Asignatura (2025-2026)
Práctica Final Programación II
Marcos García, Alejandro González, Micaela López
Rufino Prieto Alonso

PRÁCTICA FINAL PROGRAMACIÓN II

Marcos García, Alejandro González,
Micaela López

Índice

Asignatura (2025-2026)
Práctica Final Programación II
Marcos García, Alejandro González, Micaela López
Rufino Prieto Alonso

SECCIÓN 1

-Descripción de la plataforma-

En esta sección se define el alcance, el objetivo, las tecnologías, así como la metodología implementada y la planificación y organización del proyecto.

Práctica final programación II: gestión de clínica veterinaria

Objetivo

El proyecto consiste en el desarrollo de una aplicación diseñada para la gestión integral de una clínica veterinaria. El objetivo principal es digitalizar y centralizar los procesos administrativos y clínicos diarios. Este proyecto está diseñado para gestionar la base de datos de los clientes, las mascotas, el personal de la clínica y la agenda. A través de nuestra interfaz en Streamlit, aseguramos la integridad de los datos, una aplicación funcional y así una mejora en la atención a los pacientes.

Alcance

Nuestro sistema asegura a cuatro módulos funcionales que interactúan entre sí para poder cubrir las funcionalidades de la clínica veterinaria:

1.1. Gestión de clientes y mascotas

Incluimos la administración completa de la información de los dueños y sus animales, añadiendo vinculación lógica de múltiples mascotas a un único propietario (relación 1:N).
Hemos incluido la capacidad de localizar clientes rápidamente por DNI o Nombre y sus mascotas por ID, Especie o DNI del dueño. Visualización centralizada de citas pasadas y futuras desde la ficha de cada mascota.

1.2. Gestión de personal (Veterinarios)

Hemos incluido el registro detallado del personal veterinario, incluyendo especialidad y datos de contacto. También se pueden asignar citas concretas a veterinarios y gestionar el solapamiento con error.

1.3. Gestión de citas

Aplicación de creación de citas que vinculan una mascota, un cliente y un veterinario en una fecha y hora específicas. Se incluye motivo de la cita y estado.

1.4. Arquitectura y seguridad

Almacenamiento seguro en base de datos relacional. Registro del logging para funcionamiento y detección de errores.

Tecnologías

- **Python 3:** Lenguaje principal del proyecto. Elegido por su legibilidad, robustez y la amplia disponibilidad de librerías para gestión de datos y desarrollo rápido.
- **Streamlit:** Framework para crear aplicaciones de datos en Python. Permite desarrollar una interfaz gráfica web interactiva y multipágina.
- **SQLAlchemy:** Mapeador Objeto-Relacional. Abstrae las consultas SQL permitiendo trabajar con objetos Python. Facilita la migración entre gestores de bases de datos (de SQLite a PostgreSQL) y previene inyecciones SQL.
- **SQLite:** BBDD empleado por su ligereza y su fácil y rápida implementación.
- **Pandas:** Utilizado para manipular las tablas de datos y facilitar la visualización de listados y filtros en la interfaz.
- **GitHub:** Plataforma en la nube para alojar código.usado para control de versiones, facilidad para trabajar a distancia y sirve como una copia de seguridad en caso de errores, fallos, etc.
- **Bcrypt:** Librería de hashing de contraseñas. Es esencial para proteger las credenciales de acceso de los veterinarios y administradores. Se asegura de que las contraseñas nunca se almacenen en texto plano en la base de datos.

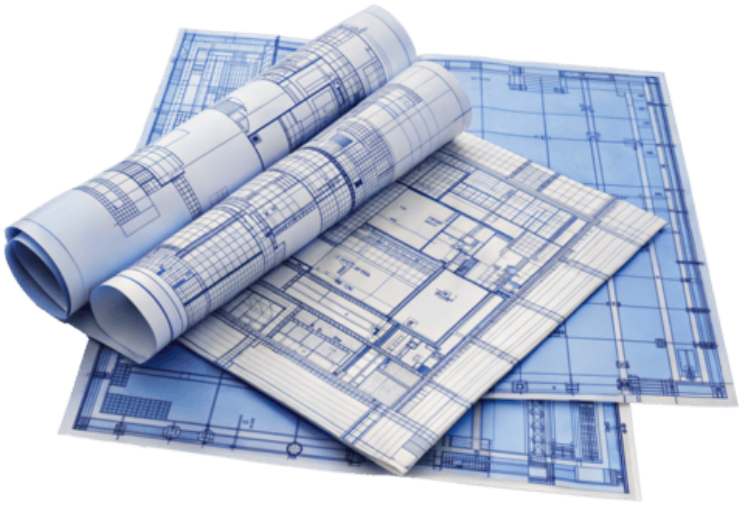
Desarrollo y organización del proyecto

La dinámica que hemos implementado ha sido la de Extreme Programming, a pesar de que la metodología se realiza mediante pequeñas entregas, nosotros al principio tuvimos que entregar tres pantallas funcionales, pero luego hemos ido programando tiempos para tenerlo todo codificado. También hemos ido revisando todo lo que hacíamos para que fuese funcional y no chocase con algo nuevo añadido.

El proyecto lo hemos organizado de manera que hemos llevado a cabo 13 reuniones, reuniéndonos todos los integrantes del grupo y centrándonos todos en una única tarea a la vez, para poder saber qué hacíamos en cada momento. Todas las reuniones se han realizado de manera online debido a la distancia entre los distintos integrantes del equipo, así, uno compartía pantalla e íbamos desarrollando el código. Las especificaciones de las tareas y cambios realizados en cada reunión se pueden ver definidas en el README incluido en el proyecto.

SECCIÓN 2

-Presentación de requisitos-



Tras revisar la descripción de la plataforma atentamente, los estudiantes presentan una especificación de requisitos tanto funcionales como no funcionales de cómo pretenden implementar estas funcionalidades en el proyecto.

Requisitos funcionales

Módulo de Clientes

RF1: El sistema debe permitir registrar nuevos clientes con los siguientes campos:

- `ID_cliente (PK)`
- `Nombre_cliente`
- `DNI_cliente`
- `Tlf_cliente`
- `Correo_electronico_cliente`

RF2: El sistema debe permitir consultar, editar y eliminar los datos de clientes existentes.
RF3: El sistema debe permitir buscar clientes por nombre o DNI.
RF4: El sistema debe mostrar el listado completo de clientes.

Métodos:

- `def crear_cliente`
- `def modificar_cliente`
- `def consultar_cliente`
- `def obtener_cliente_por_id`
- `def eliminar_cliente`
- `def listar_clientes`

Módulo de Mascotas

RF5: Cada cliente puede tener una o varias mascotas (relación 1:N).
RF6: Cada mascota tendrá los siguientes campos:

- `ID_mascota (PK)`
- `Nombre_mascota`
- `Nombre_mascota`
- `Especie_mascota`
- `Raza_mascota`
- `Edad_mascota`
- `Peso_mascota`
- `Sexo_mascota`
- `ID_cliente (FK)`

Métodos:

- `def registrar_mascota`
- `def consultar_mascota`
- `def modificar_mascota`
- `def eliminar_mascota`
- `def listar_mascotas`
- `def ver_historial_mascota`

RF7: El sistema debe permitir registrar, consultar, modificar y eliminar mascotas.
RF8: Desde la ficha de una mascota se deben poder ver sus citas realizadas y próximas.

Módulo de Veterinarios

RF9: El sistema debe permitir registrar veterinarios o empleados con los campos:

- `ID_veterinario (PK)`
- `Nombre_veterinario`
- `DNI_veterinario`
- `Cargo_veterinario`
- `Especialidad_veterinario`
- `Telefono_veterianrio`
- `Correo_electronico_veterinario`

RF10: El sistema debe permitir editar, eliminar y consultar la información de los empleados.
RF11: El sistema debe permitir asignar veterinarios a las citas de las mascotas.
RF12: Se podrá filtrar citas o estadísticas por veterinario, para analizar la carga de trabajo de cada uno.

Métodos:

- `def registrar_veterinario`
- `def consultar_veterinario`
- `def eliminar_veterinario`
- `def listar_veterinarios`

Módulo de Citas

RF13: El sistema debe permitir crear una cita veterinaria con los campos:

- `ID_cita (PK)`
- `ID_mascota (FK)`
- `ID_cliente (FK)`
- `Fecha_cita`
- `Hora_cita`
- `Estado_cita`
- `Diagnostico_cita`

RF14: El sistema debe permitir consultar y filtrar citas por fecha, veterinario, mascota o cliente.
RF15: El sistema debe permitir modificar o cancelar citas.
RF16: El sistema debe mostrar un calendario o listado cronológico de las próximas citas.

Métodos:

- `def crear_cita`
- `def modificar_cita`
- `def cancelar_cita`
- `def listar_citas`

Módulo de Análisis / Dashboard

RF17: El sistema debe mostrar gráficos interactivos (con Streamlit) sobre:

- Número de citas por mes
- Citas por veterinario
- Mascotas atendidas por especie
- Clientes nuevos por mes
- Satisfacción de los clientes
- Citas por especialidad
- Tiempo medio de atención
- Mostrar KPIs (rendimiento de la clínica)

RF18: Los gráficos deben actualizarse al modificar datos.

Requisitos no funcionales

RNF1: Los datos se almacenarán mediante SQLAlchemy y una base de datos relacional (SQLite o PostgreSQL).

RNF2: El código se organizará en módulos:

- clientes.py
- mascotas.py
- veterinarios.py
- citas.py
- analisis.py
- database.py
- main.py → Lógica principal

RNF3: Debe incluir un archivo `requirements.txt` y `README.md`.

RNF4: El sistema debe validar la entrada de datos y manejar errores (por ejemplo, fechas incorrectas).

RNF5: La interfaz será desarrollada con Streamlit, con una estructura multipágina.

RNF6: Los datos se almacenarán mediante SQLAlchemy y una base de datos relacional (SQLite o PostgreSQL).

RNF7: Debe permitir ejecutar la aplicación localmente.

RNF8: El sistema debe registrar eventos relevantes de la aplicación en un archivo de log **.log**. Debe existir un módulo **logger.py** dentro del proyecto que centralice la configuración de logging.

RNF9: Archivo de configuración de entorno **.config**

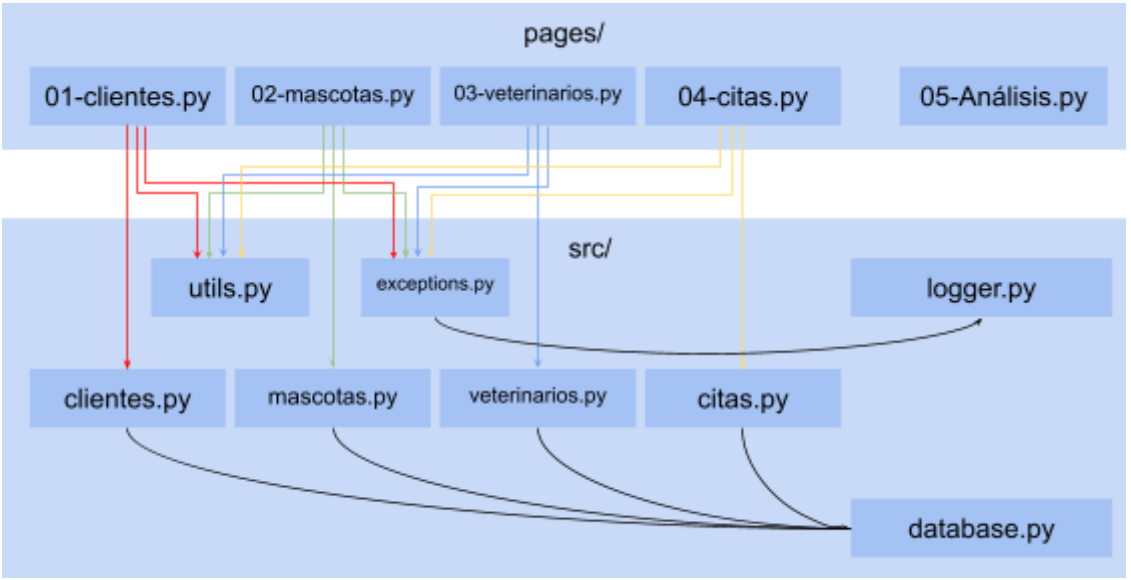
SECCIÓN 3

-Desarrollo del proyecto-

Con los requisitos en mano, se desarrolla el proyecto, incluídos en un repositorio de GitHub. En esta sección se desglosa cómo se organiza el proyecto y cuales son los diagramas de secuencia y comunicación entre archivos dentro de éste.

1. Directorio principal

El proyecto se organiza de la siguiente manera:



`pages/`

Todos los módulos dentro de este directorio contienen los ficheros responsables de estructurar el **`front-end`** a través de **`Streamlit`**. Cada módulo separa los "**`tabs`**" (pestañas) de la estructura de la página en clases. Cada una de ellas conteniendo métodos estáticos responsables de renderizar los widgets visuales así como otros métodos a los que recurren para enviar datos, validar campos o permitir interacciones. Los módulos contenidos en esta carpeta son:

- **`01-Clientes.py`**
- **`02-Mascotas.py`**
- **`03-Veterinarios.py`**
- **`04-Citas.py`**
- **`05-Análisis.py`**

`src/`

Este módulo contiene toda la lógica relacionada con las operaciones internas que ejecuta el servidor para procesar las interacciones que ejecuta el usuario. Centrándose en 5 secciones principales.

- `database.py`: Responsable de configurar la base de datos del sistema empleando `SQLAlchemy`, que se apoya en el servicio en máquina de `SQLite`. Crea el motor (engine) y empleando el sistema ORM de `SQLAlchemy` convierte las clases (`Cliente`, `Mascota`, `Veterinario` y `Cita`) utilizando como clase padre la clase `Base (decorative_base()`). Por último, inicia la sesión (`sesion = sessionmaker(bind=engine)`) que servirá como punto de conexión del servidor con la base de datos, objeto desde el que se ejecutarán todas las operaciones sobre la db.
- `utils.py`: La clase Utilidades contiene métodos estáticos que ejecutan macros consistentes en validación de datos, formateo o transformaciones.
- **módulos para cada página**: Cada uno de estos módulos son la capa inferior de lógica cada página de `Streamlit`, las páginas del módulo pages/ se comunican paralelamente con estos módulos para insertar u obtener datos. Cada módulo lo conforman dos clases principales:
  - `_Repositoriox`: Clase privada que únicamente utiliza session para realizar operaciones `CRUD` sobre la base de datos referentes a la entidad 'x' las cuales son `session.add('x')` (crear), `session.delete('x')` (eliminar) y `session.query('x')` (buscar, listar), cambiando los atributos de las entidades para actualizar registros. empleando `session.commit()` para registrar los cambios en la base de datos.
  - `_Serviciox`: Clase privada que envuelve a la clase `_Repositoriox`, añade funcionalidades extra y comprende excepciones cuando es necesario.
  - Funciones: Acceden a `_Serviciox` o `_Repositoriox` y los ejecutan como punto de llamada, son las que se emplean en streamlit.

  Los ficheros dentro de esta categoría son:
  - `clientes.py`
  - `mascotas.py`
  - `veterinarios.py`
  - `citas.py`

- `analisis.py`: Responsable de estructurar la lógica que reclama la página de análisis de `Streamlit`, devuelve queries más específicos (como número de citas por veterinario).
- `exceptions.py` y `logger.py`: Responsables de manejar excepciones comprendidas en el código y registrarlas en un log.

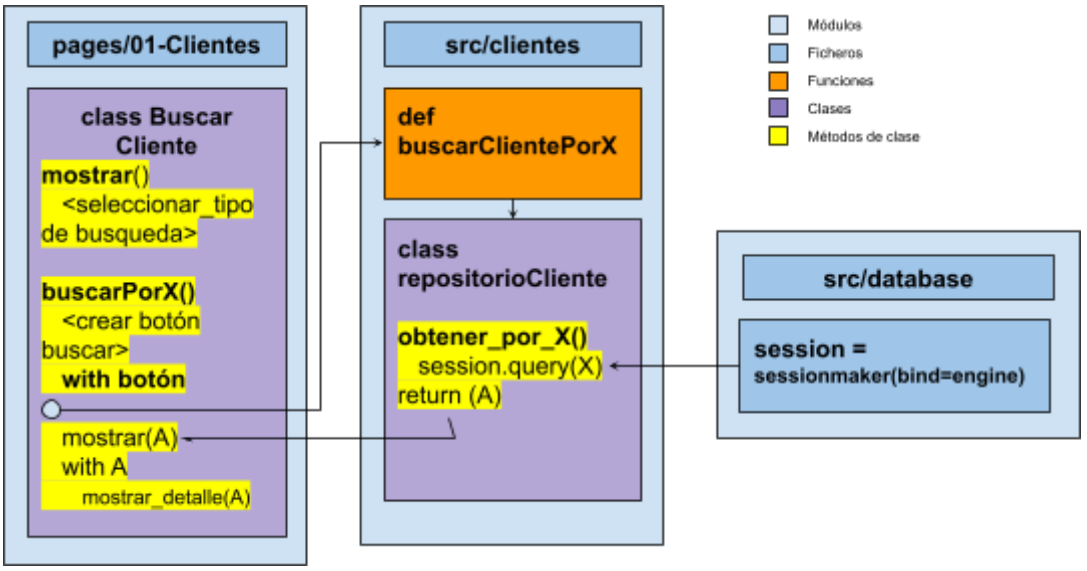
`tests/`
Contiene módulos encargados de dirigir el proyecto a través de una dinámica TDD, emplea `pytest` para comprobar que el código cumple las pruebas y el funcionamiento requerido.

otros módulos:

- `app.py`: Punto de entrada de la página del proyecto, contiene la lógica de `Streamlit` de la página de índice y utiliza `bcrypt` para desarrollar la lógica de inicio de sesión
- `clinica.db`: Archivo creado por `src/database.py` que contiene los datos del sistema en una base relacional.
- `.streamlit/config.toml`: Archivo interceptado por `Streanmlit` al ejecutar el servicio, modifica la apariencia de la página.
- `img`/: Módulo que contiene assets visuales (imágenes) que emplea `Streamlit` con st.image
- `logs/app,log`: Archivo de log

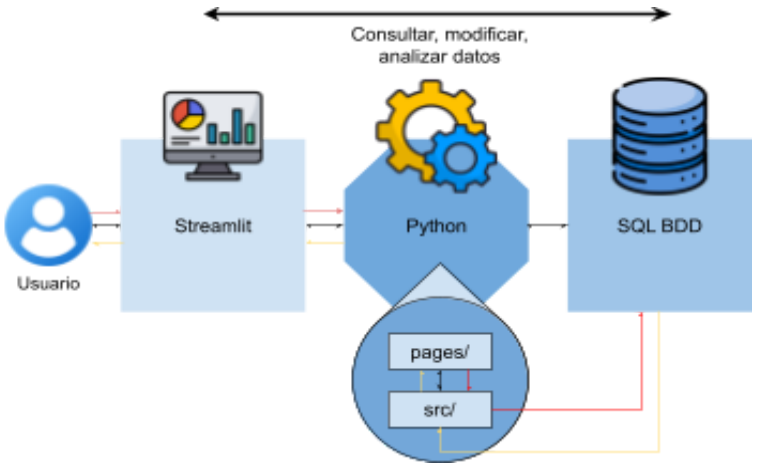
Notas:

- Cada archivo de `Streamlit` contiene una pequeña sección de CSS que modifica el fondo para incluir un gradiente
- Algunos módulos de pages/ emplean funciones de src/ que no pertenecen a su "`src/paralelo`", por ejemplo, la función `obtener_cliente_por_id` (de `src/clientes.py`) se emplea en varios módulos, no solo en `pages/01-Clientes.py`)



2. Funcionamiento (a grandes rasgos)

La plataforma maneja, desde las páginas de streamlit (front-end), la base de datos de clientes, citas, veterinarios y mascotas de una clínica veterinaria genérica, con un servidor de python (SQL-back-end). El sistema permitirá tanto consultar como manipular datos. Incluyendo módulos como buscar entidades (clientes, mascotas, veterinarios, citas) a través de distintos criterios, añadir, editar y eliminar registros y visualizar datos.



2.1. Datos comprendidos en la BBDD

Tabla	Propósito
Clientes	Guarda la información de los dueños de las mascotas.
Mascotas	Contiene los datos de cada animal y su relación con el cliente.
Veterinarios/Empleados	Registra el personal de la clínica (veterinarios, asistentes, administrativos).
Citas	Gestiona las consultas, tratamientos y servicios realizados a las mascotas.

3. Probar la plataforma

Usuarios para pruebas:

- admin → admin123
- vet → vet123