**Asignatura:** Programación Orientada a Objeto Seguro

**Sección:** TI3V21/V-IEI-N2-P3-C1/V Valparaíso IEI

**Docente:** Miguel Ángel Chacana González

**Integrantes del grupo:**

* José Miguel Astudillo Aguirre (Desarrollador)
* Michele Andrea Barriga Carrasco (Product Owner)
* Patricia Catalina Riveros Estay (Scrum Master)

**EVA03\_SmartService\_Veterinaria**

Contenido

[Introducción 3](#_Toc210943319)

[Descripción del problema 3](#_Toc210943320)

[Objetivos 3](#_Toc210943321)

[Metodología y Evidencias 3](#_Toc210943322)

[Metodología 3](#_Toc210943323)

[Sprints 3](#_Toc210943324)

[Evidencias 5](#_Toc210943325)

[Diagramas UML 6](#_Toc210943326)

[Diseño y estructura del sistema 7](#_Toc210943327)

[Capturas de pantalla del funcionamiento. 7](#_Toc210943328)

[Conclusiones y aprendizajes. 7](#_Toc210943329)

# Introducción

Con este informe nuestro equipo documentara el desarrollo del sistema de gestión de la veterinaria realizada por grupo describiendo las decisiones de diseño, metodología aplicada, arquitectura de sistema, comentarios de código y las pruebas realizadas y sirve como evidencia del cumplimiento de los requerimientos del cliente simulado.

# Descripción del problema

La empresa simulada SmartService Subsidiaria de BENEFIX requiere un sistema de gestión para optimizar la operación de una veterinaria pequeña, organizando y centralizando la administración de clientes, mascotas, citas, usuarios, este proyecto responde a la necesidad de digitalizar procesos administrativos básicos, reducir errores por manejo manual y disponer de información persistente y segura para la toma de decisiones.

# Objetivos

El proyecto debe incluir las siguientes funcionalidades:

* Gestión de usuarios: registro, autenticación y verificación de credenciales con contraseñas cifradas mediante hashing.
* Interfaz por consola: sistema de login con menú principal y submenús para cada módulo.
* Persistencia de datos: uso de una base de datos SQLite para almacenar usuarios, veterinarios, mascotas y reservas.
* Operaciones CRUD: agregar, editar, eliminar y listar registros en cada entidad.
* Reportes: generación de un reporte o resumen de la información almacenada (por pantalla y exportable a archivo).
* Estructura modular: código separado por archivos (main, controladores, sg\_veterinaria, etc.) y comentado adecuadamente.
* Evidencia de proceso ágil: entrega del informe técnico, que incluya diagrama UML, casos de prueba y documentación del uso de Scrum durante el desarrollo.

# Metodología y Evidencias

## Metodología

Para el desarrollo del proyecto se aplicó la **metodología ágil Scrum**, adaptada a un formato de **micro-sprints diarios** debido al tiempo disponible (del 9 al 15 de octubre). Esta metodología permitió una organización eficiente del equipo, entregas incrementales y una mejora continua a través de revisiones diarias.

El equipo se estructuró con los siguientes roles:

* **Product Owner (PO):** encargado de definir los requerimientos y prioridades del cliente simulado.
* **Scrum Master (SM):** responsable de coordinar el trabajo diario, eliminar bloqueos y asegurar el cumplimiento de los objetivos.
* **Desarrollador(es):** encargado(s) de la implementación del sistema, base de datos y pruebas funcionales.

## Sprints

Se establecieron **siete sprints diarios**, cada uno con un objetivo claro y entregables concretos, abarcando desde la planificación y diseño inicial hasta la entrega final del sistema funcional.  
Las tareas fueron organizadas y monitoreadas en un **tablero Kanban** con las columnas:  
**Pendiente → En curso → Validación → Terminado**, permitiendo visualizar el progreso y mantener una comunicación efectiva dentro del equipo.

**Sprint 1 – Planificación y Diseño (9 oct)**

Objetivo: Alinear alcance, roles y entornos.  
Entregables: Tipo de sistema, roles, repositorio y tablero.

Tareas:

* Elegir sistema
* Asignar roles
* Crear repositorio
* Crear tablero Kanban

**Sprint 2 – Modelado y Base de Datos (10 oct)**

Objetivo: Diseñar UML y estructura de base de datos.  
Entregables: Casos de uso, clases, bd modelo físico.

Tareas:

* Crear diagrama de casos de uso
* Crear diagrama de clases
* Diseñar modelo físico de BD
* Revisar coherencia entre diagramas
* Subir diseños al repositorio

**Sprint 3 – Autenticación (11 oct)**

Objetivo: Implementar autenticación segura.  
Entregables: Registro y login con hash.

Tareas:

* Implementar registro con hash
* Implementar login con validación
* Pruebas de inicio y cierre de sesión
* Documentar flujo de autenticación

**Sprint 4 – CRUD y Reportes (12 oct)**

Objetivo: Crear operaciones CRUD y función de reportes.  
Entregables: Listar funciones CRUD agregadas y reportes.

Tareas:

* Implementar funciones CRUD
* Conectar CRUD correctamente a BD
* Generar función de reporte/resumen filtrado y funcional
* Validar manejo de errores
* Pruebas funcionales del CRUD

**Sprint 5 – POO (13 oct)**

Objetivo: Agregar funcionalidad orientada a objetos con clases, atributos y herencias.  
Entregables: Clases, atributos y herencias.

Tareas:

* Aplicar clases, atributos y herencias
* Validar manejo de errores
* Pruebas funcionales con clases implementadas

**Sprint 6 – Integración y Pruebas Generales (14 oct)**

Objetivo: Integrar todos los módulos y probar el sistema completo.  
Entregables: Sistema integrado y estable.

Tareas:

* Integrar autenticación + CRUDs
* Probar flujo completo del sistema
* Depurar errores encontrados
* Verificar persistencia y lógica
* Preparar versión candidata final

**Sprint 7 – Documentación y Entrega (15 oct)**

Objetivo: Documentar y entregar el proyecto final.  
Entregables: Finalizar informe técnico y proyecto funcional para ser entregado.

Tareas:

* Finalizar informe técnico completando ítems faltantes
* Revisar checklist de entrega
* Entregar versión final

## Evidencias

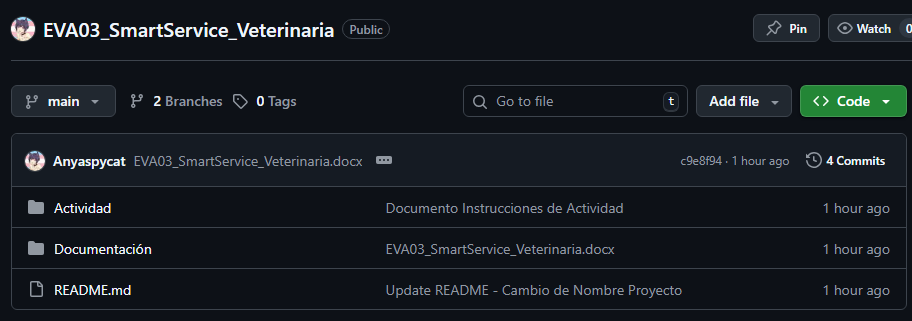
**Evidencias Sprint 1 – Planificación y Diseño:**

**Tipo de Sistema: Gestor de Reserva Veterinaria**

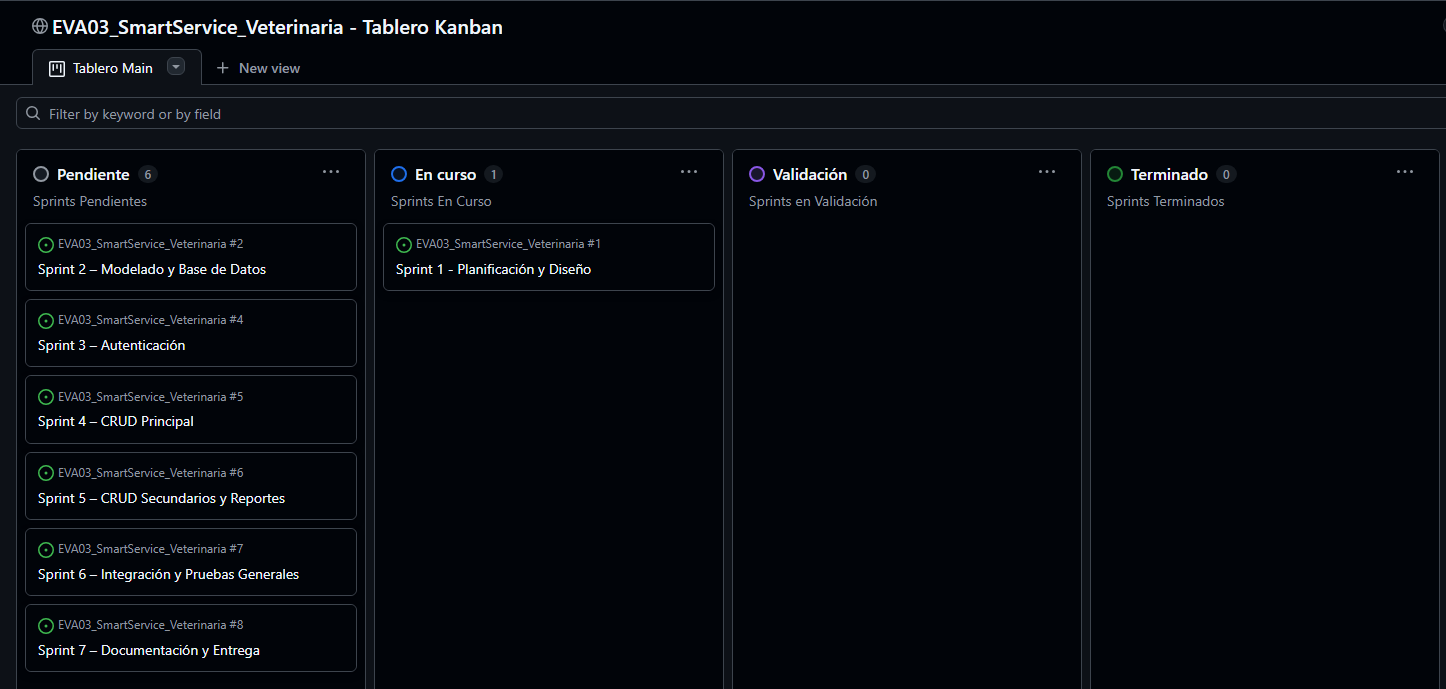
**Roles:**

* José Astudillo - Desarrollador
* Michele Barriga - Product Owner
* Patricia Riveros - Scrum Master

**Repositorio:**



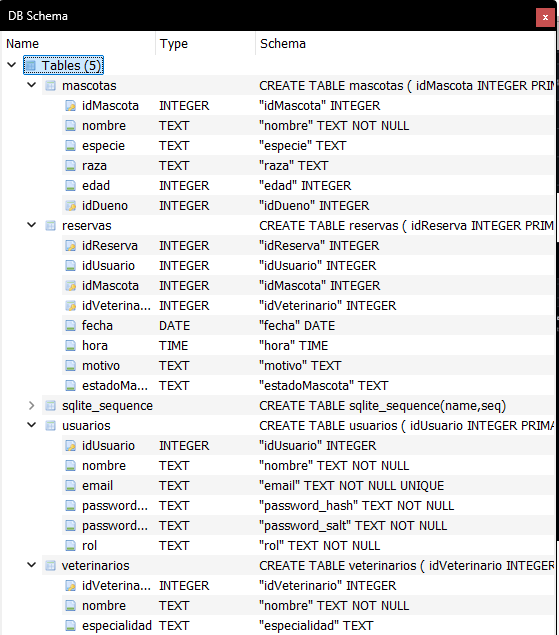
**Tablero Kanban:**



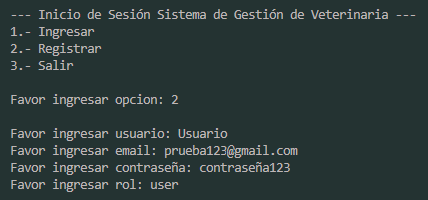
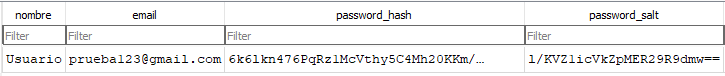
**Evidencias Sprint 2 – Modelado y Base de Datos:**

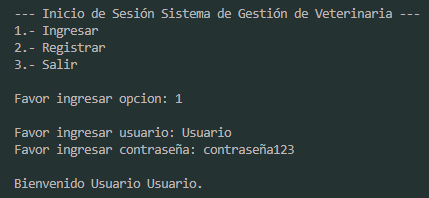
**Casos de Uso y Clases: Véase página “Diagramas UML”**

**BD Modelo Físico:**

****

**Evidencias Sprint 3 – Autenticación:**

**Ingreso correcto del registro con hash:  
  
**

**Validación correcta del registro con hash:  
**

**Flujo de autenticación:**

1. Usuario selecciona opción de “Registrar”.
2. Usuario ingresa los datos solicitados.
3. Se invoca la función “registrar\_login” y “hash\_password” para encriptar la contraseña utilizando librería hashlib y posteriormente se codifica en base64, para finalmente ser almacenada en la base de datos sg\_veterinaria.db junto al “salt” único generado en el momento, tabla usuarios.
4. Usuario selecciona opción “Ingresar”.
5. Usuario ingresa los datos solicitados.
6. Se invoca la función “verificar\_login” y “hash\_password”, primero se decodifica en base 64 para volver a encriptar la contraseña utilizando librería hashlib, esta vez utilizando el “salt” almacenado en la base de datos y se realiza comparación, validando que la autenticación se encuentre ok.
7. Usuario ingresa al sistema principal.

**Sprint 4 – CRUD y Reportes (12 oct)**

**Lista de Funciones CRUD:**

* **Tabla “mascotas”:**
  + registrar\_nueva\_mascota()
  + listar\_mascotas()
  + eliminar\_mascota()
  + actualizar\_mascota()
  + buscar\_mascotas\_por\_propietario()
  + buscar\_mascotas\_por\_especie()
  + contar\_mascotas()
* **Tabla “reservas”:**
  + crear\_reserva()
  + mostrar\_reservas()
  + modificar\_reserva()
  + eliminar\_reserva()
* **Tabla “veterinarios”:**
  + registrar\_nuevo\_veterinario()
  + listar\_veterinarios()
  + eliminar\_veterinario()
  + actualizar\_veterinario()
  + buscar\_veterinarios\_por\_especialidad()
  + buscar\_veterinarios\_por\_nombre()
  + contar\_veterinarios()
* **Tabla “usuarios”:**
  + registrar\_login()
  + verificar\_login()
  + hash\_password()

**Lista de Funciones Reportes:**

* reporte\_resumen\_general()
* exportar\_resumen\_general\_txt()

# Diagramas UML

El diagrama UML de clases que se presenta a continuación modela la estructura principal del sistema de gestión de reservas desarrollado para la veterinaria “SmartService”. Este diagrama constituye una herramienta fundamental para representar las entidades del sistema, sus atributos, métodos y las relaciones que existen entre ellas, permitiendo una comprensión clara y detallada de cómo está organizado el software a nivel lógico y orientado a objetos.

El sistema fue diseñado considerando que el veterinario es el principal usuario que interactúa con la aplicación desde la consola. Este usuario, al autenticarse, puede registrar citas para pacientes (mascotas), asignarse a sí mismo como responsable de la atención, o bien asignar a otro veterinario disponible, centralizando así la gestión operativa de la clínica.

Entre las clases principales que conforman el sistema, se encuentra la clase Usuario, que representa a todas las personas que pueden acceder al sistema, identificados por su nombre, correo electrónico, rol y contraseña encriptada. Le sigue la clase Veterinario, una especialización del usuario que posee atributos propios como su especialidad, y funcionalidades específicas como autoasignarse reservas o asignarlas a otros colegas. La clase Mascota representa a los animales registrados en el sistema, asociados a un dueño (usuario) y con información como especie, raza, edad y nombre.

La clase Reserva modela las citas médicas, vinculando una mascota con un veterinario, junto con detalles como fecha, hora, motivo de la consulta y estado de la reserva (pendiente, confirmada o cancelada). Todas estas clases están conectadas con la clase BaseDatos, la cual abstrae las operaciones de almacenamiento y recuperación de información desde una base de datos SQLite. Para garantizar la seguridad del sistema, se incluye la clase Autenticador, encargada de gestionar el registro de nuevos usuarios, la autenticación durante el inicio de sesión, y el uso de funciones de hash para proteger las credenciales.

Diagrama de Clases:

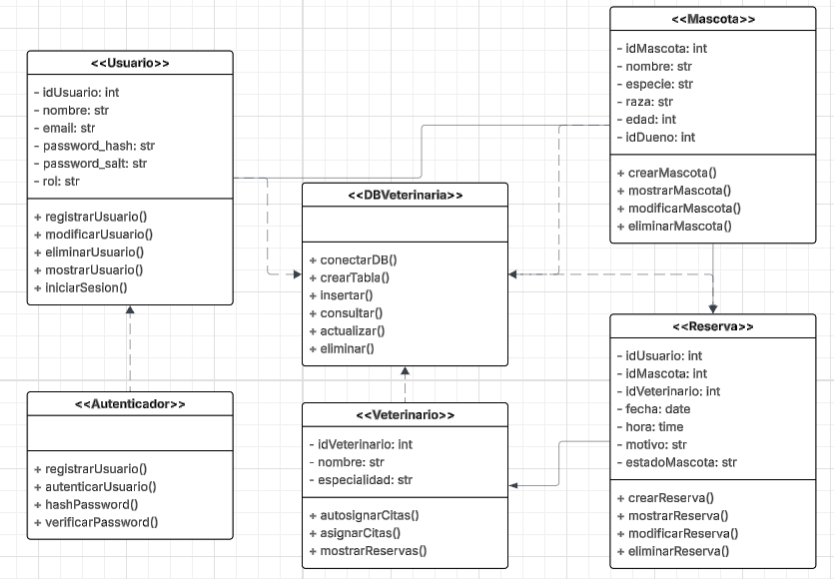
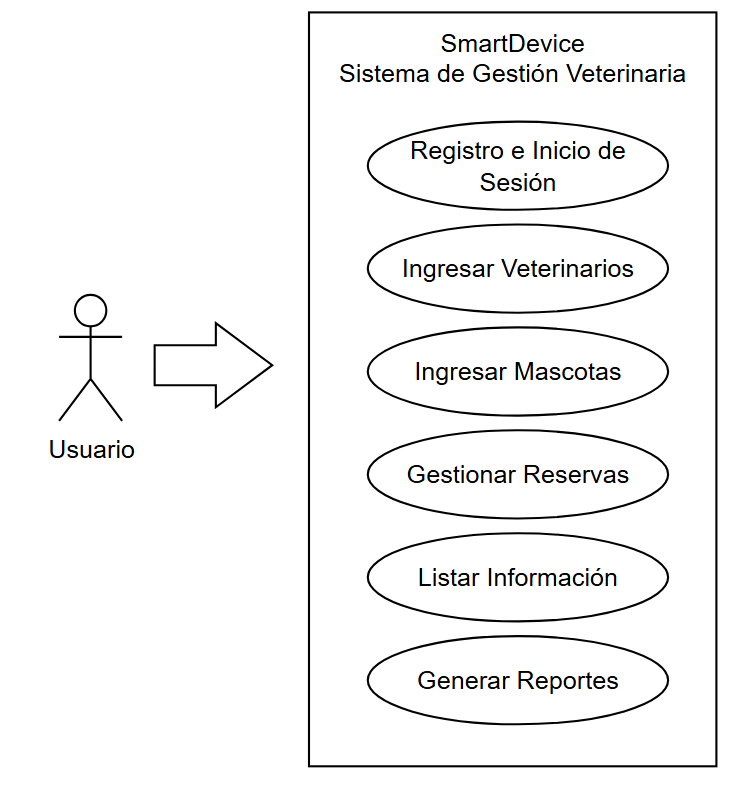


Diagrama de Casos de Uso:



# Diseño y estructura del sistema

# Capturas de pantalla del funcionamiento.

# Conclusiones y aprendizajes.