Requerimientos y Diseño – Clínica Veterinaria

Mateo Madrigal

Programación II

Índice:

1. INTRODUCCIÓN Y ALCANCE

Objetivo:

Desarrollar una aplicación para gestionar clientes, mascotas, citas y facturación de una clínica veterinaria, con persistencia en base de datos y un front-end sencillo.

Alcance:

- CRUD de Clientes y Mascotas
- Agenda de Citas (crear, listar, actualizar estado, cancelar)
- Facturación básica asociada a cita (factura + líneas)
- Dashboard con métricas (nº citas por mes, ingresos, top clientes)

2. METODOLOGÍAS Y CRITERIOS DE CALIDAD

- XP/Scrum: iteraciones cortas
- TDD (Red-Green-Refactor) para lógica de dominio y persistencia.
- SOLID en el diseño OO; separación de capas.

3. HISTORIAS DE USUARIO

1. RECEPCIONISTA

Como recepcionista, quiero registrar un cliente con sus datos de contacto, para poder asociar sus mascotas y citas.

Criterios de aceptación:

a. **Given** un formulario con nombre, DNI/NIF, teléfono y email, **when** completo campos obligatorios válidos, **then** se crea el cliente con ID único.
b. **Given** DNI duplicado, **when** guardo, **then** recibo error de duplicidad y no se crea registro

2. RECEPCIONISTA

a. Como recepcionista, quiero registrar una mascota asociada a un cliente, para llevar su historial.
b. Criterios**:** cliente existente requerido, especie/raza/nombre obligatorios; evita duplicados por (cliente, nombre, fecha_nacimiento) salvo confirmación de usuario.

3. RECEPCIONISTA

a. Como recepcionista, quiero crear una cita (cliente, mascota, veterinario, fecha/hora, motivo), para agendar visitas.
b. Criterios: impide solapes de un mismo veterinario; estado inicial = "programada"; fecha futura obligatoria.

4. VETERINARIO

a. Como veterinario, quiero marcar la cita como completada o cancelada, para actualizar el estado del calendario.
b. Criterios: sólo estados válidos {programada, completada, cancelada}; registrar motivo de cancelación.

5. VETERINARIO

a. Como administración, **q**uiero generar una factura desde una cita completada**,** para registrar ingresos.
b. Criterios**:** número de factura incremental; líneas con concepto, cantidad, precio, impuestos; totales calculados; no factura si cita no completada.

6. DUEÑO

a. Como gerente, quiero ver métricas de actividad e ingresos, para tomar decisiones.
b. Criterios: vistas por rango de fechas; KPIs básicos; gráficos interactivos.

4. REQUISITOS FUNCIONALES

- Clientes: Alta, consulta, edición, baja lógica. Validación de DNI/email.
- Mascotas: Alta, consulta, edición, baja lógica. Asociación a cliente.
- Citas: Crear, listar (filtros por fecha/veterinario/estado), reprogramar, completar, cancelar (con motivo), evitar solapes por veterinario.
- Facturación: Generar desde cita completada; gestionar líneas (concepto, cantidad, precio, IVA); calcular totales; listar y exportar factura (PDF/HTML – opcional).

5. REQUISITOS NO FUNCIONALES

- RNF-01 Persistencia: Base de datos relacional con integridad referencial (ver §7). ORM SQLAlchemy.
- Interfaz: Streamlit multipágina para MVP (posible FastAPI para API futura).
- Calidad: TDD con pytest; cobertura objetivo >80% en dominio.
- Diseño: OO con SOLID (SRP, OCP, LSP, ISP, DIP); separación de capas (dominio, infraestructura, presentación).
- Rendimiento: Listados paginados para >10k registros.
- Seguridad y datos: Validación de entrada, saneo básico; logs y manejo de excepciones; .env para credenciales.
- DevOps: Repositorio Git; GitHub Actions con linters (ruff/black) + tests.
- Documentación: README, guía de despliegue, diagrama ER/Clases.

6. ARQUITECTURA Y DISEÑO

Capas:

- Dominio: entidades, servicios de dominio (reglas negocio: solapes, estados, totales).
- Infraestructura: repositorios SQLAlchemy, migraciones (alembic opcional), adaptadores.
- Presentación: Streamlit (formularios, listados, gráficos).
- Principios: SRP (cada clase una responsabilidad), OCP (extensión sin modificar), DIP (dependencias por interfaz).

7. BASE DE DATOS

- **SQLite**