

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

ToBoVet Web App: Diseño de Sistema de Gestión para una Clínica Veterinaria

AUTOR: Carlos Javier de la Torre Monguió

Puerto Real, Marzo 2025

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA INFORMÁTICA

ToBoVet Web App: Diseño de Sistema de Gestión para una Clínica Veterinaria

AUTOR: Carlos Javier de la Torre Monguió
TUTOR: Pedro Delgado Pérez

Puerto Real, Marzo 2025

Declaración personal de autoría

Carlos Javier de la Torre Monguió con DNI 49890046X, estudiante del Grado en Ingeniería Informática en la Escuela Superior de Ingeniería de la Universidad de Cádiz, como autor de este documento académico titulado ToBoVet Web App: Diseño de Sistema de Gestión para una Clínica Veterinaria y presentado como Trabajo Final de Grado.

DECLARO QUE

Es un trabajo original, que no copio ni utilizo parte de obra alguna sin mencionar de forma clara y precisa su origen tanto en el cuerpo del texto como en su bibliografía y que no empleo datos de terceros sin la debida autorización, de acuerdo con la legislación vigente. Asimismo, declaro que soy plenamente consciente de que no respetar esta obligación podrá implicar la aplicación de sanciones académicas, sin perjuicio de otras actuaciones que pudieran iniciarse. En Puerto Real, a marzo de 2025

Fdo: Carlos Javier de la Torre Monguió

Agradecimientos

A mis amigos, porque después de tantos años diciendo “no” a todos los planes, habéis seguido a mi lado y habéis apoyado que priorizara sacarme la carrera lo antes posible. Especialmente a vosotros, Javi y Sevi, sin vuestro compromiso y apoyo, este proyecto no hubiera sido posible.

A mi pareja, por compartir tantas tardes de trabajo y ayudarme, animándome a seguir con la carrera, aclarándome mis objetivos y aspiraciones. Tu paciencia y comprensión han sido fundamentales para gestionar la frustración a lo largo de este proceso.

A mi tutor, Pedro, por acompañarme durante la mitad de la carrera y por su guía y asesoramiento a lo largo de todo el desarrollo de este trabajo.

A mis compañeros de carrera, por su constante apoyo y sus numerosas tutorías y consejos, sin vuestra ayuda estaría presentando este trabajo con canas.

Por último, a mi familia, por cuidarme durante las temporadas de exámenes, hacerme de comer cuando tenía que estudiar y permitirme la oportunidad de poder centrarme en mis estudios y forjar una carrera profesional tan enriquecedora y prometedora.

Resumen

El presente Trabajo de Fin de Grado tiene como objetivo desarrollar una aplicación web destinada a la gestión integral de una clínica veterinaria a domicilio recientemente inaugurada. Este proyecto surge de nuestro interés personal por el bienestar animal, el desarrollo web y la posibilidad de trabajar en un proyecto de gran envergadura. La clínica veterinaria mencionada tenía dificultades en la administración de clientes y servicios de manera eficiente, ya que carecía de herramientas especializadas y asequibles para poder cubrir todas las necesidades del negocio. El sistema se estructura en cuatro módulos principales: gestión de clientes y mascotas, administración de citas, facturación y pagos, e inventario. Cada módulo ha sido diseñado para optimizar las operaciones diarias, ofreciendo además flexibilidad y capacidad de adaptación. Destaca que el proyecto se ha desarrollado completamente a medida para los usuarios finales, lo que permite la implementación ágil y sencilla de funcionalidades adicionales de acuerdo a las necesidades cambiantes y de expansión del negocio.

Palabras clave: Veterinaria, gestión, web, React, Node.js, SQL, API REST

Índice general

Índice de figuras	x
-------------------	---

Índice de tablas	xii
------------------	-----

I Prolegómeno 1

1. Introducción 3

1.1. Motivación	3
1.2. Alcance y Objetivos	3
1.3. Glosario de términos	4
1.4. Organización del documento	5

2. Plan de gestión de proyecto 8

2.1. Antecedentes	8
2.1.1. Contexto	8
2.1.2. Estado de la técnica	8
2.2. Metodología de desarrollo	9
2.3. Tecnologías	10
2.4. Planificación del proyecto	14
2.4.1. Versión 0.5: Versión Inicial	14
2.4.2. Versión 1.0: Consolidación y Mejoras	14
2.4.3. Versión 2.0: Ajustes Finales y Optimización	15
2.5. Organización	15
2.5.1. Roles y Responsabilidades	16
2.5.2. Recursos Técnicos	16
2.5.3. Jira Como Herramienta	16
2.6. Costes	17
2.6.1. Coste de Desarrollo	18
2.6.2. Coste de Mantenimiento	18
2.6.3. Resumen de Costes	19
2.7. Gestión de la configuración	19
2.8. Aseguramiento de calidad	20

II Desarrollo 21

3. Requisitos del Sistema 23

3.1. Objetivos de Negocio	23
3.2. Objetivos del Sistema	23
3.3. Requisitos funcionales	24
3.4. Requisitos no funcionales	25
3.5. Requisitos de información	25

4. Análisis del Sistema	28
4.1. Modelo Conceptual de Datos	28
4.2. Modelo de Casos de Uso	28
4.2.1. Casos de Uso	31
4.3. Modelo de Interfaz de Usuario	46
5. Diseño del Sistema	48
5.1. Base de Datos	48
5.1.1. Diseño de Datos	48
5.2. Diseño de la Interfaz de Usuario	58
5.2.1. Principios de Diseño	58
5.2.2. Estilos	58
5.2.3. Componentes Visuales	59
5.2.4. Diseño Responsivo	68
6. Implementación	71
6.1. Entorno de Construcción	71
6.1.1. Entorno de Desarrollo	71
6.1.2. Lenguaje de Programación	71
6.1.3. Credenciales de Inicio de Sesión	71
6.1.4. Herramientas de Construcción y Despliegue	71
6.1.5. Control de Versiones	72
6.1.6. Repositorio de Componentes	72
6.1.7. Integración Continua	72
6.2. Código Fuente	72
6.2.1. Organización del código fuente	72
6.2.2. Extractos Significativos de Código	73
7. Pruebas del Sistema	79
7.1. Estrategia	79
7.1.1. Alcance de las Pruebas	79
7.1.2. Interpretación y Evaluación de los resultados	80
7.2. Pruebas Unitarias y de Integración	80
7.2.1. Ejemplo de Prueba Unitaria	81
7.2.2. Pruebas de Integración	83
7.3. Pruebas de Aceptación	85
7.3.1. Pruebas Funcionales	85
7.3.2. Pruebas No Funcionales	85
7.3.3. Pruebas de Accesibilidad	88
7.4. Pruebas de Usabilidad	89
7.4.1. Estrategia	89
7.4.2. Resultados	91
7.4.3. Pruebas de Diseño Responsivo	93
8. Despliegue del Sistema	96
8.1. Arquitectura Física	96
8.2. Instrucciones de despliegue	96
8.2.1. Requisitos Previos	97

8.2.2. Procedimientos de instalación	97
8.3. Instrucciones para la operación del sistema y mantenimiento del nivel de servicio	98
III Epílogo	99
9. Conclusiones	101
9.1. Objetivos alcanzados	101
9.2. Lecciones aprendidas	101
9.3. Trabajo futuro	102
Bibliografía	104
IV Anexos	107
A. Manual del desarrollador	109
A.1. Manual de instalación	109
A.1.1. MySQL	109
A.1.2. Visual Studio Code (VS Code)	110
A.1.3. Node.js y npm	112
A.1.4. React	114
A.2. Consideraciones generales sobre el desarrollo	115
A.2.1. Guías de estilo de codificación	116
A.2.2. Prácticas recomendadas para el desarrollo	116
A.2.3. Directrices para realizar pruebas	116
B. Manual de usuario	118
B.1. Escenario 1: Creación de un nuevo usuario	118
B.2. Escenario 2: Agendar una cita	119
B.3. Escenario 3: Terminar cita	121

Índice de figuras

2.1. React	10
2.2. MySQL	11
2.3. Node.js	11
2.4. Reactstrap	11
2.5. HTML y CSS	12
2.6. Jest	12
2.7. Lighthouse	12
2.8. Postman	12
2.9. Figma	13
2.10. Jira	13
2.11. Git	13
2.12. Google Cloud Platform	14
2.13. Diagrama de Gantt	15
2.14. Interfaz de Jira	17
2.15. Detalles de tarea	18
4.1. Modelo conceptual de datos	29
4.2. Modelo de Casos de Uso	30
4.3. Diagrama de navegación	46
5.1. Representación gráfica de la BD	49
5.2. Tipografía “Roboto”	59
5.3. Iconos de la librería Material Icons	59
5.5. Modal General	60
5.6. Vista Inicio Sesión	61
5.7. Vista Inicio	62
5.8. Vista Clientes	62
5.9. Vista Historial	63
5.10. Vista Artículos	63
5.11. Opciones de tabla	64
5.12. Factura en proceso	64
5.13. Factura hecha	65
5.14. Historial de Loki	66
5.15. Ejemplo de documento generado	67
7.1. Pruebas jest	83
7.2. Resultados GET Postman	88
7.3. Puntuación media según el SUS	92
7.4. Gráficos de los datos recogidos por el test SUS	92
7.5. Vista de Inicio en la pantalla del portátil del cliente	94
7.6. Vista de Inicio en el móvil del cliente	95
8.1. Configuración de la GCP	96

A.1. Nueva conexión MySQL	110
B.1. Pantalla Clientes	119
B.2. Formulario de añadir cliente	120
B.3. Cliente encontrado	120
B.4. Modal para crear cita	121
B.5. Página de Inicio	122
B.6. Editar cita	123
B.7. Próximas vacunaciones	123
B.8. Crear cita con vacuna	124

Índice de tablas

5.1. Tabla clients	50
5.2. Índices de la tabla clients	50
5.3. Tabla companies	50
5.4. Tabla client_contacts	51
5.5. Índices de la tabla client_contacts	51
5.6. Claves Foráneas de la tabla client_contacts	51
5.7. Tabla sys_users	51
5.8. Índices de la tabla sys_users	52
5.9. Tabla products	52
5.10. Tabla payments	52
5.11. Índices de la tabla payments	52
5.12. Claves Foráneas de la tabla payments	53
5.13. Tabla payments_articles	53
5.14. Índices de la tabla payments_articles	53
5.15. Claves Foráneas de la tabla payments_articles	53
5.16. Tabla pets	54
5.17. Índices de la tabla pets	54
5.18. Claves Foráneas de la tabla pets	54
5.19. Tabla tags	54
5.20. Tabla pet_tags	55
5.21. Índices de la tabla pet_tags	55
5.22. Claves Foráneas de la tabla pet_tags	55
5.23. Tabla visits	56
5.24. Índices de la tabla visits	56
5.25. Claves Foráneas de la tabla visits	56
5.26. Tabla vaccinations	57
5.27. Índices de la tabla vaccinations	57
5.28. Claves Foráneas de la tabla vaccinations	57
7.1. Resultados de Lighthouse en Ordenador	86
7.2. Resultados de Lighthouse en Móvil	87
7.3. Comparación de “Factura” con otras páginas	87
7.4. Verificación de criterios de accesibilidad según WCAG 2.1	90

Parte I

Prolegómeno

1. Introducción

A continuación, se describe la motivación del presente proyecto y su alcance. También se incluye un glosario de términos y la organización del resto de la presente documentación.

1.1. Motivación

Durante nuestros años en el instituto, participamos en programas de voluntariado que nos permitieron involucrarnos de manera directa con el bienestar animal, destacándose especialmente nuestra colaboración con una protectora en El Puerto de Santa María. Durante esa experiencia se nos planteó el siguiente dilema: ¿sería posible relacionar el aprecio por el bienestar animal con la ingeniería informática? Durante un tiempo, ambos campos parecían incompatibles, hasta que se nos propuso la creación de una clínica veterinaria a domicilio.

Se nos presentó la oportunidad de impulsar un negocio que no solo ayudaría a los animales, sino también a los veterinarios que dedican sus esfuerzos a cuidarlos. Asimismo, el proyecto ofrecía la posibilidad de aplicar los conocimientos adquiridos durante el voluntariado para abordar problemas comunes en la gestión, tales como la deficiente integración entre clientes, mascotas y pagos, la gestión manual del inventario o la inadecuada implementación de aplicaciones para generar documentos oficiales.

Además, cuando surgió la oportunidad de crear un sistema íntegramente hecho por un único informático, integrando tantas competencias adquiridas a lo largo de nuestra formación como la recogida de requisitos, el interés por aprender nuevas tecnologías y el buen diseño de una base de datos desde cero, nos resultó imposible declinarla.

Con este proyecto, pretendemos brindar al negocio una solución tecnológica integral que facilite y automatice sus procesos operativos, así como aportar un primer granito de arena a la comunidad veterinaria de la bahía de Cádiz, impulsando un negocio pionero y ético que busca mejorar la calidad de vida de los animales, sus dueños y los profesionales del sector.

1.2. Alcance y Objetivos

El objetivo principal de este proyecto es facilitar y agilizar las operaciones diarias que se realizan en una clínica veterinaria, como la gestión de citas, tratamientos, clientes o inventario. Anteriormente, este tipo de operaciones se hacían con hojas de cálculo, carpetas físicas y documentos hechos a mano, procesos que dificultan la eficiencia del negocio e impactan negativamente en el servicio que ofrecían.

La aplicación brindará características como la automatización de procesos críticos, la capacidad de acceder de forma remota en cualquier momento y lugar y la centralización de todos estos procesos. Además, la aplicación permite la escalabilidad y adaptación a las nuevas necesidades del negocio a medida que la clínica crezca. Cumpliendo estos objetivos se espera una mejora significativa de la gestión del negocio.

Por la naturaleza del proyecto, que debe permitir el acceso al sistema de forma remota y de cualquier dispositivo, cabe destacar como uno de los objetivos principales su responsividad a distintos tamaños de pantalla, especialmente para portátiles y dispositivos móviles.

La aplicación se integra en todas las áreas operativas del negocio, desde la gestión de clientes y mascotas hasta la facturación y contabilidad, siempre buscando integrar todas estas actividades en un sistema eficiente y accesible.

A continuación, se destacan los objetivos marcados por el proyecto desde el comienzo:

- **Mejora de la eficiencia.** Digitalizar los procesos manuales, reduciendo tiempos de trabajo y minimizando los errores humanos.
- **Facilitar la gestión de clientes y mascotas.** Gestionar de manera centralizada la información de los clientes y sus mascotas, incluyendo datos personales, historial médico y citas para ofrecer un mejor seguimiento.
- **Optimizar la programación de citas.** Hacer uso eficiente de los recursos humanos de los que dispone la empresa, exprimiendo al máximo los horarios disponibles de los trabajadores y evitando solapamientos
- **Creación de documentos personalizados.** Facilitar uno de los procesos más tediosos de la veterinaria, pudiendo generar documentos parametrizables con un solo *clic*

1.3. Glosario de términos

- **API:** Application Programming Interface (Interfaz de programación de aplicaciones).
- **BD:** Base de datos.
- **CRUD:** Create, Read, Update, Delete. Operaciones básicas de una base de datos.
- **GCP:** Google Cloud Platform. Conjunto de recursos físicos (CPU, Almacenamiento, RAM...) que Google ofrece mediante una suscripción a sus servicios.
- **IDE:** Integrated Development Environment (Entorno de desarrollo integrado).
- **JS:** Javascript.
- **JWT:** JSON Web Token. Estándar abierto para la creación de tokens de accesos que permiten la propagación de privilegios.

- **REST:** Representational State Transfer. Estilo arquitectónico para diseñar servicios web.
- **npm:** Node Package Manager.
- **SO:** Sistema Operativo.
- **SIE:** Sistemas de información en la empresa.
- **SQL:** Structured query language. Lenguaje para procesar información de una Base de Datos.
- **TS:** Typescript. Lenguaje de programación superconjunto de JavaScript que añade tipado estático opcional y funciones avanzadas a JavaScript.
- **UML:** Unified Modeling Language.
- **VM:** Virtual Machine (Máquina virtual). Entorno informático que funciona como sistema aislado pero se crea a partir de recursos hardware.

1.4. Organización del documento

Esta memoria detalla toda la elaboración del proyecto, desde la primera fase de análisis hasta su implementación, organizada en las secciones que se describen a continuación:

- **Capítulo 1: Introducción** En este capítulo se describe la motivación que ha llevado al desarrollo de este proyecto, junto al contexto y ámbito en el que se desarrolla el proyecto.

También se narran los alcances y objetivos, que describen cómo afecta el programa a la organización del cliente, el desarrollo del nuevo sistema y los principales objetivos que se esperan alcanzar cuando el sistema entre en producción.

Por último, se incluye un glosario de términos que usaremos durante todo el documento.

- **Capítulo 2: Plan de gestión de proyecto.** Se describen todos los aspectos relativos a la gestión del proyecto: antecedentes, metodología, costes, planificación, riesgos y aseguramiento de la calidad.
- **Capítulo 3: Requisitos del sistema.** Se presentan los objetivos y el catálogo de requisitos del nuevo sistema informático.
- **Capítulo 4: Análisis.** Este capítulo cubre el análisis del sistema de información a desarrollar, haciendo uso del lenguaje de modelado UML.
- **Capítulo 5: Diseño.** En este capítulo se recoge el diseño de la arquitectura, datos, e interfaz del sistema informático, así como el entorno de construcción y la organización del código fuente.

- **Capítulo 6: Implementación.** En este capítulo se describen el entorno de construcción y marco tecnológico del sistema, así como la estructura del código fuente y fragmentos significativos del código.
- **Capítulo 7: Pruebas.** En este capítulo se presenta el plan de pruebas del sistema de información, incluyendo los diferentes tipos de pruebas que se han llevado a cabo, ya sean manuales (mediante listas de comprobación) o automatizadas mediante algún software específico de pruebas.
- **Capítulo 8: Despliegue.** Este capítulo recoge la arquitectura física planteada para el sistema, las instrucciones para su despliegue y las instrucciones para la operación y mantenimiento del nivel de servicio.
- **Capítulo 9: Conclusiones.** En este último capítulo se detallan las lecciones aprendidas tras el desarrollo del presente proyecto y se identifican las posibles oportunidades de mejora sobre el software.

2. Plan de gestión de proyecto

En este capítulo se describen todos los aspectos relativos a la gestión del proyecto: contexto, estado de la técnica, metodología, organización, costes, planificación, riesgos y aseguramiento de la calidad.

2.1. Antecedentes

En esta sección se detalla la situación actual que origina el desarrollo o la mejora de un sistema informático. Asimismo, se describen las alternativas tecnológicas existentes.

2.1.1. Contexto

El desarrollo de sistemas informáticos pretende resolver problemas reales que presentan distintos grupos de personas, convirtiendo los tediosos y poco integrados procesos manuales en flujos de trabajo más eficientes y centralizados. Como se señala en [Sommerville \(2016\)](#), la ingeniería de software busca precisamente optimizar estos procesos para garantizar la escalabilidad, la consistencia y la fiabilidad de la información. Así, en el caso que nos ocupa, hemos detectado que la gestión de la clínica veterinaria tiende a hacerse de modo tradicional y poco optimizado, usando métodos organizativos poco eficientes como las hojas de cálculo, documentos impresos, carpetas compartidas o cuadernos de notas (especialmente en lo que respecta a la administración de citas, facturas y documentos). Estas prácticas suponen un obstáculo en términos de escalabilidad, consistencia y centralización, dado que dependen en gran medida de intervención humana, la memoria de los usuarios y documentos físicos que pueden perderse o malograrse. Como se señala en [Aubry \(2018\)](#), la transición desde métodos tradicionales a soluciones digitales es esencial para mejorar la eficiencia y centralización de la información.

El entorno tecnológico actual en el que se desarrolla este proyecto no se caracteriza por la utilización de plataformas integradas o soluciones digitales específicas para el sector veterinario. En cambio, observamos que las tecnologías aplicadas a la gestión de este servicio están poco centralizadas, y las alternativas existentes son genéricas o el coste de licencia es exagerado. Este proyecto se plantea como una respuesta a la necesidad de modernizar y unificar los procesos administrativos, transformando la experiencia de los veterinarios y los clientes.

2.1.2. Estado de la técnica

Antes de comenzar con la planificación del proyecto, se ha hecho un estudio exploratorio de las soluciones tecnológicas existentes, especialmente de una en la que el

cliente tenía experiencia previa. El análisis de la oferta nos ha revelado que existen varias aplicaciones de software libre y soluciones comerciales para la gestión de clínicas veterinarias. Sin embargo, la mayoría no se ajustaban exactamente a las necesidades específicas de una clínica veterinaria a domicilio. Algunas aplicaciones ofrecen funcionalidades limitadas, con las más potentes detrás de una barrera de pago; otras, por su parte, requieren licencias anuales a un coste exorbitado. Como es natural, estas características son inviables para negocios con escasos recursos.

Este estudio ha permitido identificar los elementos y requisitos clave que debe tener la nueva solución, además de los requisitos no funcionales como la necesidad de planear para la escalabilidad y futuras necesidades del negocio. Así, el proyecto no solo se diferencia en cuanto a su novedad, al ser una solución a medida que responde de forma directa a los problemas citados, sino que también actúa como base sobre la que poder incorporar nuevas funcionalidades conforme evolucionen las necesidades de la empresa o las demandas del mercado.

En definitiva, el estado de la técnica muestra que, a pesar de la existencia de otras alternativas, ninguna de ellas aborda de forma integral las necesidades específicas de una clínica a domicilio, lo que justifica la inversión en el desarrollo de una solución propia que combine flexibilidad, eficiencia y experiencia de usuario.

2.2. Metodología de desarrollo

Por la naturaleza del proyecto, la metodología empleada en su desarrollo se basó en un enfoque ágil, que permitió una adaptación continua a las necesidades del cliente y del negocio. Al inicio del proyecto, se llevó a cabo una reunión con el cliente para establecer qué requisitos debía cumplir la aplicación. Para esta fase se observaron tecnologías similares para definir la base del sistema y sus puntos clave. Seguidamente, a través de las directrices de metodologías ágiles recomendadas en [Schwaber y Sutherland \(2020\)](#) y adoptando principios de [Anderson \(2010\)](#) para la gestión de tareas, se creó un prototipo consolidado que servía como base para el proyecto. Durante todo el proceso de desarrollo, la retroalimentación del cliente fue continua, por lo que se realizaron rápidos *sprints* de trabajo durante las siguientes semanas para poder tener lo más cercano a una versión final para la fecha de apertura, fijada para marzo de 2024.

Para todo el proceso de desarrollo del proyecto, usamos junto al cliente la herramienta Jira ([Atlassian \(2025\)](#)) para poder comunicar la necesidad de nuevas funcionalidades con su prioridad, posibles bloqueos, rama de desarrollo y otras funcionalidades que esta plataforma permite.

Las modificaciones continuas permitieron desarrollar un sistema completamente estable. Las actualizaciones eran pequeñas y numerosas, lo que permitía tener controladas las versiones del proyecto y localizar los errores con facilidad. Una vez alcanzada la fecha de apertura, momento en el que abrió oficialmente la clínica veterinaria, el ritmo de trabajo cambió, la presión de una fecha de entrega inminente disminuyó y con ello el desarrollo del trabajo se centró más en solucionar errores que se fueran detectando.

Aun así, surgían nuevas funcionalidades por implementar que requerirían actua-

lizaciones más grandes: cambios completos de la estructura de la BD, de la página, la interfaz de usuario... Como durante esta fase priorizamos la estabilidad y el refinamiento, surgió la necesidad de hacer un pequeño servidor de pruebas en el que se duplicaran las acciones que se llevaran a cabo en producción, antes de comprobar que era lo suficientemente estable como para actualizar el sistema. Esto permitió mantener la aplicación en funcionamiento mientras se optimizaba en base a la experiencia adquirida y las nuevas necesidades del negocio.

2.3. Tecnologías

- **React.** Elegimos este framework como la biblioteca principal para el desarrollo del frontend por su capacidad de trabajar de manera eficiente con componentes reutilizables, que facilita la creación y reutilización de elementos de la interfaz. React usa el Virtual DOM para optimizar el rendimiento, haciendo que la experiencia de usuario sea más fluida, minimizando las actualizaciones directas al DOM real ([React Team \(2025a\)](#)). Durante la planificación se consideraron también otras alternativas populares. A continuación, nombramos las dos con más peso:
 - Angular. A pesar de ofrecer una solución completa y robusta, su curva de aprendizaje y complejidad en general no se adecuaban a las necesidades de nuestro proyecto, que requiere una implementación rápida y ágil.
 - Vue.js. Destaca entre los demás por su simplicidad y facilidad de integración. Además, se asemeja a React por su uso del Virtual DOM y arquitectura centrada en componentes.

Finalmente, elegimos React por su popularidad, que conlleva una amplia y consolidada documentación y una comunidad numerosa y activa. Además, optamos por utilizar React en conjunto con TS. Aunque no es una combinación con la que inicialmente estuviéramos familiarizados, optamos por ella por dos motivos principales: la oportunidad de aprender y desarrollar nuevas competencias y su creciente demanda en el mercado laboral.

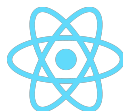


Figura 2.1: React

- **MySQL.** Para la BD seguimos un enfoque diferente, ya que los datos son el cimiento fundamental de este proyecto. Elegimos una tecnología con la que tenemos experiencia y confianza: MySQL, ya que nos permite estructurar la información en tablas interrelacionadas, lo que resulta esencial para manejar de manera efectiva la compleja red de relaciones entre las entidades de nuestro sistema ([Oracle Corporation \(2025\)](#)). La naturaleza del proyecto, que requiere integridad, consistencia y relaciones bien definidas, junto a una amplia comunidad y su soporte

para transacciones, nos hace descartar alternativas no relacionales. Dentro del ámbito de las bases relacionales optamos también por MySQL por la experiencia previa que teníamos con esta base de datos. Otras opciones, como PostgreSQL, amplían funcionalidades que consideramos innecesarias para el alcance de este proyecto.



Figura 2.2: MySQL

- **Node.js.** Node.js se usa para construir la API del proyecto, lo que nos permite ejecutar JS en el servidor y mantener una coherencia en el lenguaje de programación en todo el ámbito de la aplicación ([Node.js Foundation \(2025\)](#)). En la fase de evaluación se consideraron alternativas como PHP o Python pero como hemos comentado anteriormente, la ventaja de poder trabajar con el mismo lenguaje en el lado del cliente y del servidor, hace que Node.js sea la opción ideal para integrar la API con la BD y el frontend.



Figura 2.3: Node.js

- **Reactstrap.** Como complemento al CSS puro hemos usado Reactstrap, una implementación del famoso framework “Bootstrap” para React. Lo más destacable de esta librería y el motivo de su elección es la facilidad y consistencia con la que crea interfaces de usuarios responsivas ([Reactstrap Team \(2025\)](#), [Bootstrap Team \(2025\)](#)), un objetivo con el que debemos cumplir en este proyecto. Anteriormente se optó por Material-UI (de las que aún se usan algunos componentes), sin embargo, la simplicidad con la que Reactstrap logra interfaces adaptativas fue un factor decisivo para su elección.



Figura 2.4: Reactstrap

- **HTML/CSS:** HTML es fundamental para la base estructural de nuestro proyecto, definiendo de forma semántica el contenido y la organización de la información ([WHATWG \(2025\)](#)). CSS por su parte, es fundamental para el diseño visual y la presentación([W3C \(2025\)](#)). Su combinación es esencial para construir una experiencia de usuario robusta y clara, así es que no se plantearon alternativas en este aspecto.



Figura 2.5: HTML y CSS

- **Jest:** Para asegurar la calidad del código, elegimos Jest como nuestro framework para las pruebas unitarias y de integración, por su facilidad de integración con React y Node.js ([Facebook Inc. \(2025\)](#)). Como alternativa se manejó Mocha, que aunque es una opción robusta, requiere una configuración adicional y complementos para exprimir su potencial.



Figura 2.6: Jest

- **Lighthouse:** Para evaluar el rendimiento, la accesibilidad y las mejores prácticas en nuestra aplicación web, escogimos Lighthouse, una herramienta de auditoría automatizada de código abierto proporcionada por Google ([Google \(2025b\)](#)). Su integración con las herramientas de desarrollador de Google Chrome y su facilidad de uso nos permite obtener informes completamente detallados sobre el estado de la web sin necesidad de aplicaciones externas o configuraciones adicionales. Se consideraron alternativas especializadas como WebPageTest (a nivel de red) o Axe (a nivel de accesibilidad). Sin embargo, la facilidad de uso y el enfoque integral en múltiples métricas de Lighthouse nos hizo escoger esta tecnología.



Figura 2.7: Lighthouse

- **Postman:** Usamos Postman para realizar pruebas en la API de forma eficiente, se trata de una herramienta ampliamente utilizada para la creación, prueba y documentación de APIs ([Postman Inc. \(2025\)](#)). Su interfaz intuitiva, la fácil automatización de endpoints y la experiencia que ya teníamos con esta herramienta nos hizo escogerla frente a cualquier otra opción.



Figura 2.8: Postman

- **Figma:** Elegimos Figma como herramienta principal para el prototipado de la web, principalmente por la capacidad de trabajar a tiempo real y su interfaz basada en la web ([Figma \(2025\)](#)). Estas características nos permiten iterar rápidamente en el diseño sin necesidad de reuniones presenciales, depender de plataformas específicas o instalar un software adicional. Se tuvieron en cuenta algunas alternativas como Adobe XD, que ofrece potentes funcionalidades de diseño o Sketch, que es conocido por su facilidad de uso en entornos Mac, sin embargo, el coste de licencia del primero y la exclusividad del sistema operativo en la última, nos hizo optar finalmente por Figma.



Figura 2.9: Figma

- **Jira:** Jira se utiliza como la herramienta central de comunicación y gestión de tareas con el cliente. Esta plataforma facilita el seguimiento detallado de los parches, la implementación de las nuevas funcionalidades y el progreso continuo del proyecto ([Atlassian \(2025\)](#)). Se exploraron otras alternativas como Trello o Asana, pero ninguna permitía organizar tareas con tanto detalle o una integración tan robusta con otras herramientas de desarrollo como Git.



Figura 2.10: Jira

- **Git:** Para el manejo de control de versiones del código, usamos esta tecnología debido a su versatilidad, eficiencia y amplia adopción en la industria actual. Git permite un control preciso en todas las etapas del desarrollo del código a través de ramas, fusiones y un detallado historial de modificaciones ([The Git Project \(2025\)](#)). Por estas razones, además de su integración con otras de las herramientas usadas como Jira o VSCode, Git era una clara elección para el manejo de control de versiones.



Figura 2.11: Git

- **Google Cloud:** Para el despliegue del proyecto, hemos elegido Google Cloud Platform (GCP) utilizando una máquina virtual como entorno de ejecución ([Google \(2025a\)](#)). Para el despliegue del proyecto se consideraron otras alternativas

como Amazon Web Services o Microsoft Azure, finalmente GDP destacó frente a las alternativas por su modelo de precios, que se ajustaba a las necesidades del proyecto, su integración con otras herramientas de Google y su facilidad de uso.



Figura 2.12: Google Cloud Platform

2.4. Planificación del proyecto

Para llevar a cabo este proyecto se ha seguido un plan estructurado que abarca desde la fase inicial de concepción hasta el lanzamiento de la versión 2.0. En la [Figura 2.13](#) podemos observar el Diagrama de Gantt del proyecto, que permite visualizar una estimación de la planificación original de las tareas y su distribución a lo largo del desarrollo. Este diagrama se usó para definir correctamente las prioridades y dependencias de las actividades, así como marcar los *sprints* y versiones del proyecto.

2.4.1. Versión 0.5: Versión Inicial

Esta fase marca el inicio y la definición de la estructura fundamental del proyecto. Realizamos las siguientes actividades:

1. **Inicio y planificación inicial:** llevamos a cabo una reunión inicial con el cliente para definir objetivos, alcance y establecer las primeras iteraciones del proyecto.
2. **Diseño de la BD y reunión de requisitos:** en esta etapa elaboramos un diseño primitivo de la base de datos, estableciendo tablas y relaciones básicas, y se recogieron los requisitos esenciales a partir de la interacción con el cliente.
3. **Desarrollo del prototipo inicial:** implementamos las funciones básicas del sistema a través de:
 - a) **Diseño del backend:** desarrollo de las funcionalidades básicas de creación, modificación y eliminación de las tablas.
 - b) **Diseño del frontend:** desarrollamos una versión primitiva de la interfaz de usuario que permitiera validar el flujo de la aplicación.

Esta fase se planteó en 4 meses, desde diciembre de 2023 a marzo de 2024, fecha de inauguración de la clínica.

2.4.2. Versión 1.0: Consolidación y Mejoras

Tras la inauguración, se inició una fase de ajustes y mejoras continuas para incorporar otras funcionalidades no esenciales y optimizar el sistema.

1. **Nuevos ajustes:** correcciones y optimizaciones generales.
2. **Sprints de mejora:** iteraciones continuas de la herramienta para reestructurar los datos para mejorar la eficacia y la integridad de la BD, implementar un diseño responsivo e incorporar algunas nuevas funcionalidades para cumplir las solicitudes del cliente.

Esta fase se extendió durante aproximadamente 8 meses, desde abril de 2024 a diciembre de 2024.

2.4.3. Versión 2.0: Ajustes Finales y Optimización

En la fase final nos centramos en la consolidación del sistema, poniendo especial atención en la estabilidad y escalabilidad del proyecto. En esta fase se plantean nuevas funcionalidades y mejoras que podrían acarrear una nueva reestructuración de los datos, implementaciones de API externas, nuevas vistas y un rediseño de la interfaz de usuario. Esta fase se lleva a cabo desde enero de 2025 hasta la actualidad.

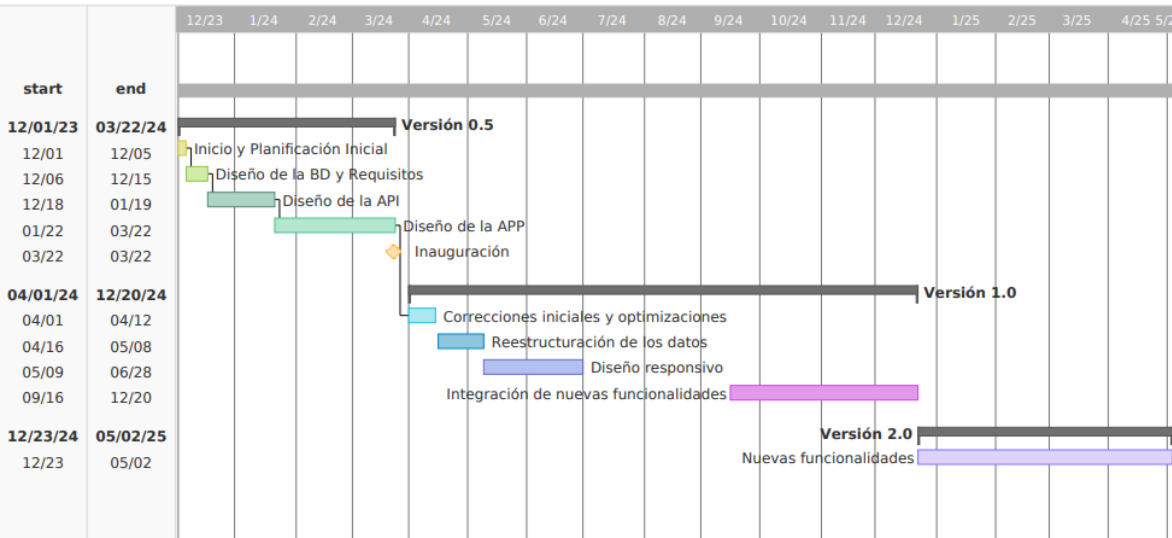


Figura 2.13: Diagrama de Gantt

2.5. Organización

En este proyecto podemos limitar las personas involucradas y sus roles a dos: el cliente (veterinario) y el desarrollador, facilitando una estructura de trabajo ágil y directa, con una comunicación continua y dinámica. A continuación, detallamos los roles y los recursos empleados en su desarrollo.

2.5.1. Roles y Responsabilidades

- **Cliente (Veterinario):** usuario principal de la página, encargado de proporcionar los requisitos funcionales y no funcionales, así como de validar las funciones implementadas mediante revisiones periódicas. Su retroalimentación continua ha sido esencial para el correcto desarrollo del proyecto.
- **Desarrollador (Responsable del proyecto):** responsable de todo el proceso de desarrollo del proyecto, así como el análisis y diseño del sistema, la implementación e integración y el mantenimiento posterior. Además, se encarga de incorporar nuevas funcionalidades solicitadas en base a las necesidades y observaciones del cliente.

2.5.2. Recursos Técnicos

Para el desarrollo del proyecto, se han utilizado una serie de recursos software y hardware que han permitido el diseño y la construcción del sistema.

- **IDE:** Visual Studio Code, empleado para escribir y organizar el código fuente
- **Gestión de proyectos:** Jira, utilizado para organizar tareas y documentar el proceso y versiones del proyecto.
- **Front-end:** React, empleado en la lógica y diseño de interfaz de usuario del proyecto.
- **Back-end:** Node.js, junto a bibliotecas adicionales para la lógica de procesamiento por parte del servidor y la conexión a MySQL.
- **Control de versiones:** Git y GitHub para el control de versiones y la gestión del código fuente del proyecto.

2.5.3. Jira Como Herramienta

Al ser Jira una herramienta relativamente novedosa, nos detendremos a explicar qué hace y por qué funciona tan bien para nuestro proyecto. Jira es una herramienta idónea para la naturaleza de nuestra aplicación porque permite gestionar la evolución del desarrollo de manera ágil y estructurada. Por el hecho de que la aplicación fue concebida mediante un desarrollo iterativo y en evolución permanente, debíamos contar con una plataforma que permitiera planificar tareas, asignar prioridades y llevar un registro de los cambios de una forma clara y estructurada. Con Jira, fuimos capaces de dividir el trabajo en pequeños *sprints*, registrar nuevas funcionalidades o errores detectados en producción y registrar todos y cada uno de los cambios que debíamos hacer en el sistema. Su capacidad para gestionar flujos de trabajo personalizados y su integración con otras herramientas nos permitieron mantener una comunicación fluida con el cliente e integrar varias de nuestras tecnologías en una misma herramienta. A continuación se muestran dos capturas:

En la primera 2.14, se muestra el tablero principal de Jira, estructurado en cuatro columnas: “Por hacer”, “En progreso”, “Bloqueado” y “En revisión de código”. Esta división nos facilita la visualización del estado actual de cada tarea y permite identificar de un solo vistazo las actividades pendientes y su progreso. Además, cada tarea presenta de forma visual información relevante, como su tipo (funcionalidad o error), su prioridad, el desarrollador asignado (especialmente útil en equipos con múltiples desarrolladores) y la existencia de subtareas asociadas.

En la segunda imagen 2.15, podemos observar el detalle de una tarea dentro de Jira. En la parte superior, se presentan atributos como la complejidad y la prioridad de la tarea, seguidos de su descripción, una plantilla recomendada por la IA de Jira para generar documentación y una sección de comentarios. En el panel lateral derecho, se incluye información adicional, como el estado de la tarea, el responsable, el *sprint* correspondiente y dos opciones de creación: una para generar una nueva rama y otra para registrar un *commit*. Al pulsar estos botones, se genera un nombre de rama y un mensaje de *commit* específicos para la tarea en cuestión.

Si bien estas funcionalidades representan solo una mínima parte de las capacidades de Jira, resulta fundamental destacar su papel en la correcta planificación y ejecución de este proyecto.

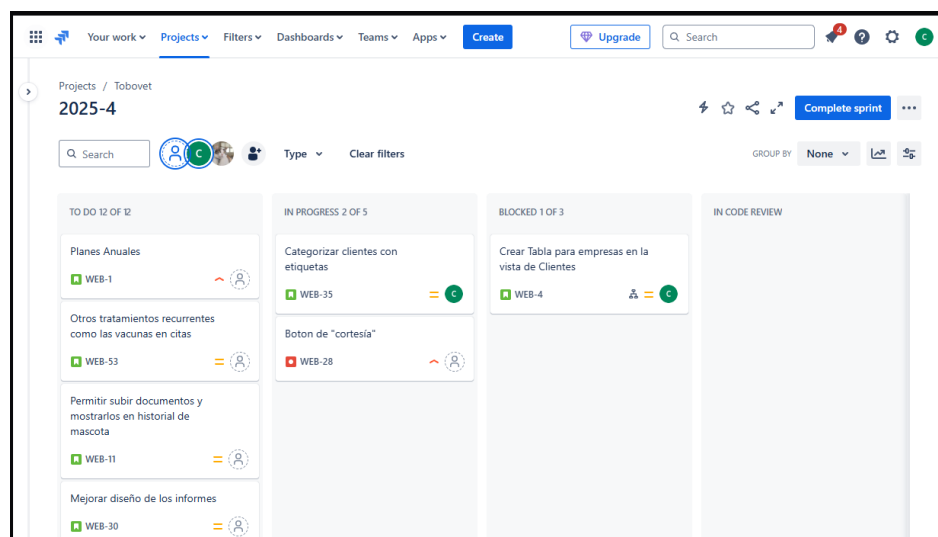


Figura 2.14: Interfaz de Jira

2.6. Costes

Para el desarrollo de este proyecto se han estimado los costes en función de los recursos empleados y el tiempo invertido. Los costes se dividen en dos categorías principales: el coste de desarrollo inicial y el coste de mantenimiento anual.

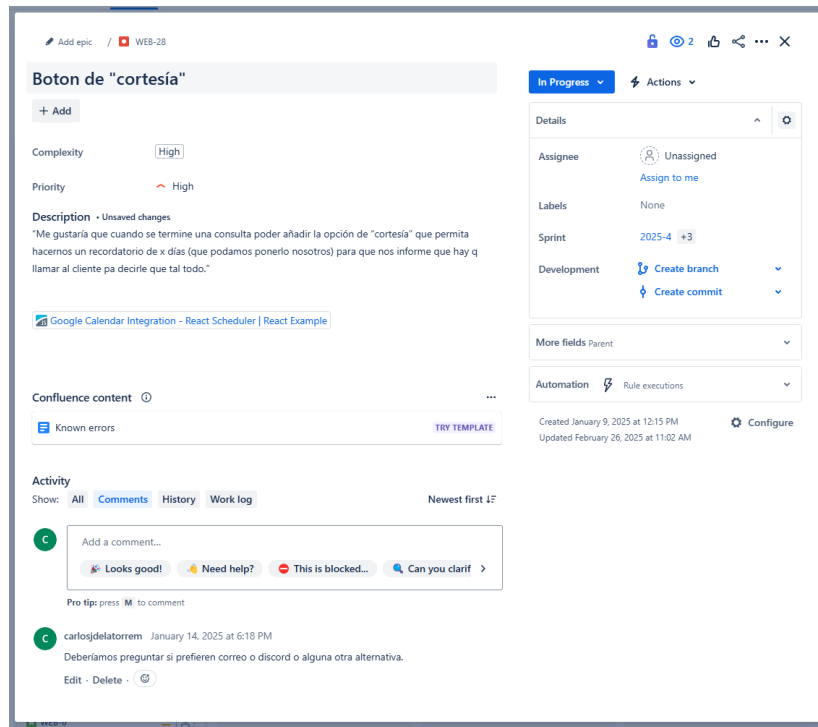


Figura 2.15: Detalles de tarea

2.6.1. Coste de Desarrollo

Dado que el desarrollo ha supuesto aproximadamente 700 horas de trabajo y que el salario de un informático *Junior* en España es de 13,85 € la hora, esto ascendería a un total de 9.695 € como coste de desarrollo.

- **Coste de desarrollo:** 9.695 €

2.6.2. Coste de Mantenimiento

El mantenimiento del sistema incluye la corrección de errores, la implementación de pequeñas mejoras y el soporte técnico. Por los costes de licencia de otras páginas de gestión veterinaria, se estima un coste anual de entre 4.800 € y 2.400 €. Para este presupuesto, intentaremos usar un umbral bajo-medio con respecto a las licencias, supongamos así, un valor anual de 3.000 €. A esto hay que añadirle el valor de tener una máquina virtual para desplegar el proyecto. Para esto, hemos escogido Google Cloud Platform, que nos supone un coste mensual de entre diez y quince euros. El primer mes ha sido un valor exacto de 13,63 €, lo que daría un valor anual de 165 € aproximadamente. A estos gastos hay que añadirle los costes de luz, internet y equipo, aunque no se han contado los gastos, podemos hacer una estimación basada en el uso real del equipo y los servicios.

- **Consumo estimado:**

- PC: 150 W

- Monitor: 30 W
- Router = 10 W
- Total = 190 W ($\sim 0,19$ kW)
- **Horas de uso diario:** ~ 5 horas
- **Días de trabajo anuales:** ~ 240 horas
- **Consumo total anual:** $0.19 \text{ kW} \times 5 \text{ h} \times 240 \text{ días} = 228 \text{ kWh}$
- **Coste anual (Suponiendo 0.20 €/kWh (Tarifa Luz Hora (2025)):** $228 \text{ kWh} \times 0.20 \text{ €/kWh} = 45,6 \text{ €}$.
- **Consumo de internet (suponiendo un uso del 20 % para el proyecto:** $480 \text{ €/año} \times 0,2 = 96 \text{ €}$
- **Coste del equipo:** Como ya teníamos un equipo, podemos aplicar una amortización suponiendo una vida útil de 7 años:
 - PC (770 €) + Monitor (150 €) + Periféricos (120 €) = 1040 €.
 - Vida útil: 7 años Coste anual = $148,50 \text{ €}$
- **Precio estimado total:** $45,60 + 96 + 148,50 = \sim 290 \text{ €/año}$
- **Coste de Mantenimiento Anual:** 3.455 €

2.6.3. Resumen de Costes

- **Coste total de desarrollo inicial:** 9.695 €
- **Coste total de mantenimiento (a un año):** 3.455 €
- **Coste total el primer año:** 13.150 €

Sin embargo, nos hemos reunido con el cliente para este apartado y hemos preguntado cuánto estarían de acuerdo en pagar por el proyecto, teniendo en cuenta factores como que el desarrollador es un *Junior* aún sin título, que son una empresa que acaba de empezar y la posibilidad de pagar un salario mensual que incluya los costes de mantenimiento.

- **Coste total de desarrollo inicial:** 4.500 €
- **Coste total de mantenimiento (a un año):** 1.500 €
- **Coste total del primer año:** 6.000 €

2.7. Gestión de la configuración

La gestión de configuración del proyecto se ha centrado en el uso de herramientas conocidas en la carrera para la gestión de versiones y la publicación de releases. Durante

el desarrollo del sistema se empleó Git como herramienta principal. Este sistema permitió mantener un historial claro y detallado de los cambios realizados, consiguiendo facilitar la identificación y corrección de errores, revirtiendo los cambios cuando fuera necesario. El repositorio se estructuró siguiendo una estrategia basada en ramas:

- **Rama principal (main):** contiene las versiones estables del proyecto y listas para ser puestas en producción.
- **Rama de desarrollo (dev):** se utilizó para integrar las funciones desarrolladas antes de ser fusionadas con la rama principal.
- **Ramas específicas de funcionalidades:** cada nueva funcionalidad tenía su rama para no interferir con el resto del proyecto.

Adicionalmente, el código se alojó en GitHub, herramienta escogida por su facilidad para trabajar en remoto, seguridad en el código fuente e integración con otras herramientas como VSCode y Jira. Finalmente, la periodicidad de nuevas *releases* fue ajustada según las necesidades del proyecto. Durante las etapas iniciales del desarrollo se realizaron lanzamientos frecuentes para facilitar la validación de requisitos. Tras la puesta en producción, las actualizaciones se centraban en grandes cambios orientados a la incorporación de nuevas funcionalidades.

2.8. Aseguramiento de calidad

El aseguramiento de la calidad en el desarrollo de este proyecto se ha basado en la implementación de prácticas y estándares reconocidos, todo con el objetivo de garantizar un ciclo de desarrollo iterativo y orientado a la calidad, minimizando riesgos y maximizando la satisfacción del cliente.

- **Documentación de requisitos.** Se elaboraron y validaron especificaciones detalladas de los requisitos funcionales y no funcionales.
- **Revisión de código y pruebas automatizadas.** Cada nueva versión en el repositorio llevaba asociada una revisión de errores, malas prácticas e inconsistencias además de un visto bueno por pruebas automatizadas para asegurar el correcto funcionamiento de componentes clave.
- **Control de versiones.** El uso de Git y GitHub hacía fácilmente accesible el control de versiones y la trazabilidad del código fuente en cada etapa del desarrollo.
- **Verificación y validación.** Incluyen revisiones técnicas, pruebas funcionales, de integración y de aceptación por parte del cliente para asegurar una retroalimentación y evolución ágil a través de prototipos, entornos controlados y control de conflictos e inconsistencias
- **Gestión de acciones correctoras y preventivas.** Cuando se identificaron problemas durante las pruebas o las revisiones, se registró el problema en Jira, se configuró según su prioridad, subtareas y estimación en tiempo y recursos y se implementó una solución.

Parte II

Desarrollo

3. Requisitos del Sistema

En este capítulo se presentan los objetivos y el catálogo de requisitos del nuevo sistema informático.

3.1. Objetivos de Negocio

El objetivo general que el sistema busca cumplir es incrementar la eficiencia operativa de la clínica y mejorar la experiencia de clientes y trabajadores mediante procesos ágiles y centralizados, así como fortalecer la posición del negocio en un entorno competitivo pero poco digitalizado. Los objetivos específicos de este proyecto son:

1. **Mejorar la eficiencia de recursos.** Reemplazar el sistema actual que se basa en notas a mano y hojas de cálculo por un sistema centralizado y automatizado, reduciendo el tiempo y los recursos necesarios para las tareas administrativas.
2. **Optimización de la programación de citas.** Reducir los problemas existentes en la organización de horarios y maximizar el tiempo útil de los trabajadores de la empresa.
3. **Mejorar la experiencia del cliente.** Facilitar la integración con el negocio a través de servicios digitales accesibles, como la gestión de citas y la información sobre el historial médico de las mascotas

3.2. Objetivos del Sistema

El sistema tiene como objetivo general proporcionar una solución tecnológica integral que facilite y automatice los procesos operativos de una clínica veterinaria. Este objetivo general se desglosa en otros objetivos específicos que marcan la dirección y el desarrollo del sistema.

1. **Digitalización de procesos operativos.** Automatizar tareas manuales como la gestión de citas, clientes y mascotas, la facturación y la generación de documentos oficiales.
2. **Accesibilidad y disponibilidad:** Permitir a los trabajadores acceder a la información en cualquier momento y desde cualquier dispositivo, promoviendo la visión de movilidad y agilidad de este negocio.
3. **Gestión eficiente de la información.** Centralizar y organizar la información clave, como el historial médico, el inventario de los productos y los registros financieros.
4. **Facilitar la generación de documentos personalizados.** Implementar funcionalidades para crear documentos oficiales de forma rápida y parametrizable.

5. **Optimización del uso de recursos humanos y materiales.** Mejorar la asignación de citas y el seguimiento de inventarios para evitar solapamientos, desabastecimientos y otros problemas operativos.
6. **Escalabilidad.** Diseñar el sistema para que pueda adaptarse a un crecimiento razonable del negocio y a nuevas necesidades que puedan surgir en el futuro.
7. **Seguridad y privacidad.** Proteger los datos sensibles de clientes y mascotas mediante el uso de buenas prácticas de desarrollo y estableciendo un sistema de inicio de sesión, encriptando las credenciales antes de introducirse en la base de datos. Estas credenciales deberán ser creadas y proporcionadas por el desarrollador, ya que el número de usuarios finales es muy limitado y no se espera que haya un aumento a corto o medio plazo.

3.3. Requisitos funcionales

El sistema desarrollado para este negocio deberá cumplir con los siguientes requisitos funcionales, organizados en categorías clave para mejor comprensión.

- **Gestión de Clientes y Mascotas.** Aparte de todos los métodos CRUD asociados a estas dos entidades, por la naturaleza del problema, hay algunas peculiaridades a tener en cuenta. Cada cliente (entiéndase cliente como “dueño” en este contexto) puede tener hasta tres contactos asociados, para asegurar que en cualquier momento puede contactarse con la mascota en caso de necesidad. A su vez, cualquier cliente puede tener asociadas un número N de mascotas, de las cuales se almacena información como su chip, especie, raza o peso. Las mascotas tendrán asociado además un historial médico incluyendo tratamientos, anteriores visitas y vacunaciones.
- **Gestión de citas.** Se deben programar citas asociadas a las mascotas especificando el cliente, motivo, fecha, hora, posibles pruebas médicas... Las citas existentes se podrán modificar o cancelar en cualquier momento.
- **Gestión de productos.** Es indispensable poder registrar y organizar los productos (tratamientos, consultas, vacaciones, etc.) con detalles como nombre, cantidad o fecha de caducidad. Es necesario además documentar el uso de estos a través del historial médico y recetas médicas personalizadas según las indicaciones del veterinario.
- **Facturación y pagos.** Crear facturas de manera ágil y parametrizable para cada cita completada, incluyendo desglose de costos de servicios y productos, además del método de pago.
- **Inicio de sesión.** Módulo de inicio de sesión a través de JWT destinado a los usuarios finales, para así garantizar la confidencialidad de los datos de los clientes y sus mascotas. Los nombres de usuario y contraseñas serán añadidas a mano por parte del desarrollador.

3.4. Requisitos no funcionales

- **Accesibilidad.** El sistema tiene que ser altamente accesible y estar diseñado para operar en dispositivos de distintos tamaños, escritorio y móvil principalmente, garantizando que los usuarios puedan interactuar con el sistema en cualquier lugar. Esto implica crear una interfaz intuitiva y adaptable, que pueda ajustarse a varios tamaños de pantalla sin afectar a la funcionalidad o experiencia de usuario.
- **Eficiencia y rendimiento.** Todas las actividades, desde el acceso y visualización de datos hasta la creación de facturas e informes, deben llevarse a cabo en plazos de respuesta razonables para no afectar a los procedimientos normales del usuario o alterar su experiencia; para ello establecemos un tiempo de respuesta de red menor a 200ms. De igual manera, el sistema tiene que ser capaz de afrontar de manera efectiva un aumento significativo del volumen de datos a medida que se expanda la clínica. La optimización y robustez se han inspirado por recomendaciones de optimización de React ([Facebook \(2025\)](#); [React Team \(2025b\)](#)) y en los principios de diseño de API robustas, tal como se detalla en [Richardson et al. \(2013\)](#).
- **Mantenibilidad.** El código debe estar bien documentado y estructurado para facilitar la incorporación de nuevas funcionalidades y la corrección de posibles futuros errores. Esto incluye el uso de tecnologías ampliamente soportadas y documentadas, y buenas prácticas de desarrollo para minimizar la dependencia de terceros.
- **Confiabilidad y disponibilidad.** El sistema debe ser confiable y accesible en todo momento, reduciendo al mínimo los periodos de inactividad. Esto implica implementar estrategias como respaldos automáticos y versionado de código para poder revertir cualquier cambio hecho.

3.5. Requisitos de información

El sistema debe gestionar la información necesaria para cumplir con las operaciones diarias del negocio. Esto incluye los siguientes datos.

1. Gestión de clientes y mascotas:

- Información de los clientes, principalmente datos personales, estado de actividad, observaciones, contactos.
- Relación cliente-contacto: asociar un máximo de tres contactos a un cliente, teniendo cada uno de ellos una dirección y un teléfono.
- Información de las mascotas como datos identificativos, historial médico, observaciones clínicas o datos de vacunación.
- Relación cliente-mascota: asociar cada mascota a un cliente y permitir múltiples mascotas por cliente.

- Planes anuales: las mascotas suscritas a planes anuales deben tener una distinción para recordarle al veterinario que debe aplicar un descuento.
2. Gestión de citas: registrar las citas con detalles como la fecha, el cliente, la mascota, el veterinario asignado y los servicios solicitados.
 3. Facturación y pagos: registro completo de los datos de la factura, como el tipo de pago, el pago total, los artículos facturados, los descuentos e impuestos aplicados y la fecha.
 4. Gestión de usuarios: datos personales, credenciales de acceso y estado de actividad.

El diseño del sistema debe asegurar la consistencia y, más importante incluso, la trazabilidad de los datos. Debemos prestar especial atención a las relaciones entre diferentes entidades para asegurar que el sistema se construye sobre una base de datos robusta y que cambie lo menos posible a lo largo de la vida del negocio.

4. Análisis del Sistema

Este capítulo cubre el análisis del sistema de información a desarrollar, haciendo uso del lenguaje de modelado UML [Object Management Group \(2017\)](#). UML es un lenguaje gráfico que se utiliza para visualizar, especificar, construir y documentar un sistema.

4.1. Modelo Conceptual de Datos

El modelo conceptual ([Figura 4.1](#)) nos servirá para identificar cómo interactúa el sistema en el cual se desenvuelve la solución. Con la ayuda de este diagrama, cualquier usuario puede comprender cómo interactúan los datos con el sistema. Brevemente, los conceptos se definen como:

- **Usuarios.** Usuarios principales y únicos del sistema (veterinarios).
- **Clientes.** Clientes de la clínica veterinaria.
- **Contactos.** Contactos asociados a los clientes de la clínica.
- **Mascotas.** Mascotas que pertenecen a los clientes.
- **Citas.** Visitas médicas, desde pruebas hasta vacunaciones u operaciones hechas por un veterinario a una mascota.
- **Facturas:** Recibos de los pagos asociados a las citas.
- **Artículos.** Artículos físicos (vacunas, antibióticos, pienso) y servicios (consulta, cita, ecografía), que conllevan un coste.

4.2. Modelo de Casos de Uso

En el siguiente modelo ([Figura 4.2](#)) presentamos una vista general de todas las acciones que el usuario puede tener con el sistema para satisfacer un propósito.

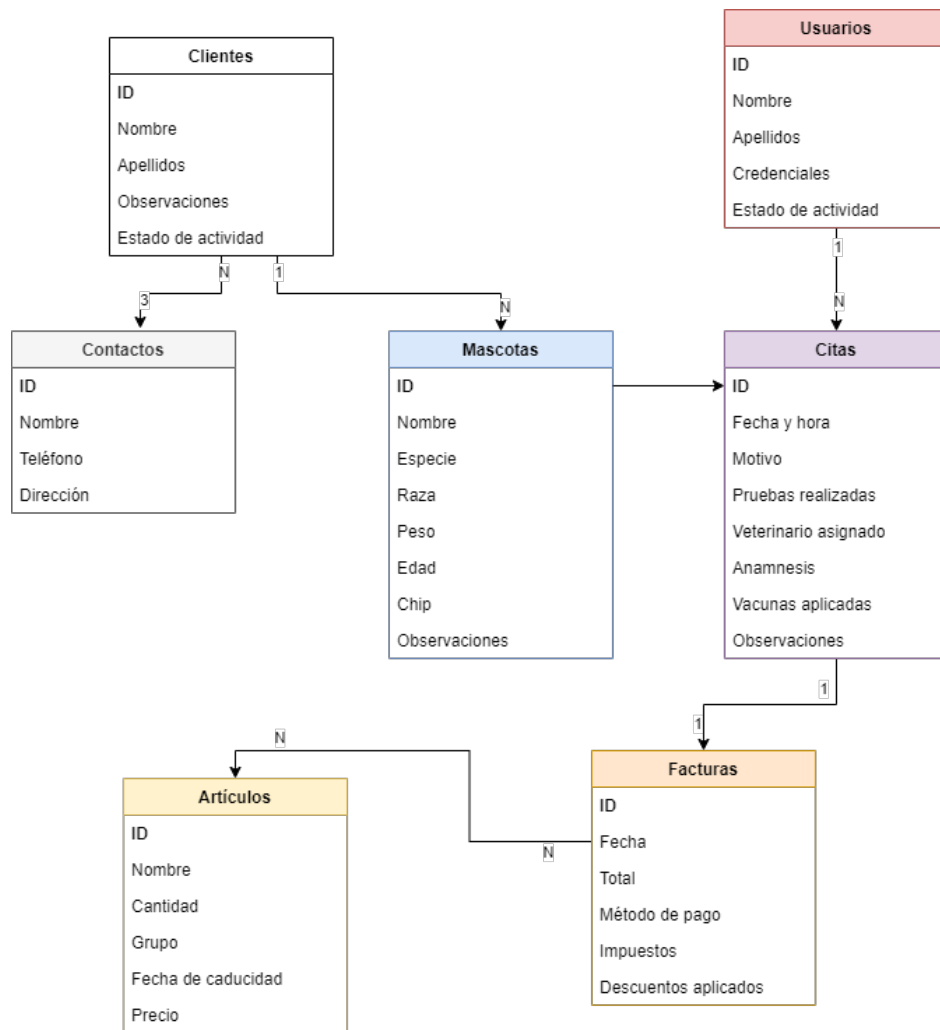


Figura 4.1: Modelo conceptual de datos

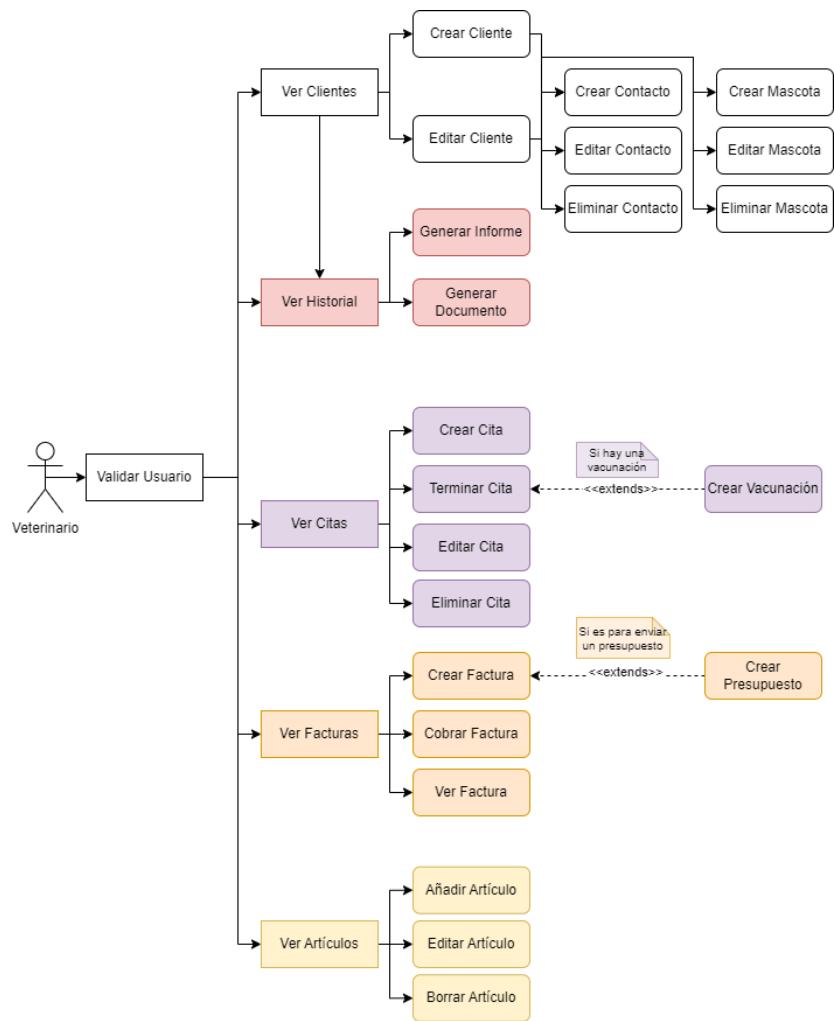


Figura 4.2: Modelo de Casos de Uso

4.2.1. Casos de Uso

En el siguiente apartado describiremos todos los modelos de Casos de Uso para los requisitos funcionales que se han descrito anteriormente. Es importante aclarar que, al ser un sistema de gestión, el único actor es el usuario “veterinario” y por tanto lo obviaremos en nuestras descripciones. Como podemos observar en la Figura 4.2, todos los casos de uso dependen de “Validar Usuario”, por lo que este caso debe considerarse como precondition en todos y cada uno de los casos de uso. Como hemos mencionado anteriormente, el sistema carece de casos de uso asociados a la gestión de usuarios, como “Crear Usuario” o “Recuperar Contraseña” esto se debe porque estas gestiones son responsabilidad única del desarrollador; los usuarios finales son muy limitados y no se espera crecimiento de la empresa a corto o medio plazo.

Caso de Uso: Validar Usuario

Descripción: Un usuario intenta autenticarse en el sistema.

Precondiciones: El sistema debe estar en funcionamiento.

Postcondiciones: El sistema valida al usuario.

Identificación de escenarios:

■ **Escenario Principal:**

1. Un usuario decide validarse en el sistema.
2. El usuario introduce su correo electrónico y contraseña.
3. El sistema valida las credenciales y navega a la página de “Inicio”.

■ **Escenarios alternativos:**

- 3a. Las credenciales del usuario son incorrectas y no se le permite acceso al sistema.

Caso de Uso: Ver Clientes

Descripción: El usuario del sistema consulta el listado de clientes registrados en la aplicación.

Precondiciones: El sistema debe estar en funcionamiento.

Postcondiciones: El sistema muestra la lista de clientes.

Identificación de escenarios:

■ **Escenario Principal:**

1. El Usuario decide consultar la lista de clientes registrados en la aplicación.
2. El usuario hace clic en la pestaña “Clientes”.
3. El sistema muestra la lista de los clientes con sus datos y acciones que tomar.

■ **Escenarios alternativos:**

- 3a. El usuario puede filtrar la lista de clientes por nombre, mascota, NHC o chip.

Caso de Uso: Crear Cliente

Descripción: El usuario del sistema introduce un nuevo cliente en la aplicación.

Precondiciones: El sistema debe estar en funcionamiento.

Postcondiciones: El sistema almacena al nuevo cliente.

Identificación de escenarios:

■ **Escenario Principal:**

1. El usuario decide añadir un nuevo cliente.
2. El usuario hace *clic* en la acción “Añadir Cliente”.
3. El usuario introduce los datos del cliente en el sistema (Nombre, Documento, Dirección, Ciudad, CP, Provincia, Observaciones).
4. El usuario pulsa el botón de “Guardar”.

■ **Escenarios Alternativos:**

- 4.a. El usuario introduce campos incorrectos, o deja algún campo clave en blanco.
 - 4.a.1. El sistema muestra el error y da la oportunidad de modificar los datos.
 - 4.a.2. El sistema vuelve al punto 3.
- *a. El usuario puede cancelar la creación del nuevo cliente en cualquier momento.

Caso de Uso: Editar Cliente

Descripción: El usuario del sistema edita los datos de un cliente existente en la aplicación.

Precondiciones: El sistema debe estar en funcionamiento y debe existir al menos un cliente.

Postcondiciones: El sistema actualiza los datos del cliente.

Identificación de escenarios:

■ **Escenario Principal:**

1. El usuario decide editar un cliente.
2. El usuario selecciona un cliente de la lista.

3. El usuario hace *clic* en la acción “Editar Cliente”.
4. El usuario modifica los datos del cliente en el sistema.
5. El usuario pulsa el botón de “Guardar”.

■ **Escenarios Alternativos:**

- 4.a. El usuario introduce campos incorrectos, o deja algún campo clave en blanco.
 - 4.a.1. El sistema muestra el error y da la oportunidad de modificar los datos.
 - 4.a.2. El sistema vuelve al punto 4.
- *a. El usuario puede cancelar la edición del cliente en cualquier momento.

Caso de Uso: Crear Mascota

Descripción: El usuario del sistema introduce una nueva mascota para un cliente en la aplicación.

Precondiciones: El sistema debe estar en funcionamiento y debe existir al menos un cliente.

Postcondiciones: El sistema almacena la nueva mascota.

Identificación de escenarios:

■ **Escenario Principal:**

1. El usuario decide añadir una nueva mascota para un cliente.
2. El usuario selecciona un cliente de la lista.
3. El usuario hace *clic* en la acción “Añadir Mascota”.
4. El usuario introduce los datos de la mascota en el sistema (Nombre, Fecha de Nacimiento, Chip, Especie, Raza, Sexo, Esterilizado, Observaciones).
5. El usuario pulsa el botón de “Guardar”.

■ **Escenarios Alternativos:**

- 4.a. El usuario introduce campos incorrectos, o deja algún campo clave en blanco.
 - 4.a.1. El sistema muestra el error y da la oportunidad de modificar los datos.
 - 4.a.2. El sistema vuelve al punto 4.
- *a. El usuario puede cancelar la creación de la nueva mascota en cualquier momento.

Caso de Uso: Editar Mascota

Descripción: El usuario del sistema edita los datos de una mascota existente en la aplicación.

Precondiciones: El sistema debe estar en funcionamiento y debe existir al menos una mascota que pertenece a ese cliente.

Postcondiciones: El sistema actualiza los datos de la mascota.

Identificación de escenarios:

■ Escenario Principal:

1. El usuario decide editar una mascota.
2. El usuario selecciona un cliente de la lista.
3. El usuario selecciona una mascota del cliente.
4. El usuario hace *clic* en la acción “Editar Mascota”.
5. El usuario modifica los datos de la mascota en el sistema.
6. El usuario pulsa el botón de “Guardar”.

■ Escenarios Alternativos:

- 5.a. El usuario introduce campos incorrectos, o deja algún campo clave en blanco.
 - 5.a.1. El sistema muestra el error y da la oportunidad de modificar los datos.
 - 5.a.2. El sistema vuelve al punto 5.
- *a. El usuario puede cancelar la edición de la mascota en cualquier momento.

Caso de Uso: Eliminar Mascota

Descripción: El usuario del sistema elimina una mascota existente en la aplicación.

Precondiciones: El sistema debe estar en funcionamiento y debe existir al menos una mascota que pertenece a ese cliente.

Postcondiciones: El sistema elimina la mascota.

Identificación de escenarios:

■ Escenario Principal:

1. El usuario decide eliminar una mascota.
2. El usuario selecciona un cliente de la lista.
3. El usuario selecciona una mascota del cliente.

4. El usuario hace *clic* en la acción “Eliminar Mascota”.
5. El sistema solicita confirmación para eliminar la mascota.
6. El usuario confirma la eliminación.
7. El sistema elimina la mascota.

■ **Escenarios Alternativos:**

- *a. El usuario puede cancelar la eliminación de la mascota en cualquier momento.

Caso de Uso: Crear Contacto

Descripción: El usuario del sistema introduce un nuevo contacto para un cliente en la aplicación.

Precondiciones: El sistema debe estar en funcionamiento y debe existir al menos un cliente.

Postcondiciones: El sistema almacena el nuevo contacto.

Identificación de escenarios:

■ **Escenario Principal:**

1. El usuario decide añadir un nuevo contacto para un cliente.
2. El usuario selecciona un cliente de la lista.
3. El usuario hace *clic* en la acción “Añadir Contacto”.
4. El usuario introduce los datos del contacto en el sistema (Nombre, Teléfono, Correo).
5. El usuario pulsa el botón de “Guardar”.

■ **Escenarios Alternativos:**

- 3.a. El cliente tiene ya tres contactos asociados.
 - 3.a.1 Se deshabilita el botón de crear contacto.
- 4.a. El usuario introduce campos incorrectos, o deja algún campo clave en blanco.
 - 4.a.1. El sistema muestra el error y da la oportunidad de modificar los datos.
 - 4.a.2. El sistema vuelve al punto 4.
- *a. El usuario puede cancelar la creación del nuevo contacto en cualquier momento.

Caso de Uso: Editar Contacto

Descripción: El usuario del sistema edita los datos de un contacto existente en la aplicación.

Precondiciones: El sistema debe estar en funcionamiento y debe existir al menos un contacto para el cliente asociado.

Postcondiciones: El sistema actualiza los datos del contacto.

Identificación de escenarios:

■ Escenario Principal:

1. El usuario decide editar un contacto.
2. El usuario selecciona un cliente de la lista.
3. El usuario selecciona un contacto del cliente.
4. El usuario hace *clic* en la acción “Editar Contacto”.
5. El usuario modifica los datos del contacto en el sistema.
6. El usuario pulsa el botón de “Guardar”.

■ Escenarios Alternativos:

- 5.a. El usuario introduce campos incorrectos, o deja algún campo clave en blanco.
 - 5.a.1. El sistema muestra el error y da la oportunidad de modificar los datos.
 - 5.a.2. El sistema vuelve al punto 5.
- *a. El usuario puede cancelar la edición del contacto en cualquier momento.

Caso de Uso: Eliminar Contacto

Descripción: El usuario del sistema elimina un contacto existente en la aplicación.

Precondiciones: El sistema debe estar en funcionamiento y existir al menos un contacto para el cliente asociado.

Postcondiciones: El sistema elimina el contacto.

Identificación de escenarios:

■ Escenario Principal:

1. El usuario decide eliminar un contacto.
2. El usuario selecciona un cliente de la lista.
3. El usuario selecciona un contacto del cliente.

4. El usuario hace *clic* en la acción “Eliminar Contacto”.
5. El sistema solicita confirmación para eliminar el contacto.
6. El usuario confirma la eliminación.
7. El sistema elimina el contacto.

■ **Escenarios Alternativos:**

- *a. El usuario puede cancelar la eliminación del contacto en cualquier momento.

Caso de Uso: Ver Historial

Descripción: El usuario del sistema consulta el historial de un cliente en la aplicación.

Precondiciones: El sistema debe estar en funcionamiento y debe existir al menos un cliente en el sistema.

Postcondiciones: El sistema muestra el historial del cliente.

Identificación de escenarios:

■ **Escenario Principal:**

1. El Caso de Uso se inicia cuando el usuario decide consultar el historial de un cliente.
2. El usuario selecciona un cliente de la lista.
3. El usuario hace *clic* en la acción “Ver Historial”.
4. El sistema muestra el historial del cliente.

Caso de Uso: Generar Informe

Descripción: El usuario del sistema genera un informe del historial de un cliente en la aplicación.

Precondiciones: El sistema debe estar en funcionamiento.

Postcondiciones: El sistema genera el informe del historial del cliente.

Identificación de escenarios:

■ **Escenario Principal:**

1. El usuario decide generar un informe del historial de un cliente.
2. El usuario selecciona un cliente de la lista.
3. El usuario hace *clic* en la acción “Generar Informe”.
4. El sistema genera el informe del historial del cliente.

5. El usuario hace *clic* en la acción “Descargar Informe”.
6. El sistema descarga el informe mediante el navegador del usuario.

Caso de Uso: Generar Documento Médico Oficial

Descripción: El usuario del sistema genera un documento para un cliente en la aplicación.

Precondiciones: El sistema debe estar en funcionamiento.

Postcondiciones: El sistema genera el documento para el cliente.

Identificación de escenarios:

■ **Escenario Principal:**

1. El usuario decide generar un documento para un cliente.
2. El usuario selecciona un cliente de la lista.
3. El usuario hace *clic* en la acción “Generar Documento”.
4. El usuario selecciona el tipo de documento a generar.
5. El usuario modifica los parámetros necesarios para el documento.
6. El usuario hace *clic* en la acción “Generar”.
7. El sistema genera el documento para el cliente.

Ver Citas

Descripción: El usuario del sistema consulta la lista de citas registradas en la aplicación.

Precondiciones: El sistema debe tener citas registradas.

Postcondiciones: El sistema muestra la lista de citas.

Identificación de Escenarios:

■ **Escenario Principal:**

1. El usuario decide consultar las citas.
2. El usuario hace *clic* en la pestaña “Citas”.
3. El sistema muestra la lista de citas con sus datos.

■ **Escenarios Alternativos:**

- 3a. El usuario puede filtrar los datos por cliente, mascota, NHC o chip.

Crear Cita

Descripción: El usuario del sistema introduce una nueva cita en la aplicación.

Precondiciones: El sistema debe tener clientes y mascotas registradas.

Postcondiciones: El sistema almacena la nueva cita.

Identificación de Escenarios:

■ Escenario Principal:

1. El usuario decide añadir una nueva cita.
2. El usuario hace *clic* en la acción “Añadir Cita”.
3. El usuario introduce los datos de la cita en el sistema (Cliente, Mascota, Fecha, Veterinario, Motivo de la consulta).
4. El usuario pulsa el botón de “Guardar”.

■ Escenarios Alternativos:

- 4.a. El usuario introduce campos incorrectos, o deja algún campo clave en blanco.
 - 4.a.1. El sistema muestra el error y da la oportunidad de modificar los datos.
 - 4.a.2. El sistema vuelve al punto 3.
- *a. El usuario puede cancelar la creación de la nueva cita en cualquier momento.

Terminar Cita

Descripción: El usuario del sistema marca una cita como completada.

Precondiciones: El sistema debe tener citas registradas.

Postcondiciones: El sistema actualiza el estado de la cita a completada.

Identificación de Escenarios:

■ Escenario Principal:

1. El usuario decide terminar una cita.
2. El usuario selecciona una cita de la lista.
3. El usuario hace *clic* en la acción “Terminar Cita”.
4. El sistema actualiza el estado de la cita a completada.

■ Escenarios Alternativos:

- 4.a. El usuario introduce campos incorrectos, o deja algún campo clave en blanco.

- 4.a.1. El sistema muestra el error y da la oportunidad de modificar los datos.
 - 4.a.2. El sistema vuelve al punto 4.
- 4.b. El usuario introduce que se ha administrado una vacuna.
 - 4.b.1 El usuario elige las vacunas administradas de un desplegable
 - 4.b.2 El sistema crea automáticamente una cita para renovar la vacunación a esa mascota.
 - 4.b.* El usuario puede cancelar la creación de la nueva vacunación en cualquier momento.
- *a. El usuario puede cancelar la finalización de la cita en cualquier momento.

Editar Cita

Descripción: El usuario del sistema edita los datos de una cita existente en la aplicación.

Precondiciones: El sistema debe tener citas registradas.

Postcondiciones: El sistema actualiza los datos de la cita.

Identificación de Escenarios:

■ Escenario Principal:

1. El usuario decide editar una cita.
2. El usuario selecciona una cita de la lista.
3. El usuario hace *clic* en la acción “Editar Cita”.
4. El usuario modifica los datos de la cita en el sistema.
5. El usuario pulsa el botón de “Guardar”.

■ Escenarios Alternativos:

- 4.a. El usuario introduce campos incorrectos, o deja algún campo clave en blanco.
 - 4.a.1. El sistema muestra el error y da la oportunidad de modificar los datos.
 - 4.a.2. El sistema vuelve al punto 4.
- *a. El usuario puede cancelar la edición de la cita en cualquier momento.

Eliminar Cita

Descripción: El usuario del sistema elimina una cita existente en la aplicación.

Precondiciones: El sistema debe tener citas registradas.

Postcondiciones: El sistema elimina la cita.

Identificación de Escenarios:

■ **Escenario Principal:**

1. El usuario decide eliminar una cita.
2. El usuario selecciona una cita de la lista.
3. El usuario hace *clic* en la acción “Eliminar Cita”.
4. El sistema solicita confirmación para eliminar la cita.
5. El usuario confirma la eliminación.
6. El sistema elimina la cita.

■ **Escenarios Alternativos:**

- *a. El usuario puede cancelar la eliminación de la cita en cualquier momento.

Ver Facturas

Descripción: El usuario del sistema consulta la lista de facturas registradas en la aplicación.

Precondiciones: El sistema debe tener facturas registradas.

Postcondiciones: El sistema muestra la lista de facturas.

Identificación de Escenarios:

■ **Escenario Principal:**

1. El Caso de Uso se inicia cuando el usuario decide consultar las facturas.
2. El usuario hace *clic* en la pestaña “Facturas”.
3. El sistema muestra la lista de facturas con sus datos.

■ **Escenarios Alternativos:**

- 3a. El usuario puede filtrar las facturas por nombre, mascota, DNI, NHC o chip.

Crear Factura

Descripción: El usuario del sistema introduce una nueva factura en la aplicación.

Precondiciones: El sistema debe tener clientes y artículos registrados.

Postcondiciones: El sistema almacena la nueva factura.

Identificación de Escenarios:

■ **Escenario Principal:**

1. El usuario decide añadir una nueva factura.
2. El usuario hace *clic* en la acción “Añadir Factura”.
3. El usuario introduce los datos de la factura en el sistema (Cliente, Artículos, Cantidad, Precio, IVA, Descuento).
4. El usuario pulsa el botón de “Guardar”.
5. Se crea una nueva factura con un número consecutivo importante para la tributación.

■ **Escenarios Alternativos:**

- 4.a. El usuario introduce campos incorrectos, o deja algún campo clave en blanco.
 - 4.a.1. El sistema muestra el error y da la oportunidad de modificar los datos.
 - 4.a.2. El sistema vuelve al punto 3.
- 4.b. El usuario marca la opción “presupuesto”.
 - 4.b.1. El usuario pulsa el botón “Guardar”.
 - 4.b.2. El sistema genera una factura sin número.
- *a. El usuario puede cancelar la creación de la nueva factura en cualquier momento.

Cobrar Factura

Descripción: El usuario del sistema marca una factura como cobrada.

Precondiciones: El sistema debe tener facturas registradas.

Postcondiciones: El sistema actualiza el estado de la factura a cobrada.

Identificación de Escenarios:

■ **Escenario Principal:**

1. El usuario decide cobrar una factura.
2. El usuario selecciona una factura de la lista.
3. El usuario hace *clic* en la acción “Cobrar Factura”.
4. El usuario selecciona el método de pago.
5. El sistema actualiza el estado de la factura a cobrada.

■ **Escenarios Alternativos:**

- *a. El usuario puede cancelar la creación de la nueva factura en cualquier momento.

Ver Factura

Descripción: El usuario del sistema consulta los detalles de una factura registrada en la aplicación.

Precondiciones: El sistema debe tener facturas registradas.

Postcondiciones: El sistema muestra los detalles de la factura.

Identificación de Escenarios:

■ Escenario Principal:

1. El Caso de Uso se inicia cuando el usuario decide consultar una factura.
2. El usuario selecciona una factura de la lista.
3. El usuario hace *clic* en la acción “Ver Factura”.
4. El sistema muestra los detalles de la factura.

Ver Artículos

Descripción: El usuario del sistema consulta la lista de artículos registrados en la aplicación.

Precondiciones: El sistema debe tener artículos registrados.

Postcondiciones: El sistema muestra la lista de artículos. **Identificación de Escenarios:**

■ Escenario Principal:

1. El Caso de Uso se inicia cuando el usuario decide consultar los artículos.
2. El usuario hace *clic* en la pestaña “Artículos”.
3. El sistema muestra la lista de artículos con sus datos.

■ Escenarios Alternativos:

- 3a. El usuario puede filtrar los artículos por nombre o grupo.

Añadir Artículo

Descripción: El usuario del sistema introduce un nuevo artículo en la aplicación.

Precondiciones: El sistema debe estar en funcionamiento.

Postcondiciones: El sistema almacena el nuevo artículo.

Identificación de Escenarios:

■ Escenario Principal:

1. El usuario decide añadir un nuevo artículo.

2. El usuario hace *clic* en la acción “Añadir Artículo”.
3. El usuario introduce los datos del artículo en el sistema (Nombre, PVP, Stock, Validez).
4. El usuario escoge un grupo de artículos de un desplegable.
5. El usuario pulsa el botón de “Guardar”.

■ **Escenarios Alternativos:**

- 4.a. El usuario introduce campos incorrectos, o deja algún campo clave en blanco.
 - 4.a.1. El sistema muestra el error y da la oportunidad de modificar los datos.
 - 4.a.2. El sistema vuelve al punto 3.
- 5.a. El usuario introduce a mano un nuevo grupo
 - 5.a.1 Se almacena un nuevo grupo en el sistema.
- *a. El usuario puede cancelar la creación del nuevo artículo en cualquier momento.

Editar Artículo

Descripción: El usuario del sistema edita los datos de un artículo existente en la aplicación.

Precondiciones: El sistema debe tener artículos registrados.

Postcondiciones: El sistema actualiza los datos del artículo.

Identificación de Escenarios:

■ **Escenario Principal:**

1. El usuario decide editar un artículo.
2. El usuario selecciona un artículo de la lista.
3. El usuario hace *clic* en la acción “Editar Artículo”.
4. El usuario modifica los datos del artículo en el sistema.
5. El usuario pulsa el botón de “Guardar”.

■ **Escenarios Alternativos:**

- 4.a. El usuario introduce campos incorrectos, o deja algún campo clave en blanco.
 - 4.a.1. El sistema muestra el error y da la oportunidad de modificar los datos.
 - 4.a.2. El sistema vuelve al punto 4.

- *a. El usuario puede cancelar la edición del artículo en cualquier momento.

Eliminar Artículo

Descripción: El usuario del sistema elimina un artículo existente en la aplicación.

Precondiciones: El sistema debe tener artículos registrados.

Postcondiciones: El sistema elimina el artículo.

Identificación de Escenarios:

■ **Escenario Principal:**

1. El usuario decide eliminar un artículo.
2. El usuario selecciona un artículo de la lista.
3. El usuario hace *clic* en la acción “Eliminar Artículo”.
4. El sistema solicita confirmación para eliminar el artículo.
5. El usuario confirma la eliminación.
6. El sistema elimina el artículo.

■ **Escenarios Alternativos:**

- *a. El usuario puede cancelar la eliminación del artículo en cualquier momento.

4.3. Modelo de Interfaz de Usuario

El reto a superar es crear el menor número de pantallas posibles sin prescindir de la legibilidad y facilidad de uso que queremos que tenga cualquier sistema. Gracias a React, sabemos que no vamos a tener problema cargando y descargando las páginas, así que la mayoría de gestiones se realizarán con modales. Esto crea una capa más de dificultad al desarrollo de la interfaz porque debemos diseñar los modales para que sean informativos y funcionales, pero sin saturar al usuario con datos o acciones innecesarias.

Inicialmente, se planificaron, además de la pantalla de Inicio de sesión, cinco secciones de navegación: “Inicio”, que funcionaría como dashboard, y “Clientes”, “Citas”, “Facturas” y “Artículos”, cada una destinada a la consulta y gestión de los elementos principales del sistema. Sin embargo, tras la primera iteración, el cliente decidió fusionar las vistas de “Facturas” y “Visitas” y modificar la visión original del dashboard, incorporando tablas que muestren de forma clara las citas programadas para el día y las futuras.

Además, debemos tener en cuenta para el desarrollo la experiencia en pantallas pequeñas, porque, como hemos señalado anteriormente, una gran parte de las gestiones se llevará a cabo en un dispositivo móvil.

A continuación, presentamos una simplificación de esta interfaz ([Figura 4.3](#)).

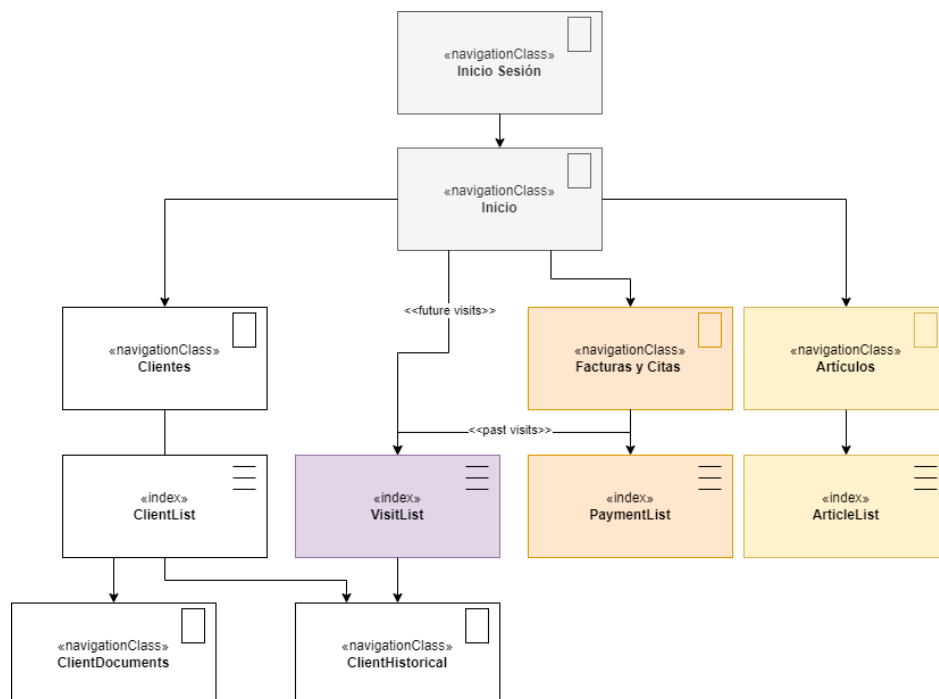


Figura 4.3: Diagrama de navegación

5. Diseño del Sistema

En este capítulo se recoge el diseño de la arquitectura, datos e interfaz del sistema informático, así como el entorno de construcción y la organización del código fuente.

5.1. Base de Datos

La funcionalidad principal de nuestro proyecto es poder consultar, insertar, editar y eliminar información. Esto nos crea la necesidad de tener una base de datos donde poder almacenar la información y trabajar con ella. Para ello, usaremos una base de datos MySQL almacenada en la misma VM que contiene la aplicación y la API, de forma que podamos acceder a ella fácilmente.

5.1.1. Diseño de Datos

En el análisis del proyecto vimos qué clases necesitábamos. Ahora que debemos construir la base de datos, es el momento de buscar fallos en el sistema y hacer las modificaciones necesarias para cumplir con las necesidades del negocio. Al comienzo de este proyecto, el esquema de la BD se ajustaba fielmente al modelo de clases que habíamos establecido en primera instancia; sin embargo, las mejoras que el cliente pedía a lo largo de la vida del proyecto obligaban a añadir nuevas clases y restricciones.

A continuación presentamos una vista esquemática de la BD ([Figura 5.1](#)). Seguidamente, aclararemos todos los campos de cada tabla y sus restricciones. Para evitar sobrecargar el proyecto, la clave primaria de todas las tablas se toma como implícita.

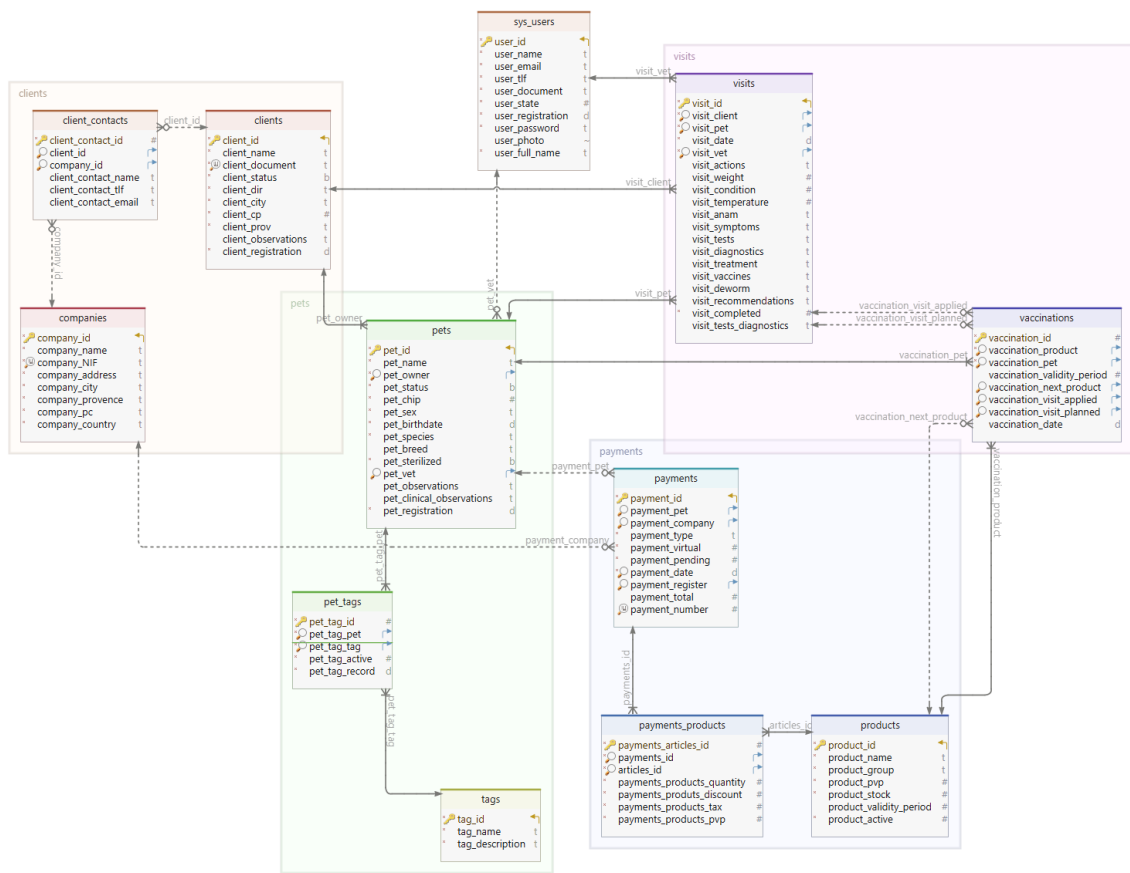


Figura 5.1: Representación gráfica de la BD

Tabla: Clients

Descripción: Clientes de la clínica veterinaria.

Tabla 5.1

Tabla clients

Idx	Column Name	Definition
*	client_id	INT AUTO_INCREMENT
	client_name	VARCHAR(45)
	client_document	VARCHAR(15)
	client_status	TINYINT(1)
	client_dir	VARCHAR(100)
	client_city	VARCHAR(45)
	client_cp	INT
	client_prov	VARCHAR(45)
	client_observations	MEDIUMTEXT
	client_registration	DATETIME DEFAULT CURRENT_TIMESTAMP

Tabla 5.2

Índices de la tabla clients

Nombre del Índice	Tipo	Columnas
idClients_UNIQUE	Unique Key	client_id
client_document_UNIQUE	Unique Key	client_document

Tabla: Companies

Descripción: Alternativa a clientes, compañías a las que se les hacen ventas de material o servicios.

Tabla 5.3

Tabla companies

Idx	Column Name	Definition
*	company_id	INT AUTO_INCREMENT
	company_name	VARCHAR(45)
	company_NIF	VARCHAR(45)
	company_address	VARCHAR(105)
	company_city	VARCHAR(45)
	company_provence	VARCHAR(45)
	company_pc	VARCHAR(45)
	company_country	VARCHAR(45)

Tabla: Client_contacts

Descripción: Contactos asociados a los clientes.

Tabla 5.4

Tabla client_contacts

Idx	Column Name	Definition
*	client_contact_id	INT AUTO_INCREMENT
	client_id	INT
	company_id	INT
	client_contact_name	VARCHAR(45)
	client_contact_tlf	VARCHAR(20)
	client_contact_email	VARCHAR(45)

Tabla 5.5

Índices de la tabla client_contacts

Nombre del Índice	Tipo	Columnas
client_contact_id_UNIQUE	Unique Key	client_contact_id
contact_client_FK_idx	Index	client_id
contact_company_FK_idx	Index	company_id

Tabla 5.6

Claves Foráneas de la tabla client_contacts

Nombre de la Clave Foránea	Referencia
contact_client_FK	client_id clients.client_id
contact_company_FK	company_id company.company_id

Tabla Sys.users

Descripción: Usuarios principales y únicos del sistema (veterinarios).

Tabla 5.7

Tabla sys_users

Idx	Column Name	Definition
*	user_id	INT AUTO_INCREMENT
	user_name	VARCHAR(45)
	user_email	VARCHAR(45)
	user_tlf	VARCHAR(15)
	user_document	VARCHAR(15)
	user_state	TINYINT DEFAULT 1
	user_registration	DATETIME DEFAULT (now())
	user_password	VARCHAR(255)
	user_photo	BLOB
	user_full_name	VARCHAR(45) DEFAULT '-'

Tabla 5.8

Índices de la tabla sys_users

Nombre del Índice	Tipo	Columnas
user_id_UNIQUE	Unique Key	user_id

Tabla: Products

Descripción: Artículos físicos y servicios que se venden a los clientes.

Tabla 5.9

Tabla products

Idx	Column Name	Definition
*	product_id	INT AUTO_INCREMENT
	product_name	VARCHAR(45)
	product_group	VARCHAR(45)
	product_pvp	FLOAT
	product_stock	INT
	product_validity_period	INT DEFAULT 365

Tabla: Payments

Tabla 5.10

Tabla payments

Idx	Column Name	Definition
*	payment_id	INT AUTO_INCREMENT
	payment_pet	INT
	payment_company	INT
	payment_type	ENUM('Efectivo','Tarjeta','Bizum','Transferencia')
	payment_pending	TINYINT DEFAULT 0
	payment_date	DATETIME
	payment_register	INT
	payment_total	FLOAT DEFAULT 0
	payment_number	INT

Tabla 5.11

Índices de la tabla payments

Nombre del Índice	Tipo	Columnas
payment_id_UNIQUE	Unique Key	payment_id
payment_number_UNIQUE	Unique Key	payment_number
payment_register_FK_idx	Index	payment_register
payment_pet_FK	Index	payment_pet
payment_company_FK	Index	payment_company

Tabla 5.12

Claves Foráneas de la tabla payments

Nombre de la Clave Foránea	Referencia
payment_company_FK	payment_company companies.company_id
payment_pet_FK	payment_pet pets.pet_id

Tabla: Payment_articles

Descripción: Tabla de asociación para conectar las facturas y los artículos vendidos en esa factura.

Tabla 5.13

Tabla payments_articles

Idx	Column Name	Definition
*	payments_articles_id	INT AUTO.INCREMENT
	payments_id	INT
	articles_id	INT
	payments_articles_quantity	INT DEFAULT 1
	payments_articles_discount	FLOAT DEFAULT 0
	payments_articles_tax	FLOAT DEFAULT 21
	payments_articles_pvp	FLOAT DEFAULT 0

Tabla 5.14

Índices de la tabla payments_articles

Nombre del Índice	Tipo	Columnas
payments_articles_payments_FK_idx	Index	payments_id
payments_articles_articles_FK_idx	Index	articles_id

Tabla 5.15

Claves Foráneas de la tabla payments_articles

Nombre de la Clave Foránea	Referencia
payments_articles_articles_FK	articles_id articles.article_id
payments_articles_payments_FK	payments_id payments.payment_id

Tabla: Pets

Descripción: Mascotas de los clientes.

Tabla 5.16

Tabla pets

Idx	Column Name	Definition
*	pet_id	INT AUTO_INCREMENT
	pet_name	VARCHAR(15)
	pet_owner	INT
	pet_status	TINYINT(1) DEFAULT 1
	pet_chip	BIGINT
	pet_sex	ENUM('M','F','O')
	pet_birthdate	DATE
	pet_species	VARCHAR(15)
	pet_breed	VARCHAR(45)
	pet_sterilized	TINYINT(1)
	pet_vet	INT
	pet_observations	MEDIUMTEXT
	pet_clinical_observations	MEDIUMTEXT
	pet_registration	DATETIME DEFAULT CURRENT_TIMESTAMP

Tabla 5.17

Índices de la tabla pets

Nombre del Índice	Tipo	Columnas
pets_id_UNIQUE	Unique Key	pet_id
pet_owner_FK	Index	pet_owner
pet_vet_FK	Index	pet_vet

Tabla 5.18

Claves Foráneas de la tabla pets

Nombre de la Clave Foránea	Referencia
pet_owner_FK	pet_owner clients
pet_vet_FK	pet_vet sys_users

Tabla: Tags

Descripción: Distintos planes anuales a los que pueden ajustarse las mascotas.

Tabla 5.19

Tabla tags

Idx	Column Name	Definition
*	tag_id	INT AUTO_INCREMENT
	tag_name	VARCHAR(45)
	tag_description	VARCHAR(45)

Tabla: Pet_tags

Descripción: Tabla que indica si una mascota tiene un plan asociado.

Tabla 5.20

Tabla pet_tags

Idx	Column Name	Definition
*	pet_tag_id	INT AUTO_INCREMENT
	pet_tag_pet	INT
	pet_tag_tag	INT
	pet_tag_active	TINYINT DEFAULT 1
	pet_tag_record	DATETIME DEFAULT CURRENT_TIMESTAMP

Tabla 5.21

Índices de la tabla pet_tags

Nombre del Índice	Tipo	Columnas
pet_tag_tag_FK_idx	Index	pet_tag_tag
pet_tag_pet_FK	Index	pet_tag_pet

Tabla 5.22

Claves Foráneas de la tabla pet_tags

Nombre de la Clave Foránea	Referencia
pet_tag_pet_FK	pet_tag_pet pets.id_pet
pet_tag_tag_FK	pet_tag_tag tags.id_tag

Tabla: Visits

Descripción: Citas médicas de los usuarios a las mascotas.

Tabla 5.23

Tabla visits

Idx	Column Name	Definition
*	visit_id	INT AUTO INCREMENT
	visit_client	INT
	visit_pet	INT
	visit_date	DATETIME
	visit_vet	INT
	visit_actions	MEDIUMTEXT
	visit_weight	FLOAT DEFAULT NULL
	visit_condition	INT DEFAULT 1
	visit_temperature	FLOAT DEFAULT NULL
	visit_anam	MEDIUMTEXT
	visit_symptoms	MEDIUMTEXT
	visit_tests	MEDIUMTEXT
	visit_diagnostics	MEDIUMTEXT
	visit_treatment	MEDIUMTEXT
	visit_vaccines	MEDIUMTEXT
	visit_deworm	MEDIUMTEXT
	visit_recommendations	MEDIUMTEXT
	visit_completed	TINYINT DEFAULT 0

Tabla 5.24

Índices de la tabla visits

Nombre del Índice	Tipo	Columnas
visit_client_FK	Index	visit_client
visit_pet_FK	Index	visit_pet
visit_vet_FK	Index	visit_vet

Tabla 5.25

Claves Foráneas de la tabla visits

Nombre de la Clave Foránea	Referencia
visit_client_FK	visit_client clients.client_id
visit_pet_FK	visit_pet pets.pet_id
visit_vet_FK	visit_vet sys_users.user_id

Tabla: Vaccinations

Descripción: Tabla para almacenar la lógica de vacunación de las mascotas.

Tabla 5.26

Tabla vaccinations

Idx	Column Name	Definition
*	vaccination_id	INT AUTO INCREMENT
	vaccination_article	INT
	vaccination_pet	INT
	vaccination_validity_period	INT DEFAULT 365
	vaccination_visit_applied	INT DEFAULT NULL
	vaccination_visit_planned	INT DEFAULT NULL
	vaccination_date	DATETIME DEFAULT NULL

Tabla 5.27

Índices de la tabla vaccinations

Nombre del Índice	Tipo	Columnas
vaccination_article_FK_idx	Index	vaccination_article
vaccination_pet_FK_idx	Index	vaccination_pet
vaccination_visit_planned_FK_idx	Index	vaccination_visit_planned
vaccination_visit_applied_FK_idx	Index	vaccination_visit_applied

Tabla 5.28

Claves Foráneas de la tabla vaccinations

Nombre de la Clave Foránea	Referencia
vaccination_article_FK	vaccination_article articles.article_id
vaccination_pet_FK	vaccination_pet pets.pet_id
vaccination_visit_applied_FK	vaccination_visit_applied visits.visit_id
vaccination_visit_planned_FK	vaccination_visit_planned visits.visit_id

5.2. Diseño de la Interfaz de Usuario

En esta sección vamos a detallar las decisiones que hemos tomado a la hora de implementar la interfaz de usuario, así como el conjunto de estilos y el conjunto de componentes visuales que la conforman.

5.2.1. Principios de Diseño

El diseño de la interfaz de usuario se basa en los siguientes principios:

- **Simplicidad.** Para evitar la sobrecarga de información, la página tiene que ser limpia, simple e intuitiva.
- **Consistencia.** Se debe mantener una coherencia en el uso de colores, tipografías e iconografía a lo largo de toda la aplicación. La iconografía y los colores, en este caso, han sido escogidos por el cliente.
- **Accesibilidad.** Aunque se han tenido en cuenta diversos aspectos de accesibilidad como el contraste, el proyecto no está pensado para que lo use una persona con discapacidad visual.
- **Responsividad.** La interfaz debe responder correctamente a diversos tamaños de dispositivos, especialmente, portátil y teléfono móvil.

5.2.2. Estilos

Colores

Los colores utilizados en la interfaz se basan en la paleta de colores de la marca:

- **Color Primario:** #4d5e3e 
- **Color Secundario:** #ffd4d4 
- **Color de Fondo:** #dadada 
- **Color de Texto:** #333333 

Tipografías

La tipografía principal utilizada en la interfaz es “Roboto” (5.2), que se utiliza en diferentes pesos y tamaños para mantener la jerarquía visual.

Iconografía

Se utilizan iconos de la librería Material Icons de React para representar acciones y elementos de la interfaz de manera visual y comprensible (5.3).

Roboto

Figura 5.2: Tipografía “Roboto”



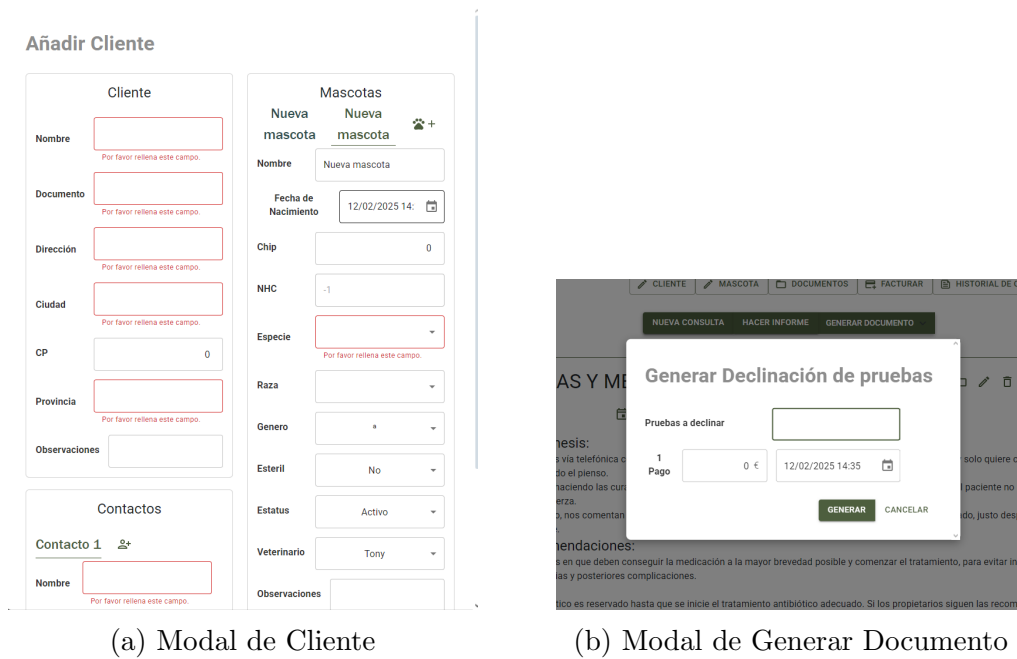
Figura 5.3: Iconos de la librería Material Icons

5.2.3. Componentes Visuales

Modales

El diseño de la interfaz se basa en gran medida en el uso de modales para la interacción del usuario. Los modales se utilizan para añadir, editar y eliminar elementos, así como para generar documentos y mostrar detalles adicionales.

- **Modal de Cliente (Figura 5.4a).** El único modal cargado de la herramienta, por petición del cliente, para poder crear cliente, mascota y contacto desde un mismo sitio.
- **Modal de Generar Documento (Figura 5.4b).** Modal para generar los documentos de pruebas oficiales. Los campos editables cambian según el documento a generar.
- **Modal de Elemento (Figura 5.5).** Exceptuando el modal de Cliente y de generación de documentos, la otra mayoría de modales siguen un modelo más simple para crear o editar.



Crear Cita

Multiples mascotas - Mascota 2 - 37

Fecha 12/02/2025 14:19

Motivo de consulta

Veterinario Tony

CREAR CANCELAR

Figura 5.5: Modal General

Vistas Principales

La aplicación cuenta con cuatro vistas principales: Inicio, Clientes, Histórico y Artículos.

- **Vista de Inicio de Sesión (Figura 5.6).** En esta vista tan solo tenemos

un componente para iniciar la sesión. Debemos prestar atención a que no hay botón para registro o para recuperar la contraseña, esto es así porque los usuarios (veterinarios) que participan en la empresa son dos y no hay planes de expansión en el futuro próximo. De esta forma, en el raro caso de que alguno de ellos olvidase la contraseña, debe pedirnos las credenciales a nosotros. Este “modelo de usuarios” limitado y estricto, prevemos que será la norma en este proyecto, al menos a medio plazo.


El formulario de inicio de sesión de TOBOVET se presenta en un recuadro centralizado. En la parte superior, se encuentra el logo de TOBOVET, que consiste en un dibujo de un perro y el texto "TOBOVET" con un símbolo de estetoscopio. Debajo del logo, el título "Iniciar Sesión" está centrado. El formulario contiene dos campos de entrada: "Correo Electrónico" y "Contraseña", ambos con bordes redondeados y un icono de ojo para alternar la visibilidad de la contraseña. En la parte inferior, hay un botón rectangular de color verde oscuro con el texto "INICIAR SESIÓN" en mayúsculas blancas.

Figura 5.6: Vista Inicio Sesión

- **Vista de Inicio (Figura 5.7).** En esta vista tenemos cuatro indicadores a modo de “dashboard” que refieren a cuatro tablas que se despliegan debajo. Estas tablas pueden ocultarse para encontrar lo necesario con facilidad.
- **Vista de Clientes (Figura 5.8).** En esta vista se incluye una tabla de todos los clientes registrados. Desde aquí podemos editar un cliente, crearle una cita, ver su historial, facturarlo o generarle un documento.
- **Vista de Histórico (Figura 5.9).** Como hemos comentado anteriormente, originalmente esta vista iba a ser una para ver todas las facturas, sin embargo, el cliente pidió también un histórico de todas las visitas pasadas en esta vista. Para cada factura podemos borrarla o verla y para cada cita podemos ver el historial completo, editarla, borrarla o terminarla.
- **Vista de Artículos (Figura 5.10).** Para esta vista tenemos un listado de artículos desde donde podemos editarlos o eliminarlos.

Todas estas vistas basadas en tabla usan el mismo componente de tablas (Figura 5.11). Este componente permite ordenar las filas, ocultar las columnas y filtrar por cualquier campo que desee el usuario.

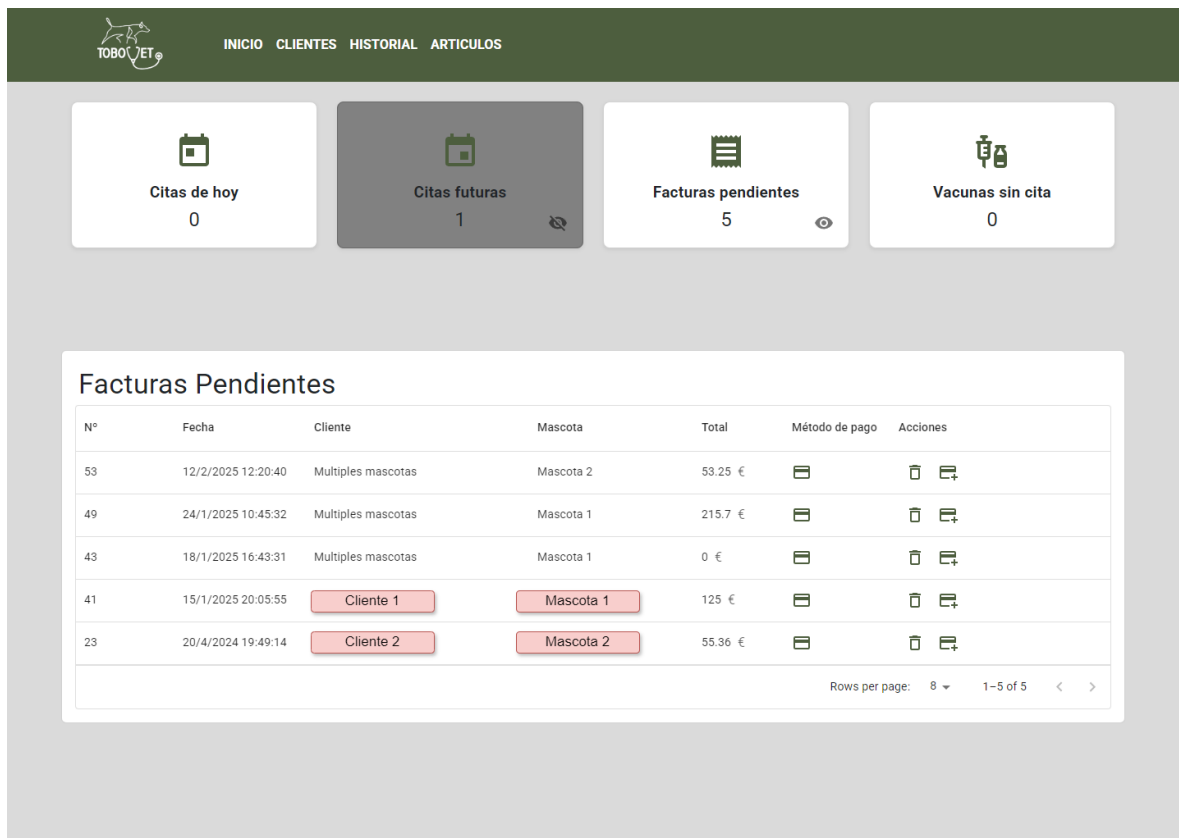


Figura 5.7: Vista Inicio

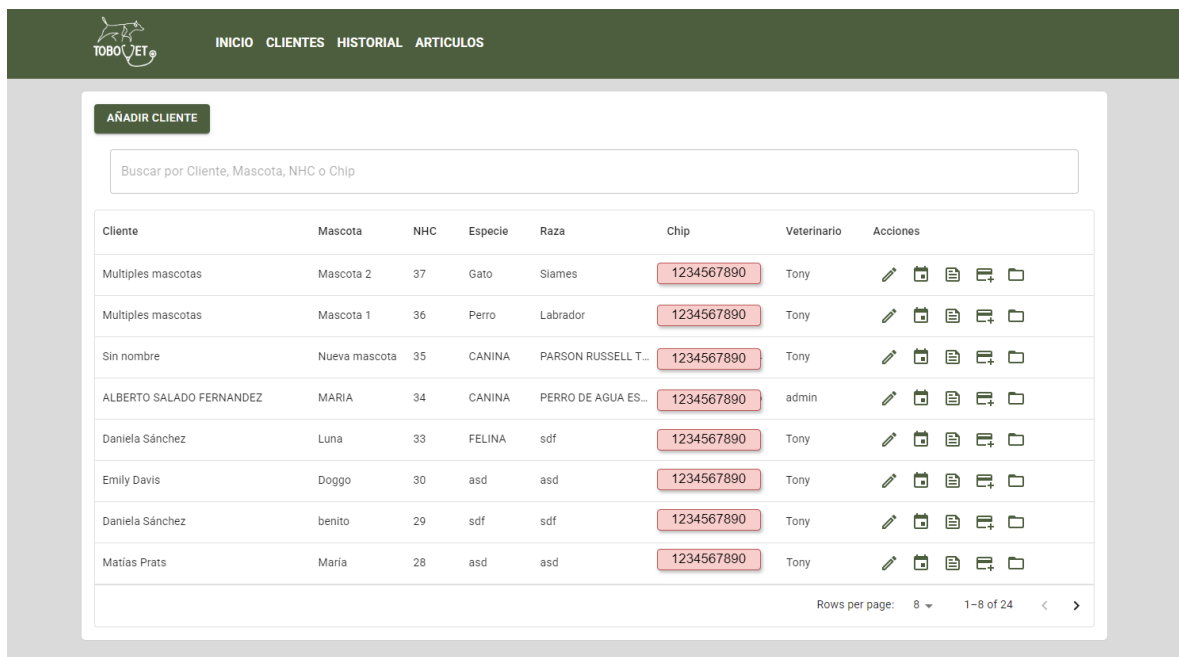



Figura 5.8: Vista Clientes



[INICIO](#)
[CLIENTES](#)
[HISTORIAL](#)
[ARTICULOS](#)

Facturas

	Código	Cliente	Mascota	Total	Fecha	Método de pago	Acciones
52		Emily Davis	Doggo	0 €	27/1/2025 18:27:25		
51	1125	Emily Davis	Doggo	120 €	27/1/2025 18:27:25		
50	1025	Emily Davis	Doggo	12 €	27/1/2025 18:25:46		
47	725	Multiples mascotas	Mascota 1	22.2 €	22/1/2025 19:20:46		
46	625	ALBERTO SALADO FERNANDEZ	MARIA	21.45 €	22/1/2025 18:16:33		
45	525	ALBERTO SALADO FERNANDEZ	MARIA	122 €	22/1/2025 18:11:41		
44	425	ALBERTO SALADO FERNANDEZ	MARIA	20 €	22/1/2025 18:09:44		
26		Jane Smith	Buddy	33.25 €	22/1/2025 18:03:50		

Rows per page: 8 1 - 8 of 41 < >

Historial de visitas

Cliente	Mascota	Fecha	Especie	Motivo de Consulta	Chip	Acciones
Multiples mascotas	Mascota 2	11/2/2025 20:12:44	Gato	Patata	1234567890	
Emily Davis	Doggo	28/1/2025 19:59:36	asd	Matrix	1234567890	
ALBERTO SALADO FERNANDEZ	MARIA	27/1/2025 19:54:23	CANINA	Vac	1234567890	
Sarah Williams	Charlie	22/1/2025 18:02:40	Cat		1234567890	
ALBERTO SALADO FERNANDEZ	MARIA	18/1/2025 17:04:16	CANINA		1234567890	
ALBERTO SALADO FERNANDEZ	MARIA	18/1/2025 17:04:16	CANINA		1234567890	
Multiples mascotas	Mascota 2	15/1/2025 20:15:31	Gato	Consulta 1	1234567890	
ALBERTO SALADO FERNANDEZ	MARIA	14/1/2025 19:30:41	CANINA	Prueba sin vacunas	1234567890	

INICIO
CLIENTES
HISTORIAL
ARTICULOS

AÑADIR ARTÍCULO

Nombre	Grupo	Precio	PVP	Stock	Caducidad	Acciones	
Interactive Dog Toy	Toys	4.75 €	5.75 €	75	365		
Gentle Pet Grooming Brush	Grooming	7.43 €	8.99 €	60	365		
Orthopedic Pet Bed	Bedding	20.66 €	25 €	25	15		
Adjustable Dog Collar	Collars	6.2 €	7.5 €	45	365		
Durable Dog Leash	Leashes	5.17 €	6.25 €	55	365		
Natural Dog Treats	Treats	10.54 €	12.75 €	40	365		
Organic Cat Food	Food	9.08 €	10.99 €	100	365		
Vitamin Supplements for Pets	Vacuna	21.07 €	25.5 €	50	365		

Rows per page: 8
1 - 8 of 51

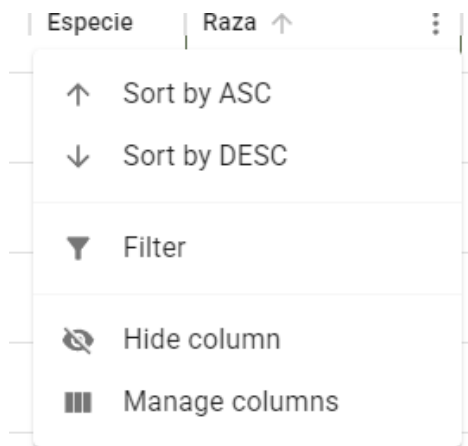


Figura 5.11: Opciones de tabla

Facturas

La interfaz permite la creación y visualización de facturas.

- **Factura en Proceso (Figura 5.12):** El usuario puede elegir los productos en un selector y estos se añadirán a la tabla.

Factura para ALBERTO SALADO FERNANDEZ

☐ Presupuesto
Total: 60.25 €

Facturar a empresa

12/02/2025 14:28

Elige un producto:

Id	Nombre	Grupo	Precio ud.	Uds.	IVA	Dto.	Total
36	Cooling Gel Pet Mat	Bedding	16.53 €	— 1 +	21 %	0 %	20 €
6	Orthopedic Pet Bed	Bedding	20.66 €	— 1 +	21 %	0 %	25 €
34	Pet Travel Water Bottle	Accessories	12.6 €	— 1 +	21 %	0 %	15.25 €

Rows per page: 8
1-3 of 3

Figura 5.12: Factura en proceso

- **Factura Generada (Figura 5.13).** El usuario genera una factura con todos los elementos de la anterior pantalla, desde aquí puede seleccionarse el tipo de cobro y tramitarlo.



TOBOVET S.L.P.

B16393159
633 443 858 / 622 001 250
C. Guatemala, 16,
Jerez de la Frontera,
España

12/2/2025

ALBERTO SALADO FERNANDEZ

XXXXXXXXXX

a

JEREZ DE LA FRONTERA
CADIZ
114010

Chip de la mascota: 985113005440536

Nombre de la mascota: MARIA

Descripción	Precio ud.	Cantidad	IVA	Total
Cooling Gel Pet Mat	16.53 €	1	21 % 3.47 €	20 €
Orthopedic Pet Bed	20.66 €	1	21 % 4.34 €	25 €
Pet Travel Water Bottle	12.6 €	1	21 % 2.65 €	15.25 €

Resumen:

Subtotal: 49.79 €

IVA: 10.46 €

Total: 60.25 €

TOBOVET S.L.P. es el Responsable del tratamiento de sus datos personales y le informa de que estos datos serán tratados de conformidad con lo dispuesto en el Reglamento (UE) 2016/679, de 27 de abril (GDPR), y la Ley Orgánica 3/2018, de 5 de diciembre (LOPDGDD), con la finalidad de prestar servicios veterinarios (en base a una relación contractual, obligación legal o interés legítimo) y conservarlos durante no más tiempo del necesario para mantener el fin del tratamiento o mientras existan prescripciones legales que dictaminen su custodia. Sus datos no se comunicarán a terceros, salvo obligación legal. Asimismo, se le informa de que puede ejercer los derechos de acceso, rectificación, portabilidad y supresión de sus datos y los de limitación y oposición a su tratamiento dirigiéndose a TOBOVET S.L.P. en CALLE GUATEMALA 16, 11407 EN JEREZ DE LA FRONTERA (CÁDIZ). E-mail: tobovet@gmail.com y el de reclamación.

Tarjeta ▼

TRAMITAR COBRO

CANCELAR

Figura 5.13: Factura hecha

Historial de Mascotas

La interfaz incluye una sección para visualizar el historial de una mascota, mostrando detalles de visitas, tratamientos y vacunas (Figura 5.14).

Historial de Loki
Cliente: JUAN ANTONIO GAMBIN

TOBO VET

35 1234567890 CANINA - PARSON RUSSELL TERRIER 1 año y 10 meses Hembra

Antonio Sevilla Ureba +34 611 611 611 test@email.com C. Guatemala, 16, Jerez, España # 1234567890

CLIENTE MASCOTA DOCUMENTOS FACTURAR HISTORIAL DE CONSUMO VACUNAS

NUEVA CONSULTA HACER INFORME GENERAR DOCUMENTO

CURAS Y MEDICACIÓN

9/10/2024, 18:24:25 Antonio Sevilla Ureba 1

Anamnesis:
Hablamos vía telefónica con los propietarios de Loki. Nos comentan que el paciente se muestra apático y solo quiere comer pollo, rechazando el pienso.
Le están haciendo las curas de la herida y administrando furacin como pueden, ya que según nos dicen, el paciente no se deja y tiene mucha fuerza.
Por último, nos comentan que aún no han comprado la amoxicilina ni el meloxicam que recetamos el sábado, justo después de colocar el drenaje.

Recomendaciones:
Insistimos en que deben conseguir la medicación a la mayor brevedad posible y comenzar el tratamiento, para evitar infecciones secundarias y posteriores complicaciones.

El pronóstico es reservado hasta que se inicie el tratamiento antibiótico adecuado. Si los propietarios siguen las recomendaciones y no se desarrollan complicaciones infecciosas graves, se espera una evolución favorable en los próximos días.

REVISION URGENCIA DEL FIN DE SEMANA

7/10/2024, 21:48:44 Antonio Sevilla Ureba 15 kg 1 38.5 °C

Anamnesis:
Acudimos a la revisión de la urgencia del pasado sábado, donde el paciente fue atendido por heridas tras un ataque de otro perro. En esta ocasión, los tutores informan que Loki ha mejorado notablemente, ya que no presenta la misma cojera observada en la extremidad

Figura 5.14: Historial de Loki

Documentos Generados

La interfaz permite la generación de documentos personalizados para los clientes (Figura 5.15).



TOBOVET

B16393159

637 172 558

Calle María Antonia Jesús Tirado 9 C,
Jerez,
España

Formulario de declinacion de pruebas diagnósticas

D/D^a **Test** con DNI **test** afirma que es el propietario del animal que se describe a continuación:

Nombre: **1234567890**

Especie: **CANINA**

Raza: **PARSON RUSSELL TERRIER**

Microchip: **1234567890**

Declino la realización de pruebas diagnósticas o de control que fuesen necesarias o propuestas por el equipo veterinario, declaro haber sido informado de la importancia de la realización de las mismas y lo hago bajo mi completa responsabilidad.

Dichas pruebas son:

Prueba 1

Prueba 2

Firmado a 12/2/2025

Cliente

Veterinario

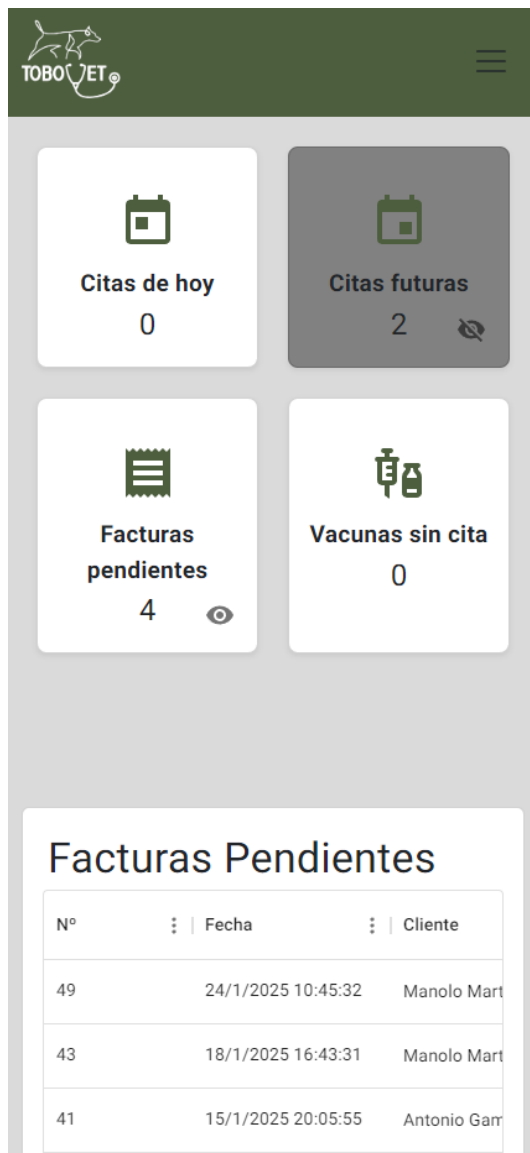
Figura 5.15: Ejemplo de documento generado

5.2.4. Diseño Responsivo

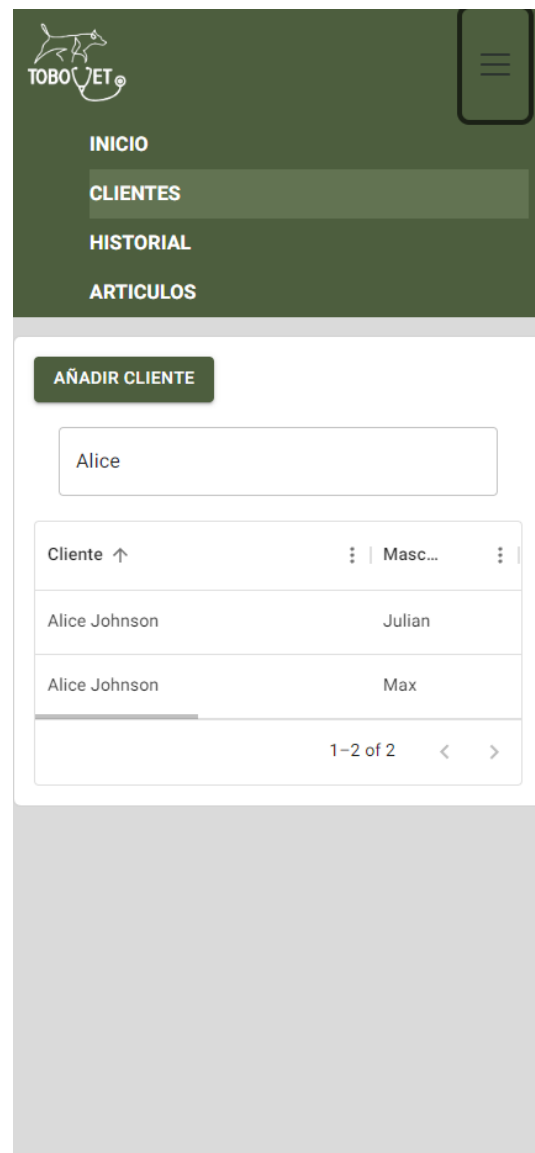
Como mencionamos anteriormente, el diseño responsivo ha sido una prioridad en nuestro proyecto, ya que consideramos fundamental que la página se adapte correctamente a dispositivos móviles, dado que todas las visitas se realizan a domicilio.

A continuación, mostramos capturas de pantalla que consideramos relevantes para ilustrar la adaptación del diseño a diferentes tamaños de pantalla.

- Figura 5.16a: vista de “Inicio” desde un dispositivo móvil.
- Figura 5.16b: vista de “Clientes” desde un dispositivo móvil.
- Figura 5.17a: generación de un documento desde un dispositivo móvil.
- Figura 5.17b: factura hecha desde un dispositivo móvil.



(a) Vista de Inicio



(b) Vista de Cliente

CLIENTE

MASCOTA

DOCUMENTOS

FACTURAR

HISTORIAL DE CONSUMO

VACUNAS

NUEVA CONSULTA

HACER INFORME

GENERAR DOCUMENTO

Generar Declinación de pruebas

Pruebas a declinar

1

Pago

0 €

04/03/2025

GENERAR

CANCELAR

(a) Generación de Documento

TOBOVET

TOBOVET S.L.P.

B16393159

633 443 858 / 622 001 250

C. Guatemala, 16,

Jerez de la Frontera,

España

4/3/2025

Cliente 1

12345678X

Calle Prueba 16

JEREZ DE LA FRONTERA

CADIZ

114010

Chip de la mascota: 123456789

Nombre de la mascota: MARIA

	Precio			
Descripción	ud.	Cantidad	IVA	Total
CAMBIADO	14.88 €	1	21 %	18 €
				3.12 €
Soft Fleece	16.53 €	1	21 %	20 €
Blanket for				3.47 €
Pets				
Resumen:				
Dirección 1 Prueba				

(b) Factura

69

6. Implementación

A continuación se describen el entorno de construcción y marco tecnológico del sistema, así como la estructura del código fuente y fragmentos significativos del código.

6.1. Entorno de Construcción

En esta sección se detalla el marco tecnológico utilizado para la construcción del sistema, incluyendo el entorno de desarrollo, el lenguaje de programación, las herramientas de ayuda a la construcción y despliegue, el control de versiones, el repositorio de componentes y la integración continua.

6.1.1. Entorno de Desarrollo

El entorno de desarrollo integrado (IDE) utilizado para este proyecto es Visual Studio Code (VSCode). VSCode es un editor de código fuente ligero pero potente, con soporte para depuración, control de versiones integrado, y una amplia gama de extensiones que mejoran la productividad del desarrollador.

6.1.2. Lenguaje de Programación

El lenguaje de programación principal utilizado en este proyecto es TypeScript, junto con React para el desarrollo del frontend y Node.js para el backend.

6.1.3. Credenciales de Inicio de Sesión

La autenticación se gestiona mediante JWT, que aseguran la integridad y la veracidad de las credenciales del usuario. Al iniciar sesión, el sistema genera un token que se adjunta a la solicitud para acceder a recursos protegidos. Esta solución se basa en las mejores prácticas recomendadas en la documentación oficial ([JWT.io](https://jwt.io) (2025)) y permite una verificación rápida y eficiente en cada interacción.

6.1.4. Herramientas de Construcción y Despliegue

Para la construcción y despliegue del sistema, se utilizan las siguientes herramientas:

- Vite: Utilizado como el empaquetador de módulos y servidor de desarrollo para el frontend ([You y Contributors](#) (2023)).

- npm: Utilizado para la gestión de dependencias y la automatización del proceso de construcción ([npm, Inc. \(2025\)](#)).

6.1.5. Control de Versiones

El control de versiones se gestiona utilizando Git, un sistema de control de versiones distribuido que permite a los desarrolladores colaborar de manera eficiente y mantener un historial detallado de los cambios en el código fuente. El repositorio del proyecto se aloja en GitHub.

6.1.6. Repositorio de Componentes

El repositorio de componentes utilizado en este proyecto es npm registry, donde se almacenan y gestionan las dependencias del proyecto.

6.1.7. Integración Continua

Para la integración continua, se utiliza GitHub Actions. Esta herramienta permite automatizar la construcción, prueba y despliegue del sistema, asegurando que los cambios en el código se integren de manera continua y sin problemas ([GitHub Inc. \(2025\)](#)).

Este entorno de construcción proporciona una base sólida para el desarrollo, construcción y despliegue del sistema, facilitando el trabajo de los desarrolladores y asegurando la calidad del software.

6.2. Código Fuente

6.2.1. Organización del código fuente

El código fuente de todo el proyecto puede dividirse en dos directorios principales, la App (tobovetApp) y la API que se conecta con la BD (tobovetApi).

Api

Este directorio contiene la lógica del servidor y la interacción con la base de datos. Está estructurado de la siguiente manera:

1. config/: Contiene archivos de configuración
2. routes/: Define las rutas de la Api
3. services/: Implementa la lógica de negocio y la interacción con la base de datos.

4. `utils/`: Contiene utilidades y funciones auxiliares

App

Este directorio contiene la aplicación cliente desarrollada con React y TypeScript. Está estructurado de la siguiente manera:

1. `public/`: Archivos públicos como imágenes
2. `views/`: Vistas principales de la aplicación
3. `utils/`: Funciones auxiliares y constantes
4. `interfaces/`: Definiciones de tipos e interfaces TypeScript
5. `assets/`: Recursos estáticos como imágenes y estilos
6. `controllers`: Controladores para interactuar con la API

6.2.2. Extractos Significativos de Código

A continuación, explicaremos el flujo del código que sigue el caso de uso “Terminar una visita”. Como comentamos anteriormente, este caso de uso puede ser el más complejo del proyecto; si en una visita se ha incluido una vacuna, debe crearse automáticamente una segunda cita para vacunar a esa mascota usando el periodo de validez de esa vacuna. Este flujo incluye:

1. Código frontend, con el botón que va a permitir realizar una petición.
2. Código de construcción de la petición.
3. Ruta en el servidor.
4. Transacción a la BD.

Código frontend

En el frontend, tenemos un componente que muestra las visitas y permite al usuario marcarlas como completadas, el botón para terminar esta visita se encuentra en el archivo *Inicio.tsx*.

```
1 <Tooltip title='Completar'>
2   <IconButton onClick={() => handleCompleteCita(row)}>
3     <TaskAlt color='primary' />
4   </IconButton>
5 </Tooltip>
```

Extracto de código 6.1: `tobovetApp/src/views/Inicio.tsx`

Construcción de la petición

Cuando el usuario hace *clic* en “Guardar Visita”, se llama a la función *handleCompleteCita*, que construye y envía la petición al servidor y actualiza las citas pendientes.

```
1 const handleCompleteCita = async (row: IVisit) => {
2   setLoading(true);
3   const dataCita: boolean = await putToApi('visits/complete
4   ', { id: row.id });
5   if (!dataCita) console.error('Error completando cita');
6   getCitas();
7   getVaccineVisits();
8 };
```

Extracto de código 6.2: tobovetApp/src/views/ClientHistorical/VisitCard.tsx

Esta función hace uso de otra llamada *putToApi*, una función general para hacer peticiones PUT al servidor.

```
1 export const putToApi = async (path: string, body: object |
2   null = null) => {
3   const headers = {
4     'Content-Type': 'application/json',
5   };
6   try {
7     const data = await axios({
8       method: 'PUT',
9       headers,
10      url: import.meta.env.VITE_BACKEND_URL + path,
11      data: body,
12    });
13
14    if (data.status === 200) {
15      return true;
16    } else {
17      console.error('Error updating ${path}');
18      return false;
19    }
20  } catch (error) {
21    console.error(error);
22    return false;
23  }
24 };
```

Extracto de código 6.3: tobovetApp/src/views/ClientHistorical/VisitCard.tsx

Ruta en el servidor

En el backend, la ruta que maneja la petición para terminar una visita se encuentra en el archivo *visits.js*.

Ruta en el Servidor

```
1 router.put("/", async function (_req, res, next) {
2   console.log("PUT visit", _req.body);
3   try {
4     res.json(await visits.updateVisit(_req.body));
5   } catch (err) {
6     console.error('Error while updating visit', err.message);
7     next(err);
8   }
9 });
```

Extracto de código 6.4: tobovetApi/routes/visits.js

Transacción a la BD

La función *updateVisit* en el servicio de visitas maneja la lógica comentada en el caso de uso y hace uso de una transacción segura (*safeQuery*). Primero, recupera las vacunas asociadas a la tabla *vaccinations* mediante una consulta a la tabla, filtrando por el identificador de visita y de mascota. Luego, construye una lista de consultas SQL que se ejecutarán en la transacción. El proceso inicia con la actualización de la tabla *visits*, haciendo uso de la función *checkNullish*, que comprueba si un elemento es nulo para escribir el valor correspondiente al insertarlo en la BD ya que MySQL y TS tratan los nulos de forma diferente. La actualización añade los campos propios al terminar una visita, como síntomas o tratamiento. Posteriormente, si se han registrado vacunas en la visita, se realizan las siguientes acciones a la tabla *vaccinations*:

- Si la vacuna ya existe en la visita, se actualiza su periodo de validez y fecha de aplicación.
- Si la vacuna no está registrada, se inserta una nueva entrada en la tabla.
- Se eliminan las citas de vacunación anteriores para evitar inconsistencias.

Finalmente, se ejecutan todas las consultas dentro de una transacción para garantizar la consistencia de los datos, si alguna operación falla, la transacción evita cambios parciales en la BD.

```
1 async function updateVisit(visit) {
2   let visitVaccines = await db.query(
3     'SELECT *
4     FROM vaccinations
5     WHERE vaccination_visit_applied=${visit.id}
6     AND vaccination_pet=${visit.petId}'
```

```

7 );
8
9 const queries = [];
10 queries.push({
11   sql: 'UPDATE visits
12     SET visit_date=?, visit_vet=?, visit_actions=?,
13       visit_weight=?, visit_condition=?, visit_temperature=?,
14       visit_anam=?, visit_symptoms=?, visit_tests=?,
15       visit_diagnostics=?, visit_tests_diagnostics=?,
16       visit_treatment=?,
17       visit_deworm=?, visit_recommendations=?, visit_completed
18     =?
19     WHERE visit_id=?',
20   params: [
21     checkNullish(dateToSQLFormat(visit.date)),
22     checkNullish(visit.vetId),
23     checkNullish(visit.visitReason),
24     checkNullish(visit.weight),
25     checkNullish(visit.condition),
26     checkNullish(visit.temperature),
27     checkNullish(visit.observations),
28     checkNullish(visit.symptoms),
29     checkNullish(visit.tests),
30     checkNullish(visit.diagnostics),
31     checkNullish(visit.testDiagnostics),
32     checkNullish(visit.treatment),
33     checkNullish(visit.deworm),
34     checkNullish(visit.recommendations),
35     checkNullish(visit.completed),
36     checkNullish(visit.id),
37   ],
38 });
39
40 if (visit.vaccines) {
41   visit.vaccines.forEach(({ id, validity }) => {
42     const vaccinationEntry = visitVaccines.find(
43       ({ vaccination_article }) => vaccination_article ===
44       id
45     );
46     if (vaccinationEntry) {
47       queries.push({
48         sql: 'UPDATE vaccinations SET
49           vaccination_validity_period=?, vaccination_date=?
50         WHERE vaccination_id=?',
51         params: [
52           checkNullish(validity),
53           checkNullish(dateToSQLFormat(visit.date)),

```

```

48         vaccinationEntry.vaccination_id,
49     ],
50     });
51     visitVaccines = visitVaccines.filter(
52         ({ vaccination_article }) => vaccination_article
53         !== id
54     );
55     } else {
56         queries.push({
57             sql: 'INSERT INTO vaccinations (vaccination_article
58             , vaccination_pet,
59             vaccination_visit_applied,
60             vaccination_validity_period, vaccination_date)
61             VALUES (?, ?, ?, ?, ?)',
62             params: [
63                 id,
64                 checkNullish(visit.petId),
65                 checkNullish(visit.id),
66                 checkNullish(visit.validity),
67                 checkNullish(dateToSQLFormat(visit.date)),
68             ],
69         });
70     }
71     });
72     visitVaccines.forEach(({ vaccination_id }) =>
73     queries.push({
74         sql: 'DELETE FROM vaccinations
75         WHERE vaccination_id=?',
76         params: [vaccination_id],
77     }));
78     });
79 }
80
81 const result = await db.safeQuery(queries);
82
83 console.log("updateVisit result", result);
84 let message = "Error in updating visit";
85
86 if (result.affectedRows) {
87     message = "Visit updated successfully";
88 }
89
90 return { message };
91 }

```

Extracto de código 6.5: tobovetApi/routes/visits.js

7. Pruebas del Sistema

En este capítulo se presenta el plan de pruebas diseñado para el sistema de información desarrollado para la gestión de la clínica veterinaria. Se detallan distintos tipos de pruebas, tanto manuales (mediante listas de comprobación) como automatizadas usando herramientas específicas. Dado el enfoque ágil adoptado en el proyecto, se implementó un proceso de validación continua y gradual a través de Jira, permitiendo que el cliente estuviera al tanto de las nuevas funcionalidades implementadas en el software. Esto facilitó que el cliente las probara en un entorno controlado y más tarde, en escenarios con procesos y datos reales. Este ciclo iterativo permitió confirmar de manera progresiva que el sistema cumple tanto los requisitos funcionales como no funcionales definidos en la fase inicial de desarrollo, así como las nuevas funcionalidades que el cliente solicitaba.

7.1. Estrategia

En esta sección se describe el alcance de las pruebas, los tipos de pruebas que se realizarán y cómo se interpretarán y evaluarán los resultados. El objetivo principal es asegurar que el sistema cumple con todos los requisitos establecidos. A continuación se describen las especificaciones del ordenador con el que se han hecho las pruebas:

- CPU: AMD Ryzen 5 4500 6 Cores 3.60 GHz.
- GPU: Nvidia GeForce RTX 4060 Ti 16 GB.
- RAM: 32 GB de Corsair Vengeance Pro.
- Monitor: Samsung Odyssey G5, 2k 144 Hz.

7.1.1. Alcance de las Pruebas

El alcance de las pruebas incluye la verificación de todas las funcionalidades principales del sistema, así como la validación de requisitos no funcionales, el rendimiento y la accesibilidad. Las pruebas se realizarán en los siguientes niveles:

- **Pruebas Unitarias:** Verificación de la funcionalidad de componentes individuales del sistema.
- **Pruebas de Integración.** Verificación de la interacción entre diferentes módulos del sistema.
- **Pruebas de Aceptación**
 - **Pruebas Funcionales.** Validación de los casos de uso del sistema desde la perspectiva del usuario final.

- **Pruebas No Funcionales.** Evaluación de la eficiencia del sistema.
- **Pruebas de Accesibilidad.** Verificación de la accesibilidad para personas con discapacidades.
- **Pruebas de Usabilidad.** Evaluación de la facilidad de uso del sistema.
- **Pruebas de diseño responsivo.** Verificación del uso del sistema en dispositivos de distintos tamaños.

7.1.2. Interpretación y Evaluación de los resultados

- **Pruebas Unitarias e Integración.** Las pruebas deben pasar sin errores. Los errores se registrarán y se corregirán antes de la siguiente fase de pruebas.
- **Pruebas de Aceptación**
 - **Pruebas Funcionales.** Todas las funcionalidades principales del sistema funcionan según lo esperado. Los errores se registrarán y se corregirán antes de la siguiente fase de pruebas.
 - **Pruebas No Funcionales.** El sistema cumple con los requisitos de rendimiento considerados.
- **Pruebas de Accesibilidad.** El sistema cumple con un subconjunto de la normativa estándar de accesibilidad en el desarrollo web [World Wide Web Consortium \(W3C\)](#) (2018).
- **Pruebas de Usabilidad.** Los usuarios pueden completar las tareas sin dificultades significativas. Se observará cualquier dificultad.
- **Pruebas de diseño responsivo.** El sistema se adapta correctamente a dispositivos de distintos tamaños en cuestiones de diseño y usabilidad.

Perfiles y Participantes

- **Desarrollador.** Responsable de escribir y ejecutar las pruebas.
- **Usuarios Finales.** Participan en las pruebas de usabilidad.
- **Testers.** Dieciocho usuarios ajenos al sistema en un rango de edad de entre 23 y 73 años, con diversidad en su conocimiento, dispositivo y rango de discapacidades auditivas, visuales o cognitivas.

7.2. Pruebas Unitarias y de Integración

Las pruebas unitarias tienen por objetivo localizar errores en cada nuevo artefacto software desarrollado, antes de que se produzca la integración con el resto de los artefactos del sistema. Para este proyecto, usaremos el framework Jest para escribir y ejecutar pruebas unitarias.

7.2.1. Ejemplo de Prueba Unitaria

Como hemos mencionado anteriormente, utilizamos Jest para llevar a cabo las pruebas unitarias. Usamos funciones de aserción como *expect(results).toBeDefined* y *toBeInTheDocument* para comprobar el correcto funcionamiento del sistema:

- *toBeDefined()*: Verifica que el valor no sea *undefined*, lo que indica que la variable existe y ha sido asignado correctamente.
- *toBeInTheDocument* (de la librería *@testing-library/jest-dom* [Testing Library \(2025\)](#)): comprueba que un elemento está presente en el DOM, asegurando que un componente se renderiza correctamente.

Para evitar sobrecargar la memoria del informe con una gran cantidad de resultados, se han seleccionado unos ejemplos representativos que prueban el funcionamiento de algunas de las funcionalidades clave. Hemos definido siete *test suites* o agrupaciones de test. A continuación, identificaremos estas siete y explicaremos brevemente la labor de cada una.

- Tests de la API
 - Clientes: está formado por diecinueve tests y comprueba que funcionen correctamente las llamadas de creación, modificación y borrado de los clientes, sus contactos y mascotas y usuarios, incluido el inicio de sesión.
 - Pagos: está formado por doce tests y comprueba que funcionen correctamente las llamadas de creación, modificación y borrado de pagos, así como su relación con los artículos.
 - Visitas: está formado por ocho tests y comprueba que funcionen correctamente las llamadas de creación, modificación y borrado de citas, así como su relación con vacunas.
- Tests de la APP
 - Cartas: está formado por veinte tests y comprueba que todos los elementos de tipo “carta” se muestran adecuadamente y que cada uno cumple su función específica.
 - Tablas: está formado por dos tests y comprueba que los elementos de tipo “tabla” se muestran adecuadamente y que la tabla puede ordenarse y modificarse adecuadamente.
 - Vistas: está formado por nueve tests y comprueba que las principales vistas de la página se muestran correctamente.
 - Modales: está formado por dieciséis tests y comprueba que todos los elementos de tipo “modal” se muestran adecuadamente y que cada uno cumple su función específica.

A continuación, se incluye una captura que demuestra que las pruebas unitarias en su conjunto se realizaron de manera exitosa y que todas las funcionalidades probadas cumplen con los requisitos establecidos (Figura 7.1). Este enfoque permite evidenciar la calidad del código sin saturar la documentación.

Prueba unitaria de la API (CRUD con client_contacts)

```
1 test('can fetch contacts', async () => {
2   const result = await db.query('SELECT * FROM
3   client_contacts');
4   expect(result).toBeDefined();
5 });
6 test('can insert a contact', async () => {
7   const result = await db.query('INSERT INTO
8   client_contacts (client_id, contact_name, contact_phone,
9   contact_email) VALUES ("9999", "Test", "123456789", "1234"
10  )');
11  expect(result).toBeDefined();
12 });
13 test('can update a contact', async () => {
14   const result = await db.query('UPDATE client_contacts SET
15   contact_name = "Test2" WHERE contact_name = "Test" AND
16   client_id = "9999"');
17   expect(result).toBeDefined();
18 });
19 test('can delete a contact', async () => {
20   const result = await db.query('DELETE FROM
21   client_contacts WHERE contact_name = "Test2" AND client_id
22   = "9999"');
23   expect(result).toBeDefined();
24 });
```

Extracto de código 7.1: tobovetApi/routes/visits.js

Prueba unitaria de la App (Componente HomeDashboard)

```
1 import { HomeDashboard } from './HomeDashboard';
2
3 test('renders HomeDashboard component', () => {
4   render(
5     <HomeDashboard
6       visitsToday={5}
7       visitsFuture={10}
8       paymentsPending={3}
9       vaccinesPending={2}
10      isTodayTableVisible={true}
11      isFutureTableVisible={true}
12      isPaymentsTableVisible={true}
13      isVaccinationsTableVisible={true}
```

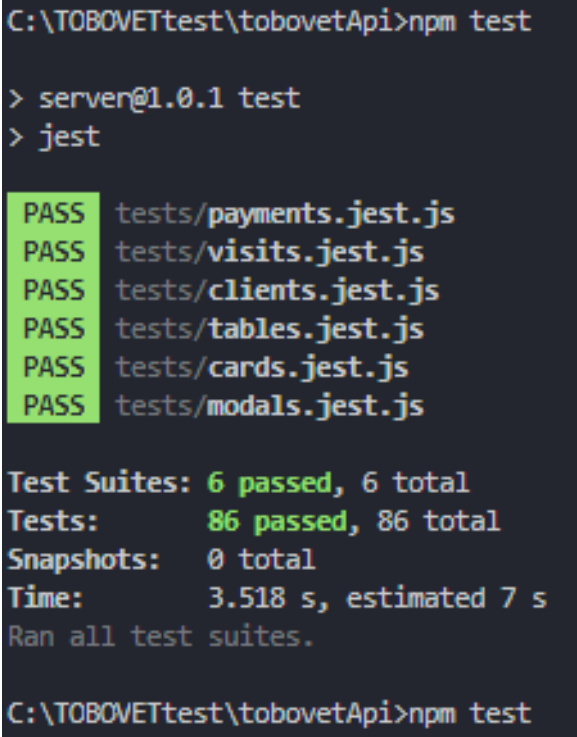
```

14     setTodayTableVisibility={() => {}}
15     setFutureTableVisibility={() => {}}
16     setPaymentsTableVisibility={() => {}}
17     setVaccinationsTableVisibility={() => {}}
18   />
19 );
20 expect(screen.getByText('Citas de hoy')).toBeInTheDocument
  ();
21 expect(screen.getByText('Citas futuras')).toBeInTheDocument
  ();
22 expect(screen.getByText('Facturas pendientes')).
  toBeInTheDocument();
23 expect(screen.getByText('Vacunas sin cita')).
  toBeInTheDocument();
24 });

```

Extracto de código 7.2: tobovetApi/routes/visits.js

Pruebas unitarias y de integración aprobadas



```

C:\TOBOVETtest\tobovetApi>npm test

> server@1.0.1 test
> jest

PASS tests/payments.jest.js
PASS tests/visits.jest.js
PASS tests/clients.jest.js
PASS tests/tables.jest.js
PASS tests/cards.jest.js
PASS tests/modals.jest.js

Test Suites: 6 passed, 6 total
Tests:       86 passed, 86 total
Snapshots:   0 total
Time:        3.518 s, estimated 7 s
Ran all test suites.

C:\TOBOVETtest\tobovetApi>npm test

```

Figura 7.1: Pruebas jest

7.2.2. Pruebas de Integración

Durante el desarrollo de la aplicación se implementó cada módulo por separado, realizando pruebas de integración con la adición de nuevas funciones. Al usar Node.js

en el backend y React en el frontend, optamos por un enfoque incremental.

Las rutas y los endpoints esenciales se definieron primero sin integrar todas las funcionalidades desde el principio. Luego, se fueron completando paulatinamente las acciones requeridas para cada módulo. Este proceso permitió realizar pruebas de integración a medida que avanzaba el desarrollo, asegurando que todos los componentes funcionaran correctamente antes de integrarlos con el resto del sistema. Estas pruebas se basaron en la comprobación manual y los logs del sistema en consola para su verificación. Se diseñaron varios escenarios de pruebas para garantizar la correcta integración, contemplando tanto casos exitosos como situaciones alternativas o de error.

Casos exitosos

- Crear un cliente y añadirle dos contactos y dos mascotas.
- Crear una cita a una mascota de un cliente que tuviera múltiples mascotas.
- Terminar una cita con más de una vacuna.
- Registrar un nuevo artículo con un nuevo grupo.
- Generar una factura con varios productos y verificar el total.
- Generar una factura, añadir algunos productos, cerrarla y volver a añadir otros antes de facturarla.
- Comprobar que el cambio de precio de un artículo no cambia las facturas previas.
- Consultar el historial de una mascota de un cliente que tuviera múltiples mascotas.
- Editar la información de un cliente, uno de sus contactos y una de sus mascotas y comprobar que esta información quedaba reflejada en todas las vistas.
- Eliminar un cliente y comprobar que sus facturas y citas anteriores no desaparecen.

Casos alternativos o de error

- Intentar añadir un cliente, contacto o mascota sin completar campos obligatorios.
- Intentar añadir más de 3 contactos a un cliente.
- Dar de baja un cliente y luego intentar ver el historial de una de sus mascotas.
- Generar una factura sin productos.
- Intentar terminar una cita sin completar campos obligatorios.

Ejemplo detallado de la prueba: Comprobar que el cambio de precio de un artículo no cambia las facturas previas

- Paso 1: Acceder a la vista de clientes y crear una factura con un producto cualquiera.
- Paso 2: Cobrar la factura y volver atrás.
- Paso 3: Acceder a la vista de artículos y modificar el precio y nombre de ese producto.
- Paso 4: Confirmar que el nombre y el precio de ese producto se mantienen en la factura.

7.3. Pruebas de Aceptación

En esta actividad se realizan las pruebas de sistema, de modo que se asegure que el sistema cumple con todos los requisitos establecidos: funcionales, de almacenamiento, reglas de negocio y no funcionales. Al adoptar un enfoque ágil, el cliente ha participado activamente en cada iteración del proceso, evaluando y validando las nuevas funcionalidades a medida que se iban desarrollando, tanto en un entorno de test como en la aplicación con casos reales, lo que ha permitido identificar y corregir errores de forma continua.

7.3.1. Pruebas Funcionales

En esta fase evaluamos todos los casos de uso del sistema, tanto en los escenarios normales como en los alternativos. Se comprobó que las acciones que el usuario tomaba se reflejaban correctamente en los datos del sistema de acuerdo a lo especificado. Como hemos comentado anteriormente, no disponemos de datos cuantitativos precisos, aun así podemos presentar un esquema general de como se realizó la validación:

- Se verificaron las funcionalidades consideradas principales, tales como la gestión de clientes y mascotas, la programación de citas y la generación de facturas.
- Cada nueva funcionalidad fue probada de forma individual mediante pruebas unitarias y luego integrada en el sistema general.
- Los errores detectados se registraron y se corrigieron en ciclos iterativos antes de proceder a nuevas implementaciones.

7.3.2. Pruebas No Funcionales

Estas pruebas pretenden comprobar el funcionamiento del sistema con respecto a los requisitos no funcionales identificados. La rapidez de carga es uno de los requisitos no funcionales más críticos en cualquier sistema, ya que interfiere directamente con todos los usuarios de este. Para evaluar este requisito de forma objetiva, se ha

empleado Lighthouse, una herramienta que, como hemos explicado anteriormente, está desarrollada por Google e incluida en todos los navegadores basados en Chromium. Las métricas que Lighthouse presenta nos permiten identificar áreas de mejora en la carga inicial, sin olvidar que, tras esta fase, la navegación dentro del sistema se realiza de forma instantánea, gracias a la optimización que ofrece el uso de tecnologías que hemos implementado en el proyecto, como React y su Virtual Dom. A continuación explicaremos brevemente lo que significa cada métrica y expondremos una tabla con los resultados obtenidos tanto para ordenador (7.1) como para dispositivos móviles (7.2) para cada una de las seis clases de navegación (4.3), en la que “Mascota” se refiere al Historial de Cliente.

Métricas medidas

- **Ren (Rendimiento).** Rendimiento general de la página calculado por las siguientes métricas:
 - **FCP (First Contentful Paint).** Tiempo que tarda en aparecer en pantalla el primer elemento del contenido.
 - **SI (Speed Index).** Rapidez con la que se van mostrando los contenidos visibles en la pantalla.
 - **LCP (Largest Contentful Paint).** Tiempo que tarda en aparecer en pantalla el elemento más grande.
 - **TBT (Total Blocking Time).** Tiempo total en el que no se pueden responder interacciones del usuario.
 - **CLS (Cumulative Layout Shift).** Estabilidad de la página, un valor bajo indica que los elementos no se mueven de forma abrupta durante la carga.
- **Acc (Accesibilidad).** Accesibilidad general para la página al tener en cuenta personas con discapacidades visuales y/o que usan herramientas de dictado para navegar por la web.
- **M. Pra (Mejores prácticas).** Valor que indica buenas prácticas a la hora de diseñar páginas web, como el uso de API actualizadas, permitir que los usuarios peguen datos en los campos de entrada, definición de charsets, carga sin errores de consola...

Tabla 7.1

Resultados de Lighthouse en Ordenador

Página	Ren(%)	FCP(s)	SI(s)	LCP(s)	TBT(ms)	CLS	Acc(%)	M.Pra(%)
Sesión	99	0.7	0.7	0.9	0	0.011	100	100
Inicio	97	0.7	1.1	1.2	10	0.034	90	100
Cientes	96	0.9	1.0	1.2	60	0.023	95	100
Historial	74	0.9	1.0	1.3	80	0.541	95	100
Artículos	97	0.9	0.9	1.2	40	0.023	95	100
Mascota	97	0.7	1.0	1.1	50	0.02	90	100
Factura	96	0.9	1.0	1.2	40	0.02	90	100

Tabla 7.2

Resultados de Lighthouse en Móvil

Página	Ren(%)	FCP(s)	SI(s)	LCP(s)	TBT(ms)	CLS	Acc(%)	M. Pra(%)
Sesión	75	3.7	3.7	4.6	70	0.011	100	100
Inicio	52	4.4	5.6	6.7	200	0	90	100
Cientes	50	4.4	5.6	6.9	200	0	95	100
Historial	47	4.9	5.7	7.3	250	0	95	100
Artículos	53	4.4	5.5	6.9	200	0	95	100
Mascota	52	4.4	5.5	7.3	200	0	90	100
Factura	49	4.6	5.4	6.9	250	0	90	100

En primer lugar, nos llama la atención el resultado negativo de CLS que ha tenido la página de “Historial”. Esto podría deberse a la carga asíncrona de datos, que hace que el contenido empuje otros elementos cuando empiezan a realizarse en dos tablas que se expanden tanto horizontalmente. Prestaremos especial atención a este apartado de aquí en adelante, ya que un alto coeficiente de CLS puede afectar negativamente a la accesibilidad de una página.

Podemos comprobar que las puntuaciones en dispositivos móviles han sido notablemente más bajas que en el ordenador. Esto nos hace ver que además de centrarnos en crear una interfaz con un diseño responsivo, deberíamos haber investigado y prestado atención a optimizaciones específicas para dispositivos móviles. Aun así, debemos tener en cuenta que Lighthouse evalúa el rendimiento desde una carga en frío; si lo planteamos así, el usuario no debería tener problemas a la hora de disfrutar de la experiencia, ya que la navegación interna es casi instantánea. Otro aspecto a tener en cuenta es que las pruebas que Lighthouse realiza para la versión móvil son simuladas bajo unas restricciones concretas de recursos, por lo que los resultados son extrapolaciones realizadas por el software ([Google \(2025b\)](#)). En general, los dispositivos móviles suelen tener menos capacidad de procesamiento y conexiones de red más lentas en comparación con los ordenadores de escritorio; así, estas diferencias en el rendimiento entre dispositivos móviles y de escritorio son habituales. Para ilustrar este problema, hacemos referencia a la tabla 7.3, donde comparamos nuestras puntuaciones de “Factura”, que es donde hemos obtenido el peor de los resultados, con una página como la de Apple o Amazon para ver que se encuentran en el mismo umbral.

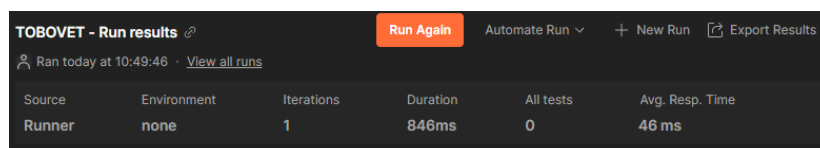
Tabla 7.3

Comparación de “Factura” con otras páginas

Página	Ren(%)	FCP(s)	SI(s)	LCP(s)	TBT(ms)	CLS
Factura	49	4.6	5.4	6.9	250	0
Amazon	36	3.8	6.7	10.8	1370	0.056
Apple	38	12.3	12.3	36.1	700	0

Pruebas de Tiempo de Respuesta

Durante las pruebas, utilizamos Postman para automatizar las llamadas a los endpoints del servidor, incluyendo operaciones GET, POST, PUT y DELETE. Estas llamadas pueden agruparse en “colecciones” y ejecutar todas automáticamente; existe también la posibilidad de automatizar pruebas de precondiciones y postcondiciones. Sin embargo, como las pruebas funcionales se han llevado a cabo con Jest, no hemos usado esta funcionalidad. Las pruebas confirmaron el correcto funcionamiento de la API, obteniendo tiempos de respuesta óptimos. En particular, las solicitudes GET 7.2 tuvieron un tiempo medio de 46 ms, mientras que las operaciones POST, PUT y DELETE registraron tiempos de 116 ms, 159 ms y 98 ms, respectivamente. Estos resultados garantizan una comunicación eficiente entre el cliente y el servidor y cumplen el requisito no funcional establecido de respuestas inferiores a 200 ms. Para no cargar el documento con imágenes necesarias, mostramos a continuación una captura de la interfaz de pruebas de Postman, en concreto de las pruebas del GET (7.2), representativa del resto de colecciones de los métodos.



Source	Environment	Iterations	Duration	All tests	Avg. Resp. Time
Runner	none	1	846ms	0	46 ms

Figura 7.2: Resultados GET Postman

7.3.3. Pruebas de Accesibilidad

Otro de los problemas de Lighthouse son las pruebas de accesibilidad, ya que son muy superficiales. La misma herramienta nos dice que necesitamos procesos manuales para auditar correctamente la página, proceso que llevaremos a cabo a continuación.

En la siguiente tabla presentamos los resultados de los conceptos más fundamentales de la normativa WCAG 2.1 (Guía para la accesibilidad en el contenido web). Esta normativa establece pautas a seguir por todos los desarrolladores que buscan adaptar su página para que el contenido sea perceptible, operable, comprensible y robusto para el mayor número de usuarios posible, incluyendo aquellos con discapacidades visuales, auditivas o neuronales. En nuestro caso, nos hemos centrado en evaluar las minusvalías visuales, recogiendo un subconjunto de los requisitos necesarios para cumplir esta normativa. Es importante señalar que este subconjunto no representa el mencionado rango, al igual que su cumplimiento no garantiza una experiencia de usuario óptima para todos los perfiles. El cumplimiento de toda la normativa WCAG 2.1 podría ser otro trabajo en sí mismo; nuestro objetivo ha sido abarcar en la medida de lo posible aquellos criterios que permiten asegurar una buena experiencia de usuario al mayor número de usuarios posible. Además, debemos recordar que la aplicación está diseñada para el uso interno de la clínica veterinaria, compuesta únicamente por veterinarios que cumplen diagnósticos visuales continuamente. Aunque la escalabilidad de la empresa obligue a contemplar otros roles como contabilidad o secretaría, adaptar la interfaz a personas

con discapacidad no ha sido una prioridad principal en este proyecto. En la siguiente tabla (7.4) se presenta un subconjunto de criterios ([World Wide Web Consortium \(W3C\) \(2018\)](#)) y qué se ha llevado a cabo para su cumplimiento.

7.4. Pruebas de Usabilidad

Para llevar a cabo las pruebas de usabilidad, hemos usado el método System Usability Scale (SUS) o Sistema de Escalas de Usabilidad. Se trata de un método propuesto por [Brooke et al. \(1996\)](#) que permite evaluar la usabilidad de cualquier sistema. Aunque es un método antiguo, se lleva usando de forma eficaz hoy en día para medir los parámetros de eficacia, eficiencia y satisfacción. Para usar este método debemos hacer uso de diez enunciados predefinidos y cinco posibles respuestas para cada uno.

7.4.1. Estrategia

Preguntas

1. Creo que me gustaría utilizar este sistema con frecuencia.
2. Encontré el sistema innecesariamente complejo.
3. Pensé que el sistema era fácil de usar.
4. Creo que necesitaría el apoyo de un técnico para poder utilizar este sistema.
5. Encontré que las diversas funciones de este sistema estaban bien integradas.
6. Pensé que había demasiada inconsistencia en este sistema.
7. Me imagino que la mayoría de la gente aprendería a utilizar este sistema rápidamente.
8. Encontré el sistema muy complicado de usar.
9. Me sentí muy seguro usando el sistema
10. Necesitaba aprender muchas cosas antes de empezar con este sistema.

Respuestas

1. Totalmente en desacuerdo.
2. En desacuerdo.
3. Neutro.
4. De acuerdo.
5. Totalmente de acuerdo.

Criterio	Descripción	Cumple
Contraste de colores	El contenido debe disponer de un contraste suficiente entre el texto y el fondo para asegurar la legibilidad.	Hemos elegido colores que tengan un nivel de contraste alto, en concreto 5.03:1 (WebAIM (2025)).
Etiquetas semánticas	Se emplean etiquetas HTML semánticas (<code><header></code> , <code><nav></code> , <code><main></code> , <code><footer></code> , <code><article></code> , etc.) para estructurar el contenido de manera lógica.	Hemos mantenido la estructura estándar de HTML para la generación de documentos a lo largo de toda la página.
Navegación por teclado	Toda la funcionalidad del sitio debe ser accesible utilizando únicamente el teclado, sin necesidad de un dispositivo apuntador.	Los elementos de Reactstrap y Material UI permiten navegación por teclado.
Alternativas textuales	Todas las imágenes y elementos multimedia deben incluir alternativas textuales descriptivas (atributo <code>alt</code>) para usuarios con discapacidad visual.	No existen elementos multimedia, y todos los iconos cuentan con un texto alternativo.
Formularios accesibles	Los formularios deben contar con etiquetas asociadas, instrucciones claras y mensajes de error comprensibles para facilitar la interacción.	Los formularios permiten navegación por teclado y tienen asociados etiquetas. Los mensajes de error son descriptivos y se muestran bajo cada campo.
Uso de ARIA	Se deben utilizar atributos ARIA apropiados en componentes complejos para mejorar la accesibilidad de la interfaz.	No se han considerado elementos lo suficientemente complejos como para añadir atributos ARIA.
Compatibilidad con lectores de pantalla	La estructura y contenido de la página deben ser fácilmente interpretables por lectores de pantalla, asegurando una navegación coherente.	La navegación por teclado y el uso de etiquetas y textos alternativos proporcionan una buena experiencia con lectores de pantalla.
Estabilidad visual	Se debe minimizar el movimiento inesperado de elementos en la pantalla (CLS bajo) para evitar confusiones durante la carga.	Se cumple con la excepción de la página “Historial”. Esto se tendrá en cuenta para futuras iteraciones del código.
Diseño responsive	La interfaz debe adaptarse correctamente a diferentes tamaños de pantalla, garantizando una experiencia óptima en dispositivos móviles y de escritorio.	El proyecto se ha diseñado con este requisito como prioridad, haciendo uso de Reactstrap para generar interfaces <i>responsive</i> .
Enlaces descriptivos	Todos los enlaces deben tener textos descriptivos que indiquen claramente su destino o función.	El proyecto no presenta enlaces externos y los enlaces internos vienen asociados con un texto visual y otro alternativo, describiendo cuál es su función.

Tabla 7.4

Verificación de criterios de accesibilidad según WCAG 2.1

Encuestados

Como mencionamos anteriormente, los encuestados están formados por un grupo de 20 personas, incluyendo usuarios finales, en un rango de edades comprendido entre los 23 y 73 años, con diversidad en su conocimiento (valorados subjetivamente en una escala del 1 al 5), dispositivo que usan y rango de discapacidades auditivas, visuales o cognitivas.

Cálculo del resultado

Para calcular el resultado se sumarán las respuestas de las preguntas impares y se les restará un 5; por otra parte, se sumarán las respuestas de los enunciados pares y se le restará ese total a 25. Finalmente, se suman ambos resultados y se multiplica por 2,5. La fórmula es:

$$\text{SUS} = 2,5 \times \left[\left(\sum_{i \in \{1,3,5,7,9\}} q_i - 5 \right) + \left(25 - \sum_{i \in \{2,4,6,8,10\}} q_i \right) \right]$$

Escenarios

A los encuestados se les pide completar un subconjunto representativo de los casos de uso del sistema. Estos son:

1. Crear un cliente nuevo con una mascota y dos contactos.
2. Darle una cita al nuevo cliente.
3. Terminar la cita anotando que la mascota ha perdido algo de peso y recibido un tratamiento antiparasitario.

7.4.2. Resultados

El análisis de las respuestas del test SUS concluyó con una puntuación media de 81.125, lo que supera el promedio establecido por Jeff Sauro de 68, indicando una usabilidad satisfactoria.

A continuación mostramos tres figuras que representan diferentes aspectos de los resultados obtenidos. La primera (Figura 7.3) compara la puntuación media adquirida por nuestro sistema (flecha negra) en comparación al promedio establecido por Jeff Sauro, ofreciendo un punto de referencia para situar el valor en su contexto. La segunda (Figura 7.4a) corresponde a un gráfico de red en el que nos permite apreciar cómo las respuestas positivas (impares) obtienen puntuaciones elevadas y las negativas (pares) un valor bajo, cumpliendo con lo deseado. Por último, la tercera (Figura 7.4b) se trata de un diagrama de dispersión que confirma la tendencia comentada anteriormente; a mayor nivel de conocimiento del usuario, mayor puntuación otorga al sistema.

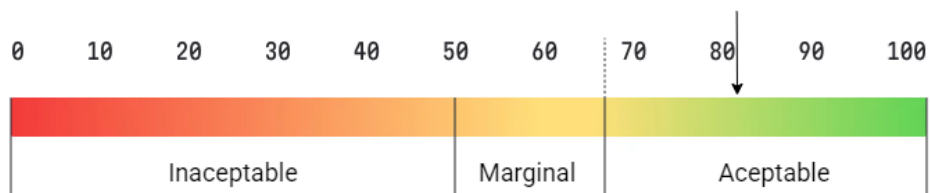
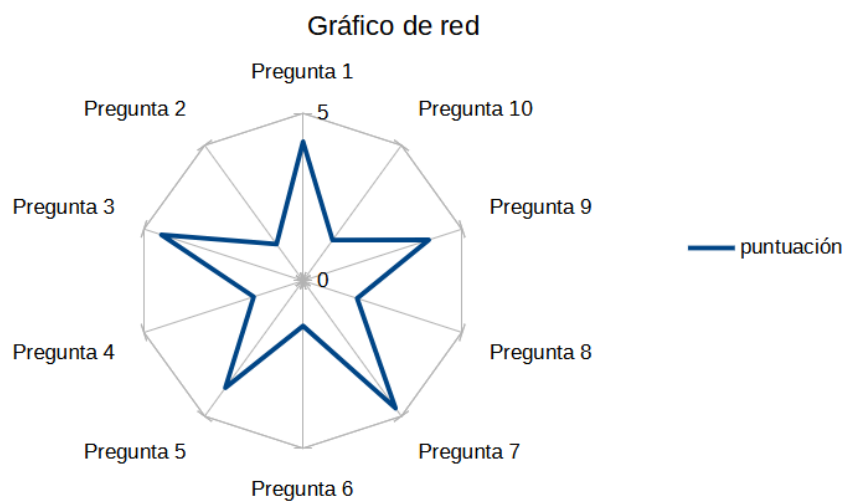
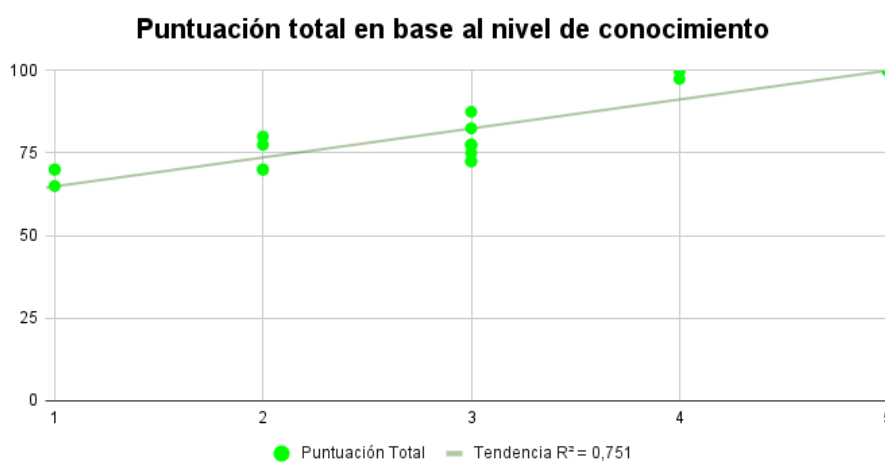


Figura 7.3: Puntuación media según el SUS



(a) Gráfico de red



(b) Gráfico de dispersión

Figura 7.4: Gráficos de los datos recogidos por el test SUS

Se observó que la percepción de la facilidad mejora conforme aumenta el nivel de conocimientos técnicos; los usuarios con niveles 4 y 5 asignaron puntuaciones elevadas a la herramienta, entre los 90-100, mientras que aquellos con niveles 1 y 2 indicaron calificaciones algo inferiores, en torno a 68-74. A pesar de estas diferencias, ninguno de los usuarios reportó dificultades críticas, lo que sugiere que el sistema es intuitivo para un amplio sector de la población.

Sin embargo, cabe destacar que había una coincidencia en todos aquellos usuarios que ponderaron debajo de un 80 al trabajo: no completaron satisfactoriamente algún caso de uso, concretamente, el tercer caso de uso planteado de “terminar una cita”. Observamos durante las pruebas que el tener que volver a la vista “Inicio” tras haber hecho la navegación y los dos primeros casos de uso en la vista “Clientes”, este grupo de usuarios no supo resolver correctamente este caso de uso. Este problema se resolvería haciendo uso de la vista “Citas” que originalmente estaba planteada. En cuanto a la edad, el dispositivo o la accesibilidad, no se detectó ningún patrón en las puntuaciones.

En resumen, el análisis realizado mediante la estrategia SUS indica que la usabilidad de la herramienta es aceptable. Los resultados sugieren que el factor más determinante es el nivel de conocimientos técnicos del usuario, lo que resulta positivo considerando la posibilidad de incorporar nuevos veterinarios de diversos perfiles y la flexibilidad de uso del sistema en cualquier momento y lugar.

7.4.3. Pruebas de Diseño Responsivo

A lo largo del proyecto, hemos destacado la importancia de garantizar que el sistema funcione sin problemas en dispositivos de diferentes tamaños, con especial atención a los móviles. En este caso, el cliente utiliza principalmente un portátil con una resolución de 1706x805 y un iPhone 14 Pro Max. Para optimizar el desarrollo y las pruebas, empleamos las DevTools de Chrome, lo que nos permitió emular estas resoluciones y verificar el correcto funcionamiento de la interfaz.

Para no saturar el grueso del proyecto, mostraremos tan solo la página de “Inicio” en ambos dispositivos; la pantalla del portátil del cliente (7.5) y su dispositivo móvil (7.6) usando las DevTools de Chrome:

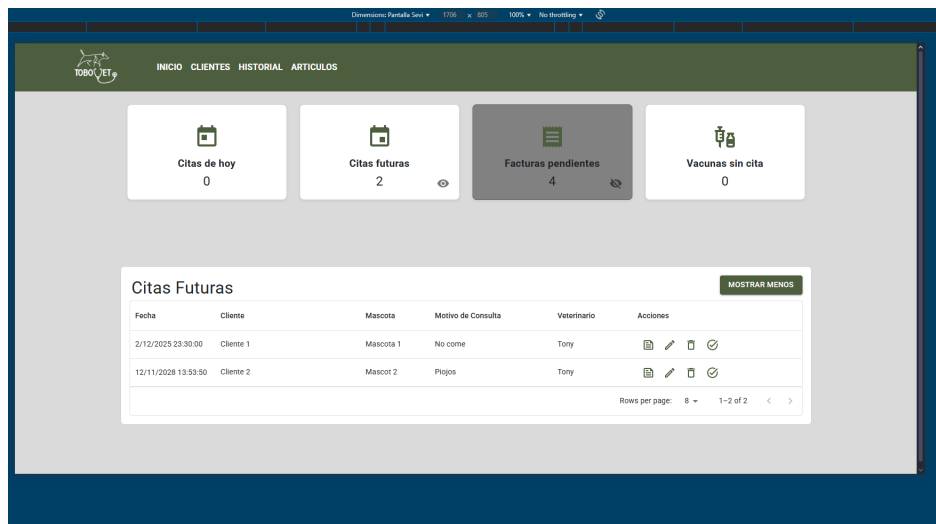
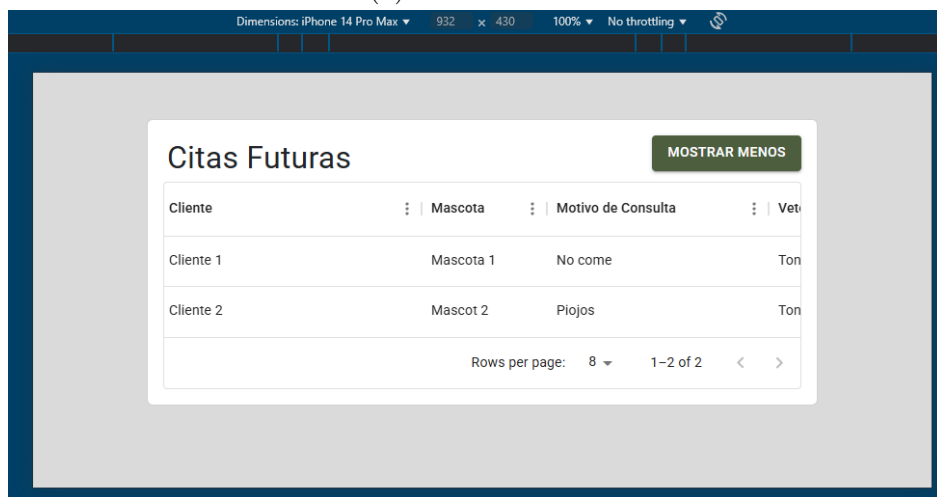


Figura 7.5: Vista de Inicio en la pantalla del portátil del cliente



(a) Modo retrato



(b) Modo paisaje

Figura 7.6: Vista de Inicio en el móvil del cliente

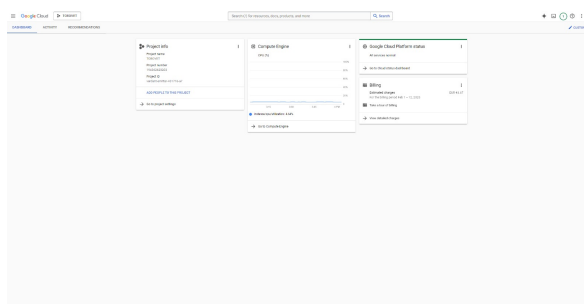
8. Despliegue del Sistema

Este capítulo recoge la arquitectura física planteada para el sistema, las instrucciones para su despliegue y las instrucciones para la operación y mantenimiento del nivel de servicio.

8.1. Arquitectura Física

El sistema se despliega en una máquina virtual (VM) en Google Cloud Platform (GCP) (8.1). Esta VM que ejecuta un sistema operativo Linux (en nuestro caso, Ubuntu), está configurada con los recursos necesarios, CPU, RAM y almacenamiento, para soportar tanto la API backend como la aplicación front-end. La infraestructura se organiza en dos directorios principales: uno para la API (/tobovetapi) y otro para la aplicación (tobovetapp). La VM tiene instalados los recursos necesarios para usar la aplicación, Node.js, npm y Git. En resumen, la arquitectura física permite que ambos componentes del sistema se ejecuten de forma simultánea y coordinada, facilitando la gestión de carga y siempre pensando en la escalabilidad del sistema.

Basic information	
Name	tobovet-vm
Instance id	7432434079818648485
Description	None
Type	Instance
Status	Running
Creation time	Aug 8, 2024, 2:08:44 PM UTC+02:00
Location	europa-west4-c
Instance template	None
In use by	None
Physical host	None
Maintenance status	—
Reservations	Automatically choose
Labels	google-ec-arc: vm_iaos-gcloud
Tags	—
Deletion protection	Disabled
Confidential VM service	Disabled
Preserved state size	0 GB
Machine configuration	
Machine type	e2-small (2 vCPUs, 2 GB Memory)
CPU platform	Intel Broadwell
Minimum CPU platform	None
Architecture	x86_64
vCPUs to core ratio	—
Custom visible cores	—
All-core turbo-only mode	—
Display device	Disabled Enable to use screen capturing and recording tools
GPUs	None
Resource policies	



(a) Información de la GCP

(b) Panel de control de la GCP

Figura 8.1: Configuración de la GCP

8.2. Instrucciones de despliegue

A continuación, repasaremos las instrucciones de despliegue para el proyecto en una máquina virtual. Para la instalación en local, es necesario seguir los pasos de

instalación en el anexo A de este documento.

8.2.1. Requisitos Previos

Para instalar el sistema, es necesario contar con:

- Una máquina virtual en GCP, configurada con una distribución Linux (por ejemplo, Ubuntu).
- Software instalado: Node.js, npm y Git
- Acceso SSH a la VM para ejecutar los comandos de despliegue y mantenimiento.

8.2.2. Procedimientos de instalación

El despliegue del sistema se ha simplificado mediante la definición de alias en la VM para automatizar tareas comunes. Los alias configurados son:

- Iniciar API:

```
1 alias tobovet_back='cd ~/tobovetapi; npm i; pkill  
-f tobovetapi; nohup npm start </dev/null &'  
2
```

Este comando instala las dependencias, finaliza procesos previos relacionados con la API y arranca el servidor

- Iniciar App:

```
1 alias tobovet_front='cd ~/tobovetapp; npm i; npm  
run build; pkill -f tobovetapp; nohup npm run preview  
&'  
2
```

Con este alias se instalan las dependencias, se genera la versión de producción de la aplicación y se inicia el front-end en modo de previsualización.

- Actualizar el Sistema:

```
1 alias update_tobovet='cd ~/tobovetapp; git pull;  
cd ~/tobovetapi; git pull'  
2
```

Este comando actualiza el código fuente de ambos componentes a partir del repositorio remoto.

8.3. Instrucciones para la operación del sistema y mantenimiento del nivel de servicio

Para asegurar el correcto funcionamiento y mantenimiento de nuestro sistema, recomendamos las siguientes prácticas:

- **Monitoreo y Gestión de Logs.** La API genera una gran cantidad de Logs para facilitar la depuración, deben revisarse periódicamente para identificar e intentar resolver las incidencias. Es recomendable en un futuro configurar un sistema de monitoreo para notificar sobre posibles errores o caídas del sistema.
- **Procedimientos de Backup.** Realizar copias de seguridad regulares, tanto del código fuente como de la BD, especialmente en *sprints* de desarrollo. Se recomienda usar scripts automatizados para este proceso.
- **Actualización y Mantenimiento.** Como hemos visto anteriormente, las actualizaciones se gestionan por alias, para facilitar la incorporación de cambios y mejora. Para ello se recomienda planificar junto al cliente ventanas de mantenimiento para poder aplicar actualizaciones sin interrumpir el servicio.
- **Reinicio y Recuperación.** En caso de incidencias, se pueden usar los alias definidos anteriormente para reiniciar los servicios. El uso del comando *nohup* asegura que los procesos continúen ejecutándose en segundo plano, incluso si se cierra la sesión SSH.

Parte III

Epílogo

9. Conclusiones

En este último capítulo se detallan las lecciones aprendidas tras el desarrollo del presente proyecto y se identifican las posibles oportunidades de mejora sobre el software desarrollado.

9.1. Objetivos alcanzados

Después de varios meses de trabajo, podemos afirmar que hemos cumplido todos los objetivos que establecimos en la introducción de este documento. El cliente afirma que sus expectativas han sido superadas y tiene una opinión muy positiva sobre el trabajo y el producto final. El sistema proporciona una herramienta útil para la gestión del negocio, además de mejorar la experiencia de usuario optimizando procesos administrativos. Entre los objetivos alcanzados destacan la implementación de un sistema de gestión para facilitar la relación entre cliente, mascota y veterinario, la automatización de procesos clave como la generación de documentos oficiales y, como necesidad no funcional, la creación de un diseño flexible y modular que permite futuras actualizaciones y ampliaciones del sistema.

9.2. Lecciones aprendidas

Desde un punto de vista tecnológico, uno de los aprendizajes más relevantes ha sido aprender el framework de React y usarlo adecuadamente para garantizar la responsividad. Iniciar un proyecto desde cero con un lenguaje con el que no se está familiarizado ha sido todo un reto y extremadamente gratificante cuando todo funcionaba como se esperaba. En términos procedimentales, la teoría reforzaba la necesidad de planificar de manera detallada, pero la experiencia ha aportado flexibilidad al marco teórico de la ingeniería del software y los métodos de desarrollo. Por ejemplo, podemos diferenciar dos claras etapas en el proceso de creación del proyecto: los primeros *sprints* que trataban de sacar adelante los objetivos propuestos para el plazo establecido y los posteriores, que, mucho más relajados, permitían mejoras, mantenimientos y la capacidad de mirar a lo anterior con la experiencia diaria como foco. Cuantitativamente, se estima que el proyecto ha requerido entre 600 y 700 horas de dedicación, repartidas en jornadas desde 40 horas semanales al inicio hasta 10 o 15 durante el mantenimiento. Esta experiencia ha sido invaluable no solo para poder añadir nuevas habilidades técnicas al portfolio, sino también para reforzar (o aprender) competencias transversales como la gestión del tiempo, la comunicación y la capacidad de adaptarse a cambios inesperados.

9.3. Trabajo futuro

Por la naturaleza de este proyecto, el trabajo futuro y las oportunidades de mejora se han ido implementando con el tiempo en el que se ha estado desarrollando esta memoria, como parte del mantenimiento del proyecto. Aun así, creemos que hay varios aspectos en los que se puede mejorar y algunos nuevos requisitos no funcionales que podrían implementarse en el trabajo que, además, servirían tanto al negocio como al desarrollador a modo de toma de contacto con otras áreas de la informática. Algunos de estos requisitos no funcionales serían:

1. Implementación de un inicio de sesión de cliente, para poder consultar el historial de su mascota, así como las próximas citas.
2. Implementación de un calendario nativo o una API con Google Calendar que permita a los veterinarios e incluso a los usuarios seguir las citas de manera aún más cómoda.
3. Rediseñar completamente la web para virar de un diseño basado en tablas a un diseño basado en tarjetas, de forma que se asemeje a una aplicación móvil.

Bibliografía

- Anderson, D. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press.
- Atlassian (2025). JIRA Software: Agile Project Management Tool. <https://www.atlassian.com/software/jira/>.
- Aubry, C. (2018). *Cree su primer sitio web: del diseño a la realización*. Objetivo: Web. ENI.
- Bootstrap Team (2025). Introduction to Bootstrap 5. <https://getbootstrap.com/>.
- Brooke, J. et al. (1996). SUS-A quick and dirty usability scale. *Usability evaluation in industry*, 189(194):4–7.
- Facebook (2025). Optimizing React Performance. <https://reactjs.org/docs/optimizing-performance.html>.
- Facebook Inc. (2025). Jest: Delightful JavaScript Testing. <https://jestjs.io/>.
- Figma (2025). Figma – Interface Design Tool. <https://www.figma.com/>.
- GitHub Inc. (2025). GitHub Actions. <https://github.com/features/actions>.
- Google (2025a). Google Cloud Platform Documentation. <https://cloud.google.com/docs>.
- Google (2025b). Lighthouse: Automated Website Quality Tool. <https://developers.google.com/web/tools/lighthouse>.
- JWT.io (2025). JSON Web Tokens. <https://jwt.io/>.
- MUI (2025). Material UI: React Component Library. <https://mui.com/>.
- Node.js Foundation (2025). Node.js Documentation. <https://nodejs.org/en/docs/>.
- npm, Inc. (2025). npm Documentation. <https://docs.npmjs.com/>.
- Object Management Group (2017). Unified Modeling Language (UML) Specification, Version 2.5.1. <https://www.omg.org/spec/UML/2.5.1/>.
- Oracle Corporation (2025). MySQL Reference Manual. <https://dev.mysql.com/doc/>.
- Postman Inc. (2025). Postman API Platform. <https://www.postman.com/>.
- React Team (2025a). A JavaScript Library for Building User Interfaces. <https://reactjs.org/>.
- React Team (2025b). React Architecture Best Practices. <https://reactjs.org/docs/faq-structure.html>.

- Reactstrap Team (2025). Reactstrap Components for React. <https://reactstrap.github.io/>.
- Richardson, L., Amundsen, M., y Ruby, S. (2013). *RESTful Web APIs: Services for a Changing World*. O'Reilly Media.
- Schwaber, K. y Sutherland, J. (2020). The Scrum Guide. <https://scrumguides.org/>.
- Sommerville, I. (2016). *Software Engineering*. Pearson, 10 edition.
- Tarifa Luz Hora (2025). Precio kWh España a 11 Marzo 2025: Evolución y Comparativa. <https://tarifaluzhora.es/info/precio-kwh>.
- Testing Library (2025). Jest-dom: Custom Jest matchers for the DOM. <https://github.com/testing-library/jest-dom>.
- The Git Project (2025). Git Documentation. <https://git-scm.com/doc>.
- W3C (2025). CSS Working Group. <https://www.w3.org/Style/CSS/>.
- WebAIM (2025). Contrast Checker. <https://webaim.org/resources/contrastchecker/>.
- WHATWG (2025). HTML Living Standard. <https://html.spec.whatwg.org/multipage/>.
- World Wide Web Consortium (W3C) (2018). Web Content Accessibility Guidelines (WCAG) 2.1. <https://www.w3.org/TR/WCAG21/>.
- You, E. y Contributors, V. (2023). Vite: Next Generation Frontend Tooling. <https://vitejs.dev/>.

Parte IV

Anexos

A. Manual del desarrollador

A continuación se recogen las instrucciones necesarias para evolucionar el software. Este manual está dirigido a los desarrolladores que pretenden extender o modificar el código fuente, con el fin de incorporar nuevas funcionalidades o modificar las ya existentes.

A.1. Manual de instalación

Este manual indica como instalar las herramientas necesarias para un sistema con el S.O Windows 11.

A.1.1. MySQL

A continuación se describirá cómo instalar MySQL Workbench en su versión 8.0.41 para hacer uso de su interfaz gráfica. Si se quiere acceder a una base de datos ya montada, puede saltarse este paso.

Requerimientos

1. Microsoft .NET Framework 4.5.2
2. Microsoft Visual C++ 2015-2022 Redistributable
3. Microsoft Windows 11 o Windows Server 2022

Instalación

Descargar el instalador oficial de [https://dev.mysql.com/downloads/windows/installer/..](https://dev.mysql.com/downloads/windows/installer/). Esta aplicación puede instalar, actualizar y manejar todos los productos de MySQL que necesitará.

1. Asegúrese de contar con una cuenta con privilegios de Administrador o Usuario avanzado.
2. Haga clic derecho sobre el archivo MSI descargado y seleccione la opción **Instalar** en el menú contextual, o bien, haga doble clic sobre el archivo.
3. En la ventana de configuración, seleccione el tipo de instalación **Completa**
4. Por defecto, MySQL Workbench se instalará en la siguiente ruta:

C:\[Program Files]\MySQL\MySQL Workbench 8.0 edition_type

donde [Program Files] es el directorio predeterminado para programas en su sistema, generalmente C:\Program Files\.

Conectarse a una BD:

Si ya tiene una BD montada, puede conectarse directamente a ella pulsando el icono de “+” y rellenando los campos necesarios en el modal que aparece en la pantalla.

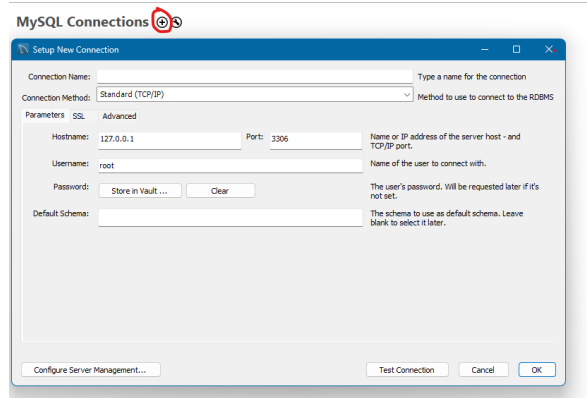


Figura A.1: Nueva conexión MySQL

Creación de una nueva base de datos

Para gestionar la información en la aplicación, es necesario crear una base de datos en MySQL. Existen dos métodos para hacerlo: usando la línea de comandos o MySQL Workbench.

1. Abra MySQL Workbench y conéctese al servidor MySQL.
2. En la pestaña **Navigator**, haga clic en **Schemas**.
3. Haga clic derecho en el área de esquemas y seleccione **Create Schema...**
4. En el campo **Schema Name**, ingrese el nombre que desee.
5. Haga clic en **Apply** y luego en **Finish** para completar la creación.
6. La base de datos estará disponible en la lista de esquemas.

Con estos pasos, la base de datos estará lista para ser utilizada en el proyecto.

A.1.2. Visual Studio Code (VS Code)

Visual Studio Code es un editor de código ligero y versátil con soporte para múltiples lenguajes de programación y una amplia variedad de extensiones que facilitan el desarrollo. A continuación, se describe el proceso de instalación y configuración recomendada.

Requerimientos

1. Microsoft Windows 11 o Windows Server 2022.
2. Acceso a una cuenta con privilegios de Administrador o Usuario avanzado.
3. Conexión a internet para la instalación de extensiones.

Instalación

Para instalar Visual Studio Code en el sistema, siga estos pasos:

1. Descargar el instalador oficial desde la <https://code.visualstudio.com/Download>. El archivo tiene el nombre:

VSCodeUserSetup-{version}.exe

donde {version} corresponde a la versión más reciente del programa.

2. Ejecutar el instalador y seguir las instrucciones en pantalla.
3. Por defecto, VS Code se instalará en:

C:\Users\{Usuario}\AppData\Local\Programs\Microsoft VS Code

donde {Usuario} corresponde al nombre del usuario en el sistema.

4. Durante la instalación, habilitar las siguientes opciones para facilitar su uso:

- **Agregar VS Code al PATH:** Permite abrir VS Code desde la terminal con el comando:

```
1 code .
2
```

Esto facilita la apertura del editor en cualquier directorio de trabajo.

- **Asociar VS Code con archivos de texto y código fuente:** Permite abrir archivos con doble clic directamente en VS Code.

5. Una vez completada la instalación, reiniciar la terminal para que los cambios en las variables de entorno surtan efecto.

Extensiones recomendadas

Para mejorar la experiencia de desarrollo, recomendamos las extensiones que se han usado para este proyecto y que pueden instalarse desde el *Marketplace* de VS Code:

- **Better Comments:** Permite categorizar el tipo de comentarios.
- **Prettier - Code formatter:** Formatea automáticamente el código para mantener una estructura clara y consistente.
- **Error Lens:** Resalta aún más las líneas que producen errores en el código.

- **Material Icon Theme:** Mejora la interfaz visual con iconos atractivos para los archivos y carpetas.
- **GitLens:** Proporciona información detallada sobre cambios en el código y su historial en Git.
- **REST Client:** Permite realizar pruebas de API directamente desde VS Code.
- **Extension Pack for Java:** Colección de extensiones para escribir y depurar código en Java.
- **GitHub Copilot:** Autocompletado con IA para un código más robusto y con más rapidez.
- **JS JSX Snippets:** Snippets para React.

Una vez instaladas estas extensiones, VS Code estará listo para su uso con un entorno optimizado para el desarrollo.

A.1.3. Node.js y npm

<https://nodejs.org/en/download/current> Para ejecutar aplicaciones desarrolladas en Node.js, es necesario instalar tanto Node.js como su gestor de paquetes npm. Se describe el proceso de instalación utilizando *fast Node Manager* (fnm).

Requerimientos

1. Microsoft Windows 11 o Windows Server 2022.
2. Acceso a una cuenta con privilegios de Administrador o Usuario avanzado.
3. Conexión a internet para descargar los paquetes necesarios.

Instalación

Para instalar Node.js y npm en el sistema, siga los siguientes pasos:

1. Instalar **fnm (fast Node Manager)** ejecutando el siguiente comando en una terminal con permisos de administrador:

```
1 winget install Schniz.fnm
2
```

2. Descargar e instalar la versión estable más reciente de Node.js utilizando fnm:

```
1 fnm install 22
2
```

Esto instalará la versión 22 de Node.js junto con npm.

3. Verificar que la instalación se realizó correctamente ejecutando los siguientes comandos:


```
1 node -v
2
```

Este comando debería devolver un resultado similar a:

v22.14.0

```
1 npm -v
2
```

El resultado esperado será:

10.9.2

Una vez completados estos pasos, el entorno de Node.js y npm estará listo para su uso.

Instalación de dependencias

Una vez instalado Node.js y npm, es necesario instalar las dependencias del proyecto para su correcto funcionamiento. Para ello, ejecute el siguiente comando en la raíz del proyecto:

```
1 npm install
```

Este comando instalará las siguientes dependencias:

Dependencias principales (dependencies) Son los paquetes esenciales para la ejecución de la aplicación:

- **@emotion/styled** - Estilos con Emotion.
- **@fontsource/roboto** - Fuente Roboto.
- **@fortawesome/fontawesome-svg-core** - Core de FontAwesome.
- **@fortawesome/react-fontawesome** - FontAwesome para React.
- **@fullcalendar/core, daygrid, interaction, react, timegrid** - Librería FullCalendar para gestión de eventos.
- **@mui/icons-material, material, styled-engine-sc, x-data-grid, x-date-pickers** - Componentes de Material UI.
- **axios** - Cliente HTTP para API.
- **bootstrap** - Framework CSS.
- **dayjs** - Manipulación de fechas.
- **react, react-dom** - Librerías base de React.
- **react-svg** - Renderizado de SVG en React.
- **react-to-pdf** - Conversión de componentes React a PDF.

- **reactstrap** - Componentes de Bootstrap para React.
- **styled-components** - Estilos en React con CSS-in-JS.

Dependencias de desarrollo (devDependencies**)** Son paquetes necesarios solo durante el desarrollo del proyecto:

- **@types/node, @types/react, @types/react-dom** - Tipados para TypeScript.
- **@typescript-eslint/eslint-plugin, @typescript-eslint/parser** - Reglas ESLint para TypeScript.
- **@vitejs/plugin-react** - Plugin de React para Vite.
- **dotenv** - Carga de variables de entorno.
- **eslint, eslint-config-prettier, eslint-plugin-import, eslint-plugin-jsx-a11y, eslint-plugin-react, eslint-plugin-react-hooks, eslint-plugin-react-refresh** - Herramientas para análisis de código y reglas de estilo.
- **prettier** - Formateo de código.
- **react-router-dom** - Enrutamiento en React.
- **sass** - Preprocesador CSS.
- **typescript** - Lenguaje TypeScript.
- **vite, vite-tsconfig-paths** - Vite y soporte para rutas en TypeScript.

Después de instalar las dependencias, el entorno estará listo para ejecutar la aplicación con:

```
1 npm start
```

A.1.4. React

React es una biblioteca de JavaScript utilizada para la creación de interfaces de usuario interactivas y eficientes. A continuación, se describe el proceso de instalación y configuración inicial de un proyecto en React.

Requerimientos

1. Node.js (versión 18 o superior) y npm.
2. Acceso a una terminal con permisos adecuados para instalar paquetes.
3. Conexión a internet para descargar dependencias.

Creación de un nuevo proyecto en React

Para iniciar un nuevo proyecto en React, siga estos pasos:

1. Abrir una terminal y ejecutar el siguiente comando:

```
1 npx create-react-app my-app
2
```

donde **my-app** es el nombre del proyecto. Este comando descargará y configurará automáticamente todas las dependencias necesarias.

2. Acceder al directorio del proyecto:

```
1 cd my-app
2
```

3. Iniciar el servidor de desarrollo:

```
1 npm start
2
```

Esto abrirá la aplicación en el navegador en la dirección:

<http://localhost:3000/>

Consideraciones adicionales

Si anteriormente se instaló **create-react-app** de forma global, se recomienda desinstalarlo para evitar conflictos y asegurarse de que siempre se utilice la última versión:

```
1 npm uninstall -g create-react-app
```

Preparación para producción

Cuando la aplicación esté lista para ser desplegada en producción, se puede generar una versión optimizada con:

```
1 npm run build
```

Esto creará una carpeta **build/** con una versión minificada del proyecto lista para su implementación en un servidor web.

A.2. Consideraciones generales sobre el desarrollo

En este apartado se detallan los aspectos importantes a tener en cuenta a la hora de modificar y extender el código fuente, así como las guías de estilo y las directrices para realizar pruebas sobre las nuevas mejoras introducidas.

A.2.1. Guías de estilo de codificación

Para mantener la coherencia y la legibilidad del código, es importante seguir las siguientes guías de estilo:

- Utilizar nombres de variables y funciones descriptivos y en inglés.
- Seguir la convención de nombres camelCase para variables y funciones, y Pascal-Case para clases.
- Indentar el código con 4 espacios en lugar de tabulaciones.
- Incluir comentarios claros y concisos para explicar la lógica del código y cualquier decisión de diseño importante.
- Mantener las funciones y métodos cortos y enfocados en una única responsabilidad.
- Evitar el uso de números mágicos; en su lugar, definir constantes con nombres descriptivos.

A.2.2. Prácticas recomendadas para el desarrollo

Al modificar o extender el código fuente, es importante seguir estas prácticas recomendadas:

- Realizar cambios incrementales y probar cada cambio de manera individual.
- Utilizar control de versiones (por ejemplo, Git) para gestionar los cambios en el código y facilitar la colaboración con otros desarrolladores.
- Crear ramas específicas para cada nueva funcionalidad o corrección de errores, y fusionarlas con la rama principal solo después de haber realizado pruebas exhaustivas.
- Reutilizar los componentes y mantener un cierto grado de modularidad.
- Mantener la documentación actualizada con cada cambio realizado en el código.

A.2.3. Directrices para realizar pruebas

Para garantizar la calidad del software, es fundamental seguir estas directrices al realizar pruebas sobre las nuevas mejoras introducidas.

Siguiendo estas consideraciones generales, los desarrolladores podrán modificar y extender el código fuente de manera eficiente y coherente, garantizando la calidad y mantenibilidad del software.

B. Manual de usuario

Este manual tiene como objetivo guiar al usuario en el uso de la aplicación una vez desplegada. A lo largo de este documento, se explicarán tres escenarios clave que ilustran el funcionamiento de ToBoVet. Para todos los casos presentados, se asume que un desarrollador ha dado de alta al usuario en el sistema y que este ha iniciado sesión correctamente.

B.1. Escenario 1: Creación de un nuevo usuario

En este escenario, se explicará cómo registrar un nuevo cliente en el sistema, junto con dos contactos asociados y dos mascotas. Este proceso es intuitivo y sigue una estructura similar a cualquier otro caso de creación dentro de la aplicación. Para ilustrarlo, se apoyará en las figuras [B.1](#) y [B.2](#).

1. Acceder a la sección de clientes

- a) Desde cualquier página, usamos la barra de navegación para acceder a la pestaña “Clientes”.
- b) En esta tabla se muestra un registro por cada mascota registrada en el sistema junto a sus datos principales, además, al final de cada registro, se encuentra distintas acciones como editar el usuario, facturar o crear una nueva cita.
- c) Para facilitar la navegación, la tabla cuenta con un sistema de filtrado en la parte superior, que permite buscar por cliente, mascota, NHC o chip.
- d) También es posible ordenar los registros haciendo *clic* sobre los nombres de las columnas o personalizar la estructura de la tabla a través del menú de opciones.
- e) En la parte inferior tenemos controles para seleccionar cuántos registros se muestran por página, así como botones para navegar entre ellas.

2. Añadir un nuevo cliente

- a) Hacer *clic* en el botón “Añadir Cliente” en la parte superior de la tabla.
- b) Se abrirá un modal con un formulario donde se deben completar los datos del propietario.
- c) En el panel derecho, se ingresan los datos de la mascota.
- d) En la parte inferior izquierda, se pueden añadir hasta 3 contactos asociados con el cliente.
- e) Si se requiere registrar más de una mascota o contacto, hacer *clic* en el botón correspondiente para agregar más campos.

- f) Una vez completada la información, pulsar “Añadir” para terminar de añadir el cliente.

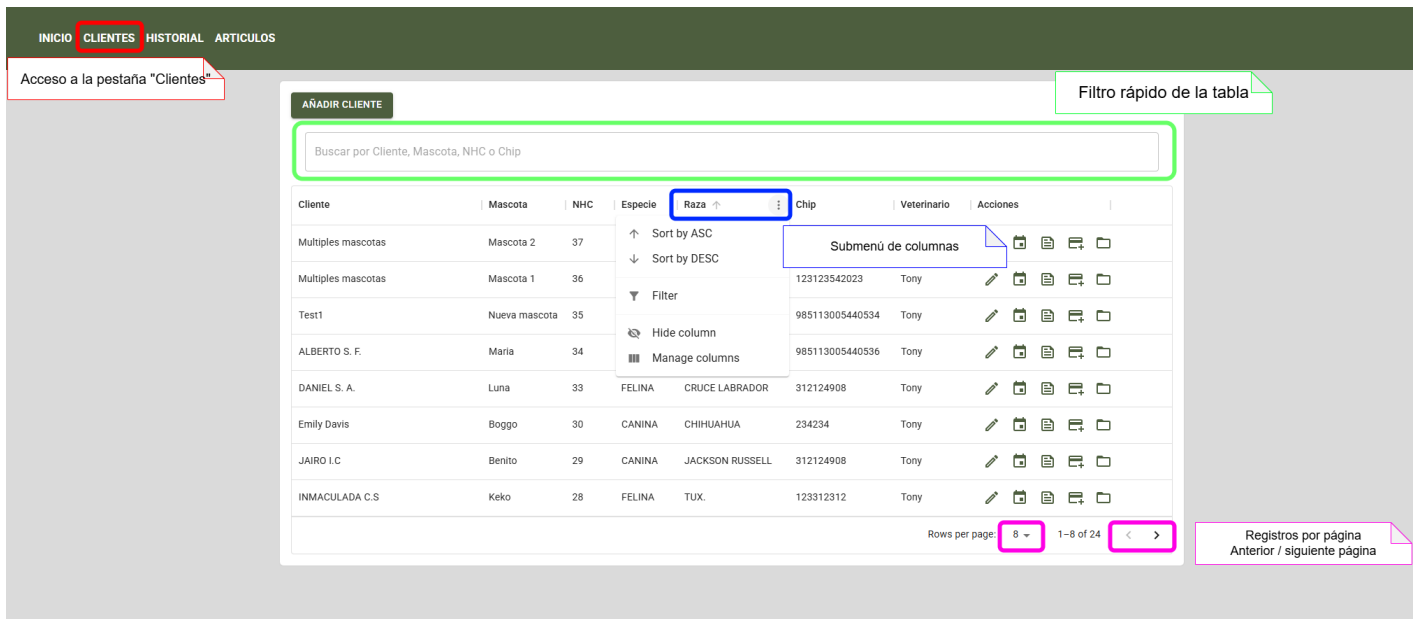


Figura B.1: Pantalla Clientes

B.2. Escenario 2: Agendar una cita

En este escenario, explicaremos el proceso de crear una cita para el cliente previamente registrado. Para ilustrarlo, haremos referencia a las figuras B.3 y B.4.

1. Buscar Cliente

- Accedemos a la pestaña “Clientes” desde la barra de navegación.
- Dentro de la pestaña “Clientes”, usamos el filtro en la parte superior de la tabla para buscar nuestro cliente, en este caso, buscaremos la mascota “Freud”.
- Hacemos *clic* en la acción “Crear Cita”

2. Crear Cita

- Se mostrará un modal para confirmar la hora y motivo de la consulta
- Tras rellenar los datos, se creará una nueva cita que podremos ver en la pestaña de “Inicio”, dependiendo de la fecha elegida, la cita mostrarse en “Citas de hoy” o “Citas futuras”.

3. Ver Citas Pasadas de un cliente

- En el mismo cliente seleccionamos la acción “Ver historial” que nos llevará a la página de historial de la mascota, donde podemos ver todas sus citas pasadas.

Añadir Cliente

Cliente

Nombre: Carlos de la Torre

Documento: 48980145Y

Dirección: Por favor rellena este campo.

Ciudad: El Puerto de Santa María

CP: 11500

Provincia: Cádiz

Observaciones:

Mascotas

Mascotas añadidas y botón para añadir otra

Freud Keko

Nombre: Keko

Fecha de Nacimiento: 10/03/2021 13:11

Chip: 123213340

Campo obligatorio sin rellenar

Especie: CANINA

Raza:

Genero: Macho

Esteril: Sí

Activo: ☒

Veterinario: Tony

Observaciones: Pelo corto

Observaciones Clínicas:

Contactos

Contacto 1 Contacto 2

Nombre: Segundo humano

Teléfono: 612723812

Correo: carlosjdlit@gmail.com

Contactos añadidos y botón para añadir otro

Añadir Cliente / Cancelar operación

AÑADIR CANCELAR

Figura B.2: Formulario de añadir cliente

AÑADIR CLIENTE

Freud

Cliente	Mascota	NHC	Especie	Raza	Chip	Veterinario	Acciones
Carlos de la Torre	Freud	39	CANINA	DRACO	12234123	Tony	

1 row selected

Rows per page: **Crear Cita** 1-1 of 1

Filtro para buscar registro

Acciones / Crear Cita

Figura B.3: Cliente encontrado

Crear Cita

Carlos de la Torre - Freud - 39

Fecha 15/03/2025 15:00

Motivo de consulta No come

Veterinario Tony

Crear Cita CREAR CANCELAR

Figura B.4: Modal para crear cita

B.3. Escenario 3: Terminar cita

En este escenario completaremos la visita recién creada añadiendo una vacunación. Nos apoyaremos en las figuras B.5, B.6, B.7 y B.8 para ilustrar el proceso.

1. Acceder a la cita programada

- Desde la barra de navegación, accedemos a “Inicio” pulsando el botón correspondiente o haciendo *clic* en el logo de ToBoVet.
- Comprobamos que la tabla que nos ocupa se encuentra visible. Si la tabla no se encontrara visible, pulsar el icono de visibilizar en el nombre de la tabla correspondiente en la parte superior de la página.
 - En caso de haber creado la cita para el mismo día, la tabla será “Citas de hoy”.
 - En caso de haber creado la cita para el futuro, la tabla será “Citas futuras”, si es dentro de más de seis meses, presionar el botón “Mostrar más” para mostrar la cita.

2. Llevar a cabo la cita

- Rellenar los campos necesarios de la cita.
- Al hacer *clic* en vacunas, todos los artículos del grupo “vacunas” se desplegarán, deberemos seleccionar los artículos necesarios de esta lista.
- Al seleccionar una vacuna, un nuevo campo aparecerá mostrando los días de eficacia que esa vacuna tiene asociados, aunque este campo puede modi-

ficarse a mano.

d) Si además se quisiera añadir un seguimiento, lo podremos hacer a través del botón “Nueva consulta”.

e) Guardar la cita.

f) Seleccionar la opción “Terminar cita”.

3. Resultado

a) Se habrá creado una próxima vacunación por cada vacuna que se le haya administrado.

b) En la tabla “Próximas vacunaciones” aparecerán estas vacunas, con solo una opción para crear la cita correspondiente.

c) Esta nueva cita, para mayor facilidad, tendrá un nuevo campo “Vacunas” donde, depende de la fecha escogida, se podrán elegir las vacunas correspondientes.

d) Una vez guardada, esta vacuna sin cita se convertirá en cita futura.

The screenshot displays the home page of a veterinary management system. At the top, there are four navigation cards: 'Citas de hoy' (1), 'Citas futuras' (2), 'Facturas pendientes' (4), and 'Vacunas sin cita' (0). The 'Citas futuras' card is highlighted with a blue box and labeled 'Ver/Esconder Tabla'. The 'Facturas pendientes' card is highlighted with a pink box and labeled 'Tabla no visible'. Below these cards, the 'Citas de hoy' table is highlighted with a green box and labeled 'Tabla que nos ocupa Botón "Terminar"'. The table has columns for Hora, Cliente, Mascota, Motivo de Consulta, Veterinario, Observaciones, and Acciones. The first row shows a consultation at 15:00:44 for Carlos de la Torre, with a dog named Freud, who is not eating, attended by Tony. The 'Acciones' column for this row contains a 'Terminar' button (a checkmark icon). Below the 'Citas de hoy' table, there is a 'Citas Futuras' table with columns for Fecha, Cliente, Mascota, Motivo de Consulta, Veterinario, and Acciones. The first row shows a future consultation on 2/12/2025 at 23:30:00 for Jessica Miller, with a dog named Bailey, who has a consultation (aaa), attended by Tony. The 'Acciones' column for this row contains a 'Terminar' button (a checkmark icon). The second row shows a future consultation on 12/11/2028 at 13:53:50 for a client named aaaadsa, with a new dog (Nueva mascota), who is far away (lejos), attended by Tony. The 'Acciones' column for this row contains a 'Terminar' button (a checkmark icon). The 'Citas Futuras' table has a 'MOSTRAR MENOS' button in the top right corner. The 'Citas de hoy' table has a 'Rows per page' dropdown set to 8 and a '1-1 of 1' indicator. The 'Citas Futuras' table has a 'Rows per page' dropdown set to 8 and a '1-2 of 2' indicator.

Figura B.5: Página de Inicio

Guardar Cita

Carlos de la Torre - Freud - nhc: 39

Fecha	15/03/2025 15:00	Tests	Pruebas generales
Veterinario	Tony	Diagnóstico	Ecografía
Motivo de consulta	No come	Pruebas Diagnósticas	
Anamnesis		Tratamiento	Selector Vacunas
Peso	23 kg	Vacunas	Rabia, Suplemento C
Condiciones (5)	4	Días hasta siguiente: Rabia	20
Temperatura	35 °C	Días hasta siguiente: Suplemento C	365
Sintomas		Desparasitación	Modificar días de eficacia
		Recomendaciones	No dar comida humana
		Completado	No

Botones de Nueva Consulta, Guardar y Cancelar

NUEVA CONSULTA

GUARDAR

CANCELAR

Figura B.6: Editar cita

Citas de hoy

0

Citas futuras

2

Facturas pendientes

4

Vacunas sin cita

1

Próxima Exp...

Cliente

Mascota

Vacuna

Día de Vac...

Acciones

4/4/2025 15:00:09

Carlos de la Torre

Freud

Rabia

15/3/2025 15:00:09

Crear cita de vacunación

1 row selected

Rows per page: 8 1-1 of 1

Figura B.7: Próximas vacunaciones

Crear Cita

Carlos de la Torre - Freud - 66

Fecha 15/03/2025 14:34

Motivo de consulta Vacunación

Veterinario Tony ▼

Vacunas

Di

Rabia

Selector Vacuna

Figura B.8: Crear cita con vacuna