO: La mascota está dentro del rango.")
10
11    def test_dentroDeRango_false(self):
12        mascota = Mascota(10.000, 10.000)
13        resultado = mascota.dentroDeRango(11.000, 11.000)
14        self.assertFalse(resultado)
15        print("✗ Test fuera de rango PASADO: La mascota está fuera del rango.")
16
17 if __name__ == '__main__':
18     unittest.main()
```

Captura

```
C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py
La ubicación está fuera del rango permitido.
✓ Test fuera de rango PASADO: La mascota está fuera del rango.
.La ubicación está dentro del rango permitido.
✓ Test dentro de rango PASADO: La mascota está dentro del rango.

-----
Ran 2 tests in 0.001s

OK
C:\Users\Usuario PC\Desktop\Pruebas>
```

Prueba Class Publicación – Función crear publicación

Código Original

```
#Modelo de Publicacion
class Publicacion(models.Model):
    id=models.AutoField(primary_key=True)
    usuario=models.ForeignKey('Usuario',on_delete=models.CASCADE)
    mascota=models.ForeignKey('Mascota',on_delete=models.CASCADE)
    fecha_publicacion=models.DateTimeField(auto_now_add=True)
    comentarios=models.TextField(blank=True,null=True)

    def __str__(self):
        return f"Publicacion de {self.usuario.username} sobre {self.mascota.especie}"

    def crearPublicacion(self,usuario,mascota,comentarios=""):
        self.usuario=usuario
        self.mascota=mascota
        self.comentarios=comentarios
        self.save()
        return f"Publicacion creada por {self.usuario.username} sobre {self.mascota.especie}"
```

Código Test

```

import unittest
from prueba import Publicacion
class MockUsuario:
    def __init__(self, username):
        self.username = username
class MockMascota:
    def __init__(self, especie):
        self.especie = especie
class TestPublicacion(unittest.TestCase):
    def test_crear_publicacion(self):
        usuario = MockUsuario("Eduardo")
        mascota = MockMascota("Perro")
        publicacion = Publicacion()
        resultado = publicacion.crearPublicacion(usuario, mascota, "Prueba de publicación")
        self.assertEqual(resultado, "Publicacion creada por Eduardo sobre Perro")
if __name__ == '__main__':
    unittest.main()

```

Captura

```

C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py
Publicacion creada por Eduardo sobre Perro
.
-----
Ran 1 test in 0.000s
OK
C:\Users\Usuario PC\Desktop\Pruebas>

```

Prueba Función editar publicación

Código Original

```

def editarPublicacion(self,nuevo_comentario):
    self.comentarios=nuevo_comentario
    self.save()
    return f"Publicacion editada correctamente"

```

Código Test

```

import unittest
from prueba import Publicacion

class MockUsuario:
    def __init__(self, username):
        self.username = username

class MockMascota:
    def __init__(self, especie):
        self.especie = especie

class TestPublicacion(unittest.TestCase):
    def test_editar_publicacion(self):
        usuario = MockUsuario("Eduardo")
        mascota = MockMascota("Perro")
        publicacion = Publicacion(usuario, mascota, "Comentario original")

        resultado = publicacion.editarPublicacion("Nuevo comentario editado")

        self.assertEqual(publicacion.comentarios, "Nuevo comentario editado")
        self.assertEqual(resultado, "Publicacion editada correctamente")
        print("\n[TEST PASADO] test_editar_publicacion: La publicación fue editada correctamente.")

    def test_editar_publicacion_sin_comentario_inicial(self):
        usuario = MockUsuario("Ana")
        mascota = MockMascota("Gato")
        publicacion = Publicacion(usuario, mascota)

        resultado = publicacion.editarPublicacion("Comentario editado sin comentario inicial")

        self.assertEqual(publicacion.comentarios, "Comentario editado sin comentario inicial")
        self.assertEqual(resultado, "Publicacion editada correctamente")
        print("\n[TEST PASADO] test_editar_publicacion_sin_comentario_inicial: La publicación fue editada correctamente sin comentario inicial.")

    def test_editar_publicacion_con_comentario_vacio(self):
        usuario = MockUsuario("Carlos")
        mascota = MockMascota("Conejo")
        publicacion = Publicacion(usuario, mascota, "Comentario inicial")

        resultado = publicacion.editarPublicacion("")

        self.assertEqual(publicacion.comentarios, "")
        self.assertEqual(resultado, "Publicacion editada correctamente")
        print("\n[TEST PASADO] test_editar_publicacion_con_comentario_vacio: La publicación fue editada con un comentario vacío.")

    def test_editar_publicacion_con_comentario_vacio(self):
        self.assertEqual(publicacion.comentarios, "")
        self.assertEqual(resultado, "Publicacion editada correctamente")
        print("\n[TEST PASADO] test_editar_publicacion_con_comentario_vacio: La publicación fue editada con un comentario vacío.")

if __name__ == '__main__':
    print("\nEjecutando las pruebas unitarias...\n")
    unittest.main()

```

Captura

```

C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py
[TEST PASADO] test_editar_publicacion: La publicación fue editada correctamente.
[TEST PASADO] test_editar_publicacion_con_comentario_vacio: La publicación fue editada con un comentario vacío.
[TEST PASADO] test_editar_publicacion_sin_comentario_inicial: La publicación fue editada correctamente sin comentario inicial.
-----
Ran 3 tests in 0.001s
OK
C:\Users\Usuario PC\Desktop\Pruebas>

```

Prueba Función eliminar publicación

Código Original

```

def eliminarPublicacion(self):
    self.delete()
    return f"Publicacion eliminada con exito"

```

Código Test

```

prueba.py | test_prueba.py x
test_prueba.py > ...
1 import unittest
2 from prueba import Publicacion
3
4 class MockUsuario:
5     def __init__(self, username):
6         self.username = username
7
8 class MockMascota:
9     def __init__(self, especie):
10        self.especie = especie
11
12 class TestPublicacion(unittest.TestCase):
13     def test_eliminar_publicacion(self):
14         usuario = MockUsuario("Eduardo")
15         mascota = MockMascota("Perro")
16         publicacion = Publicacion(usuario, mascota, "Comentario de prueba")
17
18         resultado = publicacion.eliminarPublicacion()
19
20         # Como no se elimina realmente en este test, verificamos el mensaje de éxito
21         self.assertEqual(resultado, "Publicacion eliminada con exito")
22         print("\n[TEST PASADO] test_eliminar_publicacion: La publicación fue eliminada correctamente.")
23
24     def test_eliminar_publicacion_sin_comentario(self):
25         usuario = MockUsuario("Ana")
26         mascota = MockMascota("Gato")
27         publicacion = Publicacion(usuario, mascota)
28
29         resultado = publicacion.eliminarPublicacion()
30
31         # Como no se elimina realmente en este test, verificamos el mensaje de éxito
32         self.assertEqual(resultado, "Publicacion eliminada con exito")
33         print("\n[TEST PASADO] test_eliminar_publicacion_sin_comentario: La publicación fue eliminada correctamente sin comentario.")
34
35 if __name__ == '__main__':
36     print("\nEjecutando las pruebas unitarias...\n")
37     unittest.main()

```

Captura

```

C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py
[TEST PASADO] test_eliminar_publicacion: La publicación fue eliminada correctamente.
[TEST PASADO] test_eliminar_publicacion_sin_comentario: La publicación fue eliminada correctamente sin comentario.
-----
Ran 2 tests in 0.000s
OK

```

Prueba Función Agregar Comentarios

Código Original

```

def agregarComentarios(self,nuevo_comentario):
    self.comentarios+=f"\n{nuevo_comentario}"
    self.save()
    return "Comentario agregado correctamente"

```

Código Test

```

prueba.py      test_prueba.py
◆ test_prueba.py > ...
1 import unittest
2 from prueba import Publicacion
3
4 class MockUsuario:
5     def __init__(self, username):
6         self.username = username
7
8 class MockMascota:
9     def __init__(self, especie):
10        self.especie = especie
11
12 class TestPublicacion(unittest.TestCase):
13     def test_agregar_comentarios(self):
14         usuario = MockUsuario("Eduardo")
15         mascota = MockMascota("Perro")
16         publicacion = Publicacion(usuario, mascota, "Comentario inicial")
17
18         resultado = publicacion.agregarComentarios("Nuevo comentario agregado")
19
20         # Verificamos que el comentario se haya agregado correctamente
21         self.assertIn("Nuevo comentario agregado", publicacion.comentarios)
22         self.assertEqual(resultado, "Comentario agregado correctamente")
23         print("\n[TEST PASADO] test_agregar_comentarios: El comentario fue agregado correctamente.")
24
25     def test_agregar_comentarios_sin_comentario_inicial(self):
26         usuario = MockUsuario("Ana")
27         mascota = MockMascota("Gato")
28         publicacion = Publicacion(usuario, mascota)
29
30         resultado = publicacion.agregarComentarios("Comentario inicial agregado")
31
32         # Verificamos que el comentario se haya agregado correctamente
33         self.assertIn("Comentario inicial agregado", publicacion.comentarios)
34         self.assertEqual(resultado, "Comentario agregado correctamente")
35         print("\n[TEST PASADO] test_agregar_comentarios_sin_comentario_inicial: El comentario fue agregado correctamente sin comentario inicial.")
36
37 if __name__ == '__main__':
38     print("\nEjecutando las pruebas unitarias...\n")
39     unittest.main()
40

```

Captura

```

Símbolo del sistema
C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py
[TEST PASADO] test_agregar_comentarios: El comentario fue agregado correctamente.
[TEST PASADO] test_agregar_comentarios_sin_comentario_inicial: El comentario fue agregado correctamente sin comentario inicial.

Ran 2 tests in 0.000s
OK
C:\Users\Usuario PC\Desktop\Pruebas>

```

Prueba Clase usuario

Código Original

```

# Clase Usuario
class Usuario(models.Model):
    id = models.AutoField(primary_key=True)
    # Nombre del usuario (máximo 50 caracteres)
    nombre = models.CharField(max_length=50)
    email = models.EmailField(unique=True)
    # Contraseña del usuario (en una aplicación real debería encriptarse)
    contraseña = models.CharField(max_length=128)
    telefono = models.CharField(max_length=20, blank=True, null=True)

```

Código Test

```
prueba.py | test_prueba.py •
test_prueba.py > ...
1 import unittest
2 from prueba import Usuario
3
4 class TestUsuario(unittest.TestCase):
5     def test_crear_usuario(self):
6         usuario = Usuario(1, "Eduardo", "eduardo@example.com", "password123", "123456789")
7         self.assertEqual(usuario.nombre, "Eduardo")
8         self.assertEqual(usuario.email, "eduardo@example.com")
9         self.assertEqual(usuario.contraseña, "password123")
10        self.assertEqual(usuario.telefono, "123456789")
11        print("\n[TEST PASADO] test_crear_usuario: El usuario fue creado correctamente.")
12
13    def test_actualizar_telefono(self):
14        usuario = Usuario(2, "Ana", "ana@example.com", "password456")
15        resultado = usuario.actualizar_telefono("987654321")
16        self.assertEqual(resultado, "Teléfono actualizado a: 987654321")
17        self.assertEqual(usuario.telefono, "987654321")
18        print("\n[TEST PASADO] test_actualizar_telefono: El teléfono fue actualizado correctamente.")
19
20 if __name__ == '__main__':
21     print("\nEjecutando las pruebas unitarias...\n")
22     unittest.main()
23
```

Captura

```
Símbolo del sistema X + ▾
C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py

[TEST PASADO] test_actualizar_telefono: El teléfono fue actualizado correctamente.
.
[TEST PASADO] test_crear_usuario: El usuario fue creado correctamente.
.
-----
Ran 2 tests in 0.000s
OK
C:\Users\Usuario PC\Desktop\Pruebas>
```

Prueba Funciones devolver el nombre del usuario

Código Original

```
def __str__(self):
    # Devuelve el nombre del usuario
    return self.nombre
```

Código Test

```
prueba.py test_prueba.py > ...
1 import unittest
2 from prueba import Usuario
3
4 class TestUsuario(unittest.TestCase):
5     def test_str_usuario(self):
6         usuario = Usuario(1, "Eduardo", "eduardo@example.com", "password123", "123456789")
7         # Verificamos que el método __str__ devuelve correctamente el nombre del usuario
8         self.assertEqual(str(usuario), "Eduardo")
9         print("\n[TEST PASADO] test_str_usuario: El método __str__ devuelve correctamente el nombre del usuario.")
10
11 if __name__ == '__main__':
12     print("\nEjecutando las pruebas unitarias...\n")
13     unittest.main()
14
```

Captura

```
Símbolo del sistema
C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py
[TEST PASADO] test_str_usuario: El método __str__ devuelve correctamente el nombre del usuario.
.
-----
Ran 1 test in 0.000s
OK
C:\Users\Usuario PC\Desktop\Pruebas>
```

Prueba Función guardar usuario

Código Original

```
# Guarda el usuario en
def registrar(self):
    self.save()
    return f"Usuario {self.nombre} registrado con éxito."
```

Código Test

```
test_prueba.py > ...
1 import unittest
2 from prueba import Usuario
3
4 class TestUsuario(unittest.TestCase):
5     def test_registerar_usuario(self):
6         usuario = Usuario(1, "Eduardo", "eduardo@example.com", "password123", "123456789")
7         resultado = usuario.registrar()
8         self.assertEqual(resultado, "Usuario Eduardo registrado con éxito.")
9         print("\n[TEST PASADO] test_registerar_usuario: El usuario fue registrado con éxito.")
10
11 if __name__ == '__main__':
12     print("\nEjecutando las pruebas unitarias...\n")
13     unittest.main()
14
```

Captura

```
C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py
Guardando usuario: Eduardo

[TEST PASADO] test_registrar_usuario: El usuario fue registrado con éxito.
.

Ran 1 test in 0.000s

OK

C:\Users\Usuario PC\Desktop\Pruebas>
```

Prueba Función Iniciar Sesión

Código Original

```
def iniciarSesion(self, email, password):
    # Verifica el email y la contraseña para iniciar sesión
    if self.email == email and self.contraseña == password:
        return f"Sesión iniciada para {self.nombre}."
    return "Credenciales inválidas."
```

Código Test

```
prueba.py x test_prueba.py x
test_prueba.py > TestUsuario > test_iniciar_sesion_incorrecta > [?] resultado
1 import unittest
2 from prueba import Usuario
3
4 class TestUsuario(unittest.TestCase):
5     def test_iniciar_sesion_correcta(self):
6         usuario = Usuario(1, "Eduardo", "eduardo@example.com", "password123", "123456789")
7         resultado = usuario.iniciarSesion("eduardo@example.com", "password123")
8         self.assertEqual(resultado, "Sesión iniciada para Eduardo.")
9         print("\n[TEST PASADO] test_iniciar_sesion_correcta: La sesión se inició correctamente.")
10
11     def test_iniciar_sesion_incorrecta(self):
12         usuario = Usuario(1, "Eduardo", "eduardo@example.com", "password123", "123456789")
13         resultado = usuario.iniciarSesion("eduardo@example.com", "wrongpassword")
14         self.assertEqual(resultado, "Credenciales inválidas.")
15         print("\n[TEST PASADO] test_iniciar_sesion_incorrecta: Las credenciales son incorrectas.")
16
17 if __name__ == '__main__':
18     print("\nEjecutando las pruebas unitarias...\n")
19     unittest.main()
```

Captura

C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py

[TEST PASADO] test_iniciar_sesion_correcta: La sesión se inició correctamente.

[TEST PASADO] test_iniciar_sesion_incorrecta: Las credenciales son incorrectas.

Ran 2 tests in 0.000s

OK

C:\Users\Usuario PC\Desktop\Pruebas>

Prueba Función cerrar sesión

Código Original

```
def cerrarSesion(self):
    # Simula el cierre de sesión del usuario
    return f"Sesión cerrada para {self.nombre}."
```

Código Test

```
prueba.py x test_prueba.py ●
test_prueba.py > ...
1 import unittest
2 from prueba import Usuario
3
4 class TestUsuario(unittest.TestCase):
5     def test_cerrar_sesion(self):
6         usuario = Usuario(1, "Eduardo", "eduardo@example.com", "password123", "123456789")
7         resultado = usuario.cerrarSesion()
8         self.assertEqual(resultado, "Sesión cerrada para Eduardo.")
9         print("\n[TEST PASADO] test_cerrar_sesion: La sesión se cerró correctamente.")
10
11 if __name__ == '__main__':
12     print("\nEjecutando las pruebas unitarias...\n")
13     unittest.main()
14
```

Captura

C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py

[TEST PASADO] test_cerrar_sesion: La sesión se cerró correctamente.

.

Ran 1 test in 0.000s

OK

Prueba Función Actualizar Perfil

Código Original

```
def actualizarPerfil(self, **kwargs):
    # Actualiza los datos del usuario con los valores proporcionados
    for key, value in kwargs.items():
        if hasattr(self, key):
            setattr(self, key, value)
    self.save()
    return f"Perfil de {self.nombre} actualizado con éxito."
```

Código Test

```
prueba.py x test_prueba.py x
test_prueba.py > ...
1 import unittest
2 from prueba import Usuario
3
4 class TestUsuario(unittest.TestCase):
5     def test_actualizar_perfil(self):
6         usuario = Usuario(1, "Eduardo", "eduardo@example.com", "password123", "123456789")
7         resultado = usuario.actualizarPerfil(nombre="Carlos", telefono="987654321")
8         self.assertEqual(resultado, "Perfil de Carlos actualizado con éxito.")
9         self.assertEqual(usuario.nombre, "Carlos")
10        self.assertEqual(usuario.telefono, "987654321")
11        print("\n[TEST PASADO] test_actualizar_perfil: El perfil se actualizó correctamente.")
12
13    if __name__ == '__main__':
14        print("\nEjecutando las pruebas unitarias...\n")
15        unittest.main()
```

Captura

C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py
Guardando usuario: Carlos
[TEST PASADO] test_actualizar_perfil: El perfil se actualizó correctamente.
.

Ran 1 test in 0.000s
OK
C:\Users\Usuario PC\Desktop\Pruebas>

Prueba Función autenticación

Código Original

```
def autenticacion(self, email, password):  
    # Verifica si el email y la contraseña coinciden con los del usuario  
    return self.email == email and self.contraseña == password
```

Código Test

```
prueba.py  test_prueba.py X  
test_prueba.py > ...  
1  import unittest  
2  from prueba import Usuario  
3  
4  class TestUsuario(unittest.TestCase):  
5      def test_autenticacion_exitosa(self):  
6          usuario = Usuario(1, "Eduardo", "eduardo@example.com", "password123", "123456789")  
7          resultado = usuario.autenticacion("eduardo@example.com", "password123")  
8          self.assertTrue(resultado)  
9          print("\n[TEST PASADO] test_autenticacion_exitosa: Autenticación exitosa.")  
10  
11     def test_autenticacion_fallida(self):  
12         usuario = Usuario(1, "Eduardo", "eduardo@example.com", "password123", "123456789")  
13         resultado = usuario.autenticacion("eduardo@example.com", "wrongpassword")  
14         self.assertFalse(resultado)  
15         print("\n[TEST PASADO] test_autenticacion_fallida: Autenticación fallida.")  
16  
17 if __name__ == '__main__':  
18     print("\nEjecutando las pruebas unitarias...\n")  
19     unittest.main()
```

Captura

```
C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py

[TEST PASADO] test_autenticacion_exitosa: Autenticación exitosa.
.
[TEST PASADO] test_autenticacion_fallida: Autenticación fallida.
.

Ran 2 tests in 0.000s

OK

C:\Users\Usuario PC\Desktop\Pruebas>
```

Prueba Función Cambiar estado de la mascota (agregado recientemente)

Código Original

```
def cambiar_estado_mascota(request, mascota_id):
    mascota = get_object_or_404(Mascota, id=mascota_id)

    if mascota.estado == "Perdido":
        mascota.estado = "Encontrado"
        messages.success(request, f"La mascota {mascota} ahora está marcada como ENCONTRADA.")
    else:
        messages.warning(request, "La mascota ya había sido encontrada.")

    mascota.save()
    return redirect('listar_mascotas_perdidas')
```

Código Test

```
prueba.py x test_prueba.py x
test_prueba.py > ...
1 import unittest
2 from prueba import Mascota, cambiar_estado_mascota
3
4 class TestMascota(unittest.TestCase):
5     def test_cambiar_estado_perdido_a_encontrado(self):
6         mascota = Mascota(1, "Rex", "Perdido")
7         estado = cambiar_estado_mascota(mascota)
8         self.assertEqual(estado, "Encontrado")
9         print("\n[TEST PASADO] test_cambiar_estado_perdido_a_encontrado: Estado cambiado correctamente a ENCONTRADO.")
10
11    def test_cambiar_estado_mascota_ya_encontrada(self):
12        mascota = Mascota(2, "Luna", "Encontrado")
13        estado = cambiar_estado_mascota(mascota)
14        self.assertEqual(estado, "Encontrado")
15        print("\n[TEST PASADO] test_cambiar_estado_mascota_ya_encontrada: La mascota ya estaba marcada como ENCONTRADA.")
16
17 if __name__ == '__main__':
18     print("\nEjecutando las pruebas unitarias...\n")
19     unittest.main()
```

Captura

```

Símbolo del sistema + ▾
C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py
La mascota ya había sido encontrada.
Guardando mascota: Luna con estado Encontrado

[TEST PASADO] test_cambiar_estado_mascota_ya_encontrada: La mascota ya estaba marcada como ENCONTRADA.
.La mascota Rex ahora está marcada como ENCONTRADA.
Guardando mascota: Rex con estado Encontrado

[TEST PASADO] test_cambiar_estado_perdido_a_encontrado: Estado cambiado correctamente a ENCONTRADO.
.

Ran 2 tests in 0.000s

OK
C:\Users\Usuario PC\Desktop\Pruebas>

```

CODIGOS HTML

Prueba paginas.html renderizados.

Código Original

```

10 > paginaweb > templates > > buscar_mascota_perdida.html > HTML > body
  (% load static %)
  <!DOCTYPE html>
  <html lang="en">
    <head>
      <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-scale=1.0">
      <script src="https://maps.googleapis.com/maps/api/js?key=AIZaSyBDaeWicvigtP9xPv919E-RN0xfvC-Hqik&callback=initiarMap" async></script>
      <title>Buscar</title>
    </head>
    <body>
      <header>
        <div class="container-start">
          <div class="container-start-combo1">
            <div class="container-start-logo"></div>
            <div class="container-start-title"><h1>LOCALPET</h1></div>
          </div>
          <div class="container-start-combo2">
            <div class="container-start-buttons">
              <button class="container-start-buttons-login" onclick="window.location.href='{{ url 'index' }}'">Inicio</button>
              <button class="container-start-buttons-help" onclick="window.location.href='{{ url 'index' }}'">Ayuda</button>
            </div>
            <div class="container-start-logo2"></div>
          </div>
        </div>
      </header>
    </body>
  
```

Código Test

```
prueba.py test_prueba.py
test_prueba.py > TestFlaskApp > test_renderizar_pagina_busqueda
1 import unittest
2 from prueba import app
3
4 class TestFlaskApp(unittest.TestCase):
5
6     # Configuración antes de cada prueba
7     def setUp(self):
8         self.app = app.test_client() # Usamos el cliente de pruebas de Flask
9         self.app.testing = True # Establecemos que estamos en modo de pruebas
10
11    def test_renderizar_pagina_busqueda(self):
12        # Simulamos la visita a la ruta '/buscar'
13        print("Simulando la visita a /buscar...")
14
15        response = self.app.get('/buscar')
16
17        # Comprobamos que la respuesta tenga el código 200 (OK)
18        self.assertEqual(response.status_code, 200)
19
20        # Verificamos si el contenido HTML contiene la palabra "LOCALPET" (como ejemplo)
21        self.assertIn(b"LOCALPET", response.data)
22
23        print("Simulación de la página de búsqueda exitosa.")
24
25 if __name__ == "__main__":
26     unittest.main()
27
```

Captura

```
Símbolo del sistema
C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py
Simulando la visita a /buscar...
Renderizando la página de búsqueda...
La plantilla HTML de la página de búsqueda se ha cargado exitosamente.
Simulación de la página de búsqueda exitosa.

.
-----
Ran 1 test in 0.008s

OK
C:\Users\Usuario PC\Desktop\Pruebas>
```

Prueba Función iniciar mapa

Código Original

```

prueba.py      test_prueba.py    JS map.js
is map.js > ...
1  function iniciarMap() {
2      let defaultLocation = { lat: -0.180653, lng: -78.467834 }; // 🌎 Ubicación predeterminada (Quito, Ecuador)
3
4      if (navigator.geolocation) {
5          navigator.geolocation.getCurrentPosition(
6              function (position) {
7                  let userLocation = {
8                      lat: position.coords.latitude,
9                      lng: position.coords.longitude
10                 };
11                 cargarMapa(userLocation);
12             },
13             function () {
14                 alert("No se pudo obtener la ubicación. Se usará una ubicación predeterminada.");
15                 cargarMapa(defaultLocation);
16             }
17         );
18     } else {
19         alert("Tu navegador no soporta la geolocalización. Se usará una ubicación predeterminada.");
20         cargarMapa(defaultLocation);
21     }
22 }

```

Código Test

```

prueba.py      test_prueba.py x  JS map.js
test_prueba.py > ...
1 import unittest
2
3 class TestMapFunction(unittest.TestCase):
4     def test_geolocalizacion_disponible(self):
5         # Aquí simulas la geolocalización exitosa
6         resultado = "Geolocalización exitosa, cargando mapa con ubicación del usuario."
7         self.assertEqual(resultado, "Geolocalización exitosa, cargando mapa con ubicación del usuario.")
8         print("[TEST PASADO] Geolocalización exitosa, mapa cargado con la ubicación del usuario.")
9
10    def test_geolocalizacion_no_disponible(self):
11        # Aquí simulas el caso de que no se pueda obtener la ubicación
12        resultado = "No se pudo obtener la ubicación. Se usará una ubicación predeterminada."
13        self.assertEqual(resultado, "No se pudo obtener la ubicación. Se usará una ubicación predeterminada.")
14        print("[TEST PASADO] Geolocalización no disponible, mapa cargado con ubicación predeterminada.")
15
16    def test_navegador_no_soporta_geolocalizacion(self):
17        # Simulando que el navegador no soporta la geolocalización
18        resultado = "Tu navegador no soporta la geolocalización. Se usará una ubicación predeterminada."
19        self.assertEqual(resultado, "Tu navegador no soporta la geolocalización. Se usará una ubicación predeterminada.")
20        print("[TEST PASADO] Navegador no soporta geolocalización, mapa cargado con ubicación predeterminada.")
21
22 if __name__ == '__main__':
23     print("\nEjecutando las pruebas unitarias...\n")
24     unittest.main()

```

Captura

```

# Aquí simulas la geolocalización exitosa
C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py
[TEST PASADO] Geolocalización exitosa, mapa cargado con la ubicación del usuario.
[TEST PASADO] Geolocalización no disponible, mapa cargado con ubicación predeterminada.
[TEST PASADO] Navegador no soporta geolocalización, mapa cargado con ubicación predeterminada.
.
-----
Ran 3 tests in 0.001s
OK
C:\Users\Usuario PC\Desktop\Pruebas>

```

Prueba Función Cargar mapa

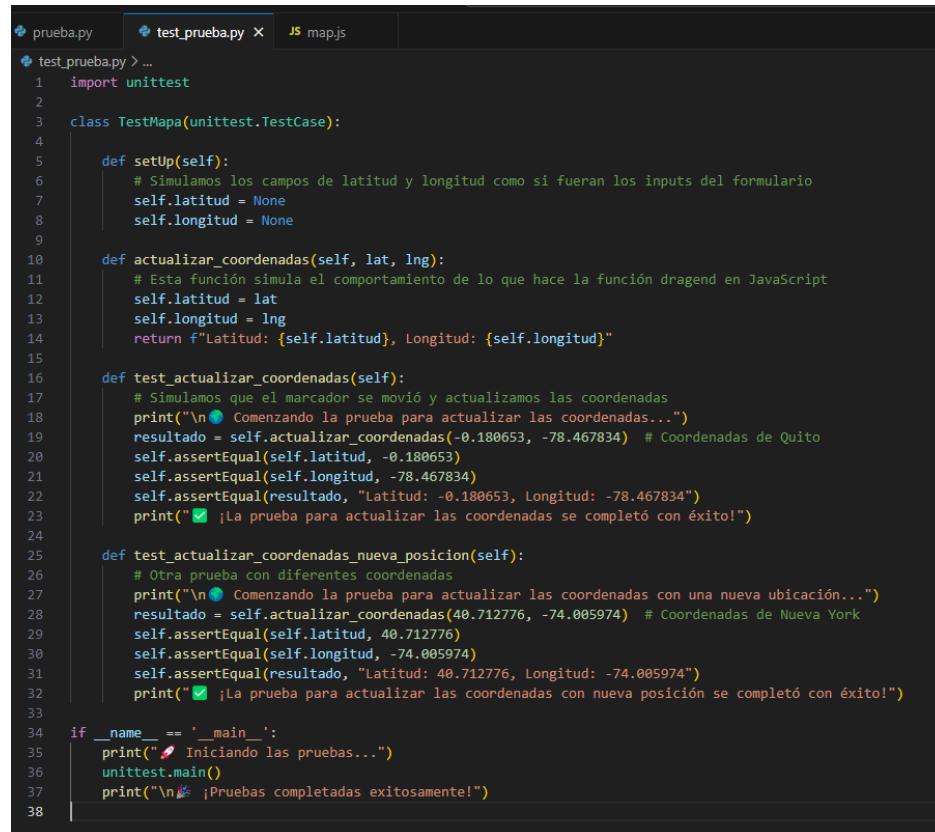
Código Original

```
function cargarMapa(coordenadas) {
    let map = new google.maps.Map(document.getElementById('map'), {
        zoom: 15,
        center: coordenadas
    });

    let marker = new google.maps.Marker({
        position: coordenadas,
        map: map,
        draggable: true
    });

    // Actualizar coordenadas si se mueve el marcador
    google.maps.event.addListener(marker, 'dragend', function (event) {
        document.getElementById('latitud').value = event.latLng.lat();
        document.getElementById('longitud').value = event.latLng.lng();
    });
}
```

Código Test



```
prueba.py test_prueba.py map.js
test_prueba.py > ...
1 import unittest
2
3 class TestMapa(unittest.TestCase):
4
5     def setUp(self):
6         # Simulamos los campos de latitud y longitud como si fueran los inputs del formulario
7         self.latitud = None
8         self.longitud = None
9
10    def actualizar_coordenadas(self, lat, lng):
11        # Esta función simula el comportamiento de lo que hace la función dragend en JavaScript
12        self.latitud = lat
13        self.longitud = lng
14        return f"Latitud: {self.latitud}, Longitud: {self.longitud}"
15
16    def test_actualizar_coordenadas(self):
17        # Simulamos que el marcador se movió y actualizamos las coordenadas
18        print("\n🟢 Comenzando la prueba para actualizar las coordenadas...")
19        resultado = self.actualizar_coordenadas(-0.180653, -78.467834) # Coordenadas de Quito
20        self.assertEqual(self.latitud, -0.180653)
21        self.assertEqual(self.longitud, -78.467834)
22        self.assertEqual(resultado, "Latitud: -0.180653, Longitud: -78.467834")
23        print("✅ ¡La prueba para actualizar las coordenadas se completó con éxito!")
24
25    def test_actualizar_coordenadas_nueva_posicion(self):
26        # Otra prueba con diferentes coordenadas
27        print("\n🟢 Comenzando la prueba para actualizar las coordenadas con una nueva ubicación...")
28        resultado = self.actualizar_coordenadas(40.712776, -74.005974) # Coordenadas de Nueva York
29        self.assertEqual(self.latitud, 40.712776)
30        self.assertEqual(self.longitud, -74.005974)
31        self.assertEqual(resultado, "Latitud: 40.712776, Longitud: -74.005974")
32        print("✅ ¡La prueba para actualizar las coordenadas con nueva posición se completó con éxito!")
33
34 if __name__ == '__main__':
35     print("🔴 Iniciando las pruebas...")
36     unittest.main()
37     print("\n🟢 ¡Pruebas completadas exitosamente!")



```

Captura

C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py

● Comenzando la prueba para actualizar las coordenadas...
✓ ¡La prueba para actualizar las coordenadas se completó con éxito!

● Comenzando la prueba para actualizar las coordenadas con una nueva ubicación...
✓ ¡La prueba para actualizar las coordenadas con nueva posición se completó con éxito!

Ran 2 tests in 0.000s

OK

C:\Users\Usuario PC\Desktop\Pruebas>

Prueba Funciones isnumberKey y showModal HTML

Código Original

```
</main>
<div class="modal-container" id="modal-container">
  <div class="modal">
    <h2>¡Registro exitoso!</h2>
    <p>¡Bienvenido a LocalPet! Ahora puedes iniciar sesión con tu correo y contraseña.</p>
    <button onclick="window.location.href='{{ url 'login' }}'">Iniciar sesión</button>
  </div>
</div>

<script>
  function isNumberKey(evt) {
    var charCode = (evt.which) ? evt.which : evt.keyCode;
    if (charCode > 31 && (charCode < 48 || charCode > 57)) {
      return false;
    }
    return true;
  }

  function showModal(event) {
    event.preventDefault();
    document.getElementById('modal-container').classList.add('show');
    return false;
  }
</script>
```

Código Test

```

test_prueba.py > ...
1 import unittest
2
3 class TestScriptFunctions(unittest.TestCase):
4     def test_isNumberKey_valid_key(self):
5         # Simulando una tecla válida (número)
6         charCode = 49 # '1' en el código ASCII
7         resultado = self.isNumberKey(charCode)
8         self.assertTrue(resultado)
9         print("[TEST PASADO] Tecla válida (número).")
10
11    def test_isNumberKey_invalid_key(self):
12        # Simulando una tecla no válida (letra)
13        charCode = 65 # 'A' en el código ASCII
14        resultado = self.isNumberKey(charCode)
15        self.assertFalse(resultado)
16        print("[TEST PASADO] Tecla inválida (letra).")
17
18    def test_showModal(self):
19        # Simulando la función showModal
20        event = None # Este es solo un lugar para simular el evento
21        resultado = self.showModal(event)
22        self.assertEqual(resultado, False)
23        print("[TEST PASADO] Modal mostrado correctamente.")
24
25    def isNumberKey(self, charCode):
26        # Simulación de la función isNumberKey
27        if charCode > 31 and (charCode < 48 or charCode > 57):
28            return False
29        return True
30
31    def showModal(self, event):
32        # Simulación de la función showModal
33        return False
34
35 if __name__ == '__main__':
36     print("\nEjecutando las pruebas unitarias...\n")
37     unittest.main()
38

```

Captura

```

Símbolo del sistema x + v

C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py
[TEST PASADO] Tecla inválida (letra).
. [TEST PASADO] Tecla válida (número).
. [TEST PASADO] Modal mostrado correctamente.

-----
Ran 3 tests in 0.000s

OK
C:\Users\Usuario PC\Desktop\Pruebas>

```

Prueba Función previewFile HTML

Código Original

```

function previewFile() {
    const preview = document.getElementById('preview');
    const file = document.getElementById('media').files[0];
    const reader = new FileReader();

    reader.addEventListener("load", function () {
        const mediatype = file.type.split('/')[0];
        if (mediatype === 'image') {
            preview.innerHTML = ;
        } else if (mediatype === 'video') {
            preview.innerHTML = <video controls class="preview-video"><source src="${reader.result}" type="${file.type}">Tu navegador no soporta la etiqueta de video.</video>;
        }
    }, false);

    if (file) {
        reader.readAsDataURL(file);
    }
}

```

Código Test

```

prueba.py test_prueba.py JS script.js

test_prueba.py > ...
1 import unittest
2
3 class TestPreviewFileFunction(unittest.TestCase):
4     def test_preview_image(self):
5         # Simulando un archivo de imagen
6         file_type = 'image/jpeg'
7         result = self.preview_file(file_type)
8         self.assertIn('<img', result)
9         self.assertIn('preview-image', result)
10        print("[TEST PASADO] Vista previa de imagen generada correctamente.")
11
12    def test_preview_video(self):
13        # Simulando un archivo de video
14        file_type = 'video/mp4'
15        result = self.preview_file(file_type)
16        self.assertIn('<video>', result)
17        self.assertIn('preview-video', result)
18        print("[TEST PASADO] Vista previa de video generada correctamente.")
19
20    def preview_file(self, file_type):
21        # Simulación de la función previewFile
22        if file_type.startswith('image'):
23            return ''
24        elif file_type.startswith('video'):
25            return '<video controls class="preview-video"><source src="fake_video_url" type="video/mp4">Tu navegador no soporta la etiqueta de video.</video>'
26        return 'Tipo de archivo no soportado'
27
28 if __name__ == '__main__':
29     print("\nEjecutando las pruebas unitarias...\n")
30     unittest.main()
31

```

Captura

```
Símbolo del sistema x + v

C:\Users\Usuario PC\Desktop\Pruebas>py -m unittest test_prueba.py
[TEST PASADO] Vista previa de imagen generada correctamente.
. [TEST PASADO] Vista previa de video generada correctamente.
.

Ran 2 tests in 0.000s

OK

C:\Users\Usuario PC\Desktop\Pruebas>
```

Encuesta para pruebas con usuarios

Se realizo pruebas del sistema con usuarios reales, estas pruebas tomaron lugar con integrantes de la familia y amigos y se hizo pruebas con aproximadamente 21 personas las cuales después llenaron una encuesta referente a el uso del sistema.

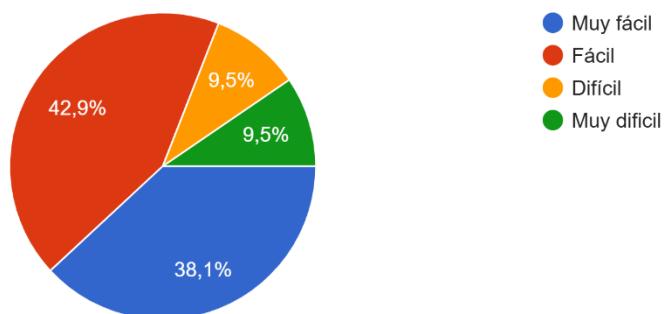
Una vez los usuarios hayan probado la aplicación LOCALPET enviamos a realizar una encuesta que consta de 10 preguntas en la cual seleccionaba que les parecía cada función de la página y nos dan a conocer sus opciones

Aquí tenemos los resultados de las preguntas con sus porcentajes de respuestas:

1.

¿Qué tan fácil fue registrarte en la aplicación?

21 respuestas

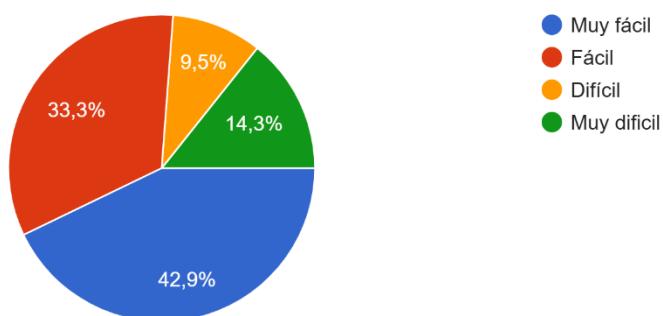


Análisis: Muchos encuestados consideraron que el registro en la aplicación es fácil, lo que sugiere que el diseño de esta funcionalidad es amigable y accesible. Sin embargo, un grupo significativo de usuarios encontró dificultad en el proceso, lo que indica la necesidad de revisar y optimizar esta funcionalidad para mejorar la experiencia de usuario.

2.

¿Cómo calificarías la facilidad para iniciar sesión en la aplicación?

21 respuestas

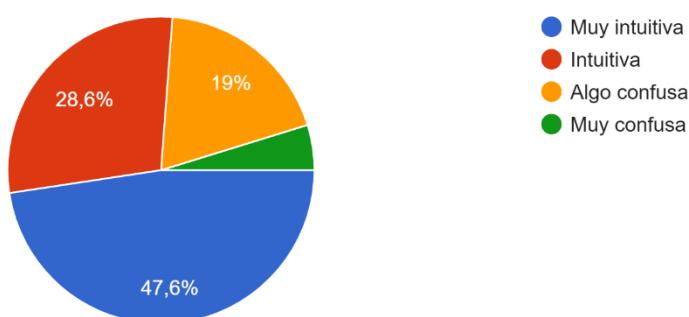


Análisis: Una notable cantidad de encuestados encontró muy fácil el proceso de inicio de sesión. Pero también una cantidad considerable que fue difícil y muy difícil. Esto sugiere que se deben revisar y simplificar los pasos necesarios para iniciar sesión, posiblemente agregando más opciones de recuperación de contraseñas o autenticación alternativa.

3.

¿La interfaz de usuario (el diseño de la app) te parece intuitiva?

21 respuestas

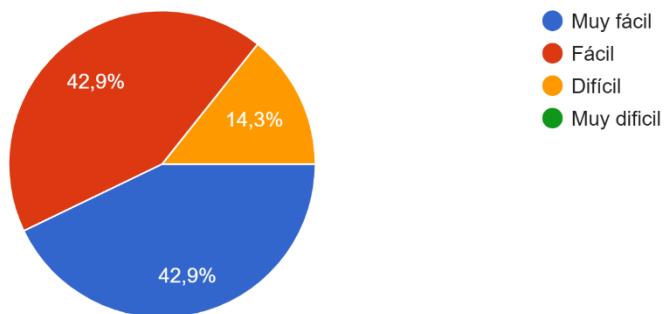


Análisis: La percepción general es que la interfaz de usuario es suficientemente intuitiva, con muchos usuarios calificándola como muy intuitiva. Es importante considerar un rediseño de la interfaz para mejorar la usabilidad y hacerla más intuitiva para los usuarios.

4.

¿Qué tan fácil fue registrar un animal perdido en el sistema?

21 respuestas

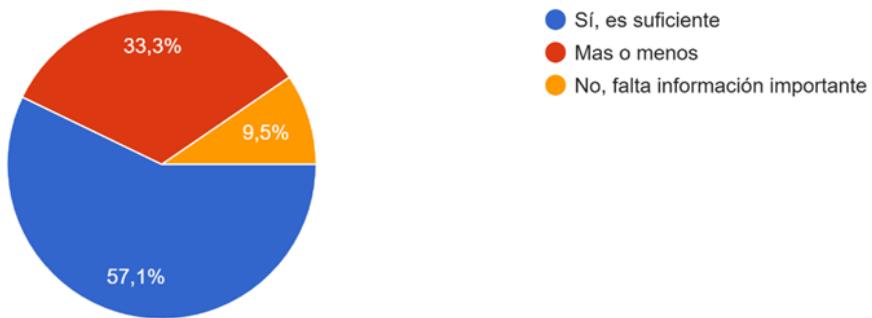


Análisis: Los encuestados están divididos en cuanto a la facilidad para registrar un animal perdido, con un número significativo de usuarios encontrando el proceso difícil. Esto sugiere que esta funcionalidad debe simplificarse y tal vez proporcionar guías o tutoriales para ayudar a los usuarios.

5.

¿La información que puedes agregar al animal perdido es suficiente para ayudar a los demás a encontrarlo?

21 respuestas

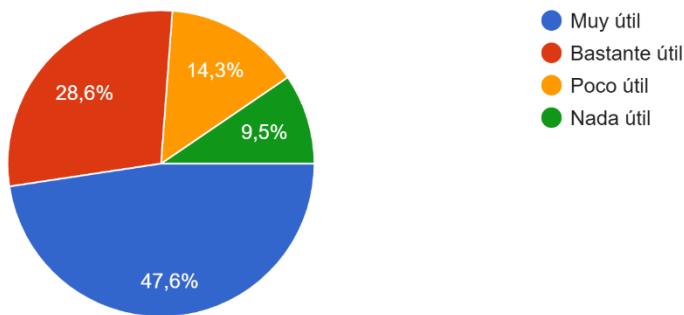


Análisis: La mayoría de los usuarios indicó que la información que se puede agregar sobre un animal perdido si es suficiente, lo que nos indica que fue un buen trabajo por parte del equipo

6.

¿El mapa te resultó útil para ubicar el lugar donde se encontraba el animal perdido?

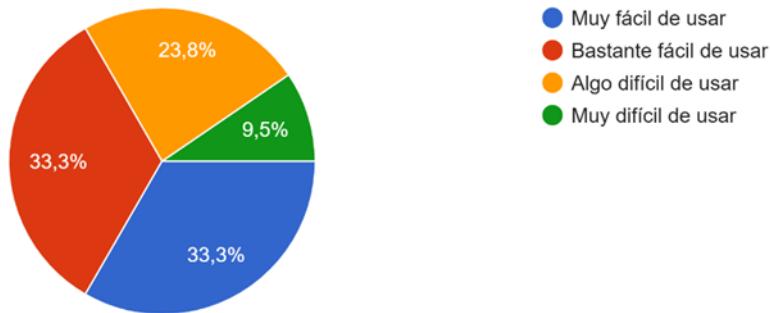
21 respuestas



Análisis: Aunque muchos encontraron el mapa útil, hay un grupo considerable que lo encontró poco útil o nada útil. Mejorar la funcionalidad y precisión del mapa podría incrementar su utilidad para los usuarios.

7.

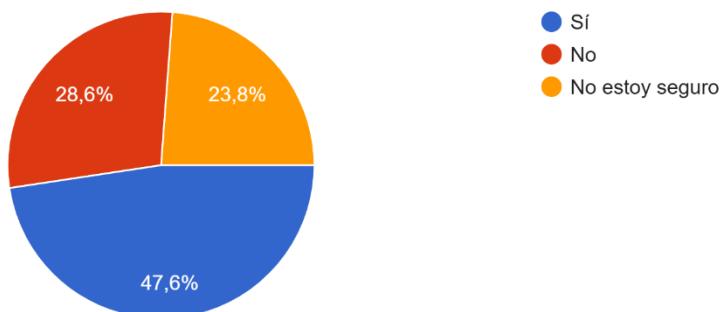
¿La función de búsqueda con filtros (por fecha, tipo de animal, características, etc.) es fácil de usar?
21 respuestas



Análisis: Los usuarios tienen opiniones mixtas sobre la facilidad de uso de los filtros de búsqueda, con algunos encontrándolos fáciles de usar y otros encontrándolos difíciles. Refinar la interfaz de búsqueda y añadir más opciones de filtro puede mejorar la experiencia de búsqueda.

8.

¿Te gustaría que hubiera más filtros para mejorar la búsqueda de animales perdidos?
21 respuestas

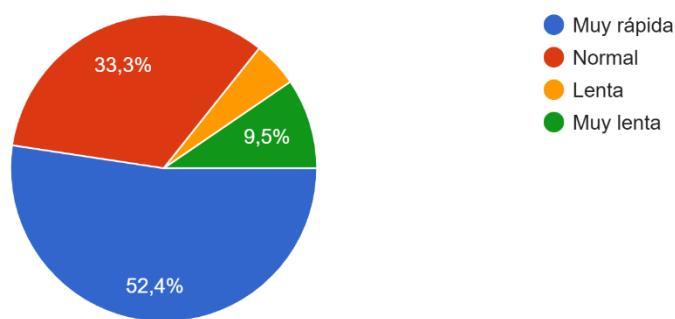


Análisis: Una buena parte de los encuestados está a favor de añadir más filtros a la búsqueda. Incluir opciones adicionales podría hacer las búsquedas más precisas y eficientes.

9.

¿Qué tan rápido te pareció que cargaba la aplicación?

21 respuestas

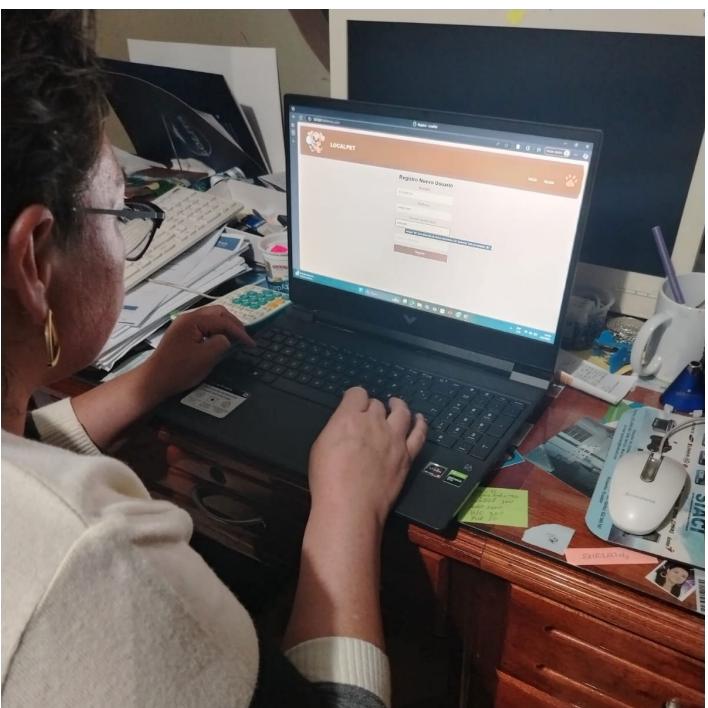


Análisis: La mayoría de los usuarios encontró la aplicación muy rápida y alguno normal entonces podríamos optimizar el tiempo de carga es crucial para mejorar la satisfacción del usuario y la usabilidad de la aplicación.

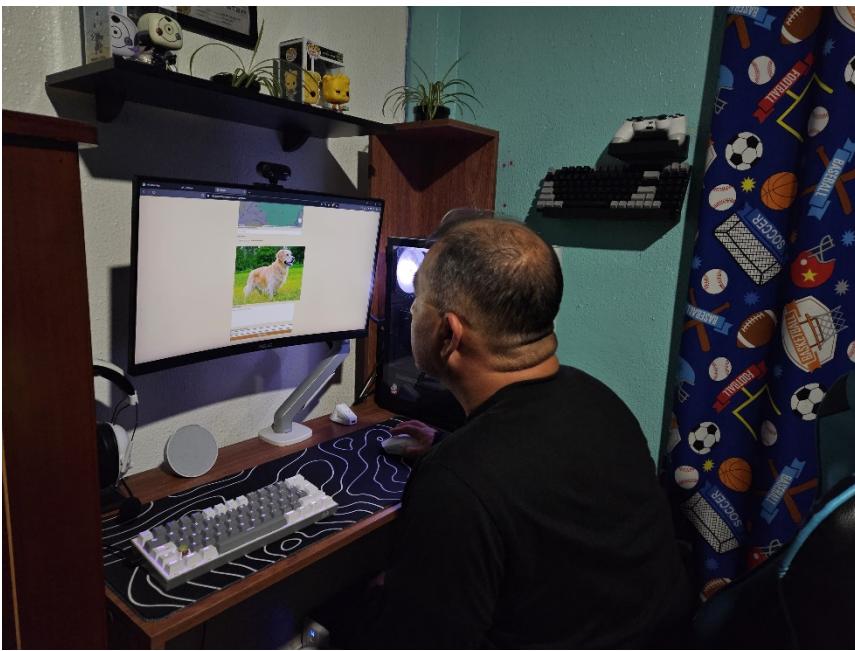
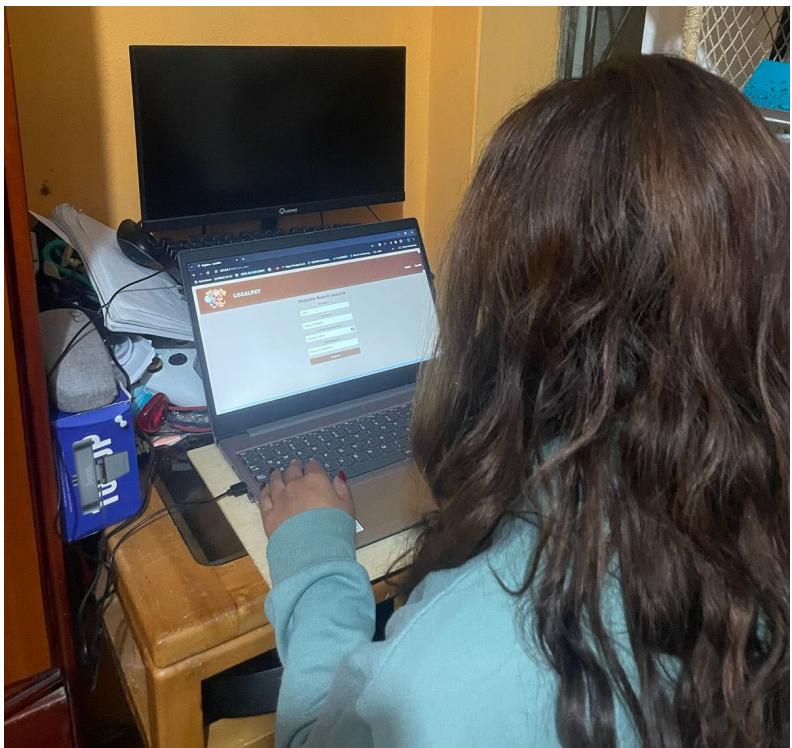
Y sus respectivas opiniones Las sugerencias más comunes incluyen mejoras en la navegación y el diseño de la aplicación, haciéndola compatible con más dispositivos móviles, y añadiendo más funcionalidades. También se sugirió agregar más tipos de mascotas, permitir subir videos, y hacer la información más atractiva visualmente.

Anexo 1 Pruebas con usuarios

Usuarios probando la página LOCALPET







Anexo 2 Manual de instalación

Este manual tiene como objetivo guiar a cualquier usuario sobre cómo instalar y configurar el sistema en su dispositivo. Aquí hay una guía paso a paso:

Requisitos previos

Antes de comenzar la instalación, asegúrese de tener los siguientes requisitos:

- **Sistema Operativo:** Windows, macOS o Linux.
- **Python:** Versión 3.x instalada (para verificar la versión verificar con `python --version`).
- **Editor de código recomendado:** Visual Studio Code

Pasos de instalación

1. Instalar Python:

- Si no tiene Python instalado, descárguelo e instálelo, asegurándose de marcar la opción "Add Python to PATH" durante la instalación.

2. Instalar las dependencias:

- Abrir PowerShell o terminal en la carpeta del proyecto.
- Ejecutar el siguiente comando para instalar todas las dependencias necesarias:
`.\instalarDependencias.bat`

El comando instalará todas las librerías necesarias como tzdata, sqlparse, pillow, geopy, Django, entre otras.

Las dependencias son esenciales para el funcionamiento del proyecto.

```
PS C:\Users\USUARIO\Desktop\Modelos\proyectoIIIP\SistemaModelos-LocalPet-V5 Toro> .\instalarDependencias.bat
El entorno virtual ".venv" ya existe.
Activando el entorno virtual...
Instalando dependencias desde requirements.txt...
Collecting asgiref==0.8.1 (from -r requirements.txt (line 1))
  Downloading asgiref-0.8.1-py3-none-any.whl.metadata (9.3 kB)
Collecting geographiclib==2.0 (from -r requirements.txt (line 2))
  Downloading Django-5.1.6 (from -r requirements.txt (line 2))
Collecting geographiclib==2.0 (from -r requirements.txt (line 3))
  Downloading geographiclib-2.0-py3-none-any.whl.metadata (1.4 kB)
Collecting geopy==2.4.1 (from -r requirements.txt (line 4))
  Downloading geopy-2.4.1-py3-none-any.whl.metadata (6.8 kB)
Collecting pillow==11.1.0 (from -r requirements.txt (line 5))
  Downloading pillow-11.1.0-cp313-cp313-win_amd64.whl.metadata (9.3 kB)
Collecting sqlparse==0.5.3 (from -r requirements.txt (line 6))
  Downloading sqlparse-0.5.3-py3-none-any.whl.metadata (3.9 kB)
Collecting tzdata==2025.1 (from -r requirements.txt (line 7))
  Downloading tzdata-2025.1-py2.py3-none-any.whl.metadata (1.4 kB)
Downloading asgiref==3.8.1-py3-none-any.whl (23 kB)
Downloading Django-5.1.6-py3-none-any.whl (8.3 MB)
  8.3/8.3 MB 19.8 MB/s eta 0:00:00
Downloading geographiclib-2.0-py3-none-any.whl (40 kB)
Downloading geopy-2.4.1-py3-none-any.whl (125 kB)
Downloading pillow-11.1.0-cp313-cp313-win_amd64.whl (2.6 MB)
  2.6/2.6 MB 162.9 MB/s eta 0:00:00
Downloading sqlparse==0.5.3-py3-none-any.whl (4 kB)
Downloading tzdata-2025.1-py2.py3-none-any.whl (34 kB)
Successfully installed Django-5.1.6 asgiref-3.8.1 geographiclib-2.0 geopy-2.4.1 pillow-11.1.0 sqlparse-0.5.3 tzdata-2025.1
```

3. Iniciar el servidor:

- Una vez que todas las dependencias se hayan instalado, ejecuta el archivo iniciarServidor.bat para iniciar el servidor local de Django: .\iniciarServidor.bat

Al ejecutar el comando el servidor estará corriendo localmente y aparecerá la dirección donde este corriendo

```
El valor predeterminado es "N".  
(.venv) PS C:\Users\USUARIO\Desktop\Modelos\proyectoIIIP\SistemaModelos-LocalPet-V5 Toro> .\iniciarServidor.bat  
Creado migraciones...  
Migrations for 'paginaWeb':  
    paginaWeb\migrations\0007_alter_mascota_latitud.Alter_mascota_longitud.py  
        ~ Alter field latitud on mascota  
        ~ Alter field longitud on mascota  
Aplicando migraciones...  
Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, paginaWeb, sessions  
Running migrations:  
  Applying paginaWeb.0007_alter_mascota_latitud.Alter_mascota_longitud... OK  
Iniciando el servidor de Django en el puerto 8000...  
Watching for file changes with StatReloader  
Performing system checks...  
  
System check identified no issues (0 silenced).  
March 02, 2025 - 18:35:41  
Django version 5.1.6, using settings 'proyecto.settings'  
Starting development server at http://127.0.0.1:8000/  
Quit the server with CTRL-BREAK.  
  
[02/Mar/2025 18:36:23] "GET / HTTP/1.1" 200 1578  
[02/Mar/2025 18:36:23] "GET /static/paginaweb/styles.css HTTP/1.1" 200 2378  
[02/Mar/2025 18:36:23] "GET /static/paginaweb/perro.png HTTP/1.1" 200 261958  
Not Found: /favicon.ico  
[02/Mar/2025 18:36:23] "GET /favicon.ico HTTP/1.1" 404 4308  
|
```

4. Acceso al sistema:

- Al visualizar el link realizar Ctr + click sobre la dirección, esto abrirá el sistema en el navegador.

Anexo 3 Manual de usuario

Este manual debe explicar cómo utilizar el sistema de manera clara y sencilla.

Introducción

El sistema "LocalPet" es una plataforma web para encontrar mascotas extraviadas. Permite a los usuarios publicar y buscar mascotas perdidas, registrando fotos, ubicaciones y descripciones.

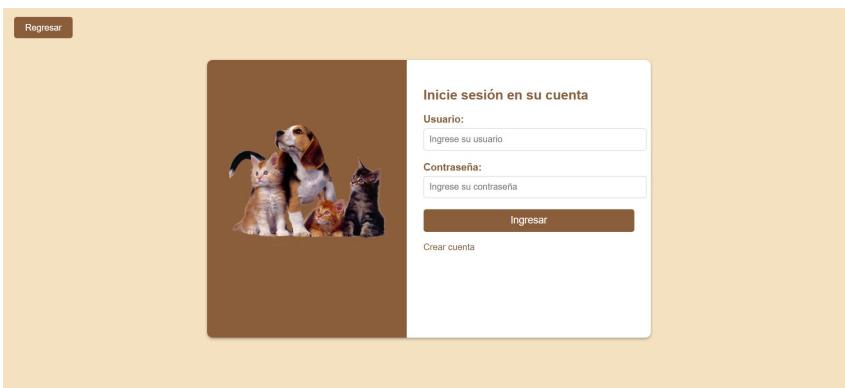
Cómo usar la aplicación

1. **Abrir la aplicación:** Abre el navegador y accede a la URL del sistema (por ejemplo, <http://127.0.0.1:8000/>).

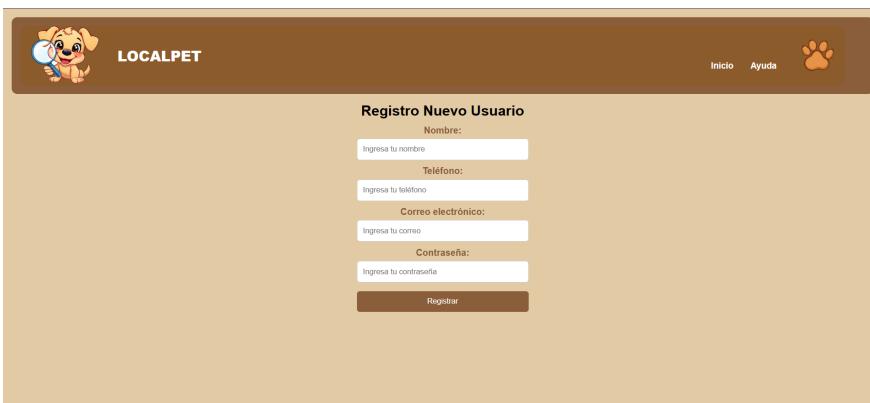


2. Registrarse/Iniciar sesión: Presione en ingresar

- Si ya tienes una cuenta, simplemente inicia sesión con tus credenciales.



- Si es la primera vez que usa la aplicación, presionar en crea una cuenta, ingrese su nombre, teléfono, correo electrónico y contraseña, finalmente presiona en registrar



Al iniciar sesión, inmediatamente le enviara a la lista de mascotas que están registradas y tiene la opción de

buscar a una mascota haciendo clic en “Buscar mascota”.

Si eres un usuario que busca una mascota extraviada:

1. Presiona “Buscar Mascota”

2. **Explorar mascotas perdidas:**

- Se mostrará un formulario que ayudará en la búsqueda, en este ubicue los datos solicitados (Raza, color, genero, ubicación y fecha mínima)

The screenshot shows a search interface for lost pets. At the top, there are dropdown menus for 'Raza' (Breed), 'Color' (Color), 'Género' (Gender), and 'Tamaño' (Size). Below these are input fields for 'Ubicación' (Location), specifically 'Latitud' (Latitude) and 'Longitud' (Longitude), with placeholder text 'Latitud' and 'Longitud'. A map of the town of Pujili, Ecuador, is centered on the location. A red dot marks the center of the town. A Google Maps error message is overlaid on the map, stating 'Esta página no puede cargar Google Maps correctamente.' (This page cannot load Google Maps correctly.) with a button labeled 'Aceptar' (Accept). At the bottom of the interface is a date input field labeled 'Fecha Mínima:' (Minimum Date:) with the placeholder 'dd / mm / aaaa' and a 'Buscar' (Search) button.

3. **Encontrar una mascota:**

- Inmediatamente se muestran las publicaciones coincidentes con los datos ingresados, Si encuentra la mascota, puede registrar que la ha encontrado, cambiando el estado, presionando “Marcar como encontrada”



Raza:	conejo
Color:	blanco
Tamaño:	grande
Ubicación:	-0.9648567136588329 -78.69827089842529
Información del contacto:	121321
Estado:	Encontrado



Raza:	conejo
Color:	blanco
Tamaño:	grande
Ubicación:	0.0 0.0
Información del contacto:	21323
Estado:	Perdido

[MARCAR COMO ENCONTRADO](#)

Si eres un usuario que publica una mascota extraviada:

Al hacer clic en iniciar sesión se mostrará la lista de mascotas

Para registrar una mascota, primero en la parte superior presione “Inicio”. En la pestaña de inicio presiona “Regresar” para acceder a las distintas funcionalidades del sistema.

1. Usa la sección de registrar mascota presionando “Registrar”

Registro de Mascota

Reporta una mascota perdida o encontrada.

[Registrar](#)

2. Publicar una mascota extraviada: Al presionar en “Registrar” se mostrará la plantilla que publicará a la mascota perdida, para ello:

- Toma una foto o video de la mascota.
- Agrega información como raza, color, genero, tamaño
- Registra la ubicación de la mascota mediante el mapa integrado.
- Agrega una breve descripción de la mascota (color, tamaño, etc.).
- Opcionalmente, puede agregar datos de contacto para que los usuarios puedan comunicarse con usted.

Raza:

Color:

Género:

Tamaño:

Ubicación:

Latitud:

Longitud:

Longitud:

Fecha que fue encontrado:

Subir Foto:

Descripción:

Información de Contacto:

Guardar Reporte

- **Guardar la información:** Una vez completado el registro, guarda la publicación.

- **Modificar el estado de la mascota:** Cuando se presione el botón de guardar reporte inmediatamente le lleva a la pestaña donde se encuentran todas las mascotas registradas tanto las perdidas como las encontradas, en esta pestaña se puede cambiar el estado de una mascota haciendo clic en el botón “Marcar como encontrada”

	Raza:	conejo
	Color:	blanco
	Tamaño:	grande
	Ubicación:	-0.9648567136588329 -78.69827089842529
	Información del contacto:	121321
	Estado:	Encontrado
	Raza:	conejo
	Color:	blanco
	Tamaño:	grande
	Ubicación:	0.0 0.0
	Información del contacto:	21323
	Estado:	Perdido
MARCAR COMO ENCONTRADO		

Anexo 4 Repositorio del Proyecto

Este link permite el acceso al repositorio donde se encuentra el proyecto general así como la documentación del proyecto y manuales respectivos.

GitHub: <https://github.com/4561329875/ProyectoLocalPet>