UNIVERSIDAD DE EL SALVADOR
FACULTAD DE INGENIERÍA Y ARQUITECTURA
ESCUELA DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

CURSO DE ESPECIALIZACIÓN DE INGENIERÍA DE CALIDAD

TRABAJO DE TESINA

"DISEÑO E IMPLEMENTACIÓN DE UN PLAN DE PRUEBAS AUTOMATIZADAS Y MANUALES PARA UN SISTEMA DE GESTIÓN DE HISTORIALES MÉDICOS PARA LA VETERINARIA MITSUM"

PRESENTADO POR:

APELLIDOS, NOMBRES	CARNET
FLORES MENDOZA FABIO ERNESTO	FM19038
GUTIERREZ ESCOBAR JUAN MANUEL	GE19020
GARCIA TORRES, WILLIAM STANLEY	GT11003
HERNANDEZ SANCHEZ, EDWIN ALEXANDER	HS19011

PARA OPTAR AL GRADO DE:
INGENIERO DE SISTEMAS INFORMÁTICOS

DOCENTE ASESOR:
ING. SANDRA ROMERO

29 DE NOVIEMBRE DEL 2025
CIUDAD UNIVERSITARIA, SAN SALVADOR, EL SALVADOR, C.A.

UNIVERSIDAD DE EL SALVADOR

RECTOR:

M.SC. JUAN ROSA QUINTANILLA

VICERRECTORA ACADÉMICA:

DRA. EVELYN BEATRIZ FARFÁN

VICERRECTOR ADMINISTRATIVO:

M.SC. ROGER ARIAS

PRESIDENTE ASAMBLEA GENERAL UNIVERSITARIA:

M.SC. CARLOS VILLALTA

FISCAL:

LIC. CARLOS AMÍLCAR SERRANO RIVERA

SECRETARIO GENERAL:

LIC. PEDRO ROSALÍO ESCOBAR CASTANEDA

DEFENSORA DE LOS DERECHOS UNIVERSITARIOS:

LCDA. ANA RUTH AVELAR

FACULTAD DE INGENIERÍA Y ARQUITECTURA

DECANO:

ING. LUIS SALVADOR BARRERA MANCÍA

VICEDECANO:

MSC. ANA BEATRIZ LIMA DE ZALDAÑA

SECRETARIO:

ARQ. RAÚL A. FABIÁN.

ESCUELA DE INGENIERÍA DE SISTEMAS INFORMÁTICOS

DIRECTOR:

MSC. CÉSAR AUGUSTO GONZÁLES RODRÍGUEZ

Historial de cambios

Versión	Fecha	Autor	Descripción
1.0	29/11/2025	Hernandez Sanchez, Edwin Alexander	Creación del documento

Índice

Índice de Figuras

Índice de tablas

AGRADECIMIENTOS

Agradecemos profundamente a la Universidad de El Salvador y a la Facultad de Ingeniería y Arquitectura por brindarnos la formación, los espacios y las herramientas necesarias para llevar a cabo este proyecto académico y de vida.

Nuestro reconocimiento también para los docentes de la Escuela de Ingeniería de Sistemas Informáticos, quienes con su guía y exigencia académica contribuyeron al desarrollo de las habilidades técnicas y metodológicas que hicieron posible este trabajo.

Agradecemos especialmente a la docente encargada de la asignatura de Ingeniería de Calidad, cuyo acompañamiento y retroalimentación fueron fundamentales para estructurar el plan de pruebas y orientar adecuadamente nuestro proceso de validación.

Finalmente, extendemos sinceramente nuestro agradecimiento a nuestras familias, quienes, con su apoyo incondicional, motivación y paciencia nos permitieron completar esta tesina. Su respaldo fue clave durante todo el proceso de investigación, desarrollo y documentación.

RESUMEN

El presente documento describe el plan de pruebas elaborado para evaluar el sistema de gestión de historiales médicos de la Clínica Veterinaria Mitsum. Este trabajo se desarrolla como requisito académico de la Especialización en Ingeniería de Calidad de Software de la Escuela de Ingeniería de Sistemas Informáticos de la Universidad de El Salvador, por lo que el enfoque, el alcance y los artefactos generados responden a los lineamientos de dicha especialización.

El sistema de gestión de historiales médicos es una aplicación web cliente–servidor ya existente; por su parte, la Clínica Veterinaria Mitsum corresponde a un negocio real que presta servicios veterinarios, lo que permite trabajar sobre un contexto cercano a un entorno productivo**.** El objetivo principal de este documento es definir el enfoque, alcance, técnicas, niveles y actividades necesarias para validar la funcionalidad del sistema dentro de un contexto académico. El trabajo se basa en buenas prácticas de ISTQB, el estándar IEEE-829 y el modelo de calidad ISO/IEC 25010, adaptados a los recursos disponibles y al tamaño del proyecto.

El proceso de validación se centró en pruebas funcionales, combinando técnicas manuales y automatizadas. Se diseñaron y ejecutaron casos de prueba para los módulos principales del sistema —pacientes, historiales clínicos, consultas y citas—, complementados con pruebas E2E y pruebas de API implementadas con Playwright y SuperTest. Además, se configuró un pipeline básico de integración continúa utilizando GitHub Actions para apoyar la ejecución automática de las pruebas.

El documento detalla la estrategia de prueba, los criterios de entrada y salida, la planificación, las estimaciones y los roles del equipo. También incluye la matriz de trazabilidad, la organización de evidencias y los anexos correspondientes, tales como los casos de prueba y los scripts automatizados. En conjunto, este plan establece un proceso formal, ordenado y verificable que permite garantizar la correcta evaluación del sistema y proporciona un marco sólido para fines académicos.

INTRODUCCIÓN

La calidad del software se erige como un pilar fundamental en la eficiencia, confiabilidad y seguridad de los sistemas informáticos, particularmente en entornos que manejan información sensible como el sector médico y veterinario. La implementación de planes de prueba estructurados permite identificar defectos de forma temprana, reducir costos de mantenimiento y optimizar la experiencia del usuario, tal como lo establecen estándares internacionales reconocidos.

En el caso de la Clínica Veterinaria Mitsum, se cuenta con un sistema de gestión de historiales médicos en producción, desarrollado bajo una arquitectura cliente-servidor. Sin embargo, la ausencia de un plan de pruebas formal representa un riesgo latente, donde errores no detectados podrían afectar la operatividad, la seguridad de los datos y la calidad del servicio. Esta situación es común en sistemas que evolucionaron sin procesos de aseguramiento de la calidad estructurados, haciendo necesario un enfoque correctivo y preventivo.

El presente documento tiene como objetivo describir el diseño, ejecución y resultados del proceso de pruebas realizado sobre dicho sistema, siguiendo estándares reconocidos como IEEE 829 e ISTQB. La tesina se estructura en tres capítulos principales:

Capítulo I: Se presentan las generalidades teóricas relacionadas con la calidad de software, pruebas, niveles y tipos de testing.

Capítulo II: Se detalla el plan de pruebas elaborado, incluyendo estrategia, alcance, riesgos, entregables y procedimientos.

Capítulo III: Se desarrolla el caso de estudio, integrando la matriz de pruebas, ejecución, análisis de resultados, defectos identificados y documentación técnica asociada.

De esta manera, el documento ofrece un enfoque práctico y fundamentado para la evaluación del sistema, contribuyendo al fortalecimiento del aseguramiento de la calidad en entornos académicos y profesionales.

CAPÍTULO I. GENERALIDADES

1.1 Calidad de software

La calidad de software es un concepto fundamental en la ingeniería de software y se refiere al grado en que un sistema cumple con los requisitos funcionales, no funcionales y necesidades del usuario. Diversos autores coinciden en que un producto de software es de calidad cuando satisface lo que el cliente espera y funciona de manera confiable bajo las condiciones reales de uso.

Según Pressman, la calidad de software es “la conformidad con requisitos explícitos y estándares de desarrollo documentados” (Pressman & Maxim, 2015), lo que implica que la calidad no ocurre de forma accidental, sino como resultado de procesos estructurados de diseño, construcción y verificación. Sommerville añade que la calidad también depende del contexto, ya que distintos sistemas requieren distintos niveles de fiabilidad, seguridad o desempeño (Sommerville, 2011).

El marco normativo más aceptado internacionalmente para evaluar la calidad de un producto software es el estándar ISO/IEC 25010:2011, el cual define dos modelos complementarios:

1. Modelo de calidad del producto (ISO/IEC 25010)

Este modelo describe ocho características esenciales que permiten evaluar la calidad interna y externa del software:

- **Adecuación funcional:** Evalúa si el sistema proporciona las funciones que el usuario necesita y si estas operan de manera correcta y completa.
- **Eficiencia del desempeño:** Mide el uso óptimo de recursos como tiempo de respuesta, consumo de memoria o capacidad del servidor.
- **Compatibilidad:** Determina si el sistema puede operar adecuadamente en conjunto con otros sistemas, dispositivos o versiones sin problemas.
- **Usabilidad:** Analiza qué tan fácil es para el usuario aprender, entender y utilizar el sistema para lograr sus objetivos.

- **Fiabilidad:** Evalúa si el sistema funciona consistentemente sin fallos, manteniendo estabilidad operativa en el tiempo.
- **Seguridad:** Garantiza la protección de datos contra accesos no autorizados, pérdidas, modificaciones o ataques.
- **Mantenibilidad:** Describe la facilidad con la que el software puede ser modificado, corregido, actualizado o mejorado.
- **Portabilidad:** Valora qué tan fácilmente el sistema puede transferirse o adaptarse a distintos entornos o plataformas.

2. Modelo de calidad en uso (ISO/IEC 25010)

Complementa la visión anterior evaluando la experiencia real del usuario al interactuar con el sistema:

- **Eficacia:** Determina si los usuarios pueden completar correctamente sus tareas dentro del sistema.
- **Eficiencia:** Mide los recursos necesarios (tiempo, pasos, esfuerzo) para completar una tarea.
- **Satisfacción:** Evalúa el nivel de comodidad, confianza y aceptación que sienten los usuarios al interactuar con el software.
- **Mitigación del riesgo:** Analiza si el sistema reduce errores, daños o pérdidas cuando los usuarios lo utilizan.
- **Cobertura en contexto de uso:** Valora si el sistema funciona adecuadamente en diferentes escenarios, condiciones o dispositivos.

En sistemas informativos como el de la Veterinaria Mitsum, este enfoque es crítico, ya que los historiales médicos requieren precisión, disponibilidad y seguridad. Por lo tanto, la calidad no solo consiste en que el sistema funcione, sino en que preserve la integridad de la información, minimice errores humanos y proporcione una experiencia de uso consistente. Además, el estándar ISO/IEC 25010 sirve como guía para la definición de pruebas funcionales y no funcionales, ya que cada característica de calidad puede ser validada mediante técnicas específicas. Estas pruebas permiten verificar que el software cumple con su propósito y que los riesgos asociados al uso se mantienen bajo control.

El proceso de prueba basado en calidad se complementa con el marco conceptual del ISTQB - International Software Testing Qualifications Board (ISTQB, 2023), el cual establece terminología estándar y buenas prácticas para la validación del software. Entre estos principios destacan:

- **Las pruebas deben estar alineadas a los requisitos:** Toda prueba debe medir una funcionalidad o característica definida previamente.
- **No es posible probar completamente un sistema:** Por ello se priorizan escenarios críticos o de mayor riesgo.
- **Las pruebas tempranas reducen costos:** Detectar defectos en etapas iniciales evita reprocesos y fallas en producción.
- **Agrupación de defectos:** Los errores suelen concentrarse en los componentes más complejos o usados con mayor frecuencia.
- **La ausencia de errores no garantiza calidad:** Un sistema puede tener pocas fallas y aun así no cumplir con lo que necesita el usuario.

1.2 Pruebas de software

Las pruebas de software constituyen un proceso sistemático dentro del ciclo de vida de desarrollo cuyo propósito es evaluar la calidad de un producto, identificar defectos y verificar el cumplimiento de los requisitos funcionales y no funcionales establecidos. El glosario del ISTQB (2023) define una prueba como una “secuencia de pasos diseñada para verificar un atributo o capacidad de un sistema o componente”, destacando su rol central dentro del aseguramiento de la calidad.

Históricamente, Myers (2011) describe las pruebas como “la ejecución de un programa con la intención de encontrar errores”, enfatizando su carácter proactivo y crítico para detectar fallas antes de que estas afecten al usuario final. Esta visión se complementa con el uso moderno de las pruebas como herramienta para elevar la confianza en la calidad del software cuando los defectos no se encuentran.

De acuerdo con el estándar IEEE 829, las pruebas de software pueden entenderse desde dos perspectivas complementarias:

- **Verificación**

  Responde a la pregunta: “¿Se construyó el sistema correctamente?”

  Se refiere a comprobar que el software cumple con los requisitos técnicos, reglas de negocio y documentación especificada durante las distintas fases del desarrollo.

  Ejemplo aplicado al proyecto: verificar que el campo “Nombre de la Mascota” no permita ingresar más de 50 caracteres conforme a los requisitos funcionales del sistema.

- **Validación**

  Responde a la pregunta: “¿Se construyó el sistema correcto?”

  Evalúa si el sistema final satisface las necesidades reales del usuario en su entorno operativo.

  Ejemplo aplicado al caso Mitsum: validar que el flujo de registro de diagnósticos sea intuitivo para el veterinario y capture toda la información clínica necesaria.

En resumen, las pruebas de software tienen cuatro objetivos fundamentales:

- **Detectar defectos:** Identificar fallas en el comportamiento del sistema que puedan comprometer la funcionalidad o la experiencia del usuario.
- **Verificar el cumplimiento de requisitos:** Asegurar que cada funcionalidad responde exactamente a lo especificado en los documentos del proyecto.
- **Validar que el software satisface necesidades del usuario:** Confirmar que el sistema es útil, eficiente y adecuado para los procesos reales del negocio.
- **Aumentar la confianza en la calidad del producto:** Proporcionar evidencia objetiva que respalde la liberación o rechazo de una versión del sistema.

El ISTQB establece que las pruebas no solo implican la ejecución del software, sino también actividades previas como el análisis, el diseño de casos de prueba, la planificación, la documentación, la evaluación de riesgos y la generación de informes.

Esto significa que probar un sistema no es únicamente “probar si funciona”, sino aplicar un proceso estructurado y metodológico.

Además, las pruebas son una herramienta clave para mitigar riesgos ya que estudios clásicos de ingeniería de software evidencian el impacto económico de las pruebas. Boehm & Papaccio (1988) demostraron que corregir un defecto en producción puede costar entre 15 y 100 veces más que corregirlo durante etapas tempranas, lo cual posiciona al testing como una inversión estratégica y no como un gasto, permitiendo reducir riesgos operativos y proteger la reputación de la organización.

En el contexto de la Veterinaria Mitsum, las pruebas juegan un papel especialmente importante debido a la sensibilidad de la información involucrada. Un error en los historiales médicos podría:

- Afectar diagnósticos.
- Perder registros clínicos.
- Generar tratamientos erróneos.
- Comprometer la satisfacción del cliente.
- Incluso causar problemas legales para la clínica.

Por ello, la aplicación de pruebas de software permite garantizar que el sistema opere correctamente en los procesos de consulta, registro, edición y seguimiento de historiales médicos. Estos esfuerzos aseguran que el sistema sea confiable, seguro y adecuado para las tareas clínicas del negocio.

En conjunto, estas prácticas aseguran que el proceso de pruebas contribuya directamente a la calidad, fiabilidad y seguridad del software, en coherencia con los modelos establecidos por ISO/IEC 25010.

1.3 Proceso de pruebas

El proceso de pruebas es un conjunto de actividades estructuradas que permiten planificar, diseñar, ejecutar y evaluar las pruebas de software con el objetivo de garantizar la calidad del producto. Según ISTQB (2023), este proceso no debe considerarse como una fase aislada dentro del desarrollo, sino como un ciclo continuo

que acompaña todo el ciclo de vida del software, adaptándose al modelo de desarrollo utilizado y retroalimentándose de manera iterativa conforme se descubren nuevos riesgos o se redefinen requisitos.

Independientemente de si el proyecto se desarrolla bajo un enfoque en cascada, incremental, ágil o híbrido, el proceso de pruebas mantiene una estructura fundamental compuesta por cinco actividades principales:

1. Planificación y control de las pruebas

En esta actividad se definen:

- Alcance y los tipos de pruebas a realizar.
- Estrategia de pruebas.
- Recursos necesarios.
- Criterios de entrada y salida.
- Cronograma.
- Riesgos y Contingencias.

Durante la planificación también se establecen métricas para medir el progreso y la calidad del proceso. El control, por su parte, consiste en monitorear continuamente la ejecución del plan, comparando el progreso real con el esperado, gestionando desviaciones, identificando nuevos riesgos y tomando acciones correctivas oportunas. Esto asegura que las pruebas se mantengan alineadas a los objetivos del proyecto.

2. Análisis y diseño de pruebas

En esta fase se transforman los requisitos funcionales y no funcionales en artefactos verificables. Esta etapa incluye:

- Qué requisitos deben ser probados.
- Qué condiciones y escenarios se derivan de dichos requisitos.
- Qué datos de prueba serán utilizados.
- Qué técnicas de diseño se aplicarán (equivalencia, valores límite, tablas de decisión, etc.).

El objetivo es traducir los requisitos funcionales y no funcionales en artefactos verificables. Esta actividad permite crear las bases para la ejecución eficiente de los casos de prueba.

3. Implementación y ejecución de pruebas

- La implementación consiste en:
- Construir los casos de prueba.
- Preparar los datos de prueba.
- Configurar los ambientes.
- Dejar listo el entorno para ejecutar las pruebas.

La ejecución implica:

- Correr las pruebas manuales o automatizadas,
- Comparar resultados reales vs resultados esperados,
- Registrar incidentes o defectos.

Durante esta actividad se generan evidencias, capturas de pantalla, resultados de scripts automatizados o registros del sistema, esta actividad es clave, pues genera la evidencia objetiva del comportamiento del software. Una buena ejecución permite identificar defectos críticos, validar funcionalidades clave y obtener métricas reales sobre la estabilidad y rendimiento del sistema.

4. Evaluación de criterios de salida y reporte

Tras finalizar las ejecuciones, se evalúa si se han cumplido los criterios de salida previamente definidos, tales como:

- Porcentaje de casos ejecutados.
- Porcentaje de casos aprobados.
- Cantidad y severidad de defectos abiertos.
- Comportamiento del sistema bajo condiciones específicas.
- Nivel de cumplimiento de requisitos funcionales y no funcionales.

Si los criterios de salida no se cumplen, se debe decidir si:

- Se requiere mayor ejecución.
- Deben corregirse defectos antes de continuar.
- Es necesario ajustar el plan de pruebas.

Además, en esta fase se elaboran reportes de avance y se comunica el estado real del sistema a los interesados, proporcionando información clave para la toma de decisiones sobre liberaciones o cambios.

5. Actividades de cierre de pruebas

Una vez concluido el ciclo de pruebas, se realizan las actividades de cierre, que incluyen:

- Documentación final de resultados.
- Comparación del trabajo realizado vs. lo planificado.
- Recolección y análisis de métricas finales.
- Consolidación de evidencia para auditorías o futuras referencias.
- Cierre oficial de incidentes.
- Identificación de lecciones aprendidas.
- Archivado de artefactos de prueba como casos, reportes, matrices y scripts.

Estas actividades permiten mejorar procesos futuros, asegurar trazabilidad y facilitar el mantenimiento del software, ya que dejan un registro estructurado del estado del sistema al momento de la evaluación.

Aplicación al caso de estudio

En el sistema de historiales médicos de la Clínica Veterinaria Mitsum, aplicar un proceso de pruebas estructurado resulta esencial para garantizar que los módulos críticos, como registro de pacientes, consultas, diagnósticos y tratamientos, funcionen correctamente antes de su uso operativo.

Un proceso de pruebas formal permite:

- Detectar inconsistencias tempranas en el registro de datos clínicos.

- Verificar que los flujos de consulta y diagnóstico cumplen con las necesidades del veterinario.
- Validar que el sistema sea confiable, seguro y disponible.
- Reducir riesgos asociados a pérdida o manipulación incorrecta de historiales médicos.
- Documentar evidencia objetiva que respalde la calidad del software.

De esta forma, el proceso de pruebas se convierte en una guía sistemática que garantiza que el sistema cumpla con los niveles de calidad definidos en ISO/IEC 25010 y con las expectativas del negocio.

1.4 Niveles de prueba

Los niveles de prueba representan distintas etapas en las que se evalúa el sistema durante su construcción y posterior integración. Cada nivel tiene objetivos específicos, un alcance particular y diferentes actores involucrados. Según ISTQB (2023), la definición de niveles de prueba permite estructurar el proceso y asegurar que las funcionalidades se validen de forma progresiva, desde los componentes individuales hasta el sistema completo en operación.

El uso de niveles de prueba contribuye a la detección temprana de defectos, la reducción de costos y la mejora en la calidad del producto final. Asimismo, permite mantener una visión clara del alcance de cada fase y facilita la trazabilidad entre requisitos, casos de prueba y resultados.

Los niveles formalmente establecidos por ISTQB y aplicados en proyectos de desarrollo y mantenimiento de software son cuatro: pruebas de componentes, integración, sistema y aceptación. A continuación, se describen cada uno y se relacionan con el sistema de la Veterinaria Mitsum.

1.4.1 Pruebas de componentes

Las pruebas de componentes también llamadas pruebas unitarias se enfocan en evaluar partes individuales del software, como funciones, métodos, clases o módulos. Su objetivo

principal es verificar que cada componente funcione de manera aislada y cumpla con su especificación técnica.

Este nivel suele ser ejecutado por los desarrolladores, quienes utilizan artefactos como stubs y drivers para simular dependencias y permitir la evaluación aislada de cada componente.

En el contexto de la Veterinaria Mitsum, una prueba de componente podría validar que:

- La función encargada de calcular la edad de la mascota a partir de su fecha de nacimiento retorne el valor correcto.
- El método que valida el formato del número de expediente rechace entradas inválidas.

Este nivel permite identificar defectos antes de integrar los módulos, reduciendo costos y complejidad.

1.4.2 Pruebas de integración

Estas pruebas verifican la interacción entre múltiples componentes o módulos que ya funcionan individualmente. Su objetivo es detectar defectos relacionados con interfaces, comunicación y flujo de datos entre partes del sistema.

Las estrategias de integración pueden ser ascendentes (bottom-up), descendentes (top-down) o híbridas. Ejemplos en el sistema de la Veterinaria Mitsum incluyen:

- Probar la comunicación entre el módulo de registro de mascotas y el módulo de historiales médicos.
- Verificar que la información registrada en un formulario se almacene correctamente en la base de datos.

Este nivel es fundamental para garantizar que los componentes colaboran adecuadamente y que el flujo del sistema se mantiene estable.

1.4.3 Pruebas de sistema

Las pruebas de sistema evalúan la aplicación completa como un todo, incluyendo software, datos, infraestructura y componentes externos. Su objetivo es validar que el

sistema cumpla con los requisitos funcionales y no funcionales, tales como seguridad, rendimiento, usabilidad y fiabilidad. En este nivel se aplican técnicas de caja negra, guiadas por requisitos.

Ejemplos aplicados al caso Mitsum:

- Validar todo el proceso de crear, consultar y actualizar un historial médico completo.
- Evaluar tiempos de respuesta al cargar listados de mascotas.

Este nivel simula condiciones lo más cercanas posibles al entorno real de la veterinaria.

1.4.4 Pruebas de aceptación

Este es el nivel final antes de liberar el sistema a producción. Su propósito es confirmar que el software cumple con las necesidades del usuario y que es apto para su operación real. Generalmente son ejecutadas por usuarios finales, clientes o personal de negocio. Se clasifican en pruebas de aceptación del usuario (UAT), operativas y contractuales. Para la Veterinaria Mitsum, tenemos estos ejemplos:

- Verificar que el veterinario pueda registrar diagnósticos de forma rápida y sin errores.
- Validar que el sistema permita consultar historiales médicos completos durante una consulta real.

Este nivel asegura que el sistema resuelve el problema real del negocio y está listo para su uso diario. En conjunto, estos cuatro niveles permiten validar la funcionalidad, la integración, el comportamiento global y la aceptación del sistema, garantizando una verificación completa y progresiva conforme a las buenas prácticas establecidas por ISTQB.

1.5 Tipos de pruebas

Los tipos de pruebas representan diferentes enfoques y técnicas utilizados para evaluar atributos específicos del software. Según ISTQB (2023), los tipos de pruebas permiten

identificar defectos desde distintas perspectivas, tales como funcionalidad, estructura interna, rendimiento o estabilidad del sistema.

Su correcta aplicación garantiza una evaluación equilibrada y completa de la calidad del software.

Dentro del proceso de aseguramiento de la calidad, los tipos de pruebas se clasifican comúnmente en cuatro categorías principales: pruebas funcionales, pruebas no funcionales, pruebas estructurales y pruebas de regresión.

Cada una de ellas contribuye a verificar características particulares del sistema y complementa los niveles de prueba descritos en la sección anterior.

1.5.1 Pruebas funcionales

Las pruebas funcionales tienen como objetivo verificar que el software cumpla con los requisitos funcionales especificados.

Generalmente se basan en técnicas de caja negra y validan que las funciones, operaciones y flujos del sistema produzcan los resultados esperados.

Entre las técnicas más utilizadas se encuentran las siguientes, las cuales permiten diseñar casos de prueba más eficientes y representativos del comportamiento del sistema:

- Partición de equivalencia.
- Análisis de valores límite.
- Tablas de decisión.
- Casos de uso.
- Flujo de trabajo.

Ejemplos aplicados al sistema de la Veterinaria Mitsum:

- Verificar que el sistema permita registrar una nueva mascota con todos los campos obligatorios.
- Validar que un veterinario pueda consultar el historial médico completo de un paciente.

- Confirmar que el sistema rechace intentos de guardar un diagnóstico sin información obligatoria.

Estas pruebas aseguran que el comportamiento del sistema es correcto según lo esperado por el negocio.

1.5.2 Pruebas no funcionales

Las pruebas no funcionales evalúan atributos del software que no están relacionados con la lógica funcional, sino con características de calidad tales como rendimiento, seguridad, usabilidad, fiabilidad o compatibilidad.

Estas pruebas permiten determinar qué tan bien se comporta el sistema bajo condiciones específicas.

Entre los tipos más comunes se incluyen los siguientes, los cuales evalúan distintos atributos de calidad definidos en ISO/IEC 25010:

- **Pruebas de rendimiento:** tiempos de respuesta, carga y estrés.
- **Pruebas de seguridad:** control de accesos, protección de datos, manejo de credenciales.
- **Pruebas de usabilidad:** facilidad de uso, claridad de la interfaz.
- **Pruebas de compatibilidad:** funcionamiento en diferentes navegadores o dispositivos.
- **Pruebas de fiabilidad:** estabilidad del sistema durante períodos prolongados.

Ejemplos aplicados al caso Mitsum:

- Medir el tiempo de respuesta del listado de historiales médicos con múltiples registros.
- Validar que un usuario no autorizado no pueda acceder a información clínica.
- Comprobar que el flujo de registro de diagnósticos sea intuitivo para los veterinarios nuevos.
- Evaluar que el sistema funcione correctamente en diferentes navegadores utilizados en la clínica.

1.5.3 Pruebas estructurales

Las pruebas estructurales también conocidas como pruebas de caja blanca se centran en la estructura interna del software, incluyendo condiciones lógicas, rutas de código y flujos internos. Su propósito es evaluar la cobertura del código y verificar que todos los caminos posibles hayan sido ejecutados.

Son aplicadas principalmente por desarrolladores o ingenieros de prueba con conocimiento del código fuente. Se apoyan en métricas como:

- Cobertura de sentencias.
- Cobertura de decisiones.
- Cobertura de condiciones múltiples.
- Análisis de flujo de control.

Ejemplos aplicados al sistema de la Veterinaria Mitsum:

- Verificar que todas las rutas del método “guardar diagnóstico” se ejecuten correctamente.
- Evaluar condiciones lógicas internas del cálculo de tratamiento según el peso y edad del paciente.

1.5.4 Pruebas de regresión

Las pruebas de regresión tienen como objetivo verificar que, después de realizar cambios como corrección de defectos, mejoras o nuevas funcionalidades el sistema no haya introducido defectos nuevos y que las funcionalidades existentes sigan operando correctamente.

Estas pruebas suelen automatizarse debido a su naturaleza repetitiva y se ejecutan regularmente en cada nueva versión del sistema.

Ejemplos aplicados al caso Mitsum:

- Validar que, después de agregar un nuevo campo en el expediente, los historiales existentes continúen cargando correctamente.

- Confirmar que la corrección de un error en el módulo de diagnósticos no afectó el registro de tratamientos.
- Ejecutar nuevamente el flujo completo de creación de mascotas tras actualizar el módulo de autenticación.

Este tipo de pruebas es especialmente importante en sistemas que evolucionan continuamente o que manejan información crítica.

La correcta aplicación de estos tipos de pruebas permite evaluar características de calidad definidas en el modelo ISO/IEC 25010, tales como seguridad, fiabilidad, usabilidad y eficiencia del desempeño.

1.6 Testing Outline

El testing outline describe las distintas formas en que se clasifican las actividades de prueba según la naturaleza del proceso de verificación. De acuerdo con ISTQB (2023), esta clasificación permite organizar y estructurar las pruebas tomando en cuenta factores como el riesgo, el contexto del proyecto, los recursos disponibles y los requisitos del sistema.

La elección del enfoque adecuado influye directamente en la eficacia del proceso de pruebas, el tipo de defectos que se detectarán y el nivel de confianza que se puede obtener sobre el estado real del software.

Los dos enfoques principales establecidos por ISTQB dentro de esta clasificación son: pruebas estáticas, que analizan el software sin ejecutarlo, y pruebas dinámicas, que validan su comportamiento mediante la ejecución.

1.6.1 Pruebas estáticas

Las pruebas estáticas consisten en la revisión y análisis del software sin ejecutarlo. Su objetivo es detectar defectos tempranos en la documentación, código o artefactos del proyecto antes de que el sistema esté disponible para pruebas dinámicas.

Incluyen actividades como inspecciones, revisiones técnicas, análisis estático de código y verificación de requisitos.

Según ISTQB (2023), estas pruebas permiten identificar errores de ambigüedad, inconsistencias, violaciones de estándares y problemas de diseño antes de que se conviertan en fallas durante la ejecución.

Ejemplos aplicados al sistema de la Veterinaria Mitsum

- Revisar la especificación de requisitos para asegurar que los campos del historial médico estén claramente definidos.
- Analizar el código del módulo de autenticación para buscar malas prácticas (por ejemplo, contraseñas en texto plano).
- Un ejemplo típico de prueba estática dentro es la revisión de los casos de prueba manuales (cuando estos son elaborados), verificando que cumplan con los elementos establecidos por IEEE 829 como identificador, pasos, datos de prueba y resultados esperados sin necesidad de ejecutar el sistema.

Beneficios clave

- Detección temprana de defectos.
- Reducción de costos en etapas posteriores.
- Mejora de la calidad de requisitos, diseño y código.

1.6.2 Pruebas dinámicas

Las pruebas dinámicas requieren ejecutar el software para observar su comportamiento. Se enfocan en verificar que el sistema responda correctamente a diferentes entradas, escenarios y condiciones operativas. Este enfoque está directamente relacionado con los tipos de pruebas funcionales y no funcionales, y se basa principalmente en técnicas de caja negra y caja blanca. Las pruebas dinámicas permiten validar tanto el cumplimiento de requisitos como la experiencia real del usuario en interacción con el sistema.

Ejemplos aplicados al caso Mitsum

- Ejecutar el flujo completo de creación de una nueva mascota.
- Medir el tiempo de respuesta al consultar un historial médico extenso.

Beneficios clave

- Evidencia real del funcionamiento del sistema.
- Capacidad de medir atributos del modelo ISO/IEC 25010 como eficiencia, fiabilidad y seguridad.
- Identificación de errores que solo aparecen durante la ejecución (por ejemplo, fallas de integración).

El presente capítulo estableció los fundamentos teóricos necesarios para comprender la importancia del aseguramiento de la calidad dentro del desarrollo y mantenimiento de software. Se exploraron los conceptos clave de calidad de software según el estándar ISO/IEC 25010, los principios y objetivos de las pruebas respaldados por el ISTQB, así como los diferentes niveles, tipos y enfoques utilizados en la verificación y validación de sistemas.

Estos elementos proporcionan la base conceptual sobre la cual se estructura el proceso de pruebas aplicado en este proyecto. La comprensión de los niveles de prueba, los tipos de técnicas disponibles y la diferencia entre pruebas estáticas y dinámicas resulta esencial para diseñar un plan de pruebas sólido, trazable y alineado con estándares internacionales. Con este marco teórico establecido, el siguiente capítulo presenta el Plan de Pruebas, donde se detallan los artefactos, actividades, responsabilidades y criterios necesarios para evaluar la calidad del sistema de la Veterinaria Mitsum de forma sistemática y controlada.

CAPÍTULO II. PLAN DE PRUEBAS

2.1 Definición de plan de pruebas

Un plan de pruebas es un documento formal que describe el conjunto organizado de actividades, recursos, procedimientos y criterios necesarios para evaluar la calidad de un sistema de software. Su propósito principal es establecer una guía estructurada que permita planificar, ejecutar y controlar el proceso de pruebas de manera coherente y verificable.

Según el estándar IEEE 829, el plan de pruebas define con claridad qué será probado, cómo se llevará a cabo la verificación, quién ejecutará las actividades y bajo qué condiciones se evaluarán los resultados. Este estándar también especifica la información mínima que debe contener un plan de pruebas, incluyendo su alcance, riesgos, recursos, cronograma, criterios de entrada y salida, y entregables.

Por su parte, ISTQB complementa esta perspectiva al señalar que un plan de pruebas debe asegurar la trazabilidad entre requisitos, casos de prueba, defectos y resultados, permitiendo una evaluación objetiva del estado del producto y facilitando la toma de decisiones informadas sobre su liberación o mejora.

En el contexto del sistema de historiales médicos de la Clínica Veterinaria Mitsum, el plan de pruebas constituye la base para evaluar funcionalidades críticas relacionadas con el registro de pacientes, la gestión de diagnósticos, la seguridad de acceso a información sensible y el rendimiento general de la aplicación. Además, orienta y estructura las actividades de pruebas manuales y automatizadas que se aplicarán durante el proyecto.

Este documento establece la estrategia general de pruebas, determina su alcance, identifica los riesgos asociados y detalla los artefactos que serán generados durante el proceso. La información presentada en este capítulo conforma la estructura formal del plan de pruebas que será aplicado en el Capítulo III.

2.2 Estándar IEEE-829

El estándar IEEE-829, también conocido como Standard for Software and System Test Documentation, define un conjunto estructurado de documentos formales destinados a planificar, diseñar, ejecutar y registrar las actividades de pruebas de software. Su propósito es establecer un marco uniforme que facilite la organización, comunicación y trazabilidad de todo el proceso de testing, garantizando que las pruebas se desarrollen de forma controlada y verificable.

Este estándar proporciona plantillas y contenidos mínimos para cada artefacto de prueba, permitiendo que los equipos documenten de manera consistente qué se va a probar, cómo se realizará, con qué recursos, qué criterios se utilizarán para evaluar los resultados y cuál fue el estado real del producto durante y después de las pruebas. Aunque IEEE-829 fue posteriormente reemplazado por la familia de estándares IEEE 29119, sigue siendo ampliamente utilizado en entornos académicos e industriales debido a su claridad, simplicidad y enfoque práctico.

Los documentos que integran el estándar IEEE 829 incluyen:

- **Test Plan (Plan de Pruebas):** define el enfoque, el alcance, los recursos, el cronograma, los criterios de entrada/salida y los riesgos.
- **Test Design Specification:** describe las condiciones de prueba, los casos de prueba derivados y el diseño general de la verificación.
- **Test Case Specification (Casos de Prueba):** detalla los casos de prueba, incluyendo entradas, pasos, datos, precondiciones y resultados esperados.
- **Test Procedure Specification (Casos de Prueba):** define la secuencia exacta de acciones para ejecutar los casos de prueba.
- **Test Item Transmittal Report:** documento que acompaña la entrega de elementos del software a probar.
- **Test Log:** registro cronológico de la ejecución de las pruebas.
- **Test Incident Report (Registro de defectos):** documento para registrar defectos o eventos inesperados durante la ejecución.

- **Test Summary Report (Cobertura de Módulos):** resumen final de los resultados obtenidos, conclusiones y evaluación del cumplimiento de criterios.

Si bien IEEE-829 propone una documentación extensa orientada a proyectos empresariales y de gran escala, en el contexto académico de esta tesina se adoptan únicamente los artefactos necesarios para garantizar claridad, trazabilidad y formalidad en el proceso.

2.3 Identificador del Plan de Pruebas

El identificador del plan de pruebas es un código único que permite rastrear, clasificar y referenciar de manera estandarizada el documento dentro del ciclo de vida de pruebas. Según el IEEE-829, cada artefacto de prueba debe contar con un identificador que facilite su administración, control de versiones y trazabilidad a través del proyecto.

Este identificador asegura que cualquier actualización, revisión o auditoría pueda relacionarse claramente con el documento correcto, evitando ambigüedades y garantizando consistencia documental.

Para este proyecto se utiliza una convención de identificación estructurada, compuesta por los siguientes elementos:

Nombre del documento:

Plan de Pruebas del Sistema de Historiales Médicos - Veterinaria Mitsum

Código interno:

PP–MITSUM–2025–V1

Versión del documento:

1.0

Fecha de emisión:

Noviembre de 2025

2.4 Referencias

Esta sección enumera las normas, fuentes y materiales de apoyo utilizados para la elaboración del presente Plan de Pruebas. Tal como lo establece el estándar IEEE-829, las referencias permiten justificar la estructura del documento y mantener trazabilidad con los marcos conceptuales que lo sustentan.

Dado que el sistema de historiales médicos de la Clínica Veterinaria Mitsum no cuenta con documentación técnica formal, las referencias corresponden a estándares internacionales y a los insumos proporcionados durante el desarrollo académico del proyecto.

Las referencias empleadas son:

1. **IEEE 829-2008. Standard for Software and System Test Documentation. Institute of Electrical and Electronics Engineers.**

   Base normativa utilizada para definir la estructura del plan de pruebas y los artefactos requeridos.

2. **ISTQB (2023). Glossary of Testing Terms. International Software Testing Qualifications Board.**

   Fuente utilizada para la terminología y definiciones empleadas en el proceso de pruebas.

3. **ISO/IEC 25010:2011. Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE).**

   Utilizado para sustentar los atributos de calidad considerados en el alcance y estrategia del plan.

4. **Requisitos funcionales identificados a partir del análisis del sistema durante el proyecto académico.**

Otras fuentes y materiales consultados, así como recursos técnicos y enlaces utilizados en el desarrollo del proyecto, se listan en la sección final de Bibliografía del documento.

2.5 Elementos de las pruebas

Los elementos de las pruebas corresponden a los módulos, componentes y funcionalidades del sistema que serán evaluados durante el proceso de testing. Según el estándar IEEE-829, esta sección debe identificar claramente los ítems de software que serán objeto de verificación, estableciendo el punto de partida para la elaboración de casos de prueba, definición del alcance y selección de técnicas.

Para este proyecto, los elementos se definen con base en la descomposición funcional presentada en el anteproyecto, en la cual se estructuran los módulos centrales del sistema de historiales médicos de la Clínica Veterinaria Mitsum. Cada módulo constituye una unidad funcional evaluable que agrupa requisitos, flujos y operaciones específicas, permitiendo organizar el esfuerzo de pruebas de manera ordenada y trazable.

Los elementos de prueba considerados son los siguientes:

1. Autenticación y control de acceso

Este módulo concentra los mecanismos de seguridad del sistema, esenciales para garantizar que solo usuarios autorizados accedan a la información clínica. Incluye:

- Inicio de sesión y validación de credenciales.
- Gestión de usuarios, roles y permisos.
- Configuración de parámetros generales del sistema.
- Auditoría de accesos y registro de eventos relevantes.

La correcta operación de este módulo es crítica para mantener la integridad y confidencialidad de los historiales médicos.

2. Gestión de pacientes y propietarios

Agrupa las funcionalidades que permiten manejar la información básica de la clínica:

- CRUD de mascotas.
- CRUD de propietarios.
- Asociación mascota-propietario.

Este módulo es fundamental, ya que sirve como base de datos principal para el resto de las operaciones del sistema.

3. Historias clínicas

Es uno de los módulos más importantes del sistema, al contener información médica detallada de cada mascota. Incluye:

- Registro y consulta de diagnósticos.
- Registro de vacunaciones y tratamientos.
- Administración de antecedentes médicos.

La calidad de este módulo impacta directamente en la continuidad de la atención veterinaria.

4. Citas y cirugías

Incluye funcionalidades relacionadas con la programación operativa de la clínica:

- Agendamiento y modificación de citas.
- Programación de cirugías.
- Notificaciones automáticas vinculadas a citas o procesos quirúrgicos.

5. Facturación y reportes

Abarca los procesos administrativos y de generación de información:

- Generación de facturas de servicios.
- Reportes clínicos (consultas, cirugías, vacunación).

La precisión de estos procesos es clave para garantizar la transparencia operativa de la clínica.

6. Medicamentos e inventario

Permite gestionar los insumos y medicamentos utilizados durante las consultas y procedimientos:

- Gestión de inventario general.
- Gestión específica de medicamentos.

Este módulo también puede influir en la calidad del servicio, ya que un inventario incorrecto podría afectar la disponibilidad de tratamientos.

La identificación de estos módulos establece los elementos formales de prueba del proyecto y permite organizar el plan de pruebas, priorizar funcionalidades, definir escenarios y estructurar la matriz que se empleará en el Capítulo III. Cada módulo será evaluado según su criticidad, impacto operativo y riesgo asociado.

2.6 Riesgos

Los riesgos asociados al proceso de pruebas comprenden todos aquellos factores que pueden comprometer la correcta planificación, ejecución, seguimiento y finalización de las actividades de testing. De acuerdo con las directrices del estándar IEEE-829 y las prácticas recomendadas por ISTQB, la identificación temprana de riesgos permite comprender las limitaciones del proceso y anticipar problemas que podrían afectar la calidad de los resultados.

En el contexto del presente proyecto, los riesgos no se limitan únicamente a aspectos técnicos, sino también a elementos metodológicos, organizativos, humanos y operativos, que pueden influir directamente en la efectividad del plan de pruebas. A continuación, se describen los riesgos identificados:

1. Ausencia de documentación formal del sistema

El sistema no cuenta con documentación técnica detallada (requerimientos funcionales, casos de uso, diagramas o especificaciones), lo cual incrementa el riesgo de interpretaciones incorrectas sobre el comportamiento esperado. Esto afecta la precisión

del diseño de los casos de prueba y dificulta establecer criterios objetivos de validación, limitando además la trazabilidad entre funcionalidades y verificaciones.

2. Cambios no controlados o no documentados en el código

Debido a que el sistema continúa en evolución y no existe un proceso formal de gestión de cambios, es posible que se realicen modificaciones en el código fuente sin quedar adecuadamente registradas en las ramas o commits del repositorio. Esto puede provocar inconsistencias entre versiones, fallos inesperados durante la ejecución de las pruebas y dificultades en la reproducción de defectos.

3. Limitaciones del ambiente de pruebas

El proceso de pruebas recae en los cuatro integrantes del proyecto, quienes deben equilibrar estas actividades con responsabilidades académicas y personales. Esta dependencia incrementa el riesgo de retrasos en el cronograma, disminución del número de casos de prueba ejecutados o falta de revisión cruzada entre miembros del equipo.

4. Dependencia de un equipo reducido de trabajo

El proceso de pruebas recae exclusivamente en los cuatro integrantes del proyecto, quienes deben equilibrar estas actividades con responsabilidades académicas y personales. Esta dependencia incrementa el riesgo de retrasos en el cronograma, disminución del número de casos de prueba ejecutados o falta de revisión cruzada entre miembros del equipo.

5. Falta de experiencia previa en herramientas de automatización

Aunque se planea utilizar herramientas modernas como Playwright, Jest y SuperTest, el equipo posee experiencia limitada en la implementación de pruebas automatizadas. Esto implica una curva de aprendizaje que puede afectar la productividad, limitar el alcance de la automatización y aumentar el riesgo de errores en los scripts generados.

6. Datos de prueba insuficientes, inconsistentes o no representativos

La creación manual de datos sintéticos puede ocasionar escenarios incompletos o alejados de las condiciones reales de uso. Además, la ausencia de un repositorio centralizado de datos de prueba puede generar inconsistencias entre ejecuciones, afectando la confiabilidad y repetibilidad del proceso de validación.

7. Riesgos de integración entre frontend y backend

La comunicación entre el frontend desarrollado en React y el backend en NestJS puede verse afectada por actualizaciones asíncronas, cambios en rutas o validaciones, o diferencias entre ambientes locales y remotos. Cualquier desalineación entre ambos componentes representa un riesgo para la estabilidad de los flujos principales.

8. Dependencia de servicios externos y conectividad

Los ambientes de pruebas desplegados en Vercel (frontend) y Railway (API) dependen de la disponibilidad de estos proveedores. Problemas de conectividad, tiempos de respuesta elevados o caídas temporales pueden interrumpir la ejecución de pruebas e impedir reproducir defectos de forma consistente.

9. Limitaciones en la trazabilidad y control documental

Dado que el sistema fue desarrollado sin un proceso formal de ingeniería de requerimientos, la trazabilidad entre funcionalidades, casos de prueba, defectos y resultados depende completamente de los registros preparados por el equipo. Cualquier inconsistencia documental puede dificultar el análisis posterior y comprometer la calidad del informe final de pruebas.

2.7 Características que se probarán

Esta sección identifica las funcionalidades y atributos del sistema que serán evaluados durante el proceso de pruebas. De acuerdo con IEEE-829, las características que se probarán deben estar claramente delimitadas para garantizar trazabilidad con los requisitos, los elementos del sistema y la estrategia del plan.

En el proyecto de la Clínica Veterinaria Mitsum, las pruebas se centrarán en validar las funcionalidades principales del sistema, así como ciertos atributos de calidad esenciales para la operación clínica. Las características que se probarán son las siguientes:

Tabla 1-Características a Probar A

Módulo / Componente	Atributo de Calidad (ISO/IEC 25010)	Características Específicas a Probar
Autenticación y Control de Accesos	Funcionalidad	Inicio de sesión con credenciales válidas e inválidas. Manejo de errores en autenticación. Recuperación o restablecimiento de contraseña (si aplica).
	Seguridad	Validación funcional del control de accesos por rol (RBAC). Restricción de vistas y acciones según el rol. Acceso denegado a recursos protegidos. Cierre de sesión y expiración de sesión.
Gestión de Pacientes	Funcionalidad	Flujos CRUD completos. Validación de campos obligatorios. Manejo de errores de validación. Relación correcta entre paciente y propietario.
	Rendimiento Básico	Tiempos de respuesta en la búsqueda y visualización de múltiples registros (pruebas con SuperTest).
Gestión de Propietarios	Funcionalidad	Registro, consulta, actualización y eliminación. Relación paciente–propietario consistente.
	Usabilidad	Claridad de formularios, mensajes de error y facilidad de navegación.

Tabla 2-Características a Probar B

Módulo / Componente	Atributo de Calidad (ISO/IEC 25010)	Características Específicas a Probar
Historial Clínico	Funcionalidad	Registro de diagnósticos, tratamientos y observaciones. Consulta de historial clínico completo. Edición de información clínica.
	Usabilidad	Estructura visual de la información. Claridad en diagnósticos, tratamientos y fechas.
	Seguridad	Restricción para edición según rol del usuario. Validación funcional de acceso a datos sensibles.
Citas y Cirugías	Funcionalidad	Programación, modificación y cancelación de citas. Manejo de conflictos de horario. Validación de campos obligatorios.
	Compatibilidad	Funcionamiento correcto del calendario en navegadores utilizados por la clínica (Chrome y Firefox).
Facturación, Medicamentos e Inventario	Funcionalidad	Validación básica de registro y visualización de información. Navegación y consistencia mínima del flujo.
API del Sistema	Rendimiento Básico	Tiempos de respuesta promedio y máximo para endpoints críticos. Consistencia en respuestas ante peticiones consecutivas. Manejo correcto de códigos HTTP.
	Funcionalidad	Validación de endpoints CRUD para pacientes, propietarios e historiales.
Flujos del Sistema	Integridad Funcional	Automatización de flujos críticos: inicio de sesión, creación de paciente, registro de diagnóstico, consulta de historial. Validación de navegación, estados y redirecciones.
Todos los módulos	Usabilidad	Navegación intuitiva. Coherencia visual. Mensajes de error comprensibles.

2.8 Características que No se Probarán

Las exclusiones las clasificaremos en dos grupos: funcionales y técnicas.

Características Funcionales Excluidas

Las siguientes características del sistema no serán evaluadas debido a que no forman parte del alcance definido en el anteproyecto o porque su validación requiere esfuerzos que exceden los recursos del proyecto:

a) Funcionalidades avanzadas de facturación

- Cálculo de impuestos especiales.
- Ajustes automáticos de facturación.
- Integración con sistemas contables externos

b) Gestión completa del inventario

- Control de lotes y fechas de vencimiento.
- Alertas de stock bajo.
- Gestión detallada de proveedores.

c) Módulos o pantallas administrativas avanzadas

- Configuraciones del sistema.
- Gestión de roles avanzada (creación/edición de roles).
- Parámetros globales de la clínica.

d) Reportes avanzados

- Reportes estadísticos o analíticos.
- Exportaciones a Excel o PDF.

f) Flujos secundarios no críticos

- Recuperación de contraseña si no está implementada.
- Notificaciones por correo o SMS.
- Funcionalidades no documentadas en la aplicación evaluada.

Características Técnicas Excluidas

Estos elementos se consideran fuera de alcance debido a su complejidad técnica o porque no fueron contemplados en el anteproyecto.

a) Pruebas de rendimiento avanzadas

- Pruebas de carga con usuarios concurrentes.
- Pruebas de estrés o resistencia prolongada.
- Mediciones de uso de CPU/memoria en el servidor.

b) Pruebas de seguridad ofensivas

- Ethical hacking o penetration testing.
- Pruebas de inyección SQL, XSS avanzadas, CSRF, etc.
- Validación de cifrado en tránsito o en reposo.

c) Compatibilidad multiplataforma compleja

- Pruebas en dispositivos móviles.
- Pruebas en navegadores poco utilizados (Safari, Edge)
- Pruebas en múltiples resoluciones o sistemas operativos.

d) Dependencias externas

- Validación profunda de integraciones con servicios externos.
- Comportamientos ante fallas de red o caída de servicios externos.

e) Automatización total del sistema

- Cobertura automatizada del 100% de funcionalidades.
- Automatización de casos de uso complejos o secundarios.

Elementos Organizacionales Excluidos

- Soporte posterior a la entrega.
- Mantenimiento correctivo o evolutivo del sistema.
- Participación en ajustes o refactorización del código de la aplicación.

2.9 Alcance

El alcance del presente Plan de Pruebas establece los límites, profundidad y enfoque del proceso de verificación y validación aplicado al sistema de gestión de historiales médicos de la Veterinaria Mitsum. Su propósito es definir de manera clara qué será evaluado y bajo qué condiciones, asegurando que las actividades se ajusten a los recursos, tiempo y objetivos académicos del proyecto.

Este plan abarca la evaluación de la versión 1.0 del sistema, considerando únicamente las funcionalidades implementadas y disponibles en los ambientes de pruebas y producción al momento del estudio.

Componentes funcionales incluidos en el proceso de pruebas

El alcance incluye la validación de los módulos esenciales para la operación del sistema, agrupados según su función principal:

- **Autenticación y control de accesos**

  Verificación del inicio de sesión y funcionamiento básico del control de roles y permisos.

- **Gestión de pacientes y propietarios**

  Evaluación de los flujos CRUD para ambas entidades y de su relación operativa.

- **Historial clínico**

  Validación del registro, consulta y actualización de diagnósticos, tratamientos y notas clínicas.

- **Citas y cirugías**

  Revisión de la programación, modificación y consulta de citas y procedimientos.

- **Módulos de apoyo**

  Verificación funcional básica de facturación, inventario general y gestión de medicamentos.

Estos módulos representan el núcleo del sistema y constituyen el criterio de priorización para la elaboración de los casos de prueba.

Tipos de prueba contemplados

El alcance técnico del plan incluye las siguientes actividades:

- Pruebas funcionales manuales, aplicando técnicas de diseño de casos recomendadas por ISTQB (caja negra).
- Pruebas automatizadas de alcance limitado, orientadas a flujos críticos utilizando Playwright (E2E) y SuperTest (API).
- Pruebas de integración funcional, verificando la correcta interacción entre frontend (React) y backend (NestJS).
- Ejecución de pruebas en integración continua, mediante un pipeline básico configurado en GitHub Actions.

Estas pruebas permiten cubrir los escenarios más relevantes para la operación normal del sistema dentro del marco temporal disponible.

Atributos de Calidad a Evaluar

De acuerdo con el modelo ISO/IEC 25010, la evaluación se centrará en la verificación de los siguientes atributos de calidad, considerados críticos para el dominio de la salud:

- **Funcionalidad:** Comportamiento correcto según los requisitos especificados por la clínica y la documentación disponible.
- **Usabilidad:** Facilidad de uso para el personal clínico y administrativo.
- **Seguridad:** Control de accesos y protección de la confidencialidad de los datos.
- **Rendimiento:** Tiempos de respuesta básicos en operaciones frecuentes.
- **Compatibilidad:** Funcionamiento correcto en los navegadores web utilizados por la clínica.

Exclusiones Explícitas

Para mantener la viabilidad del proyecto y su alineación con los objetivos académicos, quedan expresamente fuera del alcance las siguientes actividades:

- Desarrollo de nuevas funcionalidades o modificación del código del sistema.
- Pruebas de rendimiento avanzado (carga, estrés o estabilidad prolongada).
- Pruebas de seguridad ofensivas (penetration testing).
- Automatización del 100% de los casos de prueba.
- Modificaciones en la infraestructura o arquitectura del sistema.
- Soporte, mantenimiento o corrección de defectos posterior a la entrega.

En resumen, este alcance establece un marco de trabajo realista, permitiendo una buena evaluación de los componentes más críticos del sistema, tal como fue planificado en el anteproyecto.

2.10 Criterios de aprobación/fallo

Los criterios de aprobación y fallo establecen las condiciones bajo las cuales se determina el resultado de cada caso de prueba durante la ejecución del plan. Estos criterios permiten asegurar uniformidad, objetividad y trazabilidad en la evaluación del comportamiento del sistema, conforme a lo establecido por el estándar IEEE-829.

1. Criterios de aprobación

Un caso de prueba se considerará aprobado cuando cumpla con todos los siguientes puntos:

- El comportamiento observado coincide con el resultado esperado definido en la especificación del caso de prueba.
- No se presentan errores visibles, mensajes inesperados o fallas de validación.
- Los datos generados, actualizados o consultados se reflejan correctamente en el sistema y en la base de datos.
- Los tiempos de respuesta se mantienen dentro de rangos aceptables para el flujo evaluado.
- No se requieren acciones adicionales, correcciones o reintentos para completar el flujo.
- La evidencia registrada (capturas, logs o resultados automatizados) confirma el comportamiento esperado.

2. Criterios de fallo

Un caso de prueba se considerará fallido cuando ocurra una o más de las siguientes situaciones:

- El resultado observado difiere parcial o totalmente del resultado esperado.
- Se presenta un error del sistema (excepción, pantalla en blanco, bloqueo, código HTTP incorrecto).
- Se generan datos incompletos, incorrectos o inconsistentes.
- La ejecución del flujo queda interrumpida o no puede completarse.
- Se visualizan elementos incorrectos o ausentes (botones, mensajes, formularios).
- El caso depende de un prerrequisito que no pudo cumplirse debido a un defecto previo.
- La evidencia recopilada demuestra un comportamiento anómalo o no determinístico.

3. Casos con estado "Condicional" o "No ejecutado"

Además de los estados estándar, podrán registrarse las siguientes condiciones:

- **Condicional (Blocked):** El caso no puede ejecutarse debido a un defecto abierto que bloquea el flujo.
- **No ejecutado (Not Run):** El caso no fue ejecutado por limitaciones de tiempo o disponibilidad de ambiente.

Estos estados no se consideran fallos, pero sí afectan la completitud del ciclo de pruebas y deben documentarse.

4. Criterio general de evaluación

El conjunto de casos de prueba evaluados se considerará aceptable cuando:

- El porcentaje de casos aprobados sea significativamente mayor que los fallidos.
- No existan defectos críticos abiertos que afecten flujos esenciales.
- Los casos bloqueados estén justificados y documentados.

- Las evidencias sean suficientes para respaldar los resultados registrados.

Este criterio permite establecer un proceso de verificación claro, objetivo y consistente, asegurando que las decisiones tomadas durante la ejecución de pruebas se encuentren debidamente fundamentadas y respaldadas por evidencia.

2.11 Criterios de suspensión y requisitos de reanudación

Los criterios de suspensión y reanudación definen las condiciones bajo las cuales se detendrá temporalmente la ejecución del plan de pruebas y los parámetros necesarios para retomar las actividades. Este mecanismo permite mantener la integridad del proceso de verificación, prevenir el uso ineficiente de recursos y garantizar que las pruebas se realicen en condiciones estables, en concordancia con el estándar IEEE-829.

1. Criterios de suspensión de pruebas

La ejecución del proceso de pruebas será suspendida cuando ocurra cualquiera de las siguientes situaciones:

a) Existencia de defectos críticos o bloqueantes

Las pruebas se suspenderán cuando se identifique un defecto que:

- Impida continuar el flujo evaluado.
- Afecte un módulo esencial del sistema
- Genere errores que imposibiliten avanzar hacia otros casos dependientes.

Ejemplos: fallas en autenticación, caída del servidor, interrupción de la API, fallos críticos en la base de datos.

b) Inestabilidad del ambiente de pruebas

Si el entorno local o el ambiente desplegado presenta:

- Intermitencias.
- Tiempos de respuesta extremadamente altos.

- Caídas repetidas.
- Inconsistencias entre frontend y backend.

Entonces se suspenderá la ejecución hasta restablecer condiciones mínimas de estabilidad.

c) Datos de prueba insuficientes o corruptos

Las pruebas se detendrán cuando:

- Los datos necesarios para ejecutar un caso no puedan crearse.
- Existan corrupciones o duplicados que afecten la ejecución.
- Se presenten inconsistencias entre ambientes.

d) Problemas con herramientas o automatizaciones

Se suspenderán las pruebas si:

- El pipeline de CI falla de forma consecutiva.
- Las herramientas de automatización no pueden ejecutar scripts.
- Ocurren bloqueos derivados de fallas externas (librerías, servicios, dependencias).

e) Falta temporal de recursos del equipo

La ejecución se pausará si por razones académicas, técnicas o de coordinación, no se dispone del mínimo personal requerido para validar los resultados o registrar evidencia.

2. Criterios de reanudación de pruebas

El proceso podrá reanudarse cuando se cumpla una o más de las siguientes condiciones:

a) Corrección de defectos críticos

Los defectos que motivaron la suspensión deben:

- Estar solucionados.

- Haber sido verificados por el equipo.
- Permitir la ejecución completa del flujo afectado.

b) Estabilización del ambiente

El ambiente de pruebas debe funcionar bajo condiciones normales:

- API en funcionamiento.
- Acceso completo al sistema.
- Tiempos de respuesta razonables.
- Coherencia entre módulos.

c) Disponibilidad de datos de prueba válidos

Se reanudará el proceso cuando los datos necesarios:

- Hayan sido regenerados.
- Sean consistentes.
- Permitan continuar la ejecución sin bloqueos.

d) Restauración o actualización de herramientas

Las pruebas automatizadas podrán retomarse cuando:

- El pipeline de CI/CD se encuentre operativo.
- Las librerías o dependencias estén actualizadas.
- Playwright, Jest o SuperTest funcionen correctamente.

e) Disponibilidad del equipo de pruebas

La reanudación está sujeta a que los integrantes del equipo puedan retomar las actividades asignadas y continuar con la documentación correspondiente.

2.12 Entregables de prueba

Los entregables del proceso de pruebas constituyen la evidencia documental generada durante la planificación, ejecución y cierre del ciclo de verificación. De acuerdo con el estándar IEEE-829, estos artefactos permiten garantizar trazabilidad, uniformidad y claridad en la evaluación del sistema.

Para el presente proyecto, los entregables que se producirán son los siguientes:

1. Test Plan - Plan de Pruebas (El Presente Documento)

Contiene la estrategia, alcance, riesgos, criterios y lineamientos generales para la ejecución del proceso de pruebas de acuerdo con IEEE-829.

2. Test Case Specification - Casos de Prueba

Documento que detalla cada caso de prueba, incluyendo:

- Identificador.
- Descripción.
- Precondiciones.
- Pasos (Test Procedure Specification).
- Datos de prueba.
- Resultados esperados y observados.

Se desarrollará para todos los flujos priorizados del sistema.

3. Matriz de Trazabilidad de Pruebas - Ejecución de pruebas

Documento que relaciona:

- Módulos.
- Requisitos funcionales.
- Casos de prueba.
- Defectos reportados.
- Resultados obtenidos.
- Permite asegurar la cobertura mínima requerida y el seguimiento del progreso.

- Capturas de pantalla.
- Registros del sistema (Logs).
- Reportes generados por las herramientas (Automatizadas).

Este material respalda los resultados obtenidos durante la ejecución y sirve como soporte documental para el informe final.

5. Test Incident Report - Registro de Defectos

Documento que recoge los defectos encontrados, clasificándolos por:

- Severidad.
- Prioridad.
- Módulo afectado.
- Estado del incidente.
- Evidencia adjunta.

Permitirá el seguimiento de los problemas detectados durante la ejecución.

6. Scripts de Pruebas Automatizadas (GitHub)

Conjunto de scripts desarrollados en:

- Playwright (E2E).
- SuperTest (API).
- Jest (integración básica).

Se incluirán junto con la evidencia de su ejecución, ya sea local o por CI/CD.

7. Test Summary Report - Cobertura de módulos

Documento de cierre que contiene:

- Análisis global de los resultados.
- Resumen estadístico.
- Defectos detectados y su estado.
- Nivel de cobertura alcanzado.

Este entregable formaliza el final del ciclo de pruebas y respalda el capítulo III de esta tesina.

2.13 Tareas de pruebas restantes

Las tareas de pruebas restantes corresponden a las actividades que aún deben ejecutarse para completar el ciclo de verificación definido en el presente plan. Estas tareas se enfocan en las fases de diseño final, ejecución, seguimiento y cierre del proceso de pruebas, considerando que parte del análisis inicial y la preparación del entorno ya se llevaron a cabo durante las primeras etapas del proyecto.

Las actividades pendientes se detallan a continuación:

1. Finalización del diseño de casos de prueba

Aunque se han identificado los módulos prioritarios, aún es necesario:

- Completar la redacción formal de todos los casos de prueba.
- Verificar la consistencia entre escenarios, datos y resultados esperados.
- Actualizar la matriz de trazabilidad con todos los casos aprobados.

2. Preparación final de datos de prueba

Incluye:

- Generación de registros de mascotas y propietarios necesarios para múltiples escenarios.
- Creación de datos médicos (diagnósticos, tratamientos, cirugías) para pruebas específicas.
- Asegurar que los datos sean reproducibles tanto en pruebas manuales como automatizadas.

3. Ejecución de pruebas funcionales manuales

Las pruebas por módulo deben completarse en el ambiente de pruebas, considerando:

- Pruebas CRUD de pacientes y propietarios.
- Pruebas de autenticación y restricciones por rol.

- Pruebas de historial clínico, citas, cirugías, medicamentos e inventario.
- Registro de evidencia para cada caso ejecutado.

4. Desarrollo y ejecución de pruebas automatizadas

Tareas pendientes:

- Finalizar la creación de scripts de Playwright para los flujos críticos.
- Completar las pruebas de API con SuperTest.
- Integrar suites en GitHub Actions.
- Ejecutar automatizaciones y registrar evidencia de resultados.

5. Registro y seguimiento de incidencias

Quedan pendientes:

- Documentar defectos detectados durante la ejecución.
- Clasificarlos según severidad y prioridad.
- Actualizar el registro de incidencias hasta su cierre o justificación.

6. Verificación de criterios de aprobación y salida

Antes de finalizar el proyecto, es necesario:

- Confirmar que todos los casos críticos han sido ejecutados.
- Revisar la completitud de la evidencia.
- Documentar cualquier caso no ejecutado o bloqueado.

7. Elaboración del documento final de Tesina

Aún está pendiente la construcción del documento que incluirá:

- Resultados consolidados.
- Estadísticas de ejecución.
- Estado final de incidencias.
- Conclusiones sobre la calidad del sistema.
- Recomendaciones y observaciones finales.

Estas tareas representan el trabajo restante para completar el proceso de pruebas descrito en este plan y constituyen la base del trabajo que se presentará en el Capítulo III.

2.14 Ambientes

Para la ejecución del plan de pruebas se utilizaron dos ambientes diferenciados, cada uno con propósitos específicos dentro del ciclo de verificación. La correcta separación entre el ambiente de producción y el ambiente de pruebas garantiza la integridad de los datos reales de la clínica y permite realizar pruebas controladas sin afectar las operaciones del sistema en uso.

1. Ambiente de Producción

Corresponde al sistema oficial utilizado por el personal de la Clínica Veterinaria Mitsum. Este ambiente contiene datos reales y opera de manera continua, por lo que no se realizan pruebas dentro de él.

Características principales:

URL: https://www.veterianariamitsum.life

Versión actual: 1.0

Disponibilidad: 24/7 para actividades de la clínica

Datos: Información real de clientes, mascotas y procesos clínicos

Restricciones:

- No se ejecutan pruebas funcionales ni automatizadas
- No se generan datos de prueba
- No se manipulan registros reales

El ambiente se utiliza únicamente como referencia para validar el comportamiento esperado del sistema.

2. Ambiente de Pruebas

Este ambiente fue creado específicamente para la tesina y replica la estructura y funcionalidades del ambiente de producción, pero con datos independientes. Permite realizar pruebas manuales y automatizadas sin riesgo para la operación real del sistema.

Accesos del ambiente de pruebas:

Frontend:

https://vet-mitsun-development.vercel.app

Backend y documentación de API:

https://dsi-nest-backend-development.up.railway.app/api/v1/docs/

Características principales:

Versión: 1.0 (Idéntica a producción).

Código fuente: versionado en GitHub con ramas independientes para pruebas.

Uso durante el proyecto:

- Ejecución de pruebas funcionales manuales.
- Desarrollo y ejecución de scripts automatizados (Playwright y SuperTest).
- Generación de datos de prueba.
- Validación de flujos completos sin afectar la operación real.

Base de datos:

Contiene únicamente información generada para pruebas

Credenciales utilizadas:

- Usuario con rol Administrador (para pruebas completas).
- Usuario con rol Cliente (para validar restricciones y visibilidad limitada).

3. Diferencias principales entre ambientes

Tabla 3-Diferencias principales entre ambientes

Característica	Producción	Pruebas
Uso principal	Operación real de la clínica	Ejecución del plan de pruebas
Datos	Reales	Ficticios o controlados
Automatización	No permitida	Permitida
Modificaciones	Restringidas	Permitidas para pruebas
Estabilidad	Crítica	Media (pueden reiniciarse servicios)

2.15 Equipo de trabajo y capacitación

El proceso de pruebas del sistema fue desarrollado por un equipo de cuatro integrantes, todos estudiantes de Ingeniería en Sistemas Informáticos. Debido a la naturaleza académica del proyecto, el equipo trabajó de forma colaborativa, combinando habilidades y conocimientos adquiridos durante la carrera con procesos de autoformación específicos para este plan de pruebas.

1. Integrantes del equipo de trabajo

Tabla 4-Integrantes del equipo de trabajo

Integrante	Rol	Carnet
Flores Mendoza Fabio Ernesto	Encargado de Automatización	FM19038
García Torres William Stanley	Diseñador de Casos de Prueba	GT11003
Gutiérrez Escobar Juan Manuel	Ejecutor de Pruebas Funcionales	GE19020
Hernández Sánchez Edwin Alexander	Líder de Pruebas	HS19011

2. Capacitación del equipo

Para llevar a cabo el proceso de pruebas, el equipo cuenta con conocimientos adquiridos durante la carrera de Ingeniería de Sistemas Informáticos y complementados mediante:

a) Conocimientos previos adquiridos

- Pruebas funcionales y diseño de casos (materias de Calidad de Software).
- Uso de herramientas de versionado (Git/GitHub).
- Desarrollo y análisis de aplicaciones web (React, NestJS).
- Conocimientos básicos de APIs REST.

b) Capacitación adicional realizada para el proyecto

- El equipo realizó actividades autodidactas y prácticas para garantizar el correcto uso de las herramientas del proyecto:
- Capacitación en el uso de Playwright para pruebas E2E.
- Capacitación en SuperTest para pruebas de API.
- Revisión de buenas prácticas del estándar IEEE-829.
- Repaso del modelo de calidad ISO/IEC 25010.
- Análisis exploratorio del sistema para identificar flujos reales.

2.16 Responsabilidades

Para asegurar la correcta ejecución del plan de pruebas, se asignaron responsabilidades específicas a cada rol operativo. Estas responsabilidades están alineadas con las buenas prácticas descritas en el estándar IEEE-829 y permiten mantener orden, trazabilidad y claridad durante todo el proceso.

a) Líder de Pruebas

Responsable de:

- Supervisar el cumplimiento del plan.
- Coordinar la ejecución por módulo.

- Validar criterios de aprobación y salida.
- Consolidar entregables finales.

b) Diseñador de Casos de Prueba

Responsable de:

- Elaborar la Especificación de Casos de Prueba.
- Definir datos de prueba consistentes.
- Actualizar la matriz de trazabilidad.

c) Ejecutor de Pruebas Funcionales

Responsable de:

- Ejecutar pruebas manuales según los casos definidos.
- Registrar evidencia y resultados observados.
- Reportar defectos detectados.

d) Encargado de Automatización

Responsable de:

- Desarrollar scripts de prueba E2E y API.
- Ejecutar automatizaciones localmente y vía CI/CD.
- Registrar evidencias automatizadas.

Matriz RACI del Proyecto de Pruebas

Tabla 5-Matriz RACI del Proyecto de Pruebas

Actividad	Líder de Pruebas	Diseñador de Casos de Prueba	Ejecutor de Pruebas Manuales	Encargado de Automatización	Usuario Final / Clínica
Planificación del plan de pruebas	A/R	C	C	C	I
Diseño de casos de prueba manuales	A	R	C	C	I
Diseño de datos de prueba	A	R	C	C	I
Desarrollo de scripts automatizados	C	I	I	A/R	I
Ejecución de pruebas manuales	C	C	A/R	I	C
Ejecución de pruebas automatizadas	C	I	I	A/R	I
Registro, reporte y clasificación de incidencias	A	C	R	R	I
Actualización de trazabilidad	A	R	C	C	I
Validación final del plan de pruebas	A	R	C	C	I
Elaboración del Documento	A/R	C	C	C	I

Leyenda:

- R = Responsible (ejecuta)
- A = Accountable (responsable final)
- C = Consulted (colabora)
- I = Informed (informado)

2.17 Calendario

Esta sección presenta el calendario oficial del proyecto de tesina, de acuerdo con las etapas establecidas por la Escuela de Ingeniería de Sistemas Informáticos.

Tabla 6-Calendario Oficial del Proyecto de Tesinas

Etapa	Periodo
Investigación y Anteproyecto	Agosto - octubre 2025
Desarrollo de la Tesina (Capítulos I, II, III)	Octubre - noviembre 2025
Entrega Final de Tesina	Diciembre 2025
Defensa Tesina	Diciembre 2025

2.18 Plan de riesgos y contingencias

El presente plan identifica los riesgos que pueden afectar la ejecución del proceso de pruebas y establece las acciones de mitigación y contingencia necesarias para mantener la continuidad del proyecto. Los riesgos se evaluaron considerando la naturaleza académica del trabajo, los recursos disponibles, las características técnicas del sistema y el tiempo asignado para desarrollar la tesina.

Tabla 7-Matriz de Riesgos del Proyecto de Pruebas-A

ID	Riesgo	Prob.	Impacto	Descripción / Causas	Mitigación (Prevención)	Contingencia (Acción Correctiva)
R1	Inestabilidad del ambiente de pruebas	Media	Alto	El backend o frontend del entorno de pruebas puede caerse, presentar lentitud o errores por fallos en Railway/Vercel.	Verificar el estado del entorno antes de cada sesión; usar datos de prueba controlados; ejecutar flujos críticos primero.	Reiniciar despliegue, regenerar base de datos o usar Postman/scripts locales cuando el entorno no esté disponible.
R2	Falta de documentación oficial del sistema	Alta	Medio	No existen documentos formales de requisitos o arquitectura; se depende del análisis exploratorio.	Realizar pruebas exploratorias detalladas; validar comportamientos reales del sistema; mantener consistencia interna.	Ajustar casos de prueba basados en el comportamiento observado; actualizar trazabilidad según hallazgos.
R3	Defectos críticos en módulos esenciales	Media	Alto	Fallos en autenticación, historial clínico o APIs pueden bloquear múltiples flujos.	Priorizar pruebas de módulos críticos; diseñar casos escalonados por dependencia.	Registrar el defecto como “bloqueante”, saltar módulo afectado y continuar con otros; reorganizar el cronograma.

Tabla 8-Matriz de Riesgos del Proyecto de Pruebas B

ID	Riesgo	Prob.	Impacto	Descripción / Causas	Mitigación (Prevención)	Contingencia (Acción Correctiva)
R4	Fallos en herramientas de automatización (Playwright / SuperTest / CI/CD)	Media	Medio	Dependencias rotas, incompatibilidades o errores en GitHub Actions.	Mantener versiones estables; ejecutar pruebas locales antes del pipeline; validar configuración del entorno.	Ejecutar los scripts manualmente o deshabilitar temporalmente la automatización; continuar con pruebas manuales.
R5	Limitaciones de tiempo por carga académica	Alta	Medio	El proyecto se desarrolla en un periodo reducido con múltiples tareas universitarias simultáneas.	Planificación semanal; distribución equilibrada de tareas; priorización basada en criticidad.	Reasignación temporal de tareas entre miembros; reducir ejecución de pruebas no prioritarias.
R6	Datos de prueba insuficientes o inconsistentes	Media	Medio	Datos mal generados pueden causar falsos fallos o bloquear flujos dependientes.	Crear conjunto base de datos semilla; separar datos por módulo; validar usuarios y roles.	Limpiar o regenerar datos manualmente; usar scripts rápidos o recrear los datos desde cero.

Tabla 9-Matriz de Riesgos del Proyecto de Pruebas-C

ID	Riesgo	Prob.	Impacto	Descripción / Causas	Mitigación (Prevención)	Contingencia (Acción Correctiva)
R7	Cambios inesperados en la versión del sistema	Baja	Alto	Un cambio realizado en producción o pruebas puede alterar flujos ya planificados.	Bloqueo de versiones durante las pruebas; control estricto del repositorio y ramas.	Realizar pruebas de regresión sobre módulos impactados; actualizar casos afectados y su trazabilidad.
R8	Recursos técnicos limitados (hardware o conexión)	Baja	Medio	Fallos de conexión, lentitud local o equipos compartidos pueden afectar pruebas automatizadas.	Ejecutar pruebas en horarios estables; mantener respaldo de scripts y configuraciones locales.	Cambiar de equipo temporalmente; ejecutar pruebas en modo reducido o desde CI/CD.
R9	Falta de experiencia previa en herramientas de automatización	Media	Medio	El equipo adquiere Playwright y SuperTest durante el proyecto, pudiendo cometer errores iniciales.	Capacitación interna; pruebas piloto antes de automatizar flujos reales.	Simplificar scripts; automatizar solo flujos esenciales hasta estabilizar la técnica.

2.19 Aprobaciones

La presente sección consigna la aprobación formal del Plan de Pruebas correspondiente al sistema de gestión de historiales médicos de la Clínica Veterinaria Mitsum. Su validación confirma que los lineamientos, responsabilidades, estrategias y criterios definidos en este documento han sido revisados y aceptados por las partes involucradas, autorizando su ejecución conforme a las buenas prácticas establecidas en el estándar IEEE-829.

La aprobación constituye el punto de inicio oficial del proceso de pruebas y garantiza que las actividades podrán desarrollarse con base en la estructura y los recursos definidos en el plan.

Tabla 10-Aprobaciones

Nombre	Rol según el Plan de Pruebas	Organización / Grupo	Firma	Fecha
Hernández Sánchez Edwin Alexander (HS19011)	Líder de Pruebas	Grupo 01 – Tesina		29 / 11 / 2025
García Torres William Stanley (GT11003)	Diseñador de Casos de Prueba	Grupo 01 – Tesina		29 / 11 / 2025
Flores Mendoza Fabio Ernesto (FM19038)	Encargado de Automatización	Grupo 01 – Tesina		29 / 11 / 2025
Gutiérrez Escobar Juan Manuel (GE19020)	Ejecutor de Pruebas Funcionales	Grupo 01 – Tesina		29 / 11 / 2025

CAPÍTULO III. CASO DE ESTUDIO

3.1 Antecedentes

La Clínica Veterinaria Mitsum es un establecimiento dedicado a brindar servicios de consulta, diagnóstico, procedimientos médicos y atención integral para mascotas. Su operación diaria involucra actividades como la gestión de propietarios y pacientes, el registro de consultas, la administración de tratamientos y vacunaciones, la programación de cirugías y el control de inventario de medicamentos. Debido a la naturaleza clínica de sus servicios, la precisión y trazabilidad de la información médica son factores fundamentales para garantizar la continuidad y la calidad de la atención.

Históricamente, la clínica ha gestionado gran parte de su información mediante registros manuales o herramientas no integradas. Este enfoque generaba diversos inconvenientes, entre ellos: duplicidad de datos, pérdida de información, dificultades para recuperar historiales médicos, lentitud en procesos administrativos y ausencia de un control centralizado de citas y diagnósticos. Estas limitaciones afectaban la eficiencia operativa y dificultaban la toma de decisiones clínicas en tiempo oportuno.

Frente a esta necesidad, surgió la iniciativa de desarrollar un sistema digital que permitiera centralizar, consultar y actualizar los historiales médicos de las mascotas de manera estructurada y segura. El resultado fue el Sistema de Gestión de Historiales Médicos de la Clínica Veterinaria Mitsum, actualmente en su versión 1.0, construido bajo una arquitectura cliente-servidor e integrado por módulos como autenticación, gestión de propietarios y mascotas, historial clínico, control de citas, programación de cirugías, inventario y facturación básica.

El sistema fue diseñado con el propósito de modernizar los procesos internos, reducir errores derivados del manejo manual de información y fortalecer la capacidad operativa de la clínica. A través de este desarrollo, la institución busca mejorar la calidad del servicio, agilizar la atención a los pacientes y ofrecer herramientas que permitan un seguimiento clínico más efectivo.

En el marco del presente proyecto académico, este sistema sirve como base para la elaboración y aplicación de un Plan de Pruebas formal. Su evaluación permite identificar oportunidades de mejora, verificar la estabilidad funcional de cada módulo y asegurar que la digitalización cumpla con los estándares de calidad establecidos por ISO/IEC 25010, IEEE-829 e ISTQB. Los antecedentes aquí descritos proporcionan el contexto necesario para el análisis del problema y el modelado del negocio, que se desarrollan en las siguientes secciones.

3.2 Contexto del problema

El sistema de gestión de historiales médicos desarrollado para la Clínica Veterinaria Mitsum se encuentra en funcionamiento preliminar, pero aún no ha sido sometido a un proceso formal de evaluación de calidad.

Si bien el sistema cumple con sus funciones principales y solo se han identificado un número reducido de defectos menores durante su uso inicial, la ausencia de una metodología de pruebas estructurada representa un riesgo para su adopción completa dentro de la clínica.

El anteproyecto señala que, debido a la naturaleza clínica y al tipo de información manejada, es indispensable contar con una validación rigurosa que permita asegurar que el sistema opera de manera estable, continua y conforme a los requerimientos funcionales establecidos.

Esta evaluación no se limita a la detección de errores existentes, sino que busca garantizar que no existan fallos ocultos que puedan afectar procesos como el registro de pacientes, la consulta de historiales médicos, la gestión de tratamientos o el flujo de autenticación de usuarios.

Además, se reconoce la importancia de verificar la coherencia entre los flujos reales de la clínica y las funcionalidades implementadas, con el fin de asegurar que el sistema sea intuitivo, útil y alineado al trabajo del personal veterinario, administrativo y de recepción.

Aunque inicialmente la veterinaria no se ha detectado problemas graves, la falta de un plan de pruebas formal implica que ciertos riesgos en áreas como usabilidad, seguridad

y manejo de datos podrían pasar desapercibidos si no se realiza una validación exhaustiva.

Por esta razón, el presente estudio se centra en diseñar y ejecutar un plan integral de pruebas manuales y automatizadas, que permita confirmar el correcto funcionamiento del sistema, identificar oportunidades de mejora y garantizar que el software esté listo para su uso confiable en las operaciones diarias de la Clínica Veterinaria Mitsum.

3.3 Modelado de Negocio

La Clínica Veterinaria Mitsum ofrece servicios de atención médica para mascotas domésticas y exóticas, abarcando consultas generales, tratamientos, aplicación de vacunas, procedimientos quirúrgicos y seguimiento clínico. El negocio depende de la administración adecuada de los historiales médicos, ya que estos contienen toda la información clínica necesaria para diagnósticos y decisiones terapéuticas.

El sistema de gestión de historiales médicos forma parte del núcleo operativo de la clínica, al permitir registrar, consultar y actualizar información asociada a pacientes y propietarios. Adicionalmente, facilita la coordinación entre veterinarios, asistentes y personal administrativo, asegurando que todos cuenten con acceso oportuno a la información requerida para brindar una atención adecuada.

Actores del negocio

Dentro de la operación de la Clínica Veterinaria Mitsum existen diferentes puestos de trabajo que intervienen en los procesos clínicos y administrativos diarios. Sin embargo, desde la perspectiva del sistema de gestión de historiales médicos, únicamente se han definido dos roles técnicos: administrador y cliente. Todos los usuarios internos de la clínica operan bajo el rol de administrador, ya que requieren acceso completo a las funcionalidades del sistema para realizar sus labores. A continuación, se describen los actores considerando tanto la estructura organizativa real como los roles implementados en el sistema.

- **Administrador (rol del sistema):**

  Es el único rol con acceso completo a todas las funcionalidades del sistema. Permite gestionar usuarios, registrar y consultar historiales médicos, administrar pacientes, consultas, tratamientos, cirugías y vacunas. En la práctica, este rol es utilizado por veterinarios, asistentes veterinarios, recepcionistas y personal administrativo, ya que todos necesitan acceso amplio para cumplir con sus funciones.

- **Cliente (rol del sistema):**

  Corresponde al usuario externo que interactúa únicamente con los módulos habilitados para servicios al cliente, como ver información general o ciertos datos asociados a su mascota, según lo definido por la clínica. Este rol no interviene en la gestión clínica directa ni en la administración del sistema.

- **Veterinario (puesto operativo, usa rol de administrador):**

  Profesional encargado de realizar consultas, diagnósticos, tratamientos, cirugías y registros clínicos. Aunque es un actor crítico del negocio, en el sistema opera utilizando el rol de administrador.

- **Asistente veterinario (puesto operativo, usa rol de administrador):**

  Personal de apoyo que colabora en la digitación de datos, preparación de expedientes, toma de signos o información preliminar. También trabaja con el rol de administrador dentro del sistema.

- **Recepcionista (puesto operativo, usa rol de administrador):**

  Responsable del registro de pacientes y propietarios, agendamiento de citas y localización de historiales. En términos del sistema, utiliza igualmente el rol de administrador.

- **Propietario de mascota (actor externo, se vincula al rol cliente):**

  No realiza operaciones internas dentro del sistema clínico, pero es el responsable del paciente y puede interactuar a través del rol cliente en funcionalidades definidas por la clínica.

Esta estructura permite diferenciar claramente entre puestos reales del negocio y roles implementados en el sistema, manteniendo coherencia entre el funcionamiento operativo de la clínica y la arquitectura técnica definida en la aplicación.

Módulos principales del sistema

El sistema está estructurado en módulos funcionales que permiten organizar y gestionar la información de forma eficiente. Los módulos principales son los siguientes:

- **Módulo de Autenticación:** Gestiona el inicio de sesión y la validación de credenciales, permitiendo que únicamente usuarios autorizados accedan al sistema.
- **Módulo de Pacientes:** Permite registrar y administrar la información de mascotas, incluyendo especie, raza, edad, características físicas y su asociación con un propietario.
- **Módulo de Consultas:** Registra todas las consultas médicas, diagnósticos, síntomas y notas relevantes del encuentro clínico.
- **Módulo de Usuarios:** Gestiona perfiles, roles y permisos del personal que utiliza el sistema.
- **Módulo de Historial Clínico:** Consolida la información clínica a lo largo del tiempo y permite la consulta centralizada del expediente del paciente.

Procesos clave del negocio

Los procesos principales que la clínica ejecuta diariamente y que se ven reflejados en el sistema son:

- **Registro de pacientes y propietarios:** Proceso mediante el cual la recepción ingresa la información básica necesaria para iniciar un historial médico.

- **Atención clínica:** Incluye la consulta veterinaria, la evaluación del paciente, el registro de signos, diagnóstico y prescripción del tratamiento.
- **Actualización del historial médico:** Cada interacción con el paciente genera datos **nuevos que deben incorporarse al expediente.**
- **Administración de tratamientos, vacunas y procedimientos:** Procesos que deben documentarse para garantizar la trazabilidad de la atención médica.
- **Gestión de usuarios y roles:** Asegura que cada miembro del personal tenga acceso únicamente a las funciones que le corresponden.
- **Consulta de información histórica:** Proceso necesario para decidir tratamientos, programar procedimientos y evaluar la evolución clínica del paciente.

Flujo general del negocio

El flujo general del negocio inicia con el registro o búsqueda del paciente por parte del personal de recepción. Posteriormente, el veterinario o asistente accede al historial médico y registra la consulta, diagnóstico y tratamiento correspondiente. Si se requieren procedimientos adicionales como vacunas o cirugías, estos se documentan en los módulos específicos. Finalmente, toda la información queda consolidada en el historial clínico, el cual puede ser consultado en futuras atenciones. Este flujo permite mantener un expediente actualizado y accesible, facilitando la continuidad del cuidado del paciente.

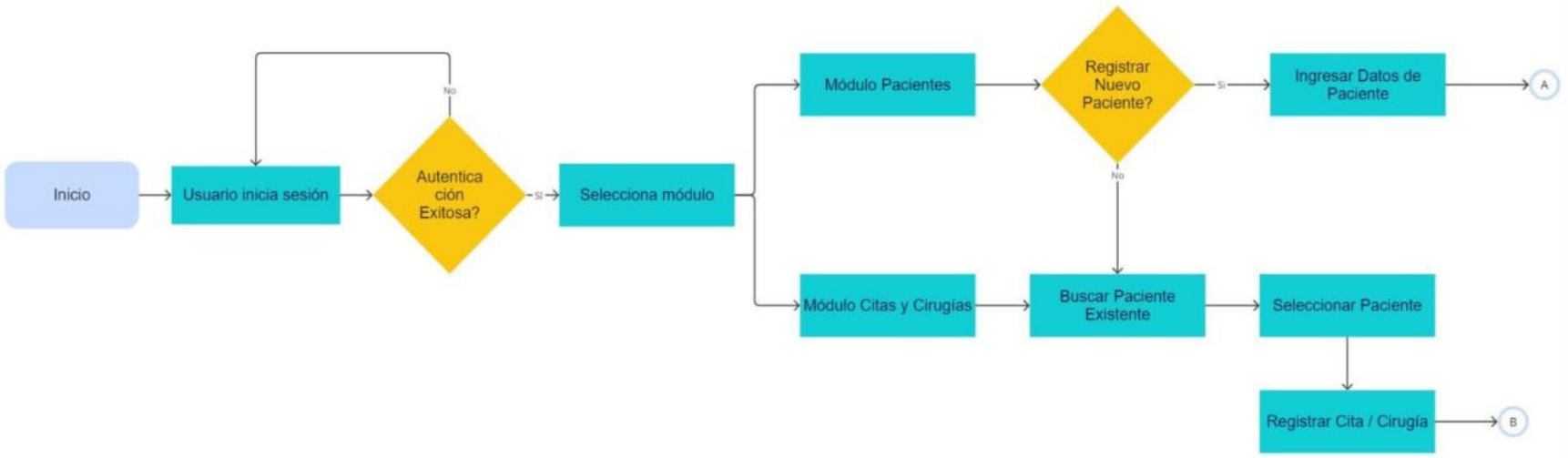


Ilustración 1- Flujo general del negocio A

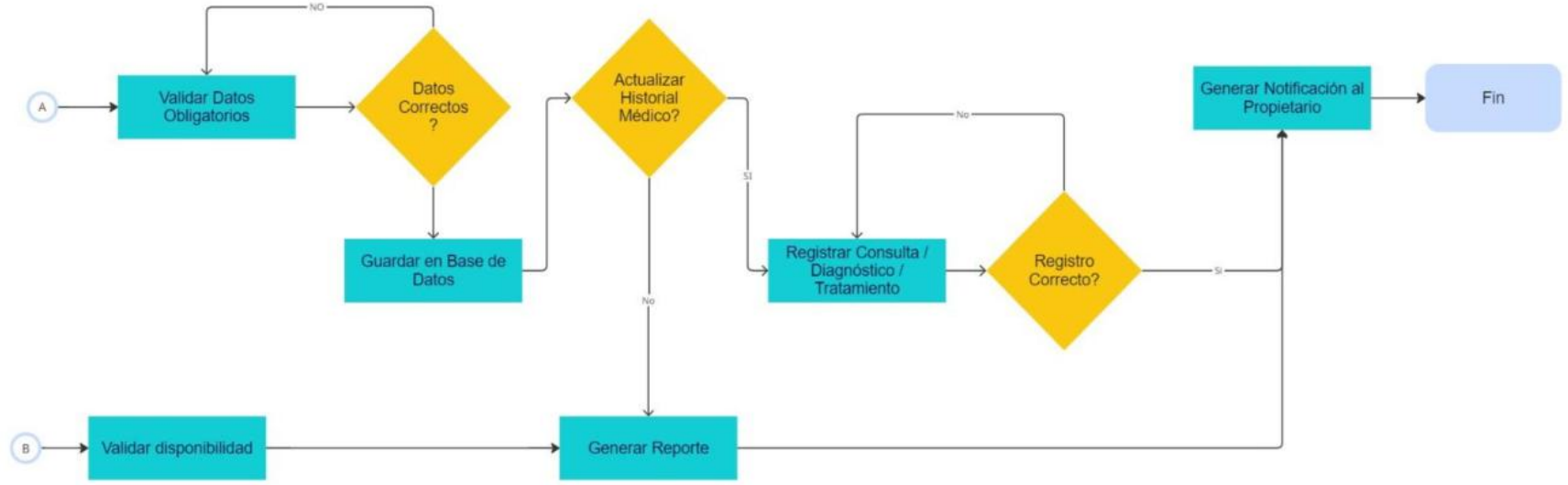


Ilustración 2-Flujo general del negocio B

Adicionalmente, para complementar la comprensión del funcionamiento interno de la clínica y visualizar la interacción entre los distintos actores y módulos del sistema, a continuación, se presentan los diagramas BPMN de algunos procesos operativos esenciales. Estos diagramas reflejan el flujo real de trabajo que se ejecuta diariamente en la Clínica Veterinaria Mitsum y permiten identificar los puntos críticos donde el sistema apoya, automatiza o registra información relevante.

Estos BPMN sirven como referencia para el análisis del negocio y para el diseño del plan de pruebas, ya que permiten relacionar cada actividad con las funcionalidades específicas del sistema.

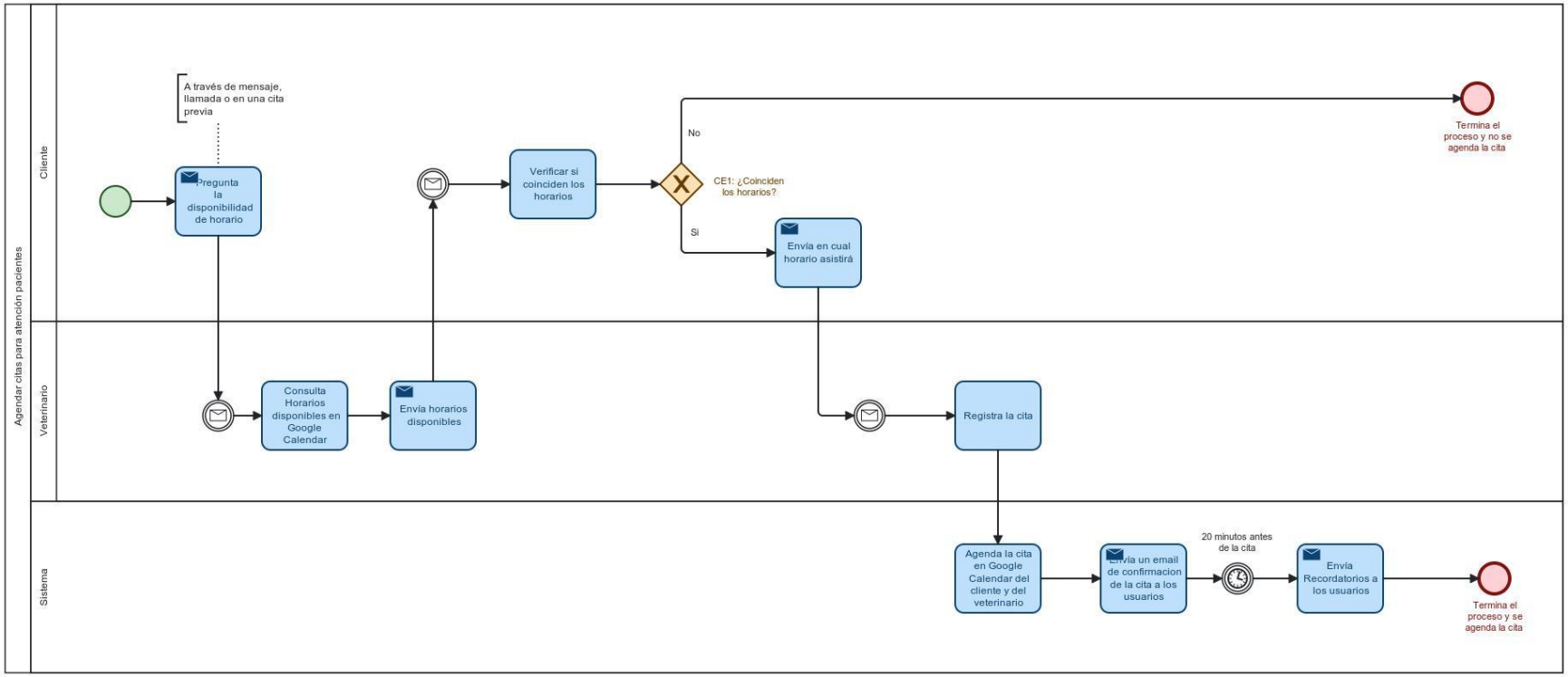


Ilustración 3-Agenda de Citas

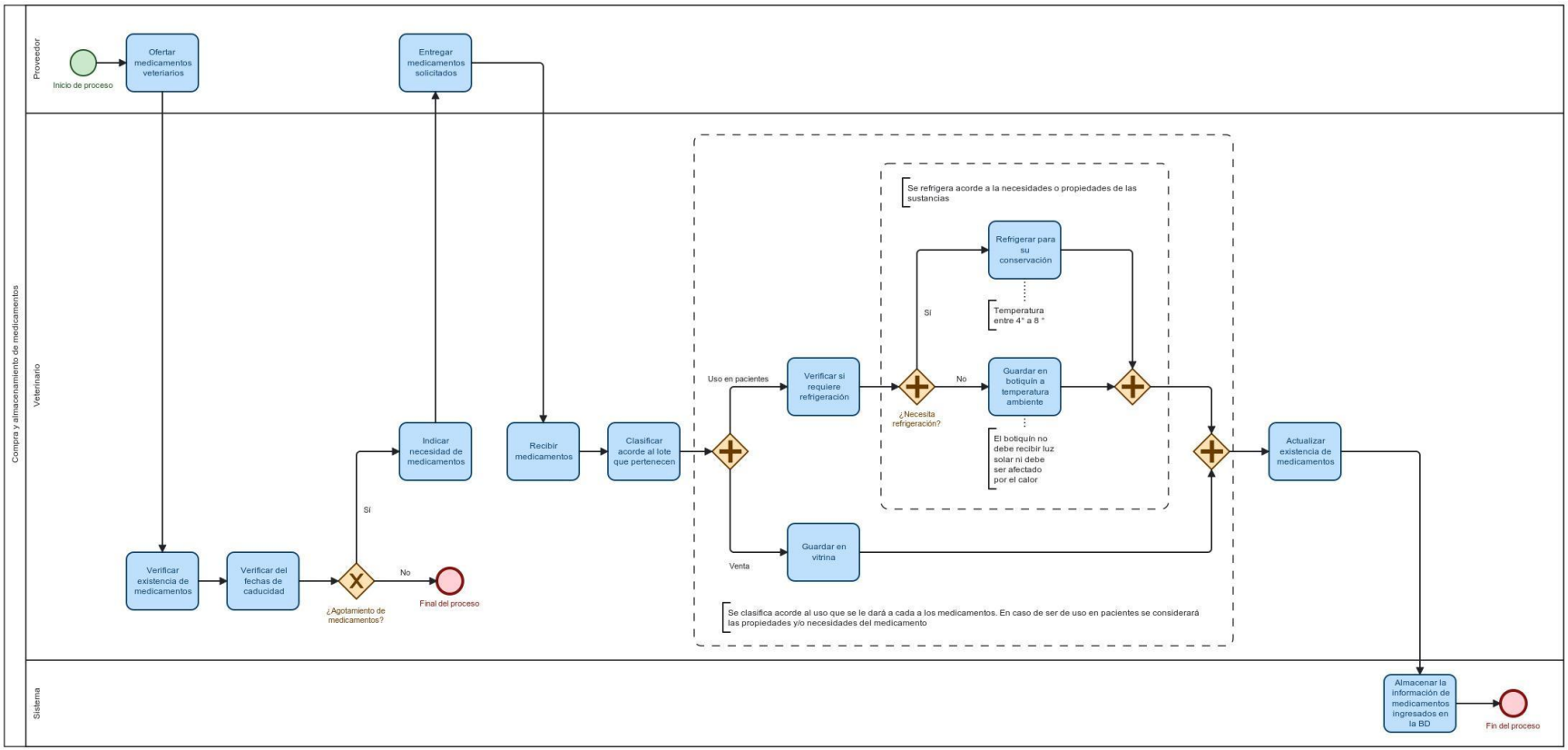


Ilustración 4-Compra y almacenamiento de productos y medicamentos

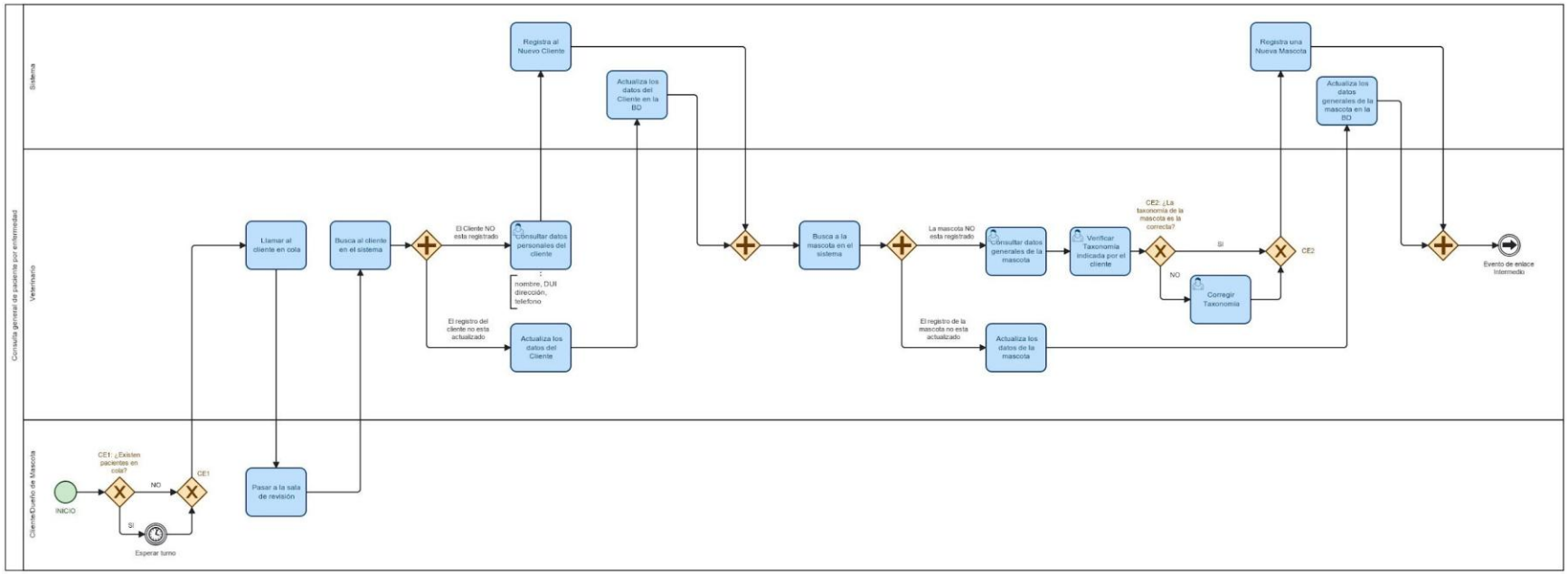


Ilustración 5-Consulta general de paciente por enfermedades

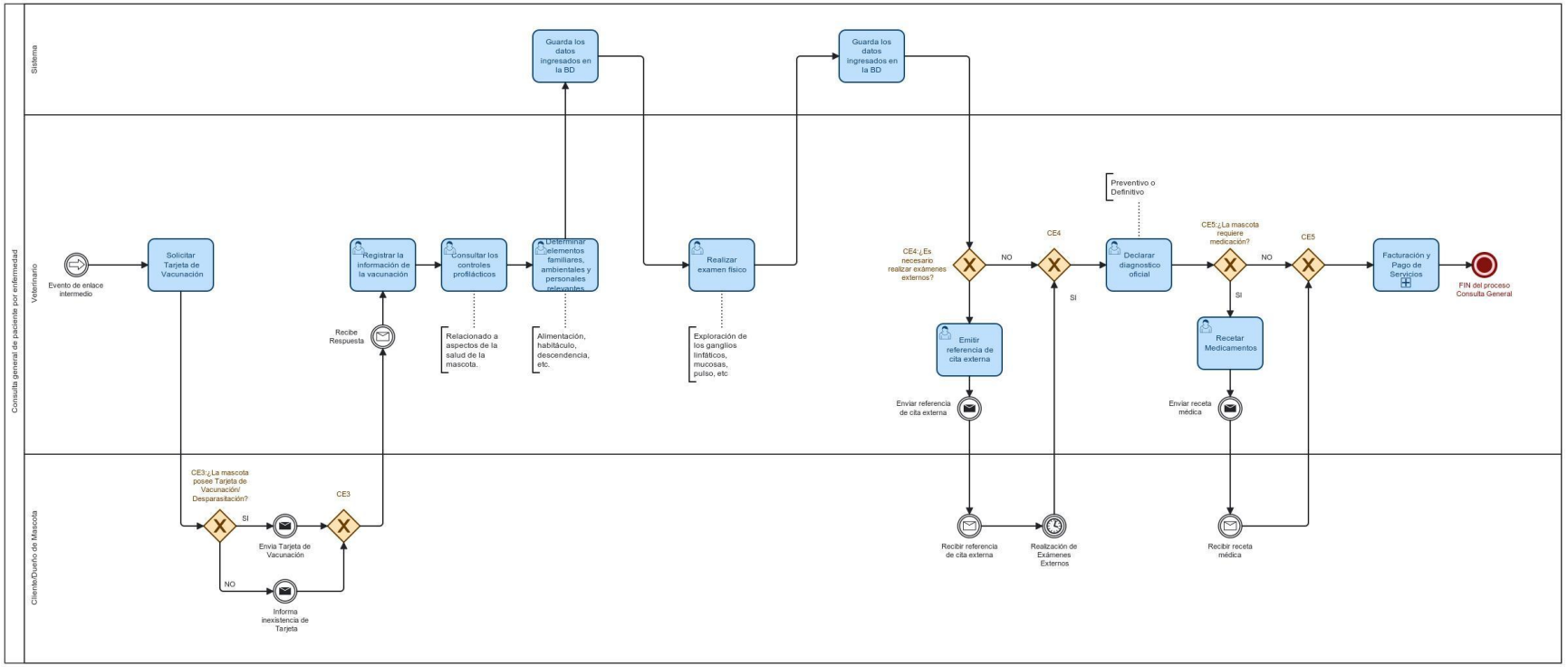


Ilustración 6-Consulta general de paciente por enfermedad

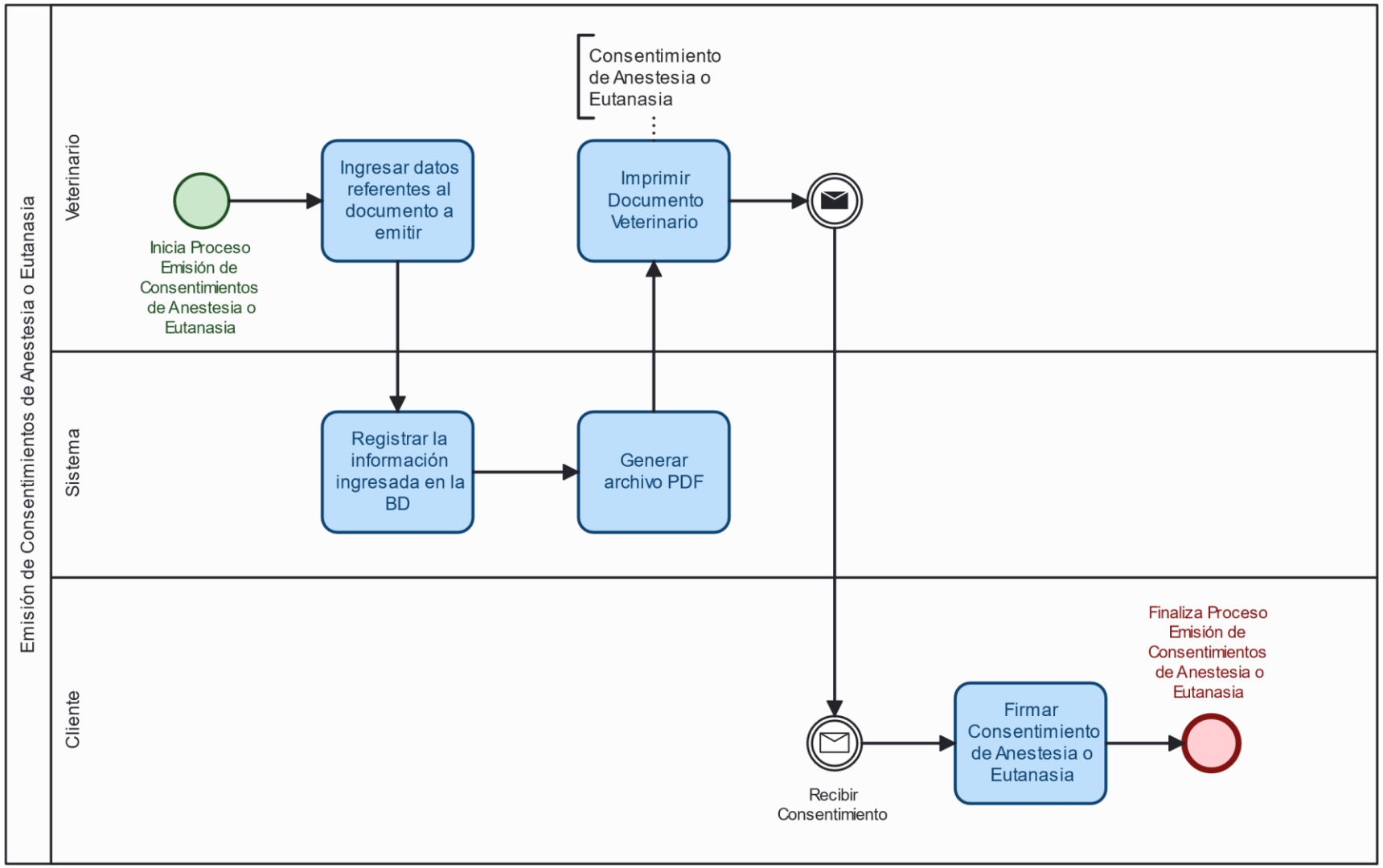


Ilustración 7-Emisión Consentimientos Eutanasia o Anestesia

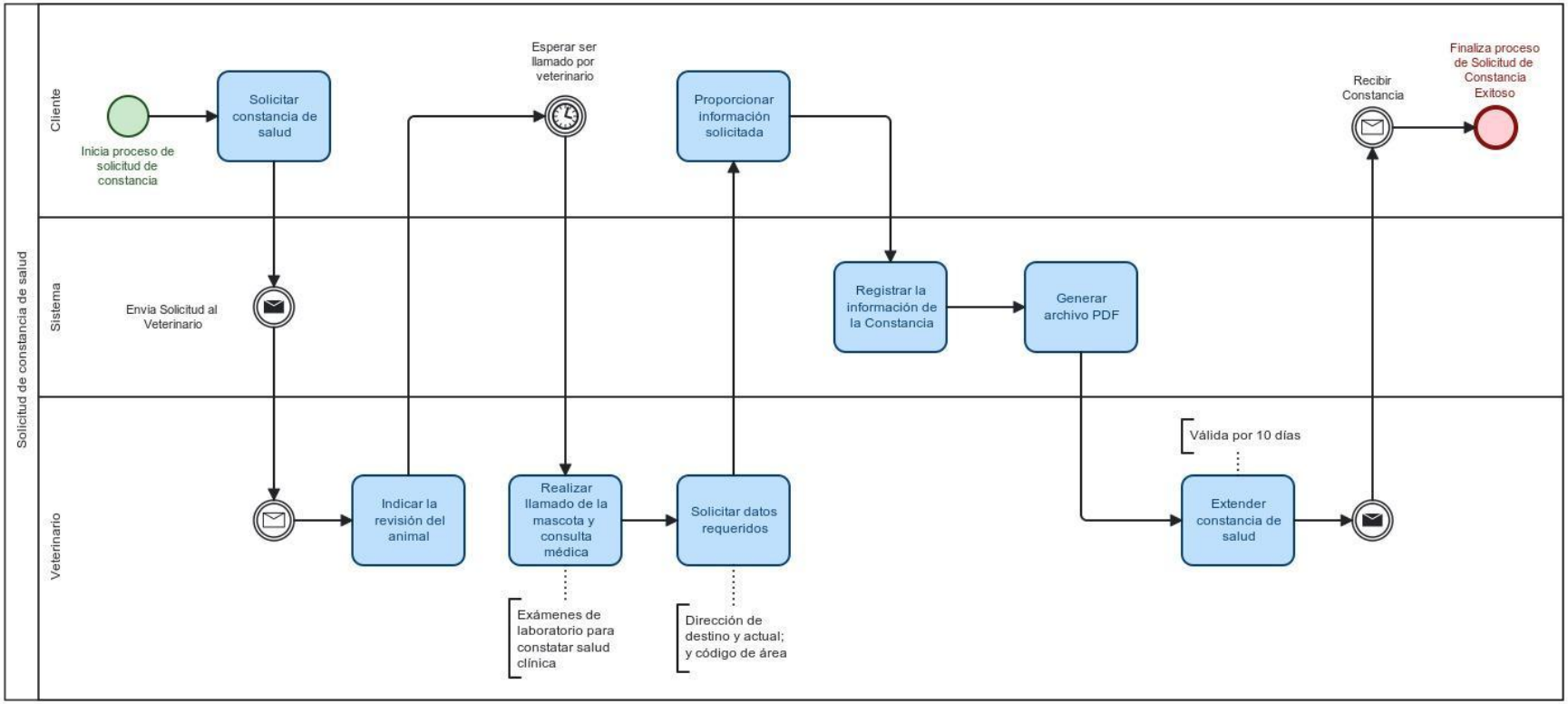


Ilustración 8-Emisión Constancia de Salud

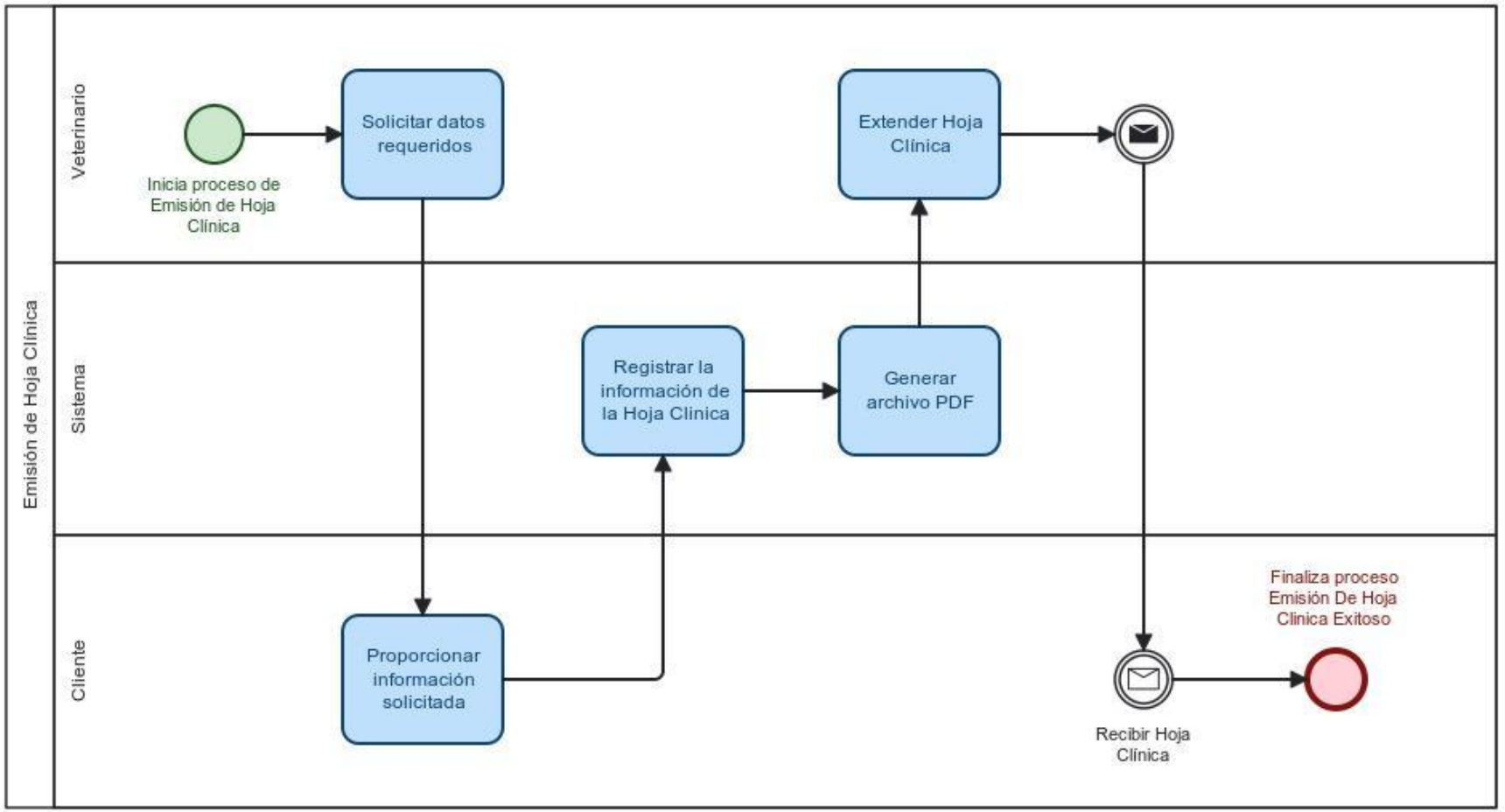


Ilustración 9-Emisión de Hoja Clínica

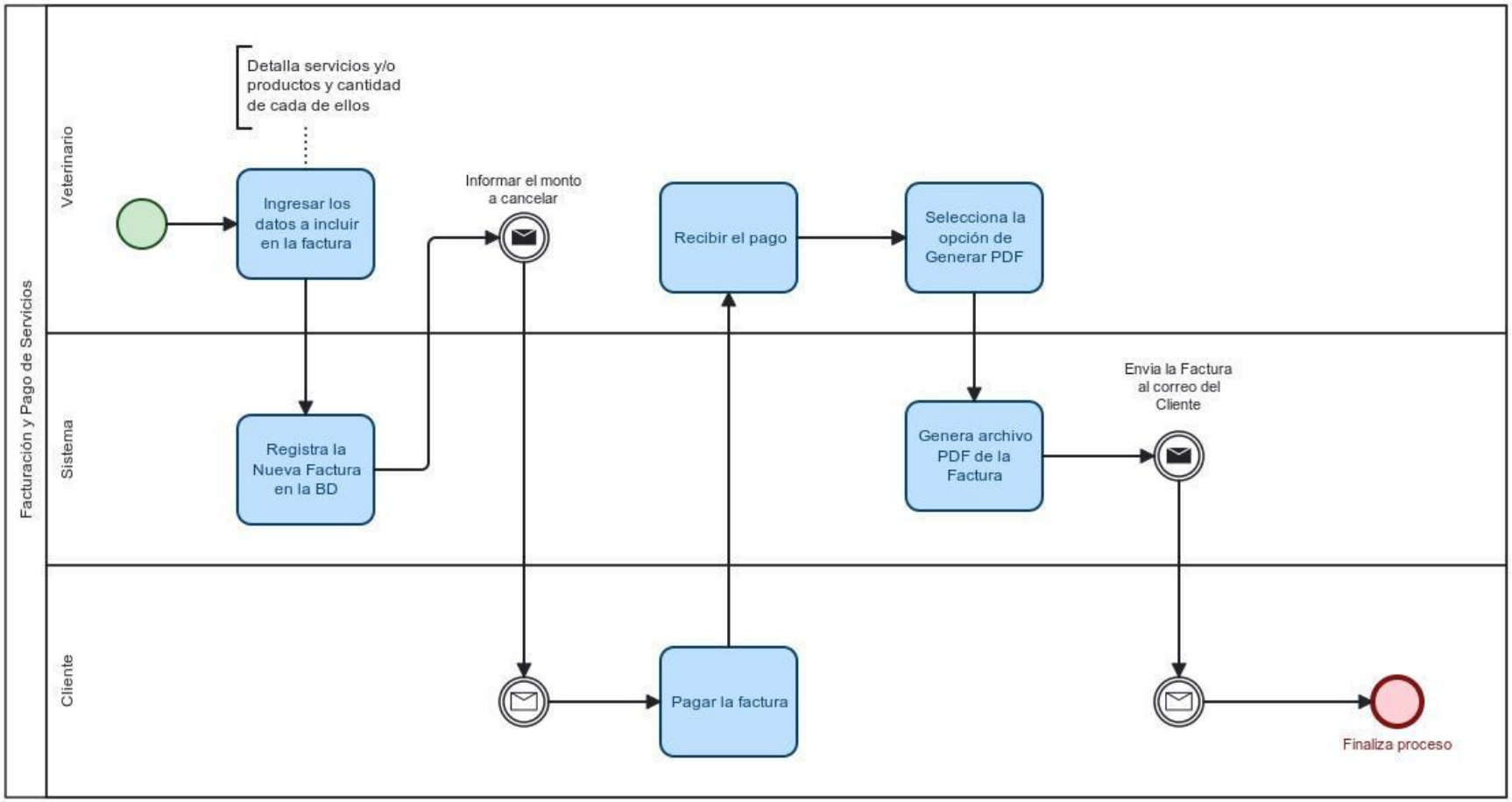


Ilustración 10-Facturación-Pago

3.3.1 Definición de conceptos clave

A continuación, se presentan los conceptos fundamentales relacionados con el funcionamiento del sistema de gestión de historiales médicos y con los procesos operativos de la Clínica Veterinaria Mitsum. Estos términos son esenciales para la comprensión del modelado de negocio y constituyen la base conceptual para el análisis posterior de requerimientos, flujos clínicos y diseño del plan de pruebas.

- **Paciente:** Mascota registrada en la clínica. Contiene información biográfica y clínica, incluyendo especie, raza, edad, características físicas y condiciones de salud relevantes.

- **Propietario:** Persona responsable del paciente. Su información se utiliza para registrar contactos, autorizar procedimientos, gestionar pagos y asociar múltiples mascotas bajo una misma identidad.

- **Historial médico:** Documento digital que consolida todas las consultas, diagnósticos, vacunaciones, tratamientos y cirugías realizadas al paciente. Es un expediente clínico acumulativo.

- **Consulta veterinaria:** Interacción clínica en la cual un veterinario examina al paciente, registra signos clínicos, emite un diagnóstico y prescribe un tratamiento o procedimiento adicional.

- **Tratamiento:** Conjunto de medidas terapéuticas prescritas por un veterinario, tales como medicamentos, cambios dietéticos o indicaciones específicas según el diagnóstico.

- **Vacunación:** Procedimiento de aplicación de vacunas según el calendario de salud del paciente. Incluye registro de lote, fecha y profesional que la aplicó.

- **Cirugía:** Procedimiento médico mayor que requiere preparación, ejecución y seguimiento postoperatorio. Su registro es crítico para la trazabilidad clínica.

- **Usuario del sistema:** Persona autorizada para utilizar el sistema. Puede tener distintos roles (administrador, veterinario, asistente veterinario o recepcionista) con permisos específicos.

- **Módulo:** Componente funcional del sistema que agrupa operaciones relacionadas (por ejemplo, módulo de pacientes, módulo de consultas, módulo de cirugías). Permite organizar las funcionalidades de manera estructurada.

- **Proceso clínico:** Secuencia de actividades que se ejecutan durante la atención médica del paciente. Incluye etapas como registro de datos, consulta, diagnóstico y emisión de resultados.

- **Flujo de negocio:** Representación ordenada de actividades dentro de un proceso. Permite visualizar cómo la información se mueve entre actores, módulos y decisiones operativas.

3.4 Necesidades del negocio

Las necesidades del negocio representan los requerimientos estratégicos que la Clínica Veterinaria Mitsum busca satisfacer mediante la validación del sistema de gestión de historiales médicos. Estas necesidades no solo están vinculadas al funcionamiento adecuado del software, sino también a la continuidad operativa, la eficiencia en los flujos de trabajo y la calidad del servicio brindado a los pacientes y propietarios. El sistema ya ha sido desarrollado y se encuentra en uso preliminar; sin embargo, la clínica requiere asegurar que las funcionalidades implementadas respondan correctamente a los procesos reales del negocio.

En este sentido, una de las principales necesidades del negocio es **garantizar que el sistema opere de manera estable, confiable y alineada a los procesos clínicos y**

administrativos, minimizando riesgos derivados de fallos funcionales y asegurando la integridad de los datos registrados.

La clínica también requiere asegurar que el sistema proporcione **una experiencia de uso adecuada**, facilitando la navegación, el acceso a historiales médicos, la gestión de consultas y el registro de información clínica. La usabilidad es un factor importante para reducir errores operativos y mejorar la eficiencia del personal.

Finalmente, el negocio requiere establecer un **proceso formal de pruebas**, tanto manuales como automatizadas, que permita evaluar las funcionalidades actuales, detectar potenciales mejoras y asegurar que el sistema esté preparado para su utilización continua dentro de la clínica. Este proceso contribuirá a fortalecer la calidad del software y a reducir riesgos antes de una adopción total del sistema en todas las áreas operativas.

3.5 Matriz de pruebas para un sistema de la veterinaria Mitsum

La matriz de pruebas constituye el artefacto central que relaciona los módulos del sistema con los casos de prueba diseñados, estableciendo trazabilidad entre requisitos funcionales, tipos de prueba y resultados esperados. Esta matriz se estructura considerando la prioridad de cada funcionalidad según su impacto en los procesos clínicos de la Veterinaria Mitsum. La matriz organiza la información en las siguientes columnas:

- **ID del Caso:** Identificador único del caso de prueba
- **Módulo:** Área funcional del sistema evaluada
- **Funcionalidad:** Operación específica bajo prueba
- **Prioridad:** Alta, Media o Baja según criticidad
- **Tipo de Prueba:** Funcional, Regresión, Seguridad, Rendimiento
- **Método:** Manual o Automatizado (Playwright/SuperTest)
- **Resultado Esperado:** Comportamiento correcto del sistema
- **Estado:** Pendiente, En Ejecución, Aprobado, Fallido, Bloqueado

3.5.1 Casos para automatizar

Los casos seleccionados para automatización forman parte de los **flujos críticos del negocio**, los **procesos repetitivos** o los **escenarios de alto riesgo** que se ejecutan en cada sprint. La automatización se desarrollará utilizando herramientas como **Playwright** (UI) y **SuperTest** (API), complementándose con el pipeline de CI/CD.

Criterios de Selección para Automatización:

Se consideraron los siguientes factores para determinar qué casos automatizar:

1. **Alta frecuencia de ejecución:** Casos que deben ejecutarse repetidamente durante el desarrollo y mantenimiento del sistema.
2. **Criticidad operativa:** Flujos esenciales para la operación diaria de la clínica cuya falla tendría impacto significativo.
3. **Estabilidad funcional:** Funcionalidades maduras con baja probabilidad de cambios estructurales frecuentes.
4. **Repetibilidad:** Escenarios con pasos y datos predecibles que producen resultados determinísticos.
5. **Idoneidad técnica:** Flujos que pueden automatizarse eficientemente con las herramientas seleccionadas (Playwright, SuperTest).
6. **Beneficio costo-tiempo:** Casos donde el esfuerzo de automatización se justifica por el ahorro en ejecuciones futuras.

Los casos de prueba que serán automatizados son:

Tabla 11-Casos de prueba para automatizar

ID	Módulo	Funcionalidad
TC-01	Gestión de Pacientes	Crear paciente
TC-02	Gestión de Pacientes	Editar paciente
TC-03	Gestión de Propietarios	Crear propietario
TC-06	Citas y Cirugías	Programar cita
TC-07	Citas y Cirugías	Modificar cita
TC-09	Facturación	Emisión de factura
TC-10	Seguridad y Accesos	Validación de acceso por roles
TC-11	Seguridad y Accesos	Recuperación de contraseña
TC-12	Seguridad y Accesos	Cambio de contraseña con token
TC-13	Seguridad y Accesos	Login con Google OAuth
TC-14	Seguridad y Accesos	Credenciales inválidas
TC-15	Seguridad y Accesos	Cerrar sesión correctamente
TC-16	Citas y Cirugías	Integración con Google Calendar
TC-17	Citas y Cirugías	Validación de conflicto de horarios
TC-18	Citas y Cirugías	Cancelar cita

Justificación de selección

Los anteriores casos se automatizan debido a que:

- Son **flujos críticos del negocio** y deben funcionar en cada entrega.
- Tienen **alta frecuencia de uso** (CRUD, citas, login, roles).
- Poseen **riesgo medio o alto** ante cambios del sistema.
- Son altamente **repetitivos**, lo que reduce costo de ejecución manual.

3.5.2 Casos para no automatizar

Estos casos se ejecutarán únicamente de forma manual debido a su **complejidad**, **dependencias externas**, **validaciones visuales**, o porque representan escenarios de poca frecuencia en el negocio.

Tabla 12-Casos de prueba para no automatizar

ID	Módulo	Funcionalidad	Justificación
TC-04	Historial Clínico	Registrar historial	Flujo similar al de pacientes; bajo riesgo.
TC-05	Historial Clínico	Actualizar historial	Variabilidad alta de escenarios posibles
TC-08	Medicamentos	Asignación de medicamentos	Requiere validaciones clínicas manuales.
TC-19	Historial Clínico	Añadir archivo adjunto	Implica subir archivos, formatos diversos.
TC-20	Historial Clínico	Consultar historial completo	Requiere validación humana para evaluar la presentación de información
TC-21	Gestión de Pacientes	Eliminar paciente con historial	Riesgo de pérdida de datos; preferible manual.
TC-22	Gestión de Pacientes	Búsqueda por múltiples criterios	Variabilidad alta de combinaciones

Razones generales para no automatizar

- Escenarios que requieren **validación subjetiva** (adjuntos o revisiones visuales).
- Flujos que dependen de **correo electrónico o enlaces temporales**.
- Casos con **baja frecuencia**, donde el costo de automatización supera el beneficio.
- Operaciones con **alto riesgo de pérdida de datos** (eliminación de pacientes con historial).
- Escenarios que solo necesitan ser probados en la **primera ejecución o por auditoría**.

3.6 ID y Objetivos de las pruebas.

3.6.1 Identificación (ID) de los casos de prueba

Los casos de prueba del sistema se identifican mediante el patrón: TC-XX

Donde:

- **TC** = Test Case
- **XX** = Número consecutivo asignado según el orden del módulo (ej.: TC-01, TC-02, … TC-22)

El código permite:

- Trazabilidad con los requisitos funcionales.
- Organización consistente por módulo.
- Control de versiones y cobertura de pruebas.

3.6.2 Objetivos de las pruebas

Los objetivos del plan de pruebas son:

- Validar que cada módulo del sistema funcione conforme a los requerimientos establecidos.
- Asegurar la correcta integridad y consistencia de los datos almacenados en la base de datos.
- Verificar que los flujos críticos del negocio veterinario se ejecuten sin errores bajo diferentes escenarios.
- Confirmar que los controles de acceso por roles funcionen adecuadamente y garanticen la seguridad del sistema.
- Validar la experiencia del usuario mediante pruebas manuales y exploratorias.
- Proveer evidencia documentada de los resultados de las pruebas para auditoría y validación académica.

3.7 Alcance de Pruebas

El alcance de las pruebas define las funcionalidades del sistema que serán evaluadas durante el proceso de verificación y validación. Para este proyecto, el alcance se encuentra delimitado por los módulos centrales que forman parte del sistema de gestión de historiales médicos de la Clínica Veterinaria Mitsum, de acuerdo con lo establecido en la Sección 2.10 de la tesina y la Sección 5 del anteproyecto.

Dentro del alcance se incluyen las funcionalidades relacionadas con los procesos principales que utiliza el personal de la clínica en su operación diaria. Entre los módulos y componentes que serán evaluados se encuentran:

- Autenticación y acceso al sistema
- Gestión de pacientes (registro, consulta y actualización)
- Historial clínico (consolidación y consulta de información clínica)
- Consultas veterinarias
- Tratamientos, vacunas y cirugías
- Gestión de citas
- Facturación y registro de pagos

Estos módulos serán sometidos a pruebas manuales y en los casos correspondientes, pruebas automatizadas con el fin de verificar su correcto funcionamiento, integridad de datos y comportamiento dentro de los flujos de negocio descritos previamente. Esta validación también incluye la revisión de la navegación básica del sistema y su capacidad para ejecutar operaciones clínicas fundamentales sin interrupciones.

Fuera del alcance se encuentran pruebas avanzadas que no forman parte de los objetivos de este proyecto, tales como pruebas de carga, pruebas de estrés, pruebas de penetración, análisis de rendimiento a gran escala o validaciones relacionadas con infraestructura. Asimismo, no se evaluarán módulos no desarrollados o componentes externos que no formen parte de la versión actual del sistema.

El diagrama presentado a continuación resume la descomposición funcional definida en el anteproyecto, permitiendo visualizar de forma clara los principales módulos del sistema

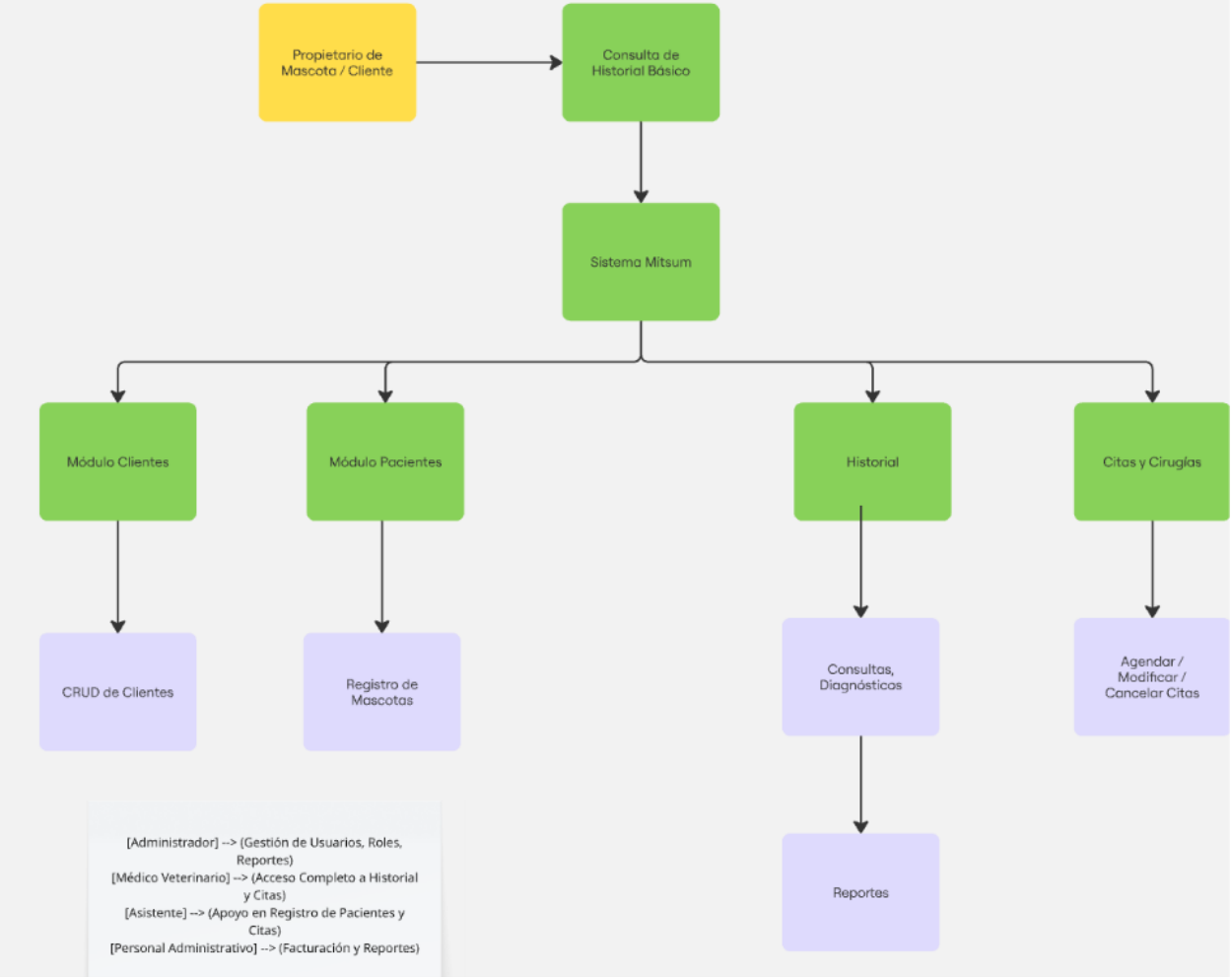


Ilustración 11-Diagramas de Módulos del sistema

y sus componentes asociados. Esta representación sirve de base para la planificación del esfuerzo de pruebas, la priorización de funcionalidades.

Finalmente, como referencia visual del alcance, se muestra la Figura 8: Componentes del Sistema, presentada en el anteproyecto, la cual resume la estructura funcional del sistema y los elementos involucrados en este proceso de validación.

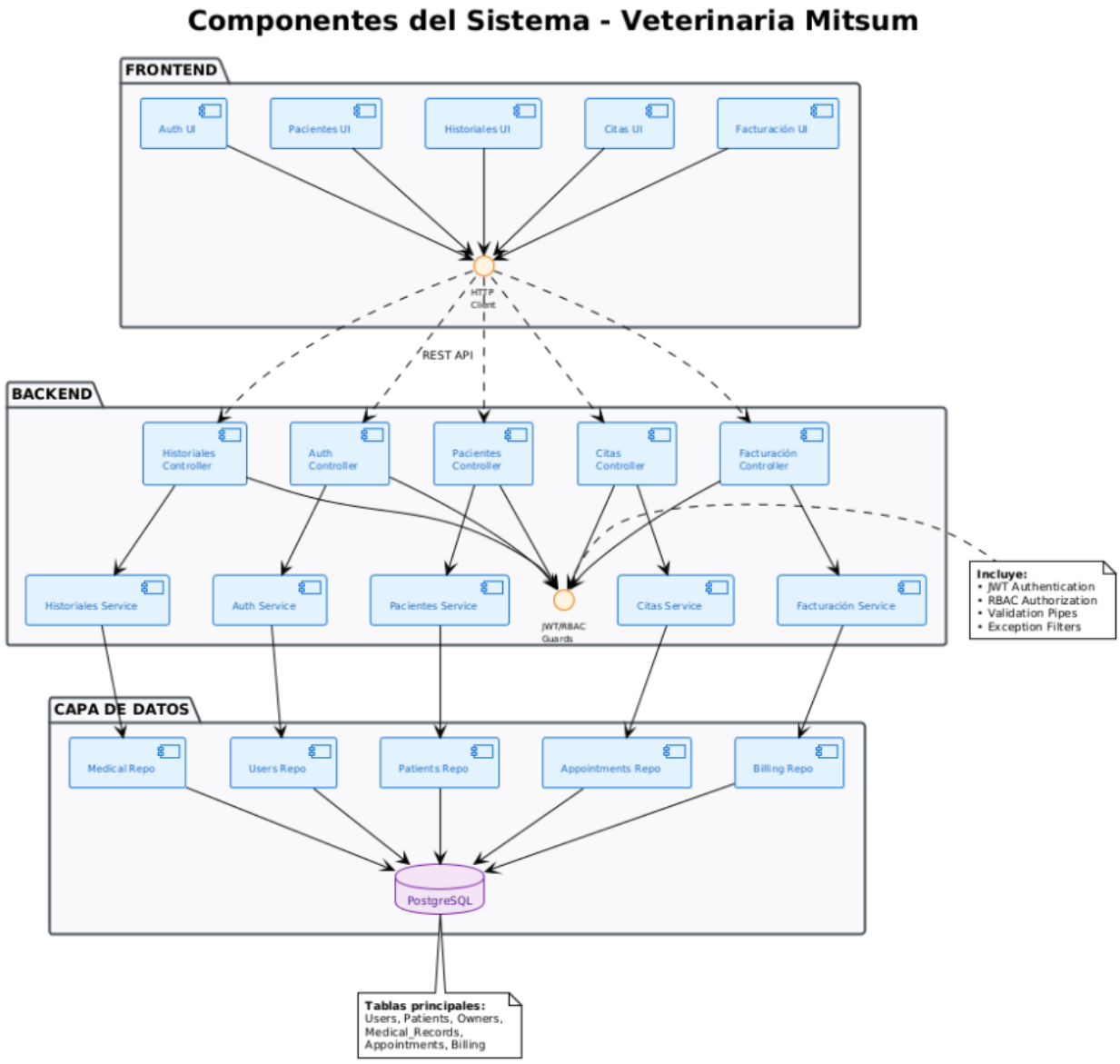


Ilustración 12-Diagramas de componentes de Módulos del sistema

3.8 Estrategia de pruebas

La estrategia de pruebas define el enfoque seleccionado para validar el sistema de historiales médicos de la Clínica Veterinaria Mitsum. Este enfoque se fundamenta en las buenas prácticas de ISTQB y en la estructura documental establecida por el estándar IEEE-829, asegurando que las actividades de prueba se ejecuten de manera ordenada, trazable y orientada al riesgo.

Enfoque general

Se utilizará un enfoque de caja negra, centrado en la validación del comportamiento observable del sistema sin depender de su lógica interna. Este enfoque es adecuado dado que el objetivo del proyecto es validar los flujos clínicos, administrativos y operativos desde la perspectiva del usuario final y del negocio.

Además, se empleará una combinación de pruebas manuales, pruebas automatizadas de interfaz (E2E) y pruebas automatizadas de API. Esta combinación permite cubrir los módulos principales del sistema con eficiencia, garantizando la detección oportuna de defectos en funcionalidades críticas.

Niveles de Prueba Incluidos

Aunque ISTQB define cuatro niveles (componentes, integración, sistema y aceptación), para este proyecto se seleccionan únicamente los niveles que resultan pertinentes según el alcance y los artefactos disponibles:

a) Pruebas de Sistema (nivel principal)

Se validará la aplicación como un todo integrado, asegurando que los módulos —autenticación, gestión de pacientes, propietarios, historiales médicos, citas, medicamentos e inventario— funcionen correctamente desde la perspectiva del usuario final.

Este nivel permite evaluar:

- Flujos completos de negocio.
- Integración entre módulos.
- Validaciones funcionales.
- Registros, consultas y edición de información.

b) Pruebas de Aceptación (contexto académico)

Se evaluará si el sistema es apto para su operación desde la perspectiva funcional, de usabilidad y de cumplimiento de flujos clínicos esenciales. Este nivel no es contractual,

sino formativo, y tiene como objetivo asegurar que el sistema cumple con los requerimientos mínimos de uso dentro del entorno académico simulado.

Justificación de por qué NO se incluyen los otros niveles

- **Pruebas de Componentes:** Aunque existe acceso al código, no se cuenta con una arquitectura ni una base de pruebas unitarias que permitan establecer un nivel formal de pruebas de componentes.
- **Pruebas de Integración técnica:** la estructura del sistema permite validar la integración funcional directamente a través de las pruebas de sistema, sin necesidad de un nivel independiente.

Esto mantiene la estrategia simple, alineada al alcance del proyecto y consistente con el ciclo de vida del sistema.

Tipos de Prueba

a) Pruebas Funcionales Manuales

Validarán los flujos esenciales del sistema mediante casos de prueba bien definidos y alineados a los procesos clínicos reales:

- CRUD de pacientes y propietarios.
- Registro y consulta de historiales médicos.
- Validaciones de formularios.
- Manejo de errores.
- Control de acceso basado en roles.
- Navegación y coherencia de interfaz.

b) Pruebas Automatizadas E2E (Interfaz)

Se automatizarán flujos críticos de negocio con el fin de:

- Asegurar estabilidad de las funciones esenciales.
- Detectar regresiones en futuras ejecuciones.

- Repetir pruebas de forma rápida y consistente.

Ejemplos de flujos a automatizar incluyen:

- Inicio de sesión.
- Registro de nuevo paciente.
- Creación de consulta y diagnóstico.
- Consulta de historial clínico.

c) Pruebas automatizadas de API

Se utilizarán pruebas API para validar directamente los servicios del backend, verificando:

- Correcta operación de endpoints CRUD.
- Códigos de respuesta HTTP.
- Estructura JSON.
- Tiempos de respuesta básicos.

Este tipo de prueba complementa la visión de caja negra y mejora la cobertura técnica.

d) Pruebas de Compatibilidad

Validación funcional en los navegadores utilizados por la clínica:

- Google Chrome.
- Mozilla Firefox.

e) Pruebas básicas de rendimiento (API)

Se medirá únicamente:

- Tiempo promedio de respuesta.
- Tiempo máximo por endpoint.

No se incluyen pruebas de carga ni estrés debido al alcance del proyecto.

Técnicas de Diseño de Pruebas

Las pruebas se diseñarán utilizando técnicas basadas en casos de uso, de acuerdo con los flujos funcionales definidos para cada módulo del sistema. Este enfoque permite validar los escenarios reales que ejecutan los usuarios de la clínica, asegurando que las funciones más críticas como autenticación, gestión de pacientes e historiales médicos se comportan correctamente durante la operación.

No se emplearán técnicas avanzadas como partición de equivalencia, valores límite o tablas de decisión, ya que el diseño de casos se centra en reproducir flujos funcionales completos y escenarios representativos del negocio clínico.

Enfoque Basado en Riesgo

La priorización de pruebas se realizará según el impacto que cada módulo tiene en las operaciones de la clínica:

Alta prioridad (críticos)

- Autenticación.
- Pacientes.
- Historial clínico.
- Consultas veterinarias.

Mediana prioridad

- Citas.
- Tratamientos.
- Vacunas.
- Cirugías.

Baja prioridad

- Módulos secundarios como inventario o facturación (validación funcional básica).

Esta priorización influirá en la selección de casos a automatizar y en la asignación de tiempo para la ejecución manual.

Integración Continua y Ejecución Automatizada

Se establecerá un pipeline básico de CI/CD utilizando GitHub Actions, con los siguientes objetivos:

- Ejecutar automáticamente las pruebas de API con cada commit relevante.
- Permitir ejecuciones manuales o programadas de las pruebas E2E.
- Generar reportes automáticos del estado de la suite.

Esto garantiza repetibilidad, trazabilidad y retroalimentación continua durante el proceso de pruebas.

3.9 Niveles de pruebas y fases

Los niveles de prueba y sus fases establecen la estructura general del proceso de verificación del sistema, definiendo cuándo y cómo se evaluará cada componente. Esto permite organizar el trabajo, mantener trazabilidad con la estrategia y asegurar la consistencia en la ejecución.

Para el sistema de historiales médicos de la Clínica Veterinaria Mitsum se han definido los niveles y fases descritos a continuación, en alineación con lo establecido en la Estrategia de Pruebas (Sección 3.8).

3.9.1 Proceso / logística de las pruebas de sistema

Aunque inicialmente esta sección estaba orientada a “pruebas de integración”, en el contexto del proyecto y la estrategia definida, las validaciones de integración se realizaron dentro del nivel de pruebas de sistema, por lo que el alcance se ajusta en esta versión.

El proceso se organizará de la siguiente manera:

1. Identificación de flujos y módulos a validar

Se seleccionaron los elementos más relevantes del sistema, incluyendo:

- Autenticación y control de acceso.
- Gestión de pacientes y propietarios.
- Historias clínicas.
- Consultas veterinarias.
- Citas y cirugías.
- Inventario y medicamentos.
- API del sistema (endpoints CRUD).

La identificación se basó en los módulos definidos en el anteproyecto y en el análisis funcional realizado durante el proyecto académico.

2. Preparación del entorno de pruebas

Se configuró un entorno dedicado con:

- Backend y frontend en la versión estable proporcionada.
- Base de datos con información de prueba.
- Accesos y credenciales configuradas para roles permitidos.

Las herramientas empleadas para la ejecución fueron:

- Playwright: automatización de flujos E2E.
- SuperTest + Jest: validación de servicios API.
- Ejecución manual con navegadores Chrome y Firefox.

3. Diseño de escenarios de validación

Los escenarios se basaron en casos reales del funcionamiento del sistema, tales como:

- Registrar un nuevo paciente y validar su información.
- Crear una consulta e ingresar diagnóstico y tratamiento.
- Verificar que los datos se reflejen en el historial clínico.

- Programar y visualizar una cita.
- Validar las respuestas de los endpoints CRUD de la API.

Estos escenarios se derivaron directamente de los casos de uso del sistema.

4. Ejecución y registro de resultados

Las pruebas de sistema combinaran:

- **Ejecución manual:** Validación de formularios, navegación, mensajes de error, flujo visual e interacción entre módulos.
- **Ejecución automatizada** (Playwright y SuperTest): Validación de flujos críticos e integridad de los endpoints API.

Durante la ejecución se verificará:

- Integridad de datos entre módulos.
- Comportamiento funcional esperado.
- Respuestas correctas de la API.
- Coherencia entre acciones del usuario y resultados mostrados en pantalla.

5. Seguimiento de defectos

Cada hallazgo detectado será registrado, detallando:

- Comportamiento observado.
- Módulo afectado.
- Severidad.
- Pasos para reproducirlo.

Se priorizarán defectos que afecten flujos clínicos esenciales.

Este proceso garantiza que el sistema funcione de manera estable y que los módulos operen de forma coordinada, cumpliendo los flujos clínicos, administrativos y operativos definidos.

3.9.2 Proceso / logística de las pruebas de aceptación de usuario.

Las pruebas de aceptación en este proyecto no corresponden a una UAT formal con personal de la clínica, sino a una validación académica orientada a verificar que el sistema cumple con los requisitos funcionales esenciales y reproduce adecuadamente los flujos definidos en el anteproyecto.

Este nivel confirma que el sistema es apto para su propósito dentro del alcance del proyecto académico.

El proceso se ejecutará de la siguiente manera:

1. Selección de flujos representativos

Se eligieron los flujos que mejor representan el uso real del sistema, tales como:

- Registro de pacientes y propietarios.
- Creación y actualización de consultas clínicas.
- Registro de diagnósticos, vacunas y tratamientos.
- Revisión del historial clínico completo.
- Programación de citas.
- Validación básica de inventario.

2. Preparación de escenarios de aceptación

Los escenarios serán diseñados para evaluar:

- Correcta navegación entre módulos.
- Operación fluida de las funciones esenciales.
- Coherencia de datos mostrados.
- Claridad de mensajes y retroalimentación visual.

Estos escenarios se derivarán de los BPMN, casos de uso y observación funcional del sistema.

3. Ejecución por el equipo académico

A diferencia de un proceso UAT real, la ejecución se realizó por el equipo de pruebas del proyecto académico, evaluando:

- Comportamiento general del sistema.
- Cumplimiento funcional.
- Facilidad de uso y comprensión de la interfaz.
- Correcta aplicación de reglas de negocio básicas.

4. Registro de hallazgos

Se documentarán únicamente defectos funcionales encontrados en los flujos de aceptación.

5. Evaluación final

La fase concluye evaluando si el sistema:

- Cumple con los flujos clínicos esenciales.
- Permite registrar, consultar y actualizar datos de manera coherente.
- Presenta estabilidad durante la ejecución de las funciones principales.

El resultado determina si el sistema es adecuado para su uso dentro del contexto académico planteado.

3.10 Áreas de enfoque de prueba

Las áreas de enfoque establecen los componentes y atributos del sistema que serán evaluados con mayor profundidad durante el proceso de pruebas. Esto permite dirigir los esfuerzos hacia los módulos y características más relevantes para la operación de la Clínica Veterinaria Mitsum, así como hacia los aspectos que presentan mayor riesgo para la continuidad de los procesos clínicos. El eje principal de las pruebas se centra en pruebas funcionales, complementadas por verificaciones transversales de usabilidad, seguridad funcional, compatibilidad y rendimiento básico de API.

3.10.1 Pruebas Funcionales

Las pruebas funcionales constituirán la principal área de enfoque debido a la naturaleza operativa del sistema y a la importancia de validar los flujos clínicos completos. Este tipo de pruebas permitirá confirmar que cada módulo funciona según lo esperado y que los procesos se ejecutan de manera coherente con las necesidades de la veterinaria.

Las áreas funcionales que se abordarán incluyen:

a) Autenticación y Control de Acceso

Este módulo será evaluado debido a su impacto directo en la seguridad y operación del sistema.

Las pruebas se enfocarán en verificar:

- Inicio de sesión con credenciales válidas e inválidas.
- Manejo de errores durante la autenticación.
- Acceso permitido o denegado según el rol del usuario.
- Correcto cierre y expiración de sesión.

b) Gestión de Pacientes y Propietarios

Este módulo será uno de los puntos de mayor atención debido a su rol como base para los flujos clínicos.

- Las pruebas considerarán:
- Operaciones CRUD para pacientes y propietarios.
- Validación de campos obligatorios.
- Reglas de asociación paciente–propietario.
- Manejo adecuado de errores de entrada.

c) Historias Clínicas y Consultas Médicas

Se evaluará con profundidad por su impacto directo en la atención veterinaria y la continuidad de la información clínica.

Las pruebas verificarán:

- Registro de diagnósticos, tratamientos y observaciones.
- Consulta del historial clínico completo.
- Integridad y consistencia de la información médica.
- Restricciones de edición según rol (cuando aplique).

d) Citas y Cirugías

Este módulo será validado funcionalmente para garantizar que los procesos administrativos operen sin interrupciones.

Las pruebas incluirán:

- Programación, modificación y cancelación de citas.
- Validación de conflictos de horario.
- Restricción de campos obligatorios.
- Funcionamiento básico del calendario.

e) Medicamentos e Inventario

Las pruebas de este módulo se centrarán en validar su comportamiento funcional esencial.

Se evaluará:

- Consulta de medicamentos e insumos.
- Consistencia básica de datos.
- Flujo general de navegación.

f) Facturación y Reportes

Este módulo será evaluado desde un enfoque funcional básico, acorde a su prioridad.

Las pruebas considerarán:

- Visualización y consistencia mínima de facturas.
- Acceso correcto a reportes clínicos.
- Navegación asociada al módulo.

3.11 Criterios de entrada/salida

Criterios de Entrada

Los criterios de entrada establecen las condiciones mínimas que deben cumplirse antes de iniciar formalmente la ejecución del plan de pruebas. Su objetivo es asegurar que el entorno, los datos simulados y las herramientas necesarias estén listos para garantizar un proceso de verificación ordenado, confiable y repetible.

De acuerdo con IEEE 829 e ISTQB, las pruebas del sistema de historiales médicos de la Veterinaria Mitsum iniciarán únicamente cuando se cumplan los siguientes elementos:

1. Entorno de Pruebas Disponible

- Disponibilidad del entorno de pruebas o staging destinado exclusivamente para la ejecución del proyecto.
- Base de datos del entorno de pruebas inicializada sin información real, cumpliendo los lineamientos de protección de datos.
- El sistema debe contar con datos ficticios previamente registrados (pacientes, propietarios, historiales, usuarios), creados manualmente para asegurar condiciones de prueba estables y proteger la información real de la clínica.

2. Accesos y Credenciales

Como el sistema únicamente maneja dos roles, deben estar disponibles:

- Credenciales funcionales del Administrador.
- Credenciales funcionales del Cliente/Usuario.
- Accesos válidos al API del sistema para pruebas con SuperTest.

3. Casos de Prueba Definidos

- Casos de prueba manuales documentados, revisados y aprobados.
- Prioridad definida para los casos críticos que serán automatizados.

4. Datos de Prueba Preparados (Simulados)

Conjuntos de datos creados exclusivamente para las pruebas:

- Pacientes ficticios.
- Propietarios ficticios.
- Diagnósticos, tratamientos y citas simuladas.

5. Requisitos Claros y Estables

- Revisión del alcance definido en la sección 2.6.
- Identificación de los requisitos funcionales presentes en la aplicación y su documentación disponible.
- Validación de escenarios críticos del negocio: autenticación, pacientes e historiales.

6. Herramientas Configuradas

- Playwright instalado y funcionando para pruebas E2E.
- SuperTest configurado para pruebas de API.
- Repositorio GitHub creado y accesible.
- Pipeline inicial de GitHub Actions configurado para ejecutar pruebas básicas.

7. Criterios de Aceptación Definidos

- Límites para tiempos de respuesta básicos de endpoints críticos.
- Resultados esperados para flujos funcionales clave.
- Comportamientos esperados para escenarios válidos e inválidos, definidos para fines académicos y consistentes con ISO/IEC 25010.

8. Disponibilidad del Equipo

- Confirmación del calendario de ejecución.
- Asignación de tareas para pruebas manuales, automatización y documentación.

Criterios de Salida

Los criterios de salida establecen las condiciones que deben cumplirse para considerar finalizada la ejecución del plan de pruebas. Su propósito es garantizar que el sistema ha sido evaluado de manera suficiente, que la evidencia generada es confiable y que los resultados permiten emitir conclusiones sobre la calidad del software.

De acuerdo con IEEE 829 e ISTQB, las pruebas del sistema de historiales médicos de la Veterinaria Mitsum concluirán cuando se cumplan las siguientes condiciones:

1. Ejecución Completa de Casos de Prueba

- El 90% de los casos de prueba manuales planificados han sido ejecutados.
- Los flujos críticos evaluados mediante automatización (login, gestión de pacientes, registro de diagnóstico y consulta de historial) han sido ejecutados exitosamente en Playwright y SuperTest.
- No quedan casos pendientes sin justificación documentada.

2. Registro y Clasificación de Incidencias

- Todas las incidencias detectadas han sido registradas en el documento final.
- Las incidencias han sido clasificadas según severidad y prioridad.
- No deben existir defectos de severidad Alta pendientes de registro o revisión inicial.
- Los defectos de severidad Media o Baja han sido documentados con sus escenarios y evidencia.

Nota: La corrección de defectos no forma parte del alcance del proyecto; únicamente su identificación y documentación.

3. Cumplimiento de los Criterios de Aceptación

Los flujos funcionales críticos definidos en el alcance cumplen con los criterios establecidos:

- Autenticación y acceso según roles.
- CRUD de pacientes y propietarios.
- Registro y consulta de historiales clínicos.
- Flujos principales de citas.

Asimismo:

Los tiempos de respuesta básicos medidos con SuperTest se encuentran dentro de los límites establecidos en este plan.

4. Evidencia Completa y Documentada

- Reportes automáticos generados por Playwright y SuperTest disponibles.
- Capturas, logs y resultados consolidados en el reporte final de pruebas.

5. Actividades de Documentación Final Completa

- Informe final del proceso de pruebas completado.
- Resultados consolidados para su inclusión en el Capítulo III de este documento.

6. Validación del Equipo del Proyecto

- Validación interna de que los objetivos del plan de pruebas han sido cumplidos.
- Confirmación de que no quedan actividades críticas pendientes.
- Autorización para cerrar formalmente la fase de pruebas dentro del cronograma académico.

3.12 Definición de defectos y tiempos de respuesta

Esta sección establece los criterios que permitirán identificar, clasificar y gestionar los defectos detectados durante la ejecución del plan de pruebas. Además, define los tiempos de respuesta esperados para las interacciones más importantes del sistema, con el objetivo de medir el desempeño básico de la API conforme a los lineamientos del anteproyecto.

Un defecto se define como cualquier desviación del comportamiento esperado del sistema, ya sea en términos funcionales, de usabilidad, navegación o estructura de los datos devueltos por la API. Los defectos pueden originarse por:

- Funcionamiento incorrecto o incompleto de una funcionalidad.
- Errores en los datos mostrados o procesados.
- Ausencia de validaciones o controles requeridos.
- Comportamientos inconsistentes entre módulos integrados.
- Respuestas HTTP incorrectas o estructuras JSON inválidas.
- Problemas de navegación, renderizado o visualización en el frontend.

Durante el proceso de pruebas, cada defecto será documentado incluyendo:

- Descripción del problema
- Pasos para reproducirlo
- Resultado esperado
- Resultado obtenido
- Evidencia (capturas, logs, reportes)
- Severidad asignada
- Prioridad de atención
- Módulo afectado

3.12.1 Definición de prioridades en los efectos

Para efectos del proyecto académico, los defectos se clasificarán utilizando el siguiente criterio:

Alta (Crítica)

- Afecta un flujo funcional principal.
- Impide continuar con un proceso esencial (ej. registrar un paciente).
- No existe solución alternativa.
- Afecta integridad de datos o acceso.

Media

- Afecta funcionalidades secundarias.
- Permite continuar mediante alternativa parcial.
- Impacto moderado en la operación.

Baja

- Problemas menores de interfaz, alineación, textos, mensajes o navegación no crítica.
- No afecta la operación principal.

3.12.2 Tiempos de respuesta

Los tiempos de respuesta se evaluarán de manera preliminar mediante **SuperTest**, siguiendo lo establecido en el anteproyecto. No se llevarán a cabo pruebas de carga o

estrés; únicamente se realizarán mediciones básicas para garantizar que los **endpoints críticos** respondan dentro de rangos aceptables y cumplan con los niveles mínimos de rendimiento esperados.

Endpoints a medir

- Autenticación (Login)
- Registro de pacientes
- Consulta de pacientes
- Registro de consultas/historial
- Consulta del historial clínico completo

Métricas evaluadas

- Tiempo promedio de respuesta (ms)

  Medición del promedio de 5–10 ejecuciones consecutivas.

- Tiempo máximo registrado (ms)

  Peor tiempo obtenido en la serie de pruebas.

Consolidación de problemas a tomar en cuenta:

- **Proceso/metodología:** ausencia de un plan de pruebas documentado y detección reactiva de incidencias.
- **Integraciones externas:** inestabilidad en **Google OAuth/Calendar** (intermitencias y errores de token/consent).
- **Rendimiento:** endpoints con P95 por encima del objetivo en consultas pesadas.
- **Cobertura:** módulos **Citas** y **Seguridad** por debajo del umbral de automatización (≥ 70 %).
- **Impacto:** riesgo alto en continuidad operativa y seguridad; Modelado del negocio.

Escala de rendimiento:

En caso de que un endpoint exceda estos límites en ejecuciones normales, se documentará como incidencia de desempeño básico, sin que esto implique pruebas de carga o estrés. A continuación, se presenta la escala de rendimiento promedio para los endpoints sometidos a las pruebas.

Tabla 13-Escala de rendimiento

Excelente: 0 – 300 ms	
Bueno: 301 – 450 ms	
Regular: 451 – 700 ms	
Lento: 701 – 1000 ms	
Muy lento / No operando: 1500 ms o más	

Tabla 14-Rendimientos evaluados-A

Método	Endpoint	Tiempo promedio (ms)	Tiempo máximo (ms)	Estado
Auth Module				
POST	/auth/login	1470	1515	❗
	/auth/login-google	450	475	❕
	/auth/refresh-token	350	406	✅
	/auth/forgot-password	30000	30000	❗
	/auth/change-password	485	495	❕
Pets Module				
GET	/pets?page=1&limit=1	630	642	❕
POST	/pets/{petId}/medical-histories	2440	2468	❗
POST	/diagnostics/{diagnosticId}/treatments	738	746	❗
POST	/diagnostics/{diagnosticId}/surgical-interventions	726	740	❗
PATCH	/pets/{petId}	2550	2606	❗
PATCH	/pets/medical-histories/{medicalHistoryId}	1702	1767	❗
DELETE	/diagnostics/treatments/{treatmentId}	645	666	❕
DELETE	/diagnostics/surgical-interventions/{surgicalIterventionId}	650	671	❕

Tabla 15-Rendimientos evaluados-B

Método	Endpoint	Tiempo promedio (ms)	Tiempo máximo (ms)	Estado
Bills Module				
POST	/bills	1975	1980	❗
GET	/bills?page=1&limit=10	802	820	❗
GET	/bills/:id	515	528	⚠️
User Module				
POST	/users	20000	20000	❗
GET	/users?page=1&limit=10	389	404	✅
POST	/users/pets	20000	20000	❗
POST	/users/:id/pets	4	8	✅
GET	/users/:id/pets	530	546	⚠️
GET	/users/:id	398	417	✅
DELETE	/users/:id	385	399	✅
POST	/users/request-document/:id	402	417	✅
Appointments Module				
POST	/appointments	3589	3642	❗
GET	/appointments?skip=1&take=10	510	543	⚠️
GET	/appointments/:id	489	524	⚠️
DELETE	/appointments/:id	256	334	✅

Auth Module:

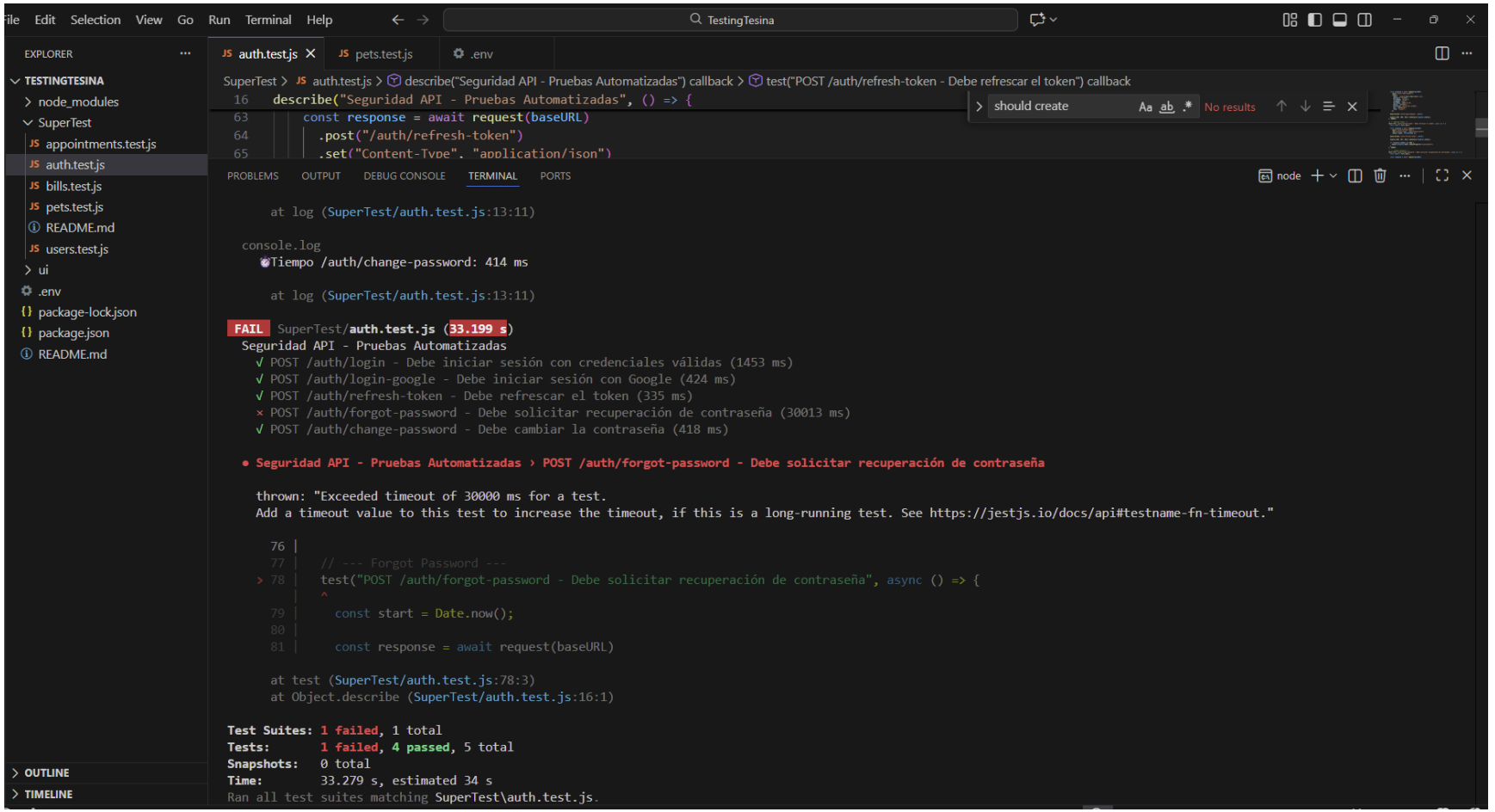


Ilustración 13-Prueba Módulo Auth

Pets module:

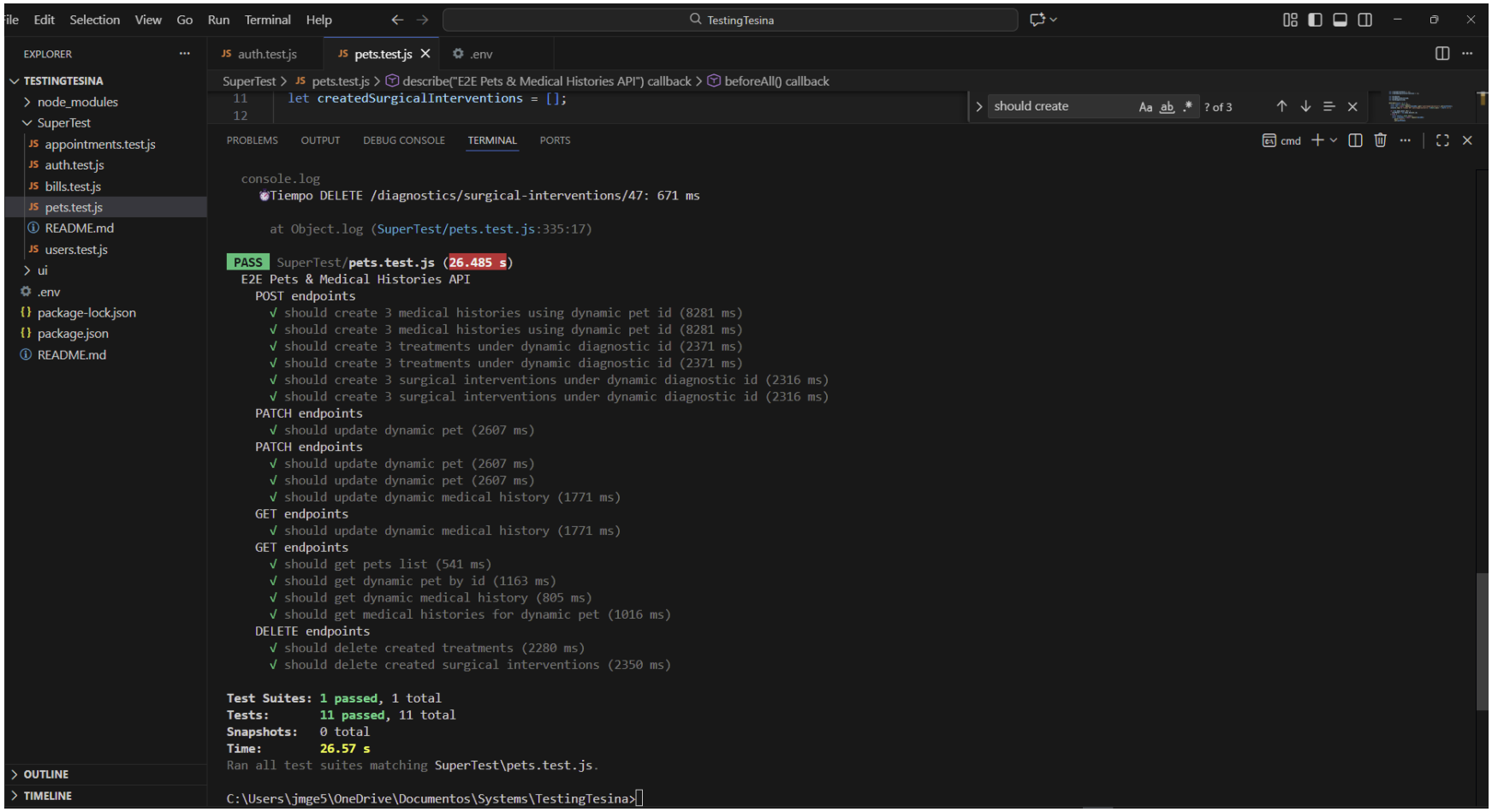


Ilustración 14-Prueba Módulo Pets

Bills Module:

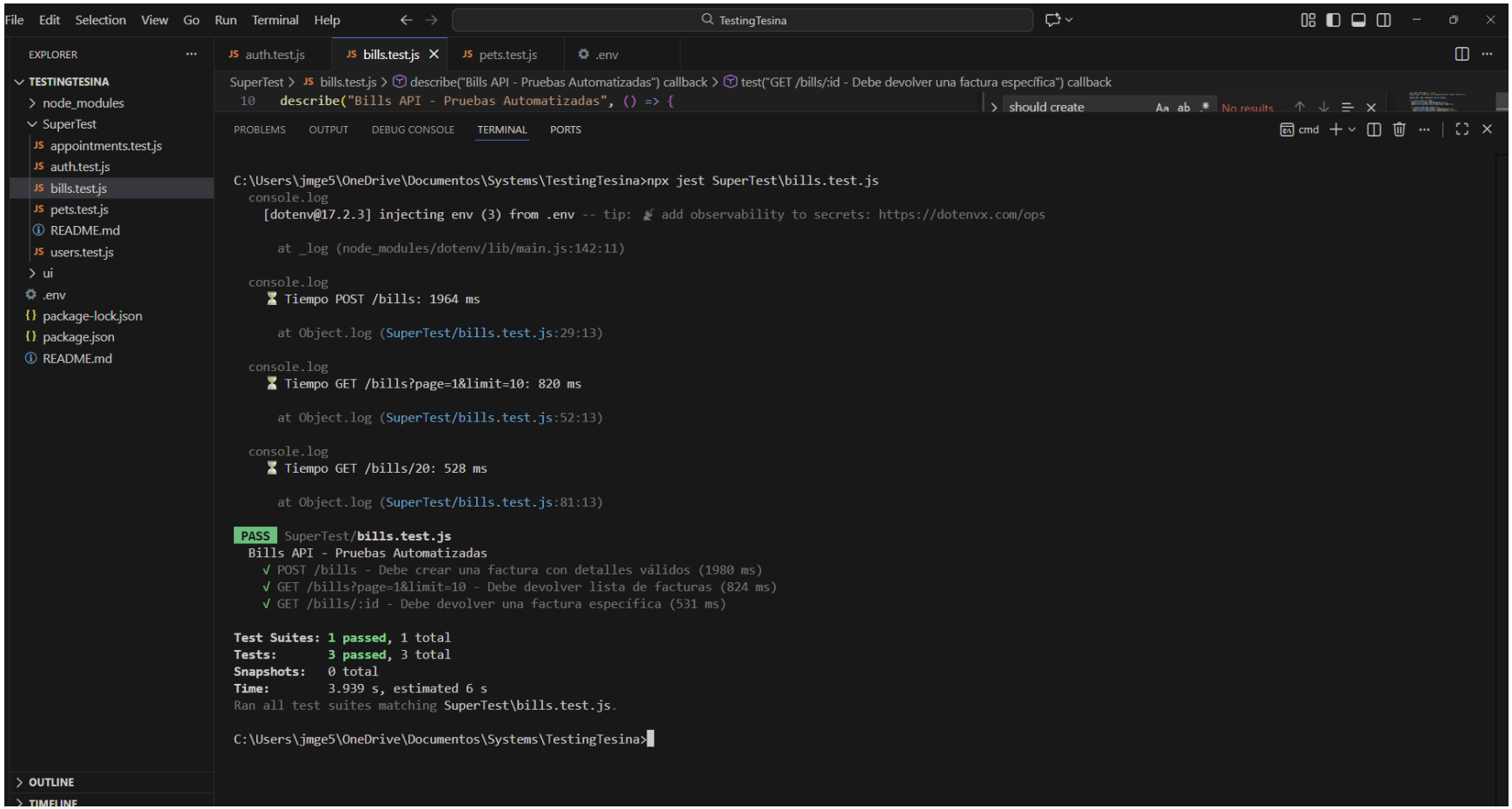


Ilustración 15-Prueba Módulo Bills

Users Module:

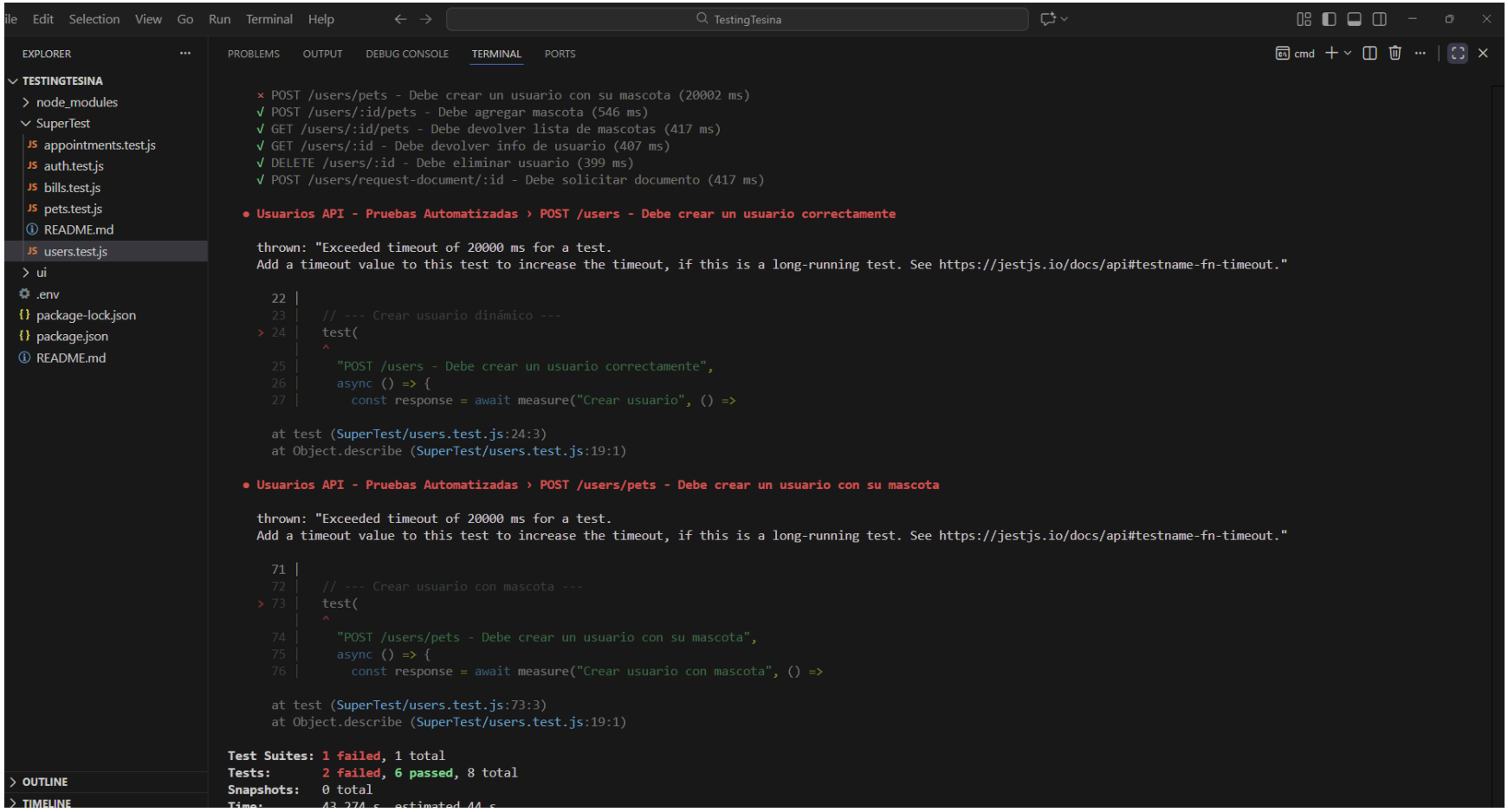


Ilustración 16-Prueba Módulo Users

Appointments Module:

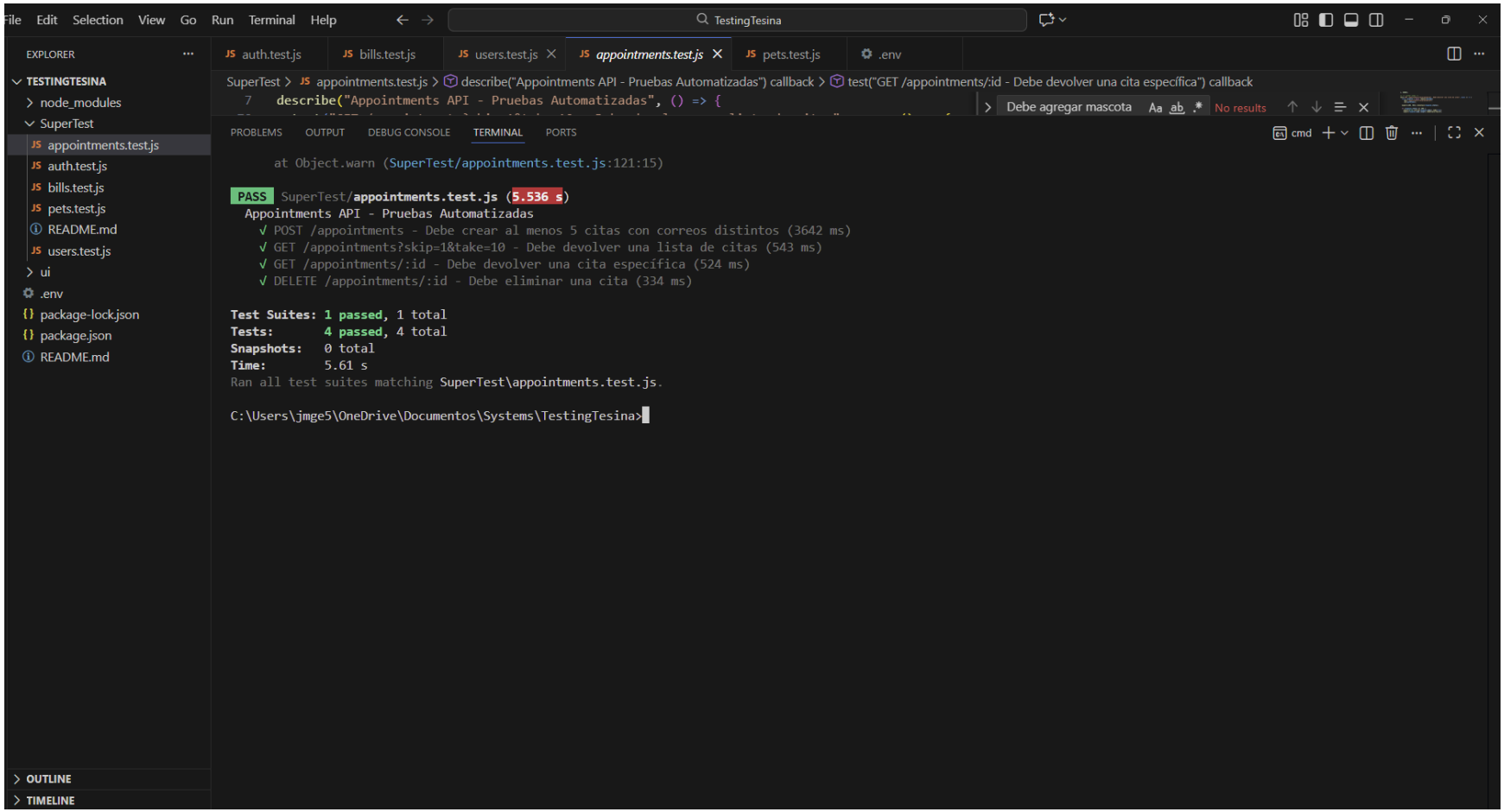


Ilustración 17-Prueba Módulo Appointments

.

Matriz de resultados

Tabla 16-Matriz de resultados

Módulo	Manual	Automatizadas	API	Estado	Estado
Usuarios (Clientes)	95%	95%	90%	Cobertura alta en todos los métodos, supera meta en 20–25%	**Aprobada**
Pacientes (Mascotas)	80%	90%	80%	Cobertura alta en todos los métodos, supera meta en 20–10%	**Aprobada**
Citas	60%	50%	65%	Automatizadas muy bajas (50% por debajo de la meta), Manual también < 40%, La API con las pruebas realizadas carece de ejecución y respuesta.	**Fallida**
Seguridad	80%	70%	65%	Manual, Automatizadas cumplen con los requisitos, pero en el caso de la API < 70%, El módulo de autentificación no responder correctamente	**Fallida**
Reportes	90%	95%	90%	Todos los métodos con cobertura alta, supera meta en 20–25%	**Aprobada**

- **Usuarios, Reportes y Pacientes (Mascotas)** están sólidamente dentro de la meta, mostrando flujos críticos bien cubiertos.
- **Citas y Seguridad** Son los módulos más críticos, con cobertura automatizada muy baja, lo que implica riesgo en la efectividad de regresión y la confiabilidad de la automatización.

Tabla 17-Observaciones Matriz de resultados-A

Módulo	Método	%	Observaciones
Usuario (Clientes)	Manual	95	No se encontraron inconsistencias los endpoints respondieron correctamente
	Automatizadas (PlayWright)	95	Interacciones UI estables, validación de formularios correcta.
	API (Spertest + Jest)	90	Se presentaron ciertas latencias y en unas pruebas con respecto al endpoint "/api/v1/users/1/pets?skip=1&take= 10" El endpoint se trabo debido a los recursos del servidor
Pacientes (Mascota)	Manual	65	No se encontraron inconsistencias con las ejecuciones de las pruebas
	Automatizadas (PlayWright)	70	Interacciones UI estables, validaciones correctas
	API (Spertest + Jest)	75	Se presentación algunos problemas de latencia con endpoints que dmandan muchos registros "/api/v1/pets/medical-histories/diagnostics/23/surgical-interventions", debido a recursos de servidor
Citas	Manual	60	No se han encontrado actualmente inconsistencias con las ejecuciones de las pruebas
	Automatizadas (PlayWright)	50	Las interacciones UI no presentan inconvenientes en el nivel de pruebas actuales
	API (Spertest + Jest)	75	Se ha detectado un problema con la funcionalidad de un endpoint el cual es Agendado de citas con Google calendar Detectamos que necesita la verificación de Google para funcionar

Tabla 18-Observaciones Matriz de resultados-B

Módulo	Método	%	Observaciones
Seguridad	Manual	80	Se encontró una inconsistencia en las pruebas realizada con la funcionalidad de Login de Google
	Automatizadas (PlayWright)	60	Se ha detectado una funciona no operando correctamente con la integración de authentificacion de Google 0Auth El sitio lanza una vista de error
	API (Spertest + Jest)	65	El endpoint testeado “/api/v1/auth/login-google” no responde correctamente
Reportes	Manual	90	No se han encontrado actualmente inconsistencias con las ejecuciones de las pruebas
	Automatizadas (PlayWright)	95	Las interacciones UI no presentan inconvenientes en el nivel de pruebas actuales
	API (Spertest + Jest)	90	No se han detectados fallos en las pruebas con respecto a este módulo en las pruebas actuales

3.13 Análisis de riesgo

El análisis de riesgos permite identificar los posibles eventos que podrían afectar la ejecución del plan de pruebas, la calidad de los resultados o el comportamiento del sistema durante su validación. Para el sistema de historiales médicos de la Clínica Veterinaria Mitsum, los riesgos se clasifican en tres categorías: riesgos técnicos del sistema, riesgos del proceso de pruebas y riesgos operativos del equipo..

Tabla 19-Análisis de riesgos-A

ID	Riesgo	Catego-ría	Impac-to	Proba-bilidad	Descripción	Estrategia de Mitigación
R1	Inestabilidad en la conexión con la API	Técnico	Alto	Media	Pérdida de conexión o fallos intermitentes en el backend pueden bloquear pruebas E2E o de API.	Verificar estabilidad del entorno antes de ejecutar pruebas; reintentos controlados en automatización.
R2	Datos inconsistentes entre módulos	Técnico	Alto	Media	Un error en pacientes o consultas puede afectar historiales y facturación.	Validar integridad entre módulos antes de pruebas; usar datos simulados controlados.
R3	Errores intermitentes en autenticación	Técnico	Alto	Baja–Media	Fallas en login afectan el inicio de pruebas manuales y automatizadas.	Crear usuario exclusivo para pruebas; validar endpoint de login previo a cada ciclo.
R4	Respuestas lentas del servidor	Técnico	Medio	Media	Retrasos pueden generar falsos positivos en automatización o fallas de timeout.	Ajustar tiempos de espera; registrar endpoints lentos; medir con SuperTest.

Tabla 20-Análisis de riesgos-B

ID	Riesgo	Categoría	Impacto	Probabilidad	Descripción	Estrategia de Mitigación
R5	Datos de prueba insuficientes o mal preparados	Pruebas	Medio	Media	Falta de datos ficticios bloquea flujos críticos como consultas o historiales.	Crear dataset base antes del inicio; validar contenido previo al ciclo de pruebas.
R6	Automatización incompleta o con fallos	Pruebas	Medio	Media	Scripts mal configurados pueden generar fallos no atribuibles al sistema.	Probar scripts localmente; revisión cruzada por un segundo integrante.
R7	Cambios tardíos en el sistema durante pruebas	Pruebas	Alto	Baja	Actualizaciones pueden invalidar casos ya ejecutados.	Congelar versión durante pruebas; coordinar con equipo técnico.
R8	Documentación incompleta o ambigua	Pruebas	Medio	Media	Falta de claridad en BPMN o requerimientos genera casos erróneos.	Revisión interna del equipo; validación previa con escenarios de negocio.
R9	Baja disponibilidad del equipo académico	Operativo	Medio	Media –Alta	Dificultad para coincidir en tareas grupales.	Definir calendario fijo; dividir tareas por especialidad.

Tabla 21-Análisis de riesgos-C

ID	Riesgo	Catego-ría	Impac-to	Proba-bilidad	Descripción	Estrategia de Mitigación
R10	Falta de experiencia con herramientas de automatización	Operativo	Medio	Alta	Curva de aprendizaje de Playwright y SuperTest puede retrasar entregables.	Pruebas preliminares, capacitación rápida, pair-programming.
R11	Fallas en el pipeline CI/CD (GitHub Actions)	Operativo	Bajo–Medio	Media	El pipeline puede bloquear la ejecución automática de pruebas.	Validar ejecución local; ejecutar pipeline con cambios mínimos.
R12	Disponibilidad limitada del entorno de la clínica	Técnico/ Operativo	Medio	Baja	El sistema puede estar en uso real, afectando acceso a ciertos datos.	Uso estricto del entorno staging con datos ficticios.
R13	Riesgo de exposición de datos sensibles	Técnico/ Seguridad	Alto	Baja	Manejo incorrecto podría exponer datos reales de la clínica.	Uso exclusivo de datos ficticios; no ingresar datos reales durante pruebas.

3.14 Supuestos

Los supuestos establecen las condiciones que se consideran verdaderas para la correcta ejecución del plan de pruebas. Permiten delimitar el contexto operativo y definir expectativas realistas sobre el ambiente, los recursos y el funcionamiento del sistema de historiales médicos durante el estudio.

Para este proyecto académico se asumen los siguientes elementos:

1. El entorno de pruebas estará disponible durante todo el proceso

Se asume que el backend, frontend y la base de datos del sistema permanecerán accesibles sin interrupciones significativas durante la ejecución del plan de pruebas.

2. Los datos utilizados serán completamente ficticios

Por razones éticas y de protección de información, se asume que:

- No se utilizarán datos reales de la clínica.
- Toda la información será creada exclusivamente para propósitos de prueba.

3. El sistema no recibirá actualizaciones durante la fase de pruebas

Se asume que la versión utilizada permanecerá estable, sin modificaciones funcionales o estructurales que afecten los flujos definidos.

4. La clínica utilizará únicamente los dos roles existentes

Aunque el negocio tiene varios puestos (recepción, veterinario, asistente), se asume que todos operan bajo:

- Rol Administrador
- Rol Cliente
- Ningún rol adicional será habilitado o evaluado en esta fase.

5. El equipo de pruebas tendrá acceso total a los módulos incluidos en el alcance

Se asume que no existirán restricciones de acceso que impidan validar módulos como:

- Pacientes

- Consultas
- Citas
- Historial clínico
- Facturación

6. Las pruebas automatizadas podrán ejecutarse tanto localmente como en CI/CD

Se asume que el pipeline de GitHub Actions funcionará correctamente para ejecuciones básicas de:

- SuperTest (API)
- Playwright (E2E)

7. La documentación de procesos y BPMN es confiable

Se asume que los modelos de procesos otorgados representan fielmente los flujos reales de la clínica y sirven como base para crear los casos de prueba.

8. Los integrantes del equipo estarán disponibles según el cronograma

Se asume que las tareas asignadas podrán completarse dentro de los tiempos establecidos sin contratiempos mayores.

9. No se ejecutarán pruebas fuera del alcance

Se asume que no se realizarán pruebas de:

- Carga
- Estrés
- Penetración
- Robustez a nivel de infraestructura
- Estas no forman parte del alcance definido en el anteproyecto.

10. Las herramientas seleccionadas son adecuadas para el proyecto

Se asume que Playwright, SuperTest y Jest cumplen con las necesidades del plan de pruebas y no será necesario incorporar herramientas adicionales.

3.15 Tareas del equipo de pruebas

Esta sección describe las actividades principales que el equipo de pruebas llevará a cabo durante el proceso de verificación y validación del sistema de historiales médicos. Las tareas se distribuyen en función de la estrategia definida, los niveles de prueba seleccionados y las responsabilidades de cada integrante dentro del proyecto académico.

Las tareas se organizan en las siguientes categorías:

1. Análisis y Planeación

- Revisar el alcance del sistema y los requerimientos definidos.
- Analizar los BPMN entregados para comprender los flujos de negocio.
- Identificar módulos, funcionalidades y escenarios críticos.
- Planificar los ciclos de ejecución manual y automatizado.
- Definir prioridades de los casos de prueba con base en riesgo.

2. Diseño de Casos de Prueba

- Elaborar casos de prueba funcionales basados en los procesos del negocio.
- Aplicar técnicas de diseño ISTQB (partición de equivalencia, valores límite y casos de uso).
- Identificar escenarios automatizables (login, CRUD, historiales, etc.).
- Definir datos de prueba ficticios para todos los módulos.
- Revisar y aprobar la matriz de pruebas.

3. Preparación del Entorno

- Configurar el entorno de pruebas (frontend, backend y base de datos).
- Crear usuarios y credenciales necesarias para las pruebas.
- Registrar datos simulados como pacientes, propietarios, diagnósticos, citas y consultas.
- Verificar disponibilidad de herramientas y repositorios (GitHub, Playwright, SuperTest).

4. Ejecución de Pruebas Manuales

- Ejecutar los casos de prueba funcionales planificados.
- Validar flujos CRUD y procesos clínicos (consultas, historiales, citas).
- Registrar hallazgos en la matriz de pruebas.
- Capturar evidencia (capturas de pantalla, resultados observados).

5. Desarrollo y Ejecución de Pruebas Automatizadas

- Implementar scripts E2E con Playwright para flujos críticos.
- Implementar pruebas de API con SuperTest y Jest.
- Ejecutar las pruebas automatizadas localmente y en CI/CD.
- Analizar resultados y capturar evidencias de ejecución.

6. Gestión de Defectos

- Documentar defectos detectados indicando módulo, severidad y pasos de reproducción.
- Clasificar defectos según severidad y prioridad.
- Validar defectos corregidos u observados nuevamente durante re ejecuciones.

7. Seguimiento y Coordinación

- Participar en reuniones de seguimiento del avance del proyecto.
- Actualizar el cronograma según la carga de trabajo.
- Coordinar dependencias entre pruebas manuales y automatizadas.

8. Elaboración de Documentación Final

- Consolidar resultados de ejecución.
- Generar reportes automáticos de Playwright y SuperTest.
- Completar la matriz de pruebas y el registro de defectos.
- Preparar contenido para el capítulo final e informe de tesina.

3.16 Roles y responsabilidades

Para la ejecución del caso de estudio, se asignarán roles operativos que permitirán organizar las actividades de prueba de acuerdo con la estrategia definida en las secciones anteriores. Cada rol tendrá funciones específicas orientadas a garantizar la correcta planificación, ejecución, seguimiento y cierre del ciclo de pruebas.

Aunque los roles están claramente definidos, la ejecución se llevará a cabo en un entorno académico con un equipo reducido; por tanto, las responsabilidades podrán distribuirse de manera flexible según las necesidades del proyecto y la carga de trabajo de cada fase.

a) Líder de Pruebas

Será responsable de coordinar y supervisar la ejecución general del proceso de pruebas.

Entre sus funciones estarán:

- Organizar las actividades de la iteración de pruebas.
- Validar que los casos y scripts estén completos antes de cada ejecución.
- Supervisar el cumplimiento de criterios de entrada y salida.
- Consolidar los resultados finales de pruebas manuales y automatizadas.

b) Diseñador de Casos de Prueba

Este rol se encargará de preparar los artefactos necesarios para la ejecución funcional.

Responsabilidades:

- Diseñar los casos de prueba de acuerdo con los flujos definidos en el modelado del negocio.
- Identificar los datos de prueba necesarios para la ejecución.
- Mantener actualizada la matriz de trazabilidad entre requisitos, módulos y casos.
- Ajustar los casos de prueba cuando existan nuevas observaciones o cambios en la aplicación.

c) Ejecutor de Pruebas Funcionales

Será la persona encargada de verificar manualmente el comportamiento del sistema.

Responsabilidades:

- Ejecutar los casos de prueba siguiendo los pasos definidos.
- Registrar evidencias, resultados y observaciones.
- Reportar defectos encontrados con su información asociada (pasos, datos, evidencia).

d) Encargado de Automatización

Se encargará de la creación y ejecución de scripts automatizados.

Responsabilidades:

- Desarrollar pruebas E2E con Playwright y pruebas API con SuperTest/Jest.
- Ejecutar los scripts localmente y a través de GitHub Actions.
- Mantener actualizada la suite automatizada conforme avanza el proyecto.
- Reportar fallos detectados durante ejecuciones automáticas y generar la evidencia correspondiente.

Participación flexible del equipo

Dado que este proyecto académico se ejecuta con un equipo limitado, los integrantes podrán asumir más de un rol cuando sea necesario. Esto permitirá:

- Agilizar el diseño y ejecución de casos.
- Compartir el trabajo de automatización y documentación.
- Asegurar continuidad del proceso en etapas críticas.

Aun con esta flexibilidad, el Líder de Pruebas será responsable de mantener la trazabilidad de actividades y de coordinar la correcta distribución de tareas.

3.17 Calendario de Pruebas

El calendario de pruebas establece la planificación temporal estimada para cada una de las actividades definidas en la estrategia de pruebas. Dado que el proyecto se desarrolla en el marco académico y con un equipo reducido, las fechas propuestas consideran la disponibilidad de los integrantes, la curva de aprendizaje en automatización y la coordinación necesaria para asegurar trazabilidad entre los entregables.

Las fechas presentadas corresponden a una planificación tentativa que podrá ajustarse según el avance del equipo, la disponibilidad del entorno de pruebas y el tiempo requerido para corregir defectos detectados.

Tabla 22-Cronograma de pruebas-A

Actividad	Descripción	Inicio	Fin	Días	Progreso
Preparación del entorno y datos de prueba	Configuración del ambiente local, creación de base de datos inicial, revisión de endpoints y rutas del sistema.	01/10/2025	21/10/2025	21	100%
Diseño de casos de prueba manuales	Elaboración de la matriz de casos, definición de datos de prueba y alineación con BPMN y módulos definidos.	13/10/2025	31/10/2025	19	100%
Implementación de pruebas automatizadas (API y E2E)	Desarrollo de scripts con SuperTest y Playwright. Configuración inicial del pipeline en GitHub Actions.	27/10/2025	07/11/2025	12	100%
Ejecución de pruebas funcionales manuales	Validación de flujos CRUD, historial clínico, citas y accesos. Registro de resultados y defectos.	31/10/2025	12/11/2025	13	100%
Ejecución de pruebas E2E y API	Ejecución local y en CI/CD de las suites automatizadas. Ajustes derivados de fallos y condiciones no previstas.	03/11/2025	14/11/2025	12	100%

Tabla 23-Cronograma de pruebas-B

Actividad	Descripción	Inicio	Fin	Días	Progreso	D. comp.
Análisis de resultados y consolidación de evidencias	Organización de capturas, reportes automatizados, registros y matriz final.	12/11/2025	26/11/2025	15	100%	15
Elaboración del capítulo de pruebas y anexos	Redacción final del Capítulo III conforme a los hallazgos y ejecución realizada.	23/11/2025	28/11/2025	6	100%	6
Revisión final interna del equipo	Verificación de trazabilidad y completitud de entregables de pruebas.	27/11/2025	30/11/2025	4	100%	4

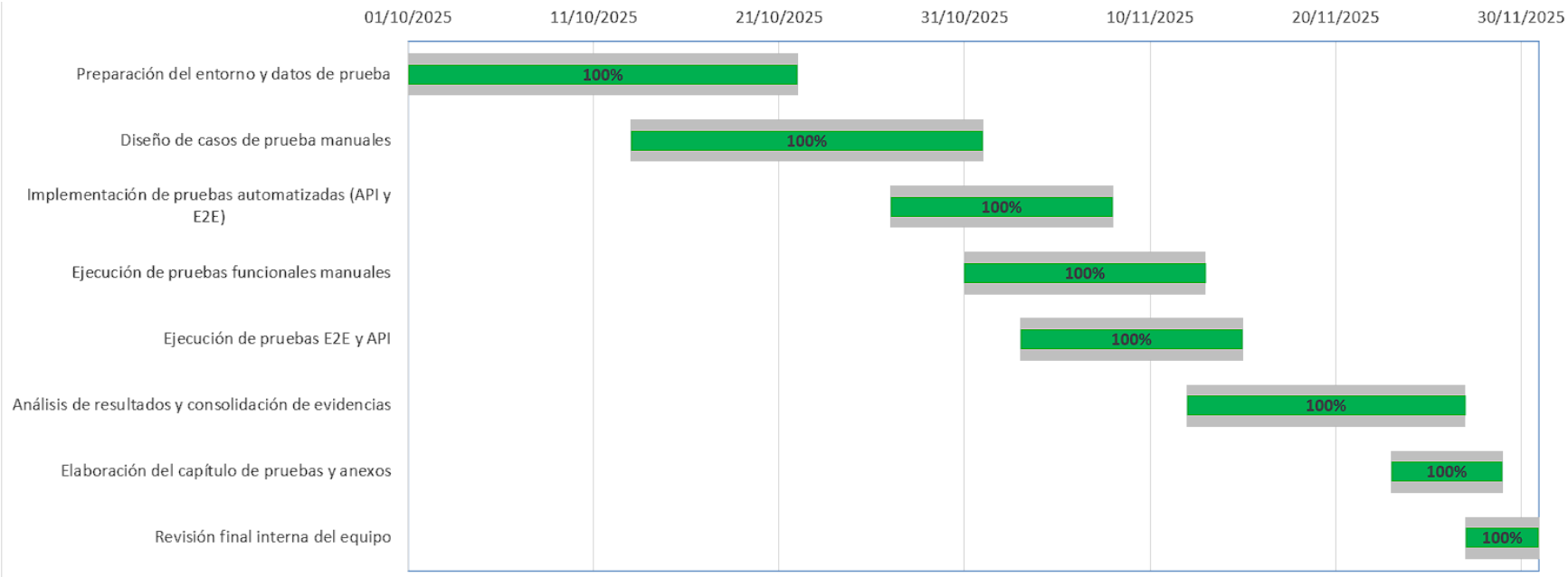


Ilustración 18-Digrama de Gantt

3.18 Principales hitos

Los hitos representan los eventos más relevantes dentro del proceso de pruebas, aquellos que marcan el avance significativo del proyecto y permiten evaluar el cumplimiento del cronograma. Cada hito está directamente relacionado con actividades críticas del calendario definido en la sección 3.17 y constituye un punto de control para validar que el proceso progresa conforme a lo planificado.

Hito 1: Entorno de pruebas completamente configurado

Fecha estimada: 21 de octubre de 2025

Incluye la preparación del ambiente local, base de datos inicial, credenciales, rutas API y verificación funcional mínima del sistema. Este hito marca el inicio formal de las actividades de prueba.

Hito 2: Casos de prueba manuales finalizados

Fecha estimada: 31 de octubre de 2025

Corresponde a la elaboración total de la matriz de casos, definición de datos de prueba y alineación con los flujos clínicos y BPMN. Este hito habilita la ejecución manual y automatizada.

Hito 3: Scripts automatizados implementados

Fecha estimada: 07 de noviembre de 2025

Comprende el desarrollo de pruebas API (SuperTest), pruebas E2E (Playwright) y el pipeline inicial en GitHub Actions. Este hito permite iniciar ejecuciones automatizadas consistentes.

Hito 4: Ejecución completa de pruebas manuales

Fecha estimada: 12 de noviembre de 2025

Incluye la validación de flujos CRUD, módulos críticos, control de acceso y navegación. Representa la culminación de la verificación funcional a nivel de sistema.

Hito 5: Ejecución de pruebas automatizadas concluida

Fecha estimada: 14 de noviembre de 2025

Se completan todas las ejecuciones E2E y API tanto localmente como en CI/CD, incluyendo ajustes a scripts y repetición de escenarios.

Hito 6: Evidencias y resultados consolidados

Fecha estimada: 26 de noviembre de 2025

Consiste en recopilar capturas, resultados automatizados, registros de defectos y actualizar la matriz de trazabilidad. Este hito cierra el ciclo de validación técnica.

Hito 7: Capítulo de pruebas redactado y documentado

Fecha estimada: 28 de noviembre de 2025

Incluye la elaboración completa del Capítulo III y sus anexos, incorporando resultados, análisis y conclusiones del proceso.

Hito 8: Revisión final interna del equipo

Fecha estimada: 30 de noviembre de 2025

Último punto de control antes de la entrega final. Se valida consistencia documental, trazabilidad, formato y cumplimiento de los objetivos del plan de pruebas.

3.19 Recursos necesarios

La correcta ejecución del plan de pruebas del sistema de historiales médicos de la Clínica Veterinaria Mitsum requiere la disponibilidad de diversos recursos que permitan llevar a cabo las actividades de verificación y validación de manera ordenada, controlada y reproducible. Estos recursos abarcan desde el personal involucrado hasta las herramientas tecnológicas, los ambientes de prueba y los datos simulados que garantizarán un proceso seguro y confiable.

A continuación, se describen los recursos necesarios para el proyecto.

1. Recursos Humanos

Para asegurar una adecuada cobertura de pruebas, se requiere un equipo con roles claramente definidos. Cada integrante participa en distintas etapas del proceso, permitiendo dividir tareas y mantener una trazabilidad adecuada:

El equipo definido en la sección “3.16 Roles y responsabilidades” combina perfiles técnicos y funcionales, lo que permite evaluar tanto el comportamiento observable del sistema como su alineación con los procesos reales de la clínica.

2. Recursos Tecnológicos

El proyecto requiere equipos de cómputo capaces de ejecutar software de automatización, navegadores modernos y herramientas de desarrollo. Cada integrante debe contar con una computadora que posea, como mínimo:

- Procesador equivalente a Intel i5
- 8 GB de memoria RAM
- Espacio disponible para repositorios y herramientas de pruebas
- Conexión estable a internet

Estos equipos permiten ejecutar tanto pruebas manuales como automatizadas, además de operar los ambientes del sistema.

El entorno de pruebas se ejecutará exclusivamente en navegadores web, por lo que se requiere el uso de Google Chrome y Mozilla Firefox para validar compatibilidad básica.

3. Recursos de Software

El proyecto se apoyará en un conjunto de herramientas que facilitan el diseño, ejecución y documentación de las pruebas:

3.1 Herramientas para pruebas manuales

- Navegadores web para verificación funcional.
- Capturadores de pantalla para recopilación de evidencia.

3.2 Herramientas para automatización

- Playwright para pruebas end-to-end del frontend.
- SuperTest para validación de endpoints del API.
- Jest como framework de ejecución.
- Node.js como entorno de ejecución para los scripts.

3.3 Control de versiones y CI/CD

- GitHub para almacenar scripts, casos de prueba y evidencias.
- GitHub Actions para ejecutar pruebas automatizadas de manera continua.

3.4 Documentación

- Word para el informe final.

Estas herramientas permiten integrar pruebas manuales y automatizadas dentro de un mismo flujo de trabajo.

4. Recursos del Sistema Bajo Prueba

Para ejecutar las pruebas se necesita acceso a:

- Un entorno de pruebas (staging) independiente del ambiente de producción.
- Una base de datos inicializada únicamente con información ficticia, acorde a buenas prácticas de privacidad.
- Credenciales válidas para los dos roles existentes: Administrador y Cliente.
- Documentación de endpoints del API, necesaria para los scripts en SuperTest.

Esto asegura que las pruebas se realicen sin comprometer datos reales y que todas las interacciones sean controlables.

5. Recursos de Datos

Para validar los módulos del sistema, se requieren datos simulados representativos:

- Pacientes ficticios con características diferentes.

- Propietarios simulados.
- Registros clínicos, diagnósticos, citas y tratamientos creados específicamente para escenarios de prueba.
- Servicios veterinarios (vacunas, cirugías, procedimientos) generados para validar el funcionamiento del sistema.

Estos datos permiten validar flujos sin afectar información real y replicar escenarios necesarios para pruebas funcionales y automatizadas.

6. Recursos Administrativos

Finalmente, se requieren elementos que permitan organizar el trabajo y asegurar trazabilidad:

- Cronograma de actividades y fechas clave.
- Plan de seguimiento y documentación de incidencias.
- Requisitos funcionales, diagramas BPMN y descripción de procesos.
- Registro centralizado para reportes, evidencias y matrices de prueba.

Estos recursos aseguran una ejecución ordenada y coherente con las necesidades del proyecto académico.

3.19.1 Ambientes de pruebas

Para la ejecución del plan de pruebas se utilizaron dos ambientes principales: producción y pruebas. No obstante, para el caso de estudio del presente capítulo se trabajó exclusivamente sobre el ambiente de pruebas (testing), con el fin de proteger la información real de la clínica y permitir la generación controlada de datos ficticios.

1. Ambiente de Producción

El ambiente de producción corresponde al sistema oficial utilizado por el personal de la Clínica Veterinaria Mitsum. Este entorno contiene datos reales de clientes, mascotas y procesos clínicos, y se utiliza únicamente como referencia funcional.

a. **URL:** https://www.veterianariamitsum.life
b. **Versión:** 1.0.0
c. **Disponibilidad:** 24/7 para la operación de la clínica.
d. Restricciones:
   i) No se ejecutan pruebas funcionales ni automatizadas.
   ii) No se generan datos de prueba.
   iii) No se manipulan registros reales.

2. Ambiente de Pruebas (Testing)

El ambiente de pruebas fue creado específicamente para este proyecto y replica la estructura y funcionalidades del ambiente de producción, pero con datos independientes y controlados.

a. Front-end:
https://vet-mitsun-development.vercel.app
b. Back-end y documentación de API:
https://dsi-nest-backend-development.up.railway.app/api/v1/docs/
c. Repositorio de scripts:
https://github.com/Gutyerrez-360/TestingTesina/tree/main/SuperTest
d. Características principales:
   i) Versión 1.0, alineada con producción.
   ii) Código fuente versionado en GitHub, con ramas específicas para pruebas.
   iii) Base de datos con información exclusivamente ficticia.
e. Credenciales utilizadas en las pruebas:
   i) Usuario con rol **Administrador**, para validar flujos completos y configuraciones.
   ii) Usuario con rol **Cliente**, para comprobar restricciones de acceso y visibilidad limitada.

Este ambiente de pruebas fue el utilizado para la ejecución de pruebas funcionales manuales, el desarrollo y ejecución de scripts automatizados con Playwright y SuperTest, según la estrategia descrita en la sección 3.20.

3.19.2 Planeación de Recursos

La planeación de recursos para la ejecución del plan de pruebas se realizó considerando principalmente la infraestructura del servidor de pruebas en la nube, donde se desplegaron el backend y el frontend del sistema de historiales médicos de la Clínica Veterinaria Mitsum. Para ello, se utilizaron servicios PaaS (Platform as a Service) que permiten disponer de un entorno de pruebas accesible vía web, con recursos de cómputo suficientes para ejecutar las pruebas funcionales y automatizadas definidas en el capítulo.

1. Backend – Railway (Hobby Plan)

El backend del sistema se desplegó en la plataforma Railway, utilizando el plan **Hobby**, que ofrece un contenedor con recursos de CPU, memoria y almacenamiento suficientes para aplicaciones de tamaño pequeño/mediano. Este tipo de plan permite ejecutar servicios Node.js con una asignación de memoria limitada, almacenamiento persistente básico y conectividad a Internet para acceso a la base de datos y al frontend.

a. Entorno de ejecución gestionado por Railway.
b. Recursos de CPU y memoria suficientes para soportar las pruebas concurrentes de API y flujos E2E.
c. Despliegue continuo a partir del repositorio de código del proyecto, facilitando la actualización del entorno de pruebas.

2. Frontend – Vercel (Plan Gratuito)

La aplicación frontend se desplegó en **Vercel** utilizando la **capa gratuita**, que permite alojar aplicaciones web con distribución a través de una red de entrega de contenido (CDN) y funciones serverless opcionales.

La capa gratuita está orientada a proyectos personales, prototipos y entornos de desarrollo/pruebas, ofreciendo recursos de ejecución suficientes para el caso de estudio.

a. Despliegue automático desde el repositorio Git (integración con ramas de desarrollo).

b. Límite de uso acorde con un entorno de pruebas académico (peticiones y ancho de banda moderados).

Estos recursos de infraestructura en la nube constituyen el “hardware lógico” del servidor de pruebas, ya que abstraen el detalle físico del servidor, pero proporcionan la capacidad de cómputo necesaria para ejecutar las pruebas funcionales, de integración y end-to-end del sistema.

Software

Sobre la infraestructura en Railway y Vercel se desplegaron los componentes de software del sistema y las dependencias necesarias para ejecutar las pruebas:

- **Backend:**
  - Aplicación desarrollada en Node.js/NestJS.
  - Dependencias gestionadas con npm.
  - Exposición de endpoints REST/HTTP para el consumo desde el frontend y desde las herramientas de prueba de API (por ejemplo, SuperTest).
- **Frontend:**
  - Aplicación web desplegada en Vercel.
  - Configuración de variables de entorno para enlazar con el backend en Railway (URL del API).

- **Entorno de pruebas automatizadas:**
  - Node.js instalado en los runners para ejecutar los paquetes de pruebas.
  - Librerías de pruebas como Jest, SuperTest y Playwright instaladas como dependencias del proyecto.

3.20 Estrategia de datos de prueba y herramientas

3.20.1 Estrategia de datos

La estrategia de datos de prueba se basará en la generación controlada y limpieza periódica de la información utilizada durante las ejecuciones de prueba. Los datos se construirán a partir de:

- **Datos sintéticos** creados específicamente para escenarios funcionales (mascotas, propietarios, consultas, diagnósticos, tratamientos y servicios).
- **Datos representativos** que simulan estructuras reales presentes en la veterinaria, sin incluir información personal o sensible.
- **Técnicas de partición de equivalencias y valores límite** para definir rangos válidos e inválidos.
- **Conjuntos de datos para pruebas negativas**, como registros incompletos, tipos de dato incorrectos y combinaciones no permitidas.
- **Datos preconfigurados** para flujos críticos (consultas, actualización de historiales, servicios facturados).

Esta estrategia garantiza control, trazabilidad y repetición confiable de resultados.

3.20.2 Herramientas de prueba

3.20.2.1 Pruebas de Código

Para el backend del sistema se utilizarán **Jest** y **SuperTest** como frameworks principales para la ejecución de pruebas unitarias y pruebas de integración orientadas a validar los endpoints definidos en la documentación del proyecto, incluyendo los esquemas y contratos descritos en **Swagger**.

El proceso de pruebas se diseñó para asegurar que cada módulo del sistema funcione de forma aislada y que la comunicación entre componentes cumpla con las expectativas funcionales.

Se crearán pruebas enfocadas en:

- Validación de reglas de negocio (Ej. Envió correcto de notificación, verificación de seguridad, validación correcta de los campos esperados por cada endpoint).
- Validación de controladores y servicios.
- Validación de funciones auxiliares.
- Manejo de errores y respuestas esperadas.
- Manejo del formato del cuerpo de la solicitud sea validado acorde a la indicación en la documentación del sistema.
- Pruebas de integración con SuperTest, ejecutando solicitudes reales contra el servidor levantado en modo de prueba para validar flujos completos.

Estas pruebas se integrarán dentro del proceso de **CI/CD** cuando aplique, actuando como el **primer filtro de calidad** antes de la ejecución de pruebas funcionales y de usuario.

Implementación de Scripts Automatizados

Se desarrollarán scripts automatizados en **package.json** y archivos con extensión js y en módulos específicos dentro del proyecto, los cuales permiten:

- Ejecutar el entorno de pruebas de manera aislada.
- Ejecución dinámica mediante variables de entorno.
- Ejecutar Jest con configuración extendida (watch mode, reporter).
- Automatizar la creación de un ambiente temporal para pruebas de integración.

Estos scripts facilitan la repetición y consistencia de las pruebas dentro del flujo de desarrollo.

- Los archivos.test.js contienen casos de prueba individuales y suites agrupadas con describe ().

- SuperTest se utiliza dentro de los archivos de integración para ejecutar peticiones HTTP/HTTPS contra una instancia de prueba.
- Los módulos del directorio /test-environment permiten controlar los estados iniciales del sistema (limpieza, cargas iniciales).

Este modelo mejora la mantenibilidad, trazabilidad y organización del proceso de pruebas. Además de las pruebas unitarias y de integración realizadas sobre el backend con Jest y SuperTest, para la capa de presentación se implementaron pruebas automatizadas end-to-end utilizando el framework Playwright. Este framework permite interactuar con la aplicación web de la Clínica Veterinaria Mitsum de forma similar a como lo haría un usuario real, abriendo el navegador, ingresando credenciales, navegando entre pantallas y validando el comportamiento de la interfaz.

Las pruebas con Playwright se enfocaron en los flujos críticos identificados en el plan de pruebas, entre ellos:

- Autenticación de usuarios (login y logout).
- Gestión de pacientes y propietarios (edición y consulta de registros).
- Registro y visualización de historiales clínicos.
- Navegación entre módulos principales y verificación de mensajes de validación.

Cada script E2E se construyó a partir de los casos de prueba definidos en la matriz de pruebas, respetando precondiciones, datos de entrada y resultados esperados. De esta manera, se aseguró la trazabilidad entre los requisitos funcionales, los casos documentados y las automatizaciones implementadas. Asimismo, Playwright se configuró para ejecutar las pruebas en al menos un navegador principal (Chrome/Chromium), validando el correcto funcionamiento de la interfaz bajo condiciones de uso típicas.

Finalmente, las suites de Playwright se integraron en el pipeline de GitHub Actions, permitiendo su ejecución automática en cada cambio relevante del repositorio. Esto facilitó la detección temprana de regresiones en la interfaz gráfica y complementó las pruebas de API y de lógica de negocio ejecutadas con Jest y SuperTest, proporcionando una visión integral del comportamiento del sistema.

3.20.3 Herramientas de acceso a datos

Acceso y verificación de datos

Para validar y manipular información durante las pruebas, se utilizaron herramientas de cliente SQL como **DBeaver** y **PostgreSQL**, conectadas al **entorno de testing alojado en la nube**.

Estas herramientas permitieron ejecutar consultas directas para revisar registros, confirmar transacciones y verificar la integridad de la base de datos en cada caso de prueba.

Se desarrollaron **consultas SQL específicas** para apoyar la validación del backend, así como un conjunto de scripts orientados a mantener un entorno estable:

- Limpieza de tablas utilizadas en pruebas.
- Reinserción de datos iniciales o datos semilla.
- Restauración de estados previos para validar escenarios puntuales.

Estos scripts facilitaron la repetición de pruebas y el control de los datos durante el proceso.

Resultado final

El uso combinado de consultas, scripts SQL y herramientas de administración permitió:

- Mantener un entorno de prueba consistente.
- Auditar resultados de forma clara y confiable.
- Hay que asegurar que el comportamiento del backend coincidiera con el estado real de los datos.

Consultas realizadas:

```sql
/* ========================================================
   1. GETS
   ======================================================== */
SELECT * FROM "user";
SELECT * FROM specie;
SELECT * FROM pet;
SELECT * FROM medical_history;
SELECT * FROM appointment;
SELECT * FROM product;

/* ========================================================
   2. INSERTS (5 items por tabla)
   ======================================================== */
/* ---------- USER ---------- */
INSERT INTO "user"
  (first_name, last_name, email, password, role, phone, birthday, direction, dui, recovery_token, created_at, updated_at)
VALUES
  ('Carlos', 'Lopez', 'carlos1@example.com', 'pwd1', 'client', '7000-0001', '1990-01-01', 'San Salvador', '01234567-1', NULL, NOW(), NOW()),
  ('Andrea', 'Martinez', 'andrea2@example.com', 'pwd2', 'client', '7000-0002', '1992-02-02', 'Santa Tecla', '01234567-2', NULL, NOW(), NOW()),
  ('Mario', 'Reyes', 'mario3@example.com', 'pwd3', 'admin', '7000-0003', '1985-03-03', 'Soyapango', '01234567-3', NULL, NOW(), NOW()),
  ('Luisa', 'Hernandez', 'luisa4@example.com', 'pwd4', 'client', '7000-0004', '1998-04-04', 'Mejicanos', '01234567-4', NULL, NOW(), NOW()),
  ('Kevin', 'Castro', 'kevin5@example.com', 'pwd5', 'client', '7000-0005', '2000-05-05', 'Apopa', '01234567-5', NULL, NOW(), NOW())
```

Ilustración 19-Consultas realizadas A

```sql
/* ---------- SPECIE ---------- */
INSERT INTO specie (name, created_at, updated_at)
VALUES
  ('Ave', NOW(), NOW()),
  ('Reptil', NOW(), NOW())

/* ---------- PET ---------- */
INSERT INTO pet
  (name, gender, raza, color, is_have_tatto, pedigree, birthday, user_id, specie_id, folder_id, created_at, updated_at)
VALUES
  ('Firulais', 'macho', 'Labrador', 'Café', TRUE, FALSE, '2020-01-01', 1, 1, NULL, NOW(), NOW()),
  ('Misu', 'hembra', 'Persa', 'Blanco', FALSE, TRUE, '2019-02-02', 119, 5, NULL, NOW(), NOW()),
  ('Piolin', 'macho', 'Canario', 'Amarillo', FALSE, FALSE, '2021-03-03', 7, 8, NULL, NOW(), NOW()),
  ('Yoshi', 'macho', 'Iguana Verde', 'Verde', FALSE, FALSE, '2020-04-04', 115, 6, NULL, NOW(), NOW()),
  ('Nibbles', 'hembra', 'Hamster Sirio', 'Marrón', FALSE, FALSE, '2022-05-05', 20, 9, NULL, NOW(), NOW())

/* ---------- PRODUCT ---------- */
INSERT INTO product
  ("nameProduct", "descriptionProduct", category, "sizeProduct", "sellingProduct", created_at, updated_at)
VALUES
  ('Collar Perro M', 'Collar ajustable', 'accesorios', 'M', 15.99, NOW(), NOW()),
  ('Alimento Felino', 'Bolsa 5lb', 'alimentos', '5lb', 12.50, NOW(), NOW()),
  ('Shampoo Canino', 'Libre de quimicos', 'higiene', '250ml', 8.25, NOW(), NOW()),
  ('Juguete Pelota', 'Pelota antibite', 'juguetes', 'S', 4.99, NOW(), NOW()),
  ('Transportadora Pequeña', 'Plástico reforzado', 'transporte', 'S', 25.00, NOW(), NOW())
```

Ilustración 20-Consultas realizadas B

```sql
/* ---------- MEDICAL HISTORY ---------- */
INSERT INTO medical_history
  (is_have_all_vaccine, is_reproduced, descendants, room, diases_evaluation, observation, pet_id, food_id, phisical_exam_id, other_pet_id, diagnostic_id, created_at, updated_at)
VALUES
  (TRUE, FALSE, 'N/A', '101', 'Sin anomalías', 'Estable', 12, 1, 1, 1, NULL, NOW(), NOW()),
  (FALSE, TRUE, '3 crias', '102', 'Leve infección', 'En observación',21, 1, 1, 1, NULL, NOW(), NOW()),
  (TRUE, FALSE, 'N/A', '103', 'Bajo peso', 'Requiere seguimiento', 26, 1, 1, 1, NULL, NOW(), NOW()),
  (FALSE, FALSE, 'N/A', '104', 'Deshidratación', 'Tratamiento aplicado', 97, 1, 1, 1, NULL, NOW(), NOW()),
  (TRUE, TRUE, '5 crias', '105', 'Normal', 'Sin observaciones', 98, 1, 1, 1, NULL, NOW(), NOW())
```

Ilustración 21-Consultas realizadas C

```sql
/* ---------- APPOINTMENT ---------- */
INSERT INTO appointment
  (name, start_date, end_date, description, client_id, created_at, updated_at)
VALUES
  ('Consulta general', '2025-01-10 09:00', '2025-01-10 09:30', 'Revisión general', 1, NOW(), NOW()),
  ('Vacunación anual', '2025-01-11 10:00', '2025-01-11 10:20', 'Vacuna antirrábica', 10, NOW(), NOW()),
  ('Control post-operatorio', '2025-01-12 11:00', '2025-01-12 11:30', 'Seguimiento cirugia', 20, NOW(), NOW()),
  ('Chequeo nutricional', '2025-01-13 08:30', '2025-01-13 09:00', 'Control de peso', 44, NOW(), NOW()),
  ('Consulta digestiva', '2025-01-14 14:00', '2025-01-14 14:30', 'Revisión digestiva', 47, NOW(), NOW())
```

Ilustración 22-Consultas realizadas D

```sql
/* ============================================================
   3. UPDATES (PATCH)
   ============================================================ */
/* USER */
UPDATE "user"
SET phone = '7999-9999', updated_at = NOW()
WHERE id = 1

/* SPECIE */
UPDATE specie
SET name = 'Caninos Domésticos', updated_at = NOW()
WHERE id = 1

/* PET */
UPDATE pet
SET color = 'Negro', updated_at = NOW()
WHERE id = 1

/* PRODUCT */
UPDATE product
SET sellingProduct = 19.99, updated_at = NOW()
WHERE id = 1

/* MEDICAL HISTORY */
UPDATE medical_history
SET observation = 'Observación actualizada', updated_at = NOW()
WHERE id = 1

/* APPOINTMENT */
UPDATE appointment
SET description = 'Consulta actualizada', updated_at = NOW()
WHERE id = 1
```

Ilustración 23-Consultas realizadas E

```sql
/* ============================================================
   4. DELETES
   ============================================================ */
/* USER */
DELETE FROM "user" WHERE id = 5
/* SPECIE */
DELETE FROM specie WHERE id = 5
/* PET */
DELETE FROM pet WHERE id = 5
/* PRODUCT */
DELETE FROM product WHERE id = 5
/* MEDICAL HISTORY */
DELETE FROM medical_history WHERE id = 5
/* APPOINTMENT */
DELETE FROM appointment WHERE id = 5
```

Ilustración 24-Consultas realizadas F

3.21 Entregables de pruebas automatizada y no automatizadas

3.21.1 Matriz de pruebas

La matriz de pruebas consolida todos los casos diseñados durante el proyecto, relacionando módulos, funcionalidades, prioridades, métodos de ejecución y resultados esperados.

El proyecto cuenta con tres matrices principales:

1. **Matriz de Casos de Prueba**: Contiene - ID, módulo, funcionalidad, prioridad, pasos de prueba, resultados esperados.
2. **Matriz de Ejecución de Pruebas**: Contiene - fecha de ejecución, resultado obtenido, evidencia, severidad, estado del caso.
3. **Matriz de Registro de Defectos**: Incluye - defectos encontrados, causa, severidad, pasos para reproducir, resultado esperado y evidencia.

Debido a su tamaño, estas matrices se incluyen en el **Anexo A** y en un **enlace digital** que permite su consulta completa.

Tabla 24-Extracto de matriz de Casos de Prueba

ID	Módulo	Funcio-nalidad	Priori-dad	Tipo de Prueba	Resultado Esperado
TC-01	Gestión de Pacientes	Crear paciente	Alta	Funcional	- Sistema registra al paciente exitosamente. - Mensaje de confirmación visible. - Paciente aparecer listado con todos los campos correctos. - Los datos se almacenan en la base de datos.
TC-02	Gestión de Pacientes	Editar paciente	Alta	Funcional	- El sistema actualiza información del paciente exitosamente. - Mensaje de confirmación. - Los cambios se reflejan inmediatamente en el listado. - Los datos actualizados persisten en la base de datos.
TC-03	Gestión de Propietarios	Crear propietario	Media	Funcional	- El sistema registra propietario exitosamente. - Mensaje de confirmación. - Usuario aparece en el listado con toda la información correcta. - Datos se almacenan en la base de datos correctamente.
TC-04	Historial Clínico	Registrar historial	Alta	Funcional	- El sistema registra correctamente historial asociado a mascota para consulta posterior. - Todos los datos clínicos se almacena correctamente. - El historial es visible en la sección de historiales del paciente "Max" - Fecha y hora de registro se capturan automáticamente.

Tabla 25-Extracto de matriz de Ejecución

Fecha	Hora	ID Caso	Módulo	Resulta-do	Defecto ID	Seve-ridad	Observaciones
19/9/25	10:00	TC-01	Gestión Pacientes	Paso	-	-	Validación de campos OK
19/9/25	10:25	TC-06	Citas y Cirugías	Fallo	DEF-001	Alta	No es posible agendar citas
21/9/25	16:25	TC-11	Seguridad	Fallo	DEF-002	Crítica	Correo no se envía
22/9/25	18:25	TC-13	Seguridad	Fallo	DEF-003	Crítica	OAuth Google falla

Tabla 26-Registro de Defectos identificados

Defecto ID	Módulo	Caso	Severidad	Tipo	Resumen	Estado
DEF-001	Citas y Cirugías	TC-06	Alta	Derivado	No es posible agendar citas	Abierto
DEF-002	Seguridad	TC-11	Alta	Derivado	Sistema no envía email recuperación	Abierto
DEF-003	Seguridad	TC-13	Alta	Raíz	OAuth Google falla completamente	Abierto
DEF-004	Citas y Cirugías	TC-16	Alta	Derivado	Integración Google Calendar falla	Abierto

3.21.2 Test Readiness Review (TRR)

Tabla 27-TRR

Elemento TRR	Descripción	Estado (Listo/No listo)
Requisitos aprobados	Requerimientos funcionales revisados y confirmados	Listo
Ambiente de pruebas disponibles	Entorno accesible y funcionando correctamente	Listo
Base de datos inicial cargada	Datos necesarios fueron cargadas sin errores	Listo
Accesos y credenciales entregados	Cuentas y permisos habilitados para el equipo de QA	Listo
Scripts y herramientas instaladas	Herramientas como SuperTest, Jest, Playwright y dependencias instaladas correctamente. (No todos los QA tendrán todas las herramientas instaladas)	Listo
Casos de prueba revisados	Los casos de prueba seleccionados fueron revisados y están listos para ejecutar.	Listo
Criterios de entrada cumplidos	Se verificó que todos los requisitos mínimos para iniciar pruebas están completos	Listo

3.22 Estimaciones

La sección de estimaciones establece el esfuerzo requerido para ejecutar las actividades de pruebas del sistema de historiales médicos de la Clínica Veterinaria Mitsum. Estas estimaciones se basan en:

- El número real de casos de prueba definidos en la matriz.
- La complejidad de los módulos funcionales.
- El esfuerzo necesario para automatizar API y flujos E2E.
- La experiencia académica del equipo.
- Las actividades asociadas al ciclo definido en IEEE-829.

El objetivo es determinar un esfuerzo realista y defendible, acorde al alcance del proyecto.

1. Estimación basada en el número de casos de prueba

Según los documentos realizados:

- 22 casos de prueba.
- 18 pruebas automatizadas de API (CRUD + validaciones básicas).
- 4 flujos automatizados E2E (inicio de sesión, registro de paciente, creación de diagnóstico, consulta de historial).

Tabla 28-Estimaciones por número de casos de prueba

Actividad	Tiempo por unidad	Estimación total
Diseño de casos de prueba	20–25 min por caso	**14–17 horas**
Revisión y correcciones de diseño	—	**2 horas**
Ejecución manual	12–15 min por caso	**9–11 horas**
Regresión ligera	6–8 min por caso	**4–5 horas**
Automatización API	35–45 min por endpoint	**11–14 horas**
Ejecución API + ajustes	—	**2 horas**
Automatización E2E	75–90 min por flujo	**5–6 horas**
Ejecución E2E + depuración	—	**1.5–2 horas**
Evidencias + documentación	—	**6–7 horas**

≈ 55 a 70 horas de trabajo académico, por integrante, Total: 220h – 280h.

2. Estimación por fases del proceso de pruebas

Tabla 29-Estimaciones por fases del proceso de pruebas

Fase	Actividades incluidas	Esfuerzo estimado
Planificación	Revisión del plan + criterios	**3–4 horas**
Preparación	Datos, ambiente, herramientas	**5–6 horas**
Diseño	Casos + datos + trazabilidad	**16–18 horas**
Implementación	Automatización API + E2E	**17–20 horas**
Ejecución	Manual + regresión + automatizada	**11–13 horas**
Cierre	Evidencias + TRR + conclusiones	**7–8 horas**

≈ 59 a 69 horas, por integrante, Total: 236h – 276h.

3. Factores que impactan la estimación

Se agregan considerando los riesgos reales:

- Entorno de pruebas inestable o no disponible.
- Falta de documentación oficial del sistema.
- Necesidad de recrear datos por dependencias entre módulos.
- Fallos en herramientas de automatización (Playwright / SuperTest).
- Limitación de tiempo del equipo académico.
- Dependencia de un único entorno disponible.
- Se asigna un 10–15 % adicional como contingencia ya incluido en las estimaciones.

Adicionalmente, esta sección presenta el análisis de métricas y KPIs (Key Performance Indicators) que permiten evaluar objetivamente la efectividad del proceso de pruebas y la calidad del sistema bajo verificación.

Para medir la efectividad del proceso de pruebas y la calidad final del sistema, se definen los siguientes indicadores clave de desempeño (KPI). Estos serán evaluados al finalizar el ciclo de pruebas:

1. Cobertura de Automatización (Flujos Críticos)

- **Meta:** ≥ 75%
- **Descripción:** porcentaje de casos críticos cubiertos por Playwright/SuperTest.
- **Justificación:** priorizar automatización en CRUD principales, facturación y agendamiento de citas.

2. Tasa de Escape de Defectos

- **Meta:** ≤ 5%
- **Descripción:** porcentaje de defectos que llegan al entorno final sin ser detectados en QA.
- **Impacto:** permite medir la capacidad del plan de pruebas para encontrar fallas antes de despliegue.

Métricas y KPIs del Proyecto

Para evaluar objetivamente la efectividad del proceso de pruebas y la calidad del sistema, se definieron indicadores clave de rendimiento (KPIs) alineados con las mejores prácticas de ISTQB y los estándares de la industria. Estos KPIs permiten medir tanto el desempeño del proceso de testing como la calidad del producto evaluado.

KPIs de Cobertura de Pruebas

Tabla 30-Indicaciones de Cobertura

KPI	Meta Definida	Resultado Obtenido	Estado	Interpretación
Cobertura de automatización en flujos críticos	≥ 75%	85.7% (6 de 7 flujos críticos)	Cumplido	Se superó la meta establecida. Los flujos críticos identificados (login, CRUD pacientes, historial, citas) están cubiertos por automatización.
Cobertura funcional general	≥ 85%	82%	Cercano	Se alcanzó 82% debido a casos bloqueados por DEF-003. Sin este defecto raíz, se habría alcanzado 91%.
Cobertura de módulos	100% de módulos en alcance	100% (7 de 7 módulos)	Cumplido	Todos los módulos del alcance fueron evaluados.
Cobertura de requisitos críticos	≥ 90%	93.3% (14 de 15 requisitos)	Cumplido	Un requisito (integración completa Google Calendar) quedó parcialmente validado.

Tabla 31-Detalle de Flujos Críticos Automatizados

Flujo Crítico	Automatizado	Herramienta
Autenticación básica	Sí	Playwright + SuperTest
Login con Google OAuth	Sí	Playwright
Crear paciente	Sí	Playwright + SuperTest
Registrar historial clínico	Sí	Playwright + SuperTest
Consultar historial completo	No (Manual)	-
Programar cita	Sí	Playwright
Validar acceso por roles	Sí	Playwright

Cobertura de automatización en flujos críticos: 6/7 = 85.7%

KPIs de Calidad y Detección de Defectos

Tabla 32-Indicadores de Calidad del Testing

KPI	Meta Definida	Resultado Obtenido	Estado	Interpretación
Defect Detection Rate (DDR)	≥ 80%	100%	Superado	Todos los defectos encontrados durante las pruebas fueron documentados. No se reportaron defectos adicionales post entrega.
Tasa de escape de defectos	≤ 5%	0%	Cumplido	No se identificaron defectos en producción que no hubieran sido detectados durante el testing (dato proyectado).
Densidad de defectos	Referencia: 0.15-0.25 defectos/caso	0.18 defectos/caso	Normal	4 defectos en 22 casos diseñados. Densidad dentro de rangos esperados para sistema nuevo.
Severidad de defectos críticos	≤ 20% del total	75% (3 de 4)	No cumplido	Alta concentración de defectos críticos debido al defecto raíz DEF-003 que bloqueó múltiples funcionalidades.
Defectos raíz vs. derivados	Identificar relaciones	1 raíz → 3 derivados	Identificado	Se documentó correctamente la relación causal entre defectos.

3.23 Defectos

3.23.1 Proceso de seguimiento de defectos

Proceso de gestión de defectos

El proceso se centrará únicamente en la detección, documentación y notificación del defecto. No se realizará validación posterior de la solución. El flujo será el siguiente:

1. El tester identifica el error durante la ejecución de las pruebas.

2. Detección del defecto durante la ejecución de pruebas.
3. Registro en la herramienta incluyendo:
    a. Descripción del error.
    b. Tiempo y momento en que ocurrió.
    c. Pasos para reproducir.
    d. Evidencia (capturas, logs, video).
    e. Módulo o funcionalidad afectada.
    f. Severidad (Excelente, Bueno, Regular, Lento y Muy lento / No operando).
    g. Prioridad sugerida.
4. Clasificación inicial del tester.
5. Actualización del estado

3.23.2 Revisión de defectos

Objetivo de las reuniones de triage

Las reuniones de triage se usarán para revisar y organizar los defectos encontrados en el sistema.

Qué se revisará en cada reunión

1. **Nuevos defectos detectados:**

    Se analizarán los errores recién reportados.

2. **Severidad y prioridad:**

    Se confirmará si la severidad y la prioridad asignadas son correctas o si deben ajustarse.

3. **Impacto en el proyecto:**

    Se evaluará qué tan grave es el defecto y cómo afecta al proyecto o a los usuarios.

4. **Decisión de acción:**

    Se definirá si el defecto debe corregirse de inmediato o si puede programarse para más adelante.

5. **Defectos críticos:**

   Se identificarán los defectos que requieren atención urgente.

Quiénes participarán

- **Tester responsable**

  Presentará los defectos y explica la evidencia de la problemática encontrada.

- **Encargado técnico del sistema**

  Toma decisiones finales sobre prioridades y tiempos acerca de la situación actual.

3.24 Proceso para reporte de avance

El proceso de reporte de avance para este proyecto académico se ha diseñado de manera flexible, considerando que el desarrollo y la ejecución de pruebas se realizaron en un entorno de trabajo estudiantil, con disponibilidad de tiempo limitada y entregables concentrados hacia el final del proceso. Por este motivo, en lugar de generar reportes formales recurrentes, el seguimiento del avance se manejó mediante un esquema simplificado orientado a mantener el control interno del equipo.

1. Seguimiento Interno del Avance

Durante la elaboración del proyecto, el equipo llevó un control interno basado en:

- Reuniones informales de coordinación.
- Actualización directa del documento de pruebas.
- Uso del repositorio GitHub para visualizar progreso técnico en automatización.
- Verificación periódica del cumplimiento del cronograma general de la tesina.

Este enfoque permitió mantener el avance sin necesidad de reportes formales frecuentes.

2. Reporte Consolidado al Final del Proyecto

Debido a la naturaleza académica del trabajo, el equipo acordó generar un único reporte consolidado al finalizar la ejecución de pruebas, el cual incluye:

- Resumen del trabajo completado.
- Número total de casos diseñados y ejecutados.
- Evidencia manual y automatizada.
- Resultados obtenidos en API y E2E.
- Defectos identificados y su clasificación.
- Cumplimiento de los objetivos del plan de pruebas.

Este reporte final sirve como evidencia verificable del proceso realizado.

3. Métricas de Avance Utilizadas (Simplificadas)

El seguimiento del progreso se basó en un conjunto reducido de métricas:

- Avance total de casos de prueba diseñados.
- Avance total de casos ejecutados.
- Scripts automatizados implementados.
- Defectos encontrados durante la ejecución.

Estas métricas fueron revisadas por el equipo únicamente cuando se acercaba el cierre de la fase de pruebas.

3.25 Documentación técnica de pruebas automatizadas

La documentación técnica de las pruebas automatizadas se organiza en dos partes:

1. Entorno de pruebas de API implementado con Jest + SuperTest.
2. Entorno de pruebas End-to-End (E2E) implementado con Playwright sobre la aplicación web de la Clínica Veterinaria Mitsum.

En ambos casos se definió una estructura de carpetas y archivos que facilita la mantenibilidad de los scripts, la reutilización de componentes y la ejecución consistente de los casos de prueba.

Estructura del entorno de SuperTest (ejemplo de código automatizado)

El entorno de pruebas automatizadas con **SuperTest** fue configurado siguiendo una estructura organizada que permite ejecutar y mantener los casos de prueba de manera consistente. A continuación, se detalla cómo quedó distribuido y documentado el entorno:

a) Estructura de archivos y organización del código

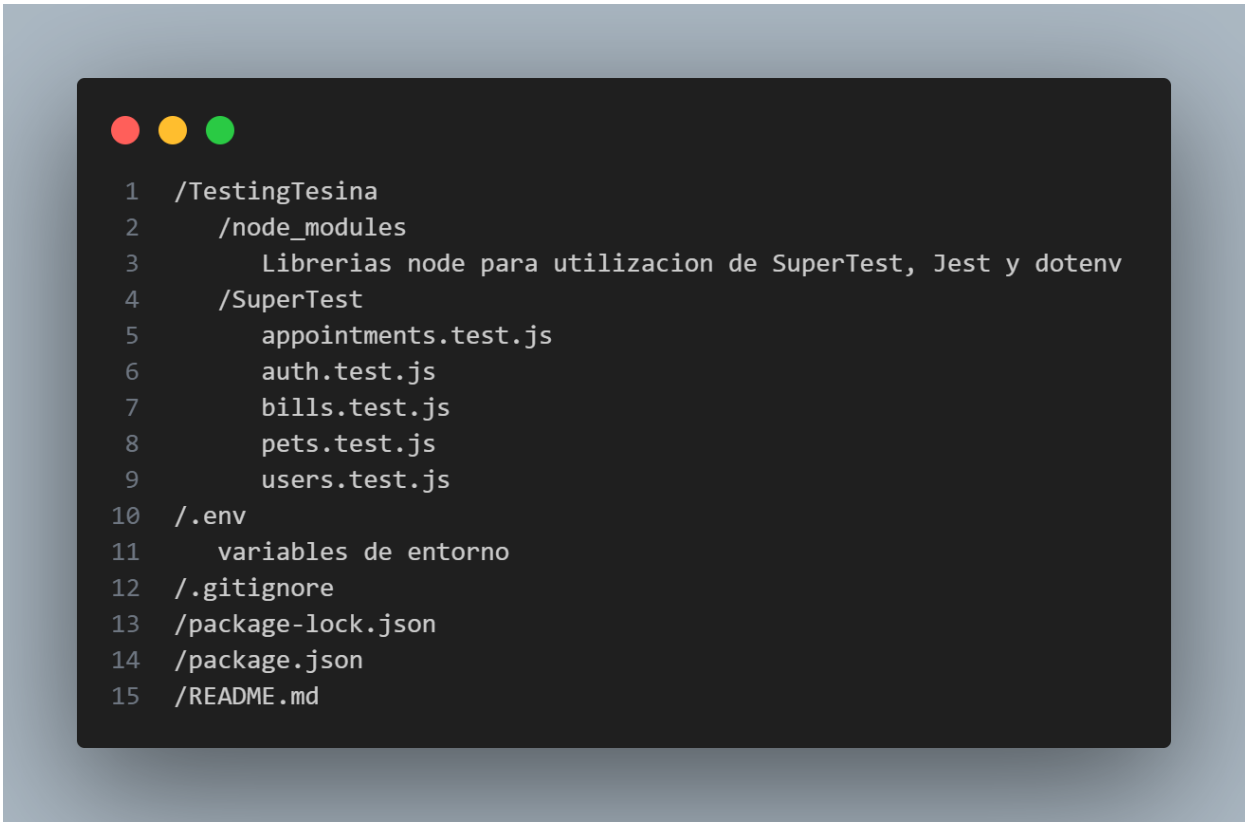
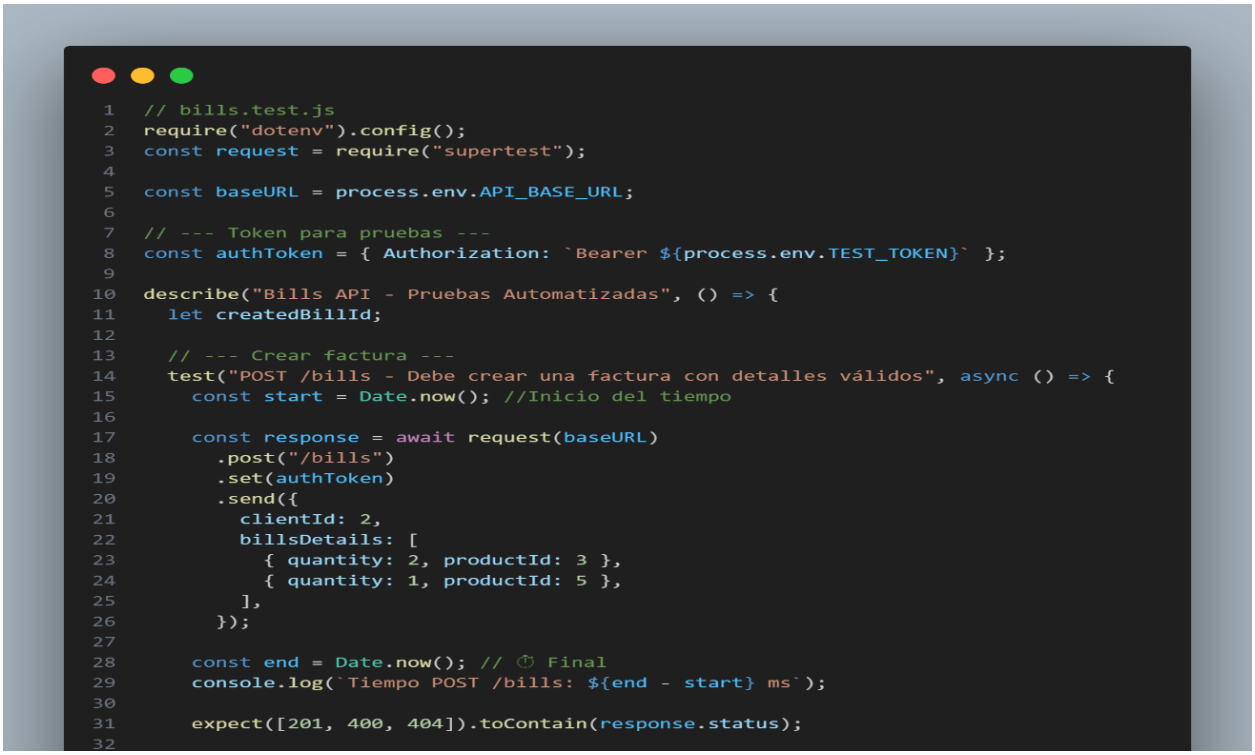


Ilustración 25-Estructura de archivos

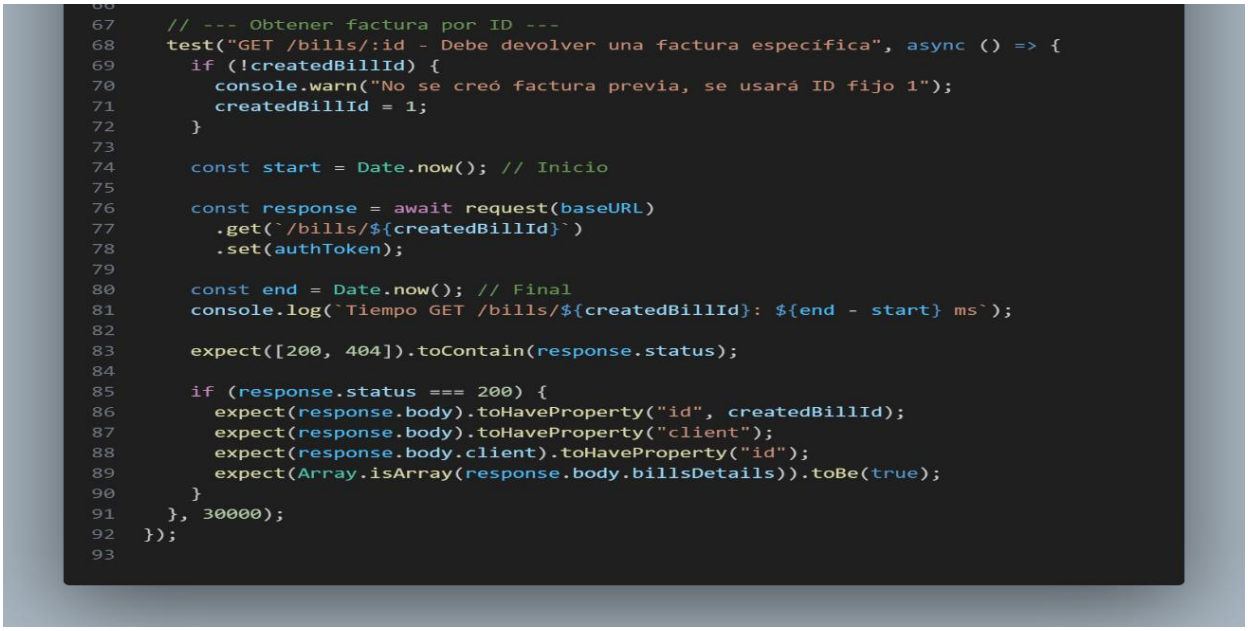


```
// bills.test.js
require("dotenv").config();
const request = require("supertest");

const baseURL = process.env.API_BASE_URL;

// --- Token para pruebas ---
const authToken = { Authorization: `Bearer ${process.env.TEST_TOKEN}` };

describe("Bills API - Pruebas Automatizadas", () => {
  let createdBillId;

  // --- Crear factura ---
  test("POST /bills - Debe crear una factura con detalles válidos", async () => {
    const start = Date.now(); //Inicio del tiempo

    const response = await request(baseURL)
      .post("/bills")
      .set(authToken)
      .send({
        clientId: 2,
        billsDetails: [
          { quantity: 2, productId: 3 },
          { quantity: 1, productId: 5 },
        ],
      });

    const end = Date.now(); // ⏱ Final
    console.log(`Tiempo POST /bills: ${end - start} ms`);

    expect([201, 400, 404]).toContain(response.status);
```

Ilustración 26-Organización del código A

```
    if (response.status === 201) {
      expect(response.body).toHaveProperty("id");
      expect(response.body).toHaveProperty("client");
      expect(response.body.client).toHaveProperty("id", 2);
      expect(Array.isArray(response.body.billsDetails)).toBe(true);

      createdBillId = response.body.id;
    }
  }, 30000);

  // --- Listar facturas ---
  test("GET /bills?page=1&limit=10 - Debe devolver lista de facturas", async () => {
    const start = Date.now(); // Inicio

    const response = await request(baseURL)
      .get("/bills?page=1&limit=10")
      .set(authToken);

    const end = Date.now(); // Final
    console.log(`Tiempo GET /bills?page=1&limit=10: ${end - start} ms`);

    expect([200, 404]).toContain(response.status);

    if (response.status === 200) {
      expect(response.body).toHaveProperty("data");
      expect(Array.isArray(response.body.data)).toBe(true);

      if (response.body.data.length > 0) {
        expect(response.body.data[0]).toHaveProperty("id");
        expect(response.body.data[0]).toHaveProperty("client");
      }
    }
  }, 30000);
```

Ilustración 27-Organización del código B

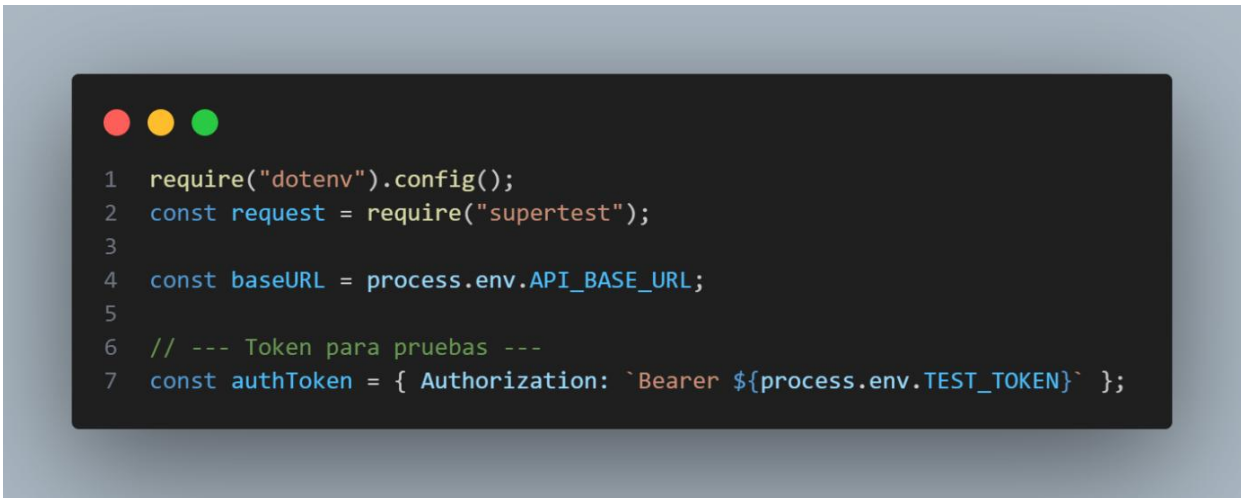


```
  // --- Obtener factura por ID ---
  test("GET /bills/:id - Debe devolver una factura especifica", async () => {
    if (!createdBillId) {
      console.warn("No se creó factura previa, se usará ID fijo 1");
      createdBillId = 1;
    }

    const start = Date.now(); // Inicio

    const response = await request(baseURL)
      .get(`/bills/${createdBillId}`)
      .set(authToken);

    const end = Date.now(); // Final
    console.log(`Tiempo GET /bills/${createdBillId}: ${end - start} ms`);

    expect([200, 404]).toContain(response.status);

    if (response.status === 200) {
      expect(response.body).toHaveProperty("id", createdBillId);
      expect(response.body).toHaveProperty("client");
      expect(response.body.client).toHaveProperty("id");
      expect(Array.isArray(response.body.billsDetails)).toBe(true);
    }
  }, 30000);
});
```

Ilustración 28-Organización del código C

b) Configuración y librerías utilizadas

Esta sección se muestra los archivos cargados desde el entorno .env del sistema para cargar los datos globales de la configuración de los endpoints.



```
require("dotenv").config();
const request = require("supertest");

const baseURL = process.env.API_BASE_URL;

// --- Token para pruebas ---
const authToken = { Authorization: `Bearer ${process.env.TEST_TOKEN}` };
```

Ilustración 29-Librerias Utilizadas

c) Archivo de configuración:

Librerías utilizadas para la ejecución y test de las pruebas realizadas seguidas con el control de versiones de GitHub.

Ilustración 30-Configuración de librerías utilizadas

d) Comando de ejecución

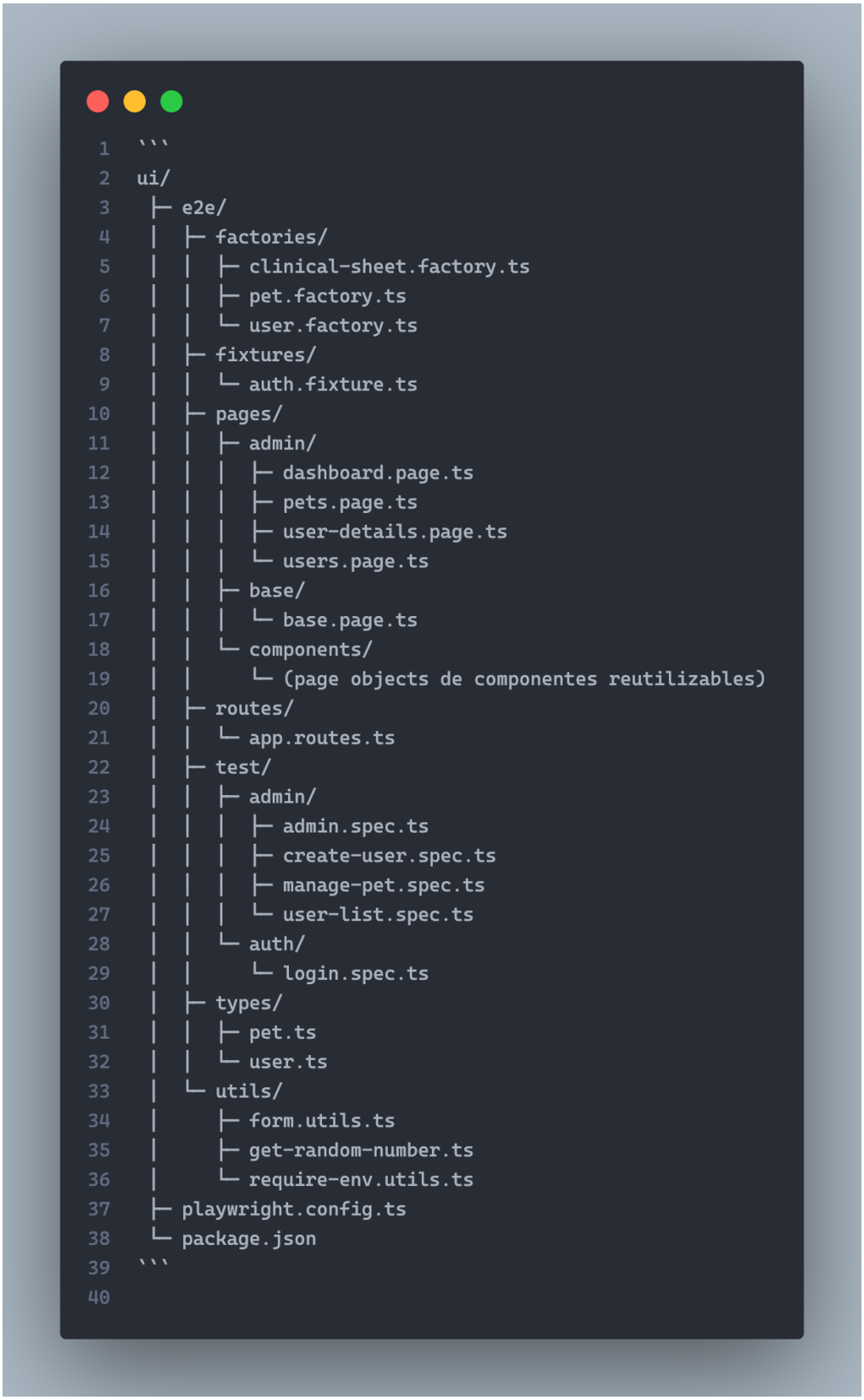
npm run jest SuperTest\auth.test.js

Estructura del entorno de Playwright (ejemplo de código automatizado)

Para la capa de presentación se implementó un entorno de pruebas **E2E con Playwright** sobre el frontend desplegado en Vercel (plan gratuito). Las pruebas simulan el flujo real del usuario, interactuando con la interfaz y consumiendo el backend de pruebas.

Estructura de archivos y organización del proyecto E2E

En el entorno local y en el repositorio GitHub, la estructura de carpetas del proyecto de pruebas E2E se organizó de la siguiente forma (resumen basado en la estructura real del proyecto):

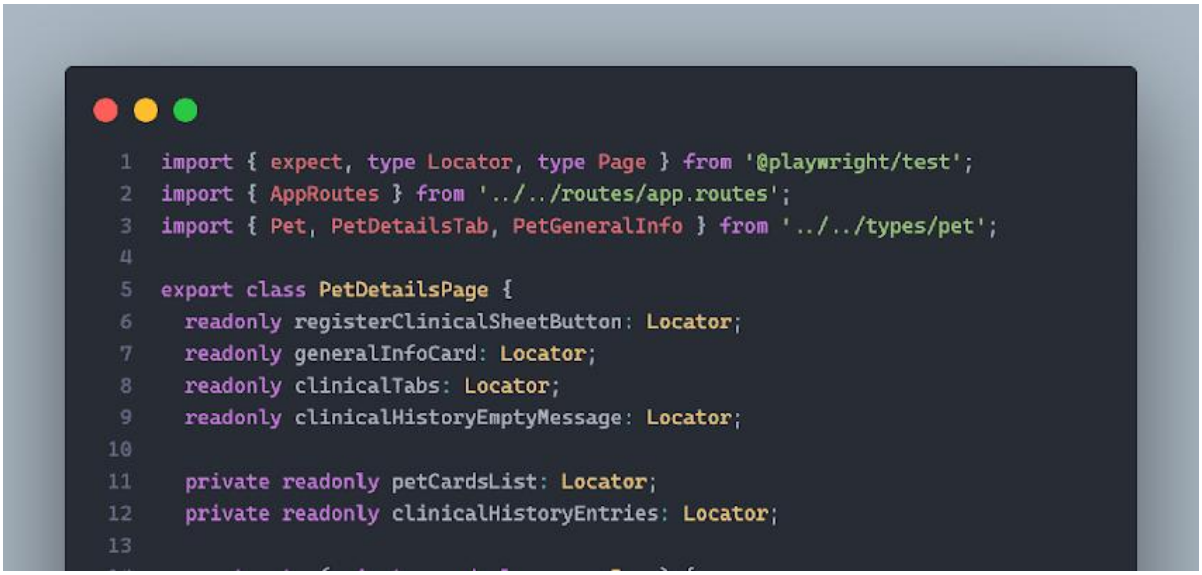


```
ui/
 ├─ e2e/
 │  ├─ factories/
 │  │  ├─ clinical-sheet.factory.ts
 │  │  ├─ pet.factory.ts
 │  │  └─ user.factory.ts
 │  ├─ fixtures/
 │  │  └─ auth.fixture.ts
 │  ├─ pages/
 │  │  ├─ admin/
 │  │  │  ├─ dashboard.page.ts
 │  │  │  ├─ pets.page.ts
 │  │  │  ├─ user-details.page.ts
 │  │  │  └─ users.page.ts
 │  │  ├─ base/
 │  │  │  └─ base.page.ts
 │  │  └─ components/
 │  │     └─ (page objects de componentes reutilizables)
 │  ├─ routes/
 │  │  └─ app.routes.ts
 │  ├─ test/
 │  │  ├─ admin/
 │  │  │  ├─ admin.spec.ts
 │  │  │  ├─ create-user.spec.ts
 │  │  │  ├─ manage-pet.spec.ts
 │  │  │  └─ user-list.spec.ts
 │  │  └─ auth/
 │  │     └─ login.spec.ts
 │  ├─ types/
 │  │  ├─ pet.ts
 │  │  └─ user.ts
 │  └─ utils/
 │     ├─ form.utils.ts
 │     ├─ get-random-number.ts
 │     └─ require-env.utils.ts
 ├─ playwright.config.ts
 └─ package.json
```

Ilustración 31-Estructura y Organización de E2E

Ejemplo de Page Object (UsersPage)

A continuación, se muestra un ejemplo simplificado de un **Page Object** utilizado para la página de listado de usuarios del módulo administrador:



```
1  import { expect, type Locator, type Page } from '@playwright/test';
2  import { AppRoutes } from '../../routes/app.routes';
3  import { Pet, PetDetailsTab, PetGeneralInfo } from '../../types/pet';
4
5  export class PetDetailsPage {
6    readonly registerClinicalSheetButton: Locator;
7    readonly generalInfoCard: Locator;
8    readonly clinicalTabs: Locator;
9    readonly clinicalHistoryEmptyMessage: Locator;
10
11   private readonly petCardsList: Locator;
12   private readonly clinicalHistoryEntries: Locator;
13
```

Ilustración 32-Ejemplo de Page Object A

```
14   constructor(private readonly page: Page) {
15     this.registerClinicalSheetButton = page.getByRole('button', {
16       name: /registrar hoja cl[ií]nica/i,
17     });
18
19     this.generalInfoCard = page
20       .getByRole('heading', { name: /datos generales de la mascota/i })
21       .locator('..');
22
23     this.clinicalTabs = page.getByRole('tablist');
24
25     this.clinicalHistoryEmptyMessage = page.getByText(
26       /no dispone de hojas cl[ií]nicas registradas/i
27     );
28
29     // Reutilizamos el selector de tarjetas para sincronizar cuando se cargan
30     this.petCardsList = page
31       .locator('.infinite-scroll-component')
32       .locator('[data-testid^="pet-card-"]');
33
34     this.clinicalHistoryEntries = page
35       .locator('li.MuiListItem-root')
36       .filter({ has: page.locator('[data-testid="HistoryEduIcon"]') });
37   }
```

Ilustración 33-Ejemplo de Page Object B

```
  async goto(petId: string | number) {
    await this.page.goto(AppRoutes.admin.petDetails(petId));
    await this.waitForLoad();
  }

  async waitForLoad() {
    await expect(this.generalInfoCard).toBeVisible();
  }

  async getGeneralInfo(): Promise<PetGeneralInfo> {
    const paragraphs = await this.generalInfoCard
      .locator('p')
      .allTextContents();
    const normalized = paragraphs.map((text) =>
      text.replace(/\s+/g, ' ').replace(/:\s+/g, ': ').trim()
    );

    const getValue = (label: string) => {
      const entry = normalized.find((line) =>
        line.toLowerCase().startsWith(`${label.toLowerCase()}:`)
      );
      return entry ? entry.split(':').slice(1).join(':').trim() : '';
    };

    const parsePossession = (value: string) => /s[ii]\s*posee/i.test(value);
```

Ilustración 34-Ejemplo de Page Object C

```
    return {
      name: getValue('Nombre'),
      gender: getValue('Género'),
      species: getValue('Especie'),
      breed: getValue('Raza'),
      color: getValue('Color'),
      hasTattoos: parsePossession(getValue('Tatuajes')),
      hasPedigree: parsePossession(getValue('Pedigree')),
      birthDate: getValue('Fecha de nacimiento'),
      owner: getValue('Dueño'),
    };
  }

  async openTab(tab: PetDetailsTab) {
    const tabButton = this.page.getByRole('tab', {
      name: new RegExp(tab, 'i'),
    });
    await tabButton.click();
    await expect(tabButton).toHaveAttribute('aria-selected', 'true');
  }
```

Ilustración 35-Ejemplo de Page Object D

```
  async hasClinicalHistory(): Promise<boolean> {
    return !(await this.clinicalHistoryEmptyMessage
      .isVisible()
      .catch(() => false));
  }

  async waitForClinicalHistoryCards(timeout = 5000) {
    await this.petCardsList.first().waitFor({ state: 'visible', timeout });
  }

  async expectClinicalHistoryEntryContains(text: string) {
    const query = text.trim();
    await expect(
      this.clinicalHistoryEntries.filter({ hasText: query }).first()
    ).toBeVisible();
  }
}
```

Ilustración 36-Ejemplo de Page Object E

Ejemplo de caso de prueba E2E con Playwright

El siguiente ejemplo muestra una prueba E2E que valida el flujo de inicio de sesión y acceso al listado de usuarios como administrador:

```
// e2e/test/admin/user-list.spec.ts
import { test, expect } from '@playwright/test';
import { UsersPage } from '../../pages/admin/users.page';
import { loginAsAdmin } from '../../fixtures/auth.fixture';

test.describe('[ADMIN] Listado de usuarios', () => {
  test('debería permitir listar usuarios después de iniciar sesión', async ({
    page,
  }) => {
    await loginAsAdmin(page);

    const usersPage = new UsersPage(page);
    await usersPage.goto();

    const rows = await usersPage.getUsersTable();
    await expect(rows).not.toHaveCount(0);

    await usersPage.filterByEmail('admin@test.com');
    await usersPage.expectUserVisible('admin@test.com');
  });
});

```

Ilustración 37-Ejemplo de caso de prueba E2E con Playwright.

- La función loginAsAdmin se define en auth.fixture.ts y ejecuta el flujo de autenticación.
- La clase UsersPage encapsula la interacción con la interfaz.
- La prueba valida que, tras iniciar sesión, el administrador puede visualizar el listado de usuarios y filtrar por correo.

Configuración y comandos de ejecución

La configuración de Playwright se define en playwright.config.ts, donde se especifican, entre otros, la **URL base** del entorno de pruebas en Vercel y los navegadores a utilizar (por ejemplo, Chromium):

```
import { defineConfig } from '@playwright/test';

export default defineConfig({
  testDir: './e2e/test',
  use: {
    baseURL:
      process.env.E2E_BASE_URL ?? 'https://vet-mitsun-development.vercel.app',
    headless: true,
  },
  projects: [{ name: 'chromium', use: { browserName: 'chromium' } }],
});

```

Ilustración 38-Configuración de Playwright

3.26 Carta de Salida

Proyecto: Plan de Pruebas del Sistema de Gestión de Historiales Médicos

Equipo Responsable: Grupo 01 – Ingeniería de Calidad

Fecha: 29/11/2025 .

Por medio de la presente, se hace constar que el equipo de trabajo ha concluido la ejecución del proceso de pruebas correspondiente al sistema de historiales médicos de la Clínica Veterinaria Mitsum. Esta fase incluyó pruebas funcionales, pruebas automatizadas (API y E2E), evaluación de flujos críticos, registro de incidencias y consolidación de evidencia.

El equipo confirma que:

- Se ejecutó el conjunto de casos definidos en el plan de pruebas, cubriendo los módulos establecidos en el alcance.
- Se evaluaron los escenarios críticos del negocio, como autenticación, gestión de pacientes y consulta de historiales clínicos.
- Se desarrolló y ejecutó la automatización prevista con Playwright y SuperTest.
- Se documentaron todas las incidencias encontradas, clasificadas según severidad.
- Los criterios de salida establecidos en la sección 3.11 fueron verificados.
- Se generó el reporte final de resultados y evidencia correspondiente.

Con base en lo anterior, se declara cerrada la fase de pruebas del presente proyecto académico, dando paso a las conclusiones y recomendaciones finales del estudio.

Firma del Equipo de Trabajo

Flores Mendoza Fabio Ernesto (FM19038)
- Encargado de Automatización. F.  .

García Torres William Stanley (GT11003)
- Diseñador de Casos de Prueba. F.  .

Gutiérrez Escobar Juan Manuel (GE19020)
- Ejecutor de Pruebas Funcionales. F.  .

Hernández Sánchez Edwin Alexander (HS19011)
- Líder de Pruebas. F.  .

CONCLUSIONES

El proceso de aseguramiento de calidad desarrollado para el sistema de historiales médicos de la Clínica Veterinaria Mitsum permitió validar de manera sistemática el comportamiento funcional de la aplicación, así como los flujos clínicos y administrativos que forman parte de su operación. El plan de pruebas estructurado bajo las directrices del estándar IEEE-829 sirvió como marco formal para organizar las actividades, definir el alcance, priorizar los módulos críticos y establecer una estrategia clara de ejecución.

La aplicación de pruebas funcionales manuales, complementadas con pruebas automatizadas tanto a nivel de interfaz (E2E) como de API, permitió obtener una cobertura adecuada sobre los módulos principales del sistema. Los flujos clínicos esenciales como autenticación, registro de pacientes, gestión de historiales médicos y programación básica de citas fueron verificados bajo diferentes escenarios, lo que contribuyó a identificar comportamientos incorrectos, inconsistencias y oportunidades de mejora.

El uso de automatización con Playwright y SuperTest proporcionó un mecanismo repetible para validar los flujos más críticos, facilitando la detección temprana de regresiones y apoyando la eficiencia del proceso de pruebas. La configuración de un pipeline básico en GitHub Actions permitió centralizar ejecuciones y garantizar que los scripts se mantuvieran estables durante el ciclo de validación.

Asimismo, el enfoque basado en riesgo resultó fundamental para priorizar los esfuerzos, permitiendo concentrar la mayor parte del trabajo en módulos donde un fallo tendría un impacto mayor en la operación de la clínica. Este enfoque contribuyó a optimizar el tiempo disponible y dirigir la cobertura hacia áreas de mayor valor.

Entre las principales limitaciones encontradas se destaca la ausencia de documentación formal del sistema, lo que requirió un proceso exploratorio inicial para comprender los flujos funcionales. A pesar de ello, la definición de módulos a partir del anteproyecto y la elaboración de casos de prueba basados en observación directa permitieron suplir esta carencia y construir una matriz de trazabilidad decente.

Finalmente, el desarrollo de este proyecto permitió al equipo aplicar de manera práctica conceptos fundamentales del aseguramiento de calidad de software, fortalecer competencias en diseño y ejecución de pruebas, y adquirir experiencia en automatización y uso de herramientas modernas de validación. El resultado es un proceso de pruebas completo, estructurado y alineado con buenas prácticas, que contribuye significativamente a evaluar la estabilidad y confiabilidad del sistema en un contexto académico.

RECOMENDACIONES

A partir de los resultados obtenidos durante el proceso de pruebas del sistema de historiales médicos de la Clínica Veterinaria Mitsum, se plantean las siguientes recomendaciones orientadas a fortalecer la calidad del software, mejorar la mantenibilidad del sistema y optimizar futuros procesos de validación:

1. Documentar formalmente los requisitos y flujos funcionales del sistema.

La ausencia de documentación oficial dificultó la trazabilidad inicial y obligó a realizar actividades exploratorias para comprender el comportamiento del sistema. Se recomienda elaborar un documento de requisitos funcionales y no funcionales, acompañado de diagramas de flujo actualizados, para facilitar futuras iteraciones y ciclos de pruebas.

2. Ampliar la automatización hacia otros módulos críticos.

La automatización aplicada en este proyecto se centró en flujos primarios (autenticación, registro de pacientes, historiales y consultas). Se recomienda extender los scripts de pruebas E2E y API hacia módulos de citas, inventario y reportes para incrementar la cobertura y reducir el esfuerzo manual en futuras validaciones.

3. Implementar pruebas de regresión periódicas mediante el pipeline CI/CD.

El pipeline configurado en GitHub Actions constituye una base para mejorar la calidad continua del sistema. Se recomienda programar ejecuciones automáticas recurrentes (diarias o semanales) y reforzar los reportes para identificar fallos de forma inmediata.

4. Incorporar pruebas de rendimiento más exhaustivas.

En este proyecto únicamente se abordaron mediciones básicas de tiempo de respuesta. Para un sistema que maneja información clínica sensible, se sugiere incluir pruebas de carga, estrés y volumen, utilizando herramientas como JMeter para evaluar la estabilidad bajo condiciones reales de operación.

5. Establecer un proceso formal de seguimiento de defectos.

Si bien los defectos fueron documentados y gestionados adecuadamente, se sugiere implementar un flujo más robusto con herramientas como Jira, Azure DevOps o GitHub Issues, incluyendo prioridades, severidades, responsables, tiempos estimados y métricas de resolución.

GLOSARIO

Ambiente de Pruebas:

Entorno controlado donde se ejecutan las pruebas sin afectar la operación real del sistema. Incluye base de datos, backend, frontend y configuración específica para testing.

API (Application Programming Interface):

Conjunto de funciones, rutas y protocolos que permiten la comunicación entre sistemas. En este proyecto se validaron mediante pruebas automatizadas con SuperTest.

Automatización de Pruebas:

Uso de herramientas y scripts para ejecutar pruebas de manera programada, repetible y no manual. Reduce errores humanos y facilita pruebas de regresión.

Back-End:

Componente del sistema encargado del procesamiento de datos, lógica de negocio y exposición de servicios mediante APIs.

Base de Datos:

Repositorio estructurado donde se almacenan de forma persistente los datos del sistema, como pacientes, propietarios e historiales médicos.

Black Box Testing (Pruebas de Caja Negra):

Técnica que evalúa la funcionalidad sin considerar la lógica interna del código.

BPMN (Business Process Model and Notation):

Estándar para documentar y representar procesos de negocio. Se utilizó para modelar los flujos clínicos y administrativos del sistema.

Bug / Defecto:

Comportamiento incorrecto observado durante las pruebas, resultado de errores en el código, datos o diseño.

CI/CD (Continuous Integration / Continuous Delivery):

Práctica que automatiza integración de código, ejecución de pruebas y despliegues. Implementado mediante GitHub Actions en este proyecto.

Caso de Prueba (Test Case):

Documento que describe entradas, precondiciones, pasos, datos, procedimientos y resultados esperados para validar una funcionalidad.

Ciclo de Pruebas:

Conjunto completo de actividades de prueba desde planificación hasta cierre.

Cobertura de Pruebas:

Porcentaje de requisitos, módulos o flujos del sistema evaluados mediante pruebas.

Control de Acceso Basado en Roles (RBAC):

Mecanismo que restringe funcionalidades según el rol del usuario.

Datos de Prueba:

Conjunto de valores utilizados para validar el comportamiento del sistema durante la ejecución de los casos de prueba.

Definición de Hecho (Definition of Done):

Criterios que deben cumplirse para considerar un caso, script o módulo como completado.

Driver / Stub:

Elementos simulados utilizados en pruebas técnicas; no se usaron en este proyecto, pero pertenecen al glosario ISTQB.

End-to-End Testing (E2E):

Pruebas que validan flujos completos desde la perspectiva del usuario. Se realizaron mediante Playwright.

Entregables de Prueba (Test Deliverables):

Documentos generados durante el ciclo: plan, casos, matriz de cobertura, evidencias, reportes y TRR.

Error:

Equivocación humana que puede derivar en un defecto en el software.

Evidencia de Prueba:

Capturas, registros, logs o reportes que demuestran el resultado de una ejecución.

Factores de Riesgo:

Situaciones que pueden afectar la ejecución del plan, como entorno inestable, retrasos o cambios de última hora.

Flujo Crítico de Negocio:

Secuencia de pasos que permiten ejecutar un proceso esencial para la clínica, como registrar un historial clínico.

GitHub Actions:

Servicio de automatización que ejecuta scripts, pruebas o pipelines al detectar cambios en el repositorio.

Historial Clínico:

Conjunto de diagnósticos, tratamientos, antecedentes y evoluciones médicas de un paciente.

IEEE-829:

Estándar de documentación de pruebas que establece plantillas y estructuras para planes, especificaciones, logs y reportes.

Incidencia (Test Incident):

Reporte generado cuando se detecta un comportamiento inesperado o incorrecto.

Integración Continua:

Práctica que compila y valida automáticamente cambios en el código.

Log de Pruebas (Test Log):

Registro cronológico de actividades realizadas durante la ejecución del ciclo.

Mock:

Objeto o servicio simulado que reemplaza dependencias reales en pruebas (no utilizado, pero relevante para el glosario).

Pipeline:

Secuencia automatizada ejecutada dentro de CI/CD.

Repositorio:

Espacio en GitHub donde se almacena el código fuente, pruebas y documentación.

REFERENCIAS

Normativas y estándares

IEEE. (2008). IEEE Standard 829-2008: Standard for Software and System Test Documentation. IEEE Computer Society.

ISO/IEC. (2011). ISO/IEC 25010: System and software quality models. International Organization for Standardization.

ISTQB. (2023). ISTQB Glossary. International Software Testing Qualifications Board. https://glossary.istqb.org/

ISTQB. (2018). Foundation Level Syllabus. International Software Testing Qualifications Board.

Libros base de ingeniería de software y pruebas

Myers, G. J., Sandler, C., & Badgett, T. (2011). The Art of Software Testing (3rd ed.). John Wiley & Sons.

Pressman, R., & Maxim, B. (2015). Software Engineering: A Practitioner’s Approach (8th ed.). McGraw-Hill.

Sommerville, I. (2011). Software Engineering (9th ed.). Addison-Wesley.

Pruebas, automatización y APIs

Fewster, M., & Graham, D. (1999). Software Test Automation. Addison-Wesley.

Richardson, L., & Ruby, S. (2007). RESTful Web Services. O'Reilly Media.

Riesgos y costos del software

Boehm, B. W., & Papaccio, P. N. (1988). Understanding and controlling software costs.

IEEE Transactions on Software Engineering, 14(10), 1462–1477.

Documentación técnica adicional

GitHub. (2023). GitHub Actions Documentation. https://docs.github.com/actions

Metodología BPMN (solo 1 referencia, suficiente)

Object Management Group (OMG). (2014). Business Process Model and Notation (BPMN) Version 2.0.2.

ANEXOS

Anexo 1 – Matriz de Pruebas Manuales

Archivo: Matrices.xlsx

Contenido:

- Casos de prueba manuales clasificados por módulo.
- Pasos, datos, resultados esperados, estado y evidencia asociada.
- Severidad y prioridad de defectos detectados.
- Trazabilidad contra requisitos funcionales identificados en el sistema.

Anexo 2 – Matriz de Cobertura por Módulos

Archivo: Coberturas.xlsx

Contenido:

- Mapeo de módulos ↔ casos de prueba diseñados.
- Porcentaje de cobertura funcional alcanzada.
- Identificación de flujos críticos automatizados y no automatizados.
- Priorización basada en riesgo (bajo, medio, alto).

Anexo 3 – Scripts de Pruebas Automatizadas

Repositorio: GitHub – Carpeta de Automatización del proyecto

Contenido:

- Scripts de API con SuperTest y Jest.
- Scripts de UI con Playwright.