



UNIVERSIDAD NACIONAL DEL CALLAO
Facultad de Ingeniería Industrial y de Sistemas
Escuela Profesional de Ingeniería de Sistemas

PATITAS SEGURAS: PLATAFORMA DE ALERTA, RANKING Y ADOPCIÓN

Diseño modular con base de datos SQLite y evidencias para Lima
y Callao

Estudiante 1 (código 1234)

Estudiante 2 (código 1235)

Estudiante 3 (código 1236)

Estudiante 4 (código 1237)

Docente: Reinoso Palacios Artemio Rubén
Curso: Programación Orientada a Objetos
Modalidad: Proyecto aplicado

Informe del proyecto de curso

Ciclo académico 2024-II

Callao, Perú

Agradecimientos

Agradecemos al docente Reinoso Palacios Artemio Rubén por la guía metodológica brindada en el curso de Programación Orientada a Objetos. Reconocemos el trabajo conjunto del equipo (Estudiante 1, Estudiante 2, Estudiante 3 y Estudiante 4) y el apoyo de las familias y amistades que impulsaron la culminación de este prototipo orientado a mejorar el bienestar animal en Callao y Lima.

Resumen

Este informe documenta el desarrollo de *Patitas Seguras*, una aplicación de escritorio creada en el curso de Programación Orientada a Objetos para centralizar reportes de maltrato, listas negras de veterinarias, adopciones y un ranking comunitario de servicios veterinarios en Lima y Callao. El trabajo se estructuró en módulos reutilizables y se ofrece una versión con base de datos SQLite, pensada para ejecutarse sin depender de servidores corporativos. Se describen la motivación social, el estado del arte sobre bienestar animal en la región, la arquitectura de la solución, los hallazgos preliminares y las líneas futuras.

Palabras clave: bienestar animal, Lima, Callao, SQLite, Patitas Seguras, lista negra de veterinarias

Abstract

This report presents *Patitas Seguras*, a desktop application built in the Object-Oriented Programming course to consolidate animal abuse alerts, blacklists for veterinary clinics, adoption notices, and a community-driven ranking of veterinary services in Lima and Callao (Peru). We modularized the solution and added a SQLite-based deployment to remove the dependency on corporate SQL servers. The document details the social motivation, regional background on animal welfare, system architecture, preliminary findings, and future work.

Keywords: animal welfare, Lima, Callao, SQLite, veterinary ranking, civic tech

Índice general

1. Introducción	1
1.1. Problema y propósito	1
1.2. Objetivos generales y alcance	1
1.3. Estructura del documento	2
2. Estado del arte y contexto regional	3
2.1. Normativa y planes nacionales	3
2.2. Situación en Lima y Callao	3
2.3. Referentes tecnológicos	3
3. Materiales y métodos	5
3.1. Recursos de software	5
3.2. Arquitectura modular	5
3.3. Modelo de datos y persistencia	5
3.4. Experiencia de usuario y navegación	6
3.5. Módulo de adopción	6
3.6. Módulo de mascotas perdidas o encontradas	6
3.7. Módulo de listas negras	6
3.8. Módulo de ranking de veterinarias	7
3.9. Seguridad, integridad y calidad	7
3.10. Plan de evolución y pruebas	7
4. Resultados	9
4.1. Prototipo original	9
4.2. Script alternativo con SQLite	9
4.3. Hallazgos y validación	9
5. Conclusiones y trabajos futuros	11
Bibliografía y referencias	12

A. Ejemplos en \LaTeX	13
A.1. Referencias, citas y bibliografía	13
A.1.1. Referencias	13
A.1.2. Bibliografía	13
A.2. Figuras	14
A.3. Tablas	14
A.4. Código	15
A.5. Listas	16
A.6. Ecuaciones	16

Índice de figuras

A.1. Ejemplo de figura	14
----------------------------------	----

Índice de tablas

A.1. Ejemplo de tabla	15
---------------------------------	----

Capítulo 1

Introducción

1.1. Problema y propósito

En Lima Metropolitana y el Callao los reportes de adopción, alertas de mascotas perdidas y denuncias de maltrato suelen fragmentarse en canales cerrados y publicaciones efímeras. Esa dispersión genera un “laberinto” de desinformación, repite esfuerzos y expone datos personales. La falta de un punto central para consolidar, filtrar y geolocalizar eventos incrementa el riesgo de estafas y retrasa la respuesta comunitaria.

“Patitas Seguras” se concibió como una aplicación de escritorio que reduce esa fricción: concentra reportes, permite ubicarlos en mapa y mantiene un historial verificable. El proyecto busca transformar procesos que hoy demoran días en acciones de minutos mediante un flujo *end-to-end* que une registro, consulta y localización geográfica.

1.2. Objetivos generales y alcance

El sistema se plantea como una herramienta integral de gestión y cuidado animal, con módulos especializados para adopción, reportes de mascotas perdidas o encontradas, listas negras de maltrato y ranking comunitario de veterinarias. Opera sobre una pila tecnológica ligera (Python, Tkinter, Pillow, TkinterMapView y SQLite) para priorizar viabilidad en entornos sin conectividad permanente.

El alcance funcional se agrupa en tres líneas: (a) gestión de información con operaciones CRUD, (b) geolocalización y representación cartográfica de eventos, y (c) mecanismos comunitarios de prevención mediante reputación y alertas. El objetivo operativo es ofrecer catálogos filtrables, mapas interactivos y persistencia local con criterios claros de ordenamiento y trazabilidad.

1.3. Estructura del documento

El documento se organiza en cinco capítulos: el Cap.1 expone el contexto y alcance; el Cap.2 sintetiza la literatura y normativa relevante; el Cap.3 detalla arquitectura, modelo de datos, módulos y criterios de calidad; el Cap.4 resume implementaciones y hallazgos; finalmente, el Cap.5 plantea conclusiones y una hoja de ruta futura.

Capítulo 2

Estado del arte y contexto regional

2.1. Normativa y planes nacionales

La Ley N° 30407 establece el marco peruano de protección y bienestar animal, obligando a denunciar el maltrato y promoviendo la tenencia responsable ley30407. Complementariamente, el Servicio Nacional Forestal y de Fauna Silvestre (SERFOR) publicó en 2023 un diagnóstico que subraya la falta de registros unificados de mascotas y la necesidad de articular a gobiernos locales serfor2023diagnostico. Estas referencias justifican el componente de listas negras y la georreferenciación de reportes.

2.2. Situación en Lima y Callao

Estudios locales indican un aumento de denuncias por maltrato y sobrepoblación canina en distritos populosos de Lima y el Callao, lo que incrementa el riesgo de rabia y otras zoonosis ops2022rabia. La Municipalidad Metropolitana de Lima emitió ordenanzas que promueven la identificación de mascotas, pero la adopción de herramientas digitales sigue siendo limitada ordenanzaLima. En este escenario, una aplicación de escritorio con base SQLite facilita pilotos en laboratorios académicos y cabinas municipales sin conectividad permanente.

2.3. Referentes tecnológicos

Existen iniciativas de mapeo ciudadano basadas en *open data* y sistemas ligeros (por ejemplo, dashboards de dengue y COVID-19) que inspiran el uso de `tkintermapview` y esquemas relacionales compactos whoZero2023. La propuesta adopta patrones de arquitectura MVC y refuerza la trazabilidad mediante catálogos de usuarios y auditoría de votos.

Capítulo 3

Materiales y métodos

3.1. Recursos de software

El sistema se construyó con Python 3.11 y bibliotecas estables de escritorio: `tkinter` para la interfaz, `Pillow` para tratamiento de imágenes y `tkintermapview` para mapas interactivos. La persistencia recae en `sqlite3` con un archivo local `data.db` que elimina dependencias de servidor y permite replicar el prototipo en cabinas o laboratorios sin conectividad. Este enfoque prioriza viabilidad y portabilidad, manteniendo una experiencia fluida mediante estructuras en memoria.

3.2. Arquitectura modular

El diseño sigue un esquema monolítico modular con tres capas diferenciadas. La capa de presentación define la ventana principal, el dashboard y las pantallas de cada módulo (adopción, perdidos/encontrados, blacklist y ranking), reutilizando componentes como tarjetas, formularios y mapas. La lógica de negocio concentra reglas de filtrado, validación básica y cálculo de promedios de estrellas, mientras que la capa de datos inicializa tablas, carga catálogos a memoria y gestiona inserciones o eliminaciones controladas. El flujo global arranca con la preparación del entorno y carga de datos, pasa por navegación de módulos en el dashboard y concluye con commits y cierre seguro de la conexión.

3.3. Modelo de datos y persistencia

SQLite se eligió como motor embebido para lograr persistencia real sin despliegues adicionales. El esquema considera tablas para adopción, veterinarias, listas negras (personas y establecimientos), mascotas perdidas y votos. Al iniciar la aplicación se ejecutan consultas `SELECT *` que poblan listas en memoria (por ejemplo, `db_adopcion`, `db_perdidos` o `db_veterinarias`) para renderizar tarjetas con latencia mínima. Las operaciones de escri-

tura validan campos obligatorios, usan sentencias parametrizadas y actualizan tanto la base como las estructuras en memoria antes de confirmar con `commit`. Se contempla eliminación controlada (p. ej., `DELETE FROM mascotas_perdidas WHERE id = ?`) y se destaca la necesidad futura de llaves foráneas, *checksums* de imágenes y auditoría de acciones.

3.4. Experiencia de usuario y navegación

La UI se trata como componente de ingeniería: la identidad visual y los componentes reutilizables reducen errores y aceleran la decisión. La pantalla inicial presenta un dashboard con accesos directos a cada capacidad, y la navegación superior limpia y reemplaza paneles para mantener una sola ventana principal. Las tarjetas muestran fotografía y datos clave; los formularios incluyen controles estandarizados y los mapas permiten marcar coordenadas con un clic. La retroalimentación visual confirma envío de solicitudes o registros para evitar duplicados.

3.5. Módulo de adopción

El catálogo de adopción convierte publicaciones dispersas en una vista filtrable por especie y etapa de vida. Cada tarjeta prioriza imagen, nombre y etiquetas de salud, mientras que la vista detalle habilita la acción “Quiero adoptar” mediante un formulario básico de contacto. El filtrado se realiza en memoria con complejidad lineal, lo que permite refrescos inmediatos y facilita la extensión hacia nuevos atributos como distrito o tamaño.

3.6. Módulo de mascotas perdidas o encontradas

Este módulo evita la “ceguera geográfica” al registrar puntos exactos en el mapa a través de `tkintermapview`. Cada reporte incluye estado (perdido/encontrado), descripción, datos de contacto, foto y coordenadas, y se muestra como tarjeta con acceso a detalle y mapa. Se admite eliminación controlada por identificador durante la fase de prototipo, con la recomendación de evolucionar hacia estados (p. ej., resuelto) para preservar trazabilidad.

3.7. Módulo de listas negras

La blacklist funciona como mecanismo de prevención comunitaria. Para personas denunciadas se registran identificadores, descripción, evidencia y ubicación; para veterinarias denunciadas se almacenan nombre, motivo y coordenadas. Ambos flujos generan tarjetas y mapas que permiten identificar puntos críticos. Se señala el riesgo ético y legal de este tipo

de registros y se proponen controles como estados de validación, moderación y políticas de uso claras.

3.8. Módulo de ranking de veterinarias

El ranking establece un sistema de reputación basado en estrellas y comentarios. Las tablas `veterinarias` y `votos` se vinculan para recalcular promedios cada vez que se inserta una reseña. La interfaz ordena las tarjetas por calificación promedio y muestra detalle con ubicación en mapa. Se reconocen riesgos de manipulación y se sugiere controlar duplicidad de votos, registrar identidad y moderar lenguaje en versiones futuras.

3.9. Seguridad, integridad y calidad

Los formularios validan campos obligatorios, rangos de coordenadas y formatos de teléfono, mientras que las operaciones en base usan sentencias parametrizadas y clausuras seguras para evitar corrupción del archivo. Se recomienda incluir manejo explícito de errores (imágenes, apertura de base, escritura) con mensajes claros y bitácoras. Para mitigar riesgos de difamación o exposición de datos sensibles, el sistema debe mostrar advertencias visibles y adoptar moderación antes de publicar reportes.

3.10. Plan de evolución y pruebas

El roadmap contempla cuatro fases: (1) refactorizar en paquetes y reforzar validaciones y logging; (2) incorporar roles, estados de reporte y auditoría para confiabilidad comunitaria; (3) habilitar sincronización y API REST cuando sea necesario escalar; y (4) sumar analíticas y visualizaciones avanzadas. Las pruebas sugeridas incluyen suites unitarias para la capa de datos y reglas de dominio, pruebas de integración sobre flujos de navegación y checklist visual por pantalla, además de empaquetado con PyInstaller para despliegue en escritorio.

Capítulo 4

Resultados

4.1. Prototipo original

El archivo `main.py` conserva el diseño de interfaz con paleta de colores consistentes, menú principal y módulos de login, listas negras y ranking. Las consultas dependen de SQL Server y actualmente requieren ajustes de cadena de conexión para funcionar fuera del entorno académico.

4.2. Script alternativo con SQLite

Se implementó `Code/patitas_sqlite.py` como clon funcional que crea y puebla automáticamente la base `patitas.db`. El script mantiene los flujos de inicio de sesión, registro de usuarios, altas en listas negras y calificación de veterinarias. La función `seed_data()` agrega dos entradas de prueba (Callao y Lima) para validar el mapa y los promedios.

4.3. Hallazgos y validación

La modularidad permitió reutilizar componentes de interfaz y aislar la lógica de base de datos. El cambio a SQLite reduce fricción de despliegue y facilita demostraciones sin credenciales corporativas. Pendientes futuros incluyen incorporar formularios completos de adopción y pérdida, así como automatizar la carga de evidencias (fotos y documentos) por distrito.

Capítulo 5

Conclusiones y trabajos futuros

El proyecto *Patitas Seguras* integra denuncia ciudadana, georreferenciación y reputación colaborativa para mejorar el bienestar animal en Lima y Callao. La versión en SQLite demuestra que es posible ejecutar el prototipo en entornos aislados manteniendo los flujos previstos en el desarrollo original.

Como trabajo futuro se plantea: (1) completar los formularios de adopción y mascotas perdidas, (2) incorporar validación de archivos multimedia y resguardo en la base, (3) sumar analíticas de calor por distrito y (4) exponer servicios web REST para integración con aplicaciones móviles.

Apéndice A

Ejemplos en L^AT_EX

Este primer apéndice presenta ejemplos en L^AT_EX de cómo incluir referencias, citas bibliográficas, figuras, tablas o código. Este apéndice se deberá eliminar de la memoria antes de entregar el trabajo.

A.1. Referencias, citas y bibliografía

A.1.1. Referencias

Para poder referenciar un elemento dentro de la memoria hay que marcarlo con una etiqueta (`\label{[id]}`) que lo identifique inequívocamente. Es habitual utilizar identificadores representativos, por ejemplo, para marcar la introducción podemos utilizar una etiqueta como `\label{chap:introduccion}`

Una vez marcado el elemento (capítulo, sección, figura, tabla, ...) en L^AT_EX, utilizaremos el comando `ref` indicando a qué etiqueta queremos referenciar (`\ref{[id]}`). Por ejemplo, de esta manera podemos referenciar al capítulo de la introducción con [1](#). Podemos acompañarlo de un texto como “...como se vió en el capítulo [1](#)...” o bien utilizar el comando `\autoref{[i]}`, que incluiría el tipo del elemento y aparecería en el texto como [Capítulo 1](#), o incluso referenciarlo por el nombre del elemento con `\nameref{[id]}`, lo que se mostraría como [Introducción](#).

A.1.2. Bibliografía

Para añadir una cita bibliográfica al documento tendremos que asegurarnos que en el fichero de bibliografía (bibliografía.bib) se encuentre la entrada bibliográfica correspondiente. Una vez que tengamos nuestra entrada bibliográfica utilizaremos el comando de cita de L^AT_EX indicando el identificador de la referencia bibliográfica, por ejemplo: `\cite{aikg}`, que se se visualizaría como **aikg**.

Cuando queramos citar más de un trabajo en un mismo punto del documento se deberán

añadir a la cita separados por comas, por ejemplo: `\cite{chen_2018, Qin_2020}`, que se visualizaría como **chen_2018; Qin_2020**. En estos casos es habitual añadir las citas en orden cronológico de más antigua a más reciente.

Tal y como está configurada esta plantilla, por defecto el comando `\cite` realizará una cita textual (`\citet` si se activa el paquete `natbib` en lugar de `biblatex`), es decir, la cita forma parte del texto. Este tipo de citas se suelen realizar en frases como:

“Algunos ejemplos de grafos de conocimiento se presentan en **ji2020survey**”

Cuando la cita no forma parte del texto si no que se utiliza para reforzar una afirmación realizada en una frase, se debe utilizar una cita entre paréntesis (o corchetes). Para ello se usa el comando `\parencite` en `biblatex` o `\citep` en `natbib`, por ejemplo:

“Existe una gran variedad de aplicaciones de los grafos de conocimiento (**ji2020survey**).”

A.2. Figuras

Para añadir una figura al documento utilizaremos el entorno `figure`, indicando la posición que debe tener dicha figura en el documento (h: here, t: top, b: botom; p: page), se recomienda en lo posible evitar de “!” que ignora todos los ajustes de los parámetros. El orden en el que se indiquen cada una de las opciones se tendrá en cuenta para colocar la figura, es decir si se indicase el orden `[htbp]` la figura primero se intentará colocar en el lugar que ocupa en el documento, si no se puede se intentará colocar al inicio (top) de la página, en caso que tampoco sea posible se intentará colocar al final (bottom) de la página y, por último en caso que no sea posible ninguna de las anteriores se colocará al inicio de una página nueva.



Figura A.1: Esta figura tiene una descripción al pie muy larga, por lo que añadiremos un título breve utilizando para ello los corchetes tras el comando `\caption`. La etiqueta de la figura (label) se incluirá al inicio del flotante de la figura para que cualquier referencia cruzada (ref) a la misma lleve al inicio del flotante.

A.3. Tablas

Para añadir una tabla se utilizará el entorno `table`, indicando al inicio de la tabla el título de la misma utilizando el `caption`. Un ejemplo puede verse en la Tabla [A.1](#).

Tabla A.1: Esta tabla presenta un ejemplo con tres columnas y formato formal.

Model	Accuracy	Precision	Recall	F1-score	AUC
Modelo 1	0,33	0,75	0,72	0,42	0,21
Modelo 2	0,01	0,63	0,60	0,50	0,10
Modelo 3	0,03	0,93	0,33	0,04	0,42

A.4. Código

Para añadir código en la memoria utilizaremos el paquete `listing`, que permite mostrar código formateado en diversos lenguajes (Java, Python, C, ...).

```

1 import numpy as np
2
3 def incmatrix(genl1, genl2):
4     m = len(genl1)
5     n = len(genl2)
6     M = None #to become the incidence matrix
7     VT = np.zeros((n*m,1), int) #dummy variable
8
9     #compute the bitwise xor matrix
10    M1 = bitxormatrix(genl1)
11    M2 = np.triu(bitxormatrix(genl2),1)
12
13    for i in range(m-1):
14        for j in range(i+1, m):
15            [r,c] = np.where(M2 == M1[i,j])
16            for k in range(len(r)):
17                VT[(i)*n + r[k]] = 1;
18                VT[(i)*n + c[k]] = 1;
19                VT[(j)*n + r[k]] = 1;
20                VT[(j)*n + c[k]] = 1;
21
22            if M is None:
23                M = np.copy(VT)
24            else:
25                M = np.concatenate((M, VT), 1)
26
27            VT = np.zeros((n*m,1), int)
28
29    return M

```

Código A.1: Título del fragmento de código

A.5. Listas

Como en cualquier procesador de textos debemos diferenciar dos tipos de listas. Las listas no numeradas (o de viñetas) se definirán mediante entornos `itemize`.

- Elemento no numerado
- Elemento no numerado
- Elemento no numerado

Las listas numeradas se definirán mediante entornos `enumerate`. En ambos casos, cada elemento de la lista se genera utilizando el comando `item`.

1. Elemento 1

Con el comando `\` podemos insertar saltos de línea sin cambiar de párrafo.

2. Elemento 2. Con el comando `\textbf` podemos **enfaticar con negrita** un texto y con el comando `\textit` podemos *enfaticar con itálica* un texto.

3. Elemento 3

A.6. Ecuaciones

Cuando una ecuación va a ser referenciada desde el texto, en principio más de una vez, será necesario nombrar o numerar dicha ecuación (ver [Ecuación A.1](#)). Para añadir una ecuación numerada utilizaremos el entorno `equation`.

$$\begin{pmatrix} w_1, & w_2, & \dots, & w_n \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} + b = 0 \rightarrow \mathbf{w}^t \mathbf{x} + b = 0 \quad (\text{A.1})$$

Cuando la ecuación no va a ser referenciada desde el texto, podemos utilizar la versión de entorno no numerada `equation*`

$$\frac{1}{2} \|\mathbf{w}\| = \frac{1}{2} \sqrt{\sum w_i^2}$$

Es también posible añadir ecuaciones en línea con el texto, para ello se debe incluir la expresión matemática en L^AT_EX encerrada entre símbolos de dólar. Por ejemplo, la ecuación anterior se podría representar también en línea: $\frac{1}{2} \|\mathbf{w}\| = \frac{1}{2} \sqrt{\sum w_i^2}$.