Trabajo de Grado Gestor de Inventario de Materia Prima para Pequeñas y Medianas Empresas con Módulo de Consulta Inteligente LLM

Germán David Murillas Mondragón

Universidad del Valle — Sede Tuluá
Escuela de Ingeniería en Sistemas y Computación
Ingeniería de Sistemas
Octubre de 2026

Trabajo de Grado Gestor de Inventario de Materia Prima para Pequeñas y Medianas Empresas con Módulo de Consulta Inteligente LLM

Germán David Murillas Mondragón

Director
Héctor Fabio Ocampo Arbeláez

Universidad del Valle — Sede Tuluá
Escuela de Ingeniería en Sistemas y Computación
Ingeniería de Sistemas
Octubre de 2026

Tesis presentada por
Germán David Murillas Mondragón
Como requisito parcial para la obtención del título de
Ingeniero de Sistemas

Héctor Fabio Ocampo Arbeláez
Director

Jurado

Jurado

Agradezco al profesor Héctor Fabio Ocampo Arbeláez, director de este trabajo de grado, por su guía metodológica, sus observaciones precisas y su exigencia constante durante el desarrollo del proyecto. Su dirección fue fundamental para que este trabajo alcanzara el nivel de rigor que una tesis de ingeniería demanda.

Agradezco a la panificadora del Valle del Cauca y a su administradora, que generosamente abrieron las puertas de su bodega para que pudiera entender, desde la realidad operativa de una PYME colombiana, cuáles son los problemas que realmente necesitan solución. Sin esa experiencia de campo, Pymetory sería un ejercicio académico abstracto en lugar de una herramienta diseñada para resolver necesidades concretas.

Agradezco a la Universidad del Valle, y en particular a la Escuela de Ingeniería en Sistemas y Computación de la Sede Tuluá, por proporcionar el espacio académico y los recursos que hicieron posible este proyecto.

Finalmente, agradezco a mi familia por su apoyo incondicional durante todos estos años de formación. Este logro es también suyo.

Tabla de Contenido

1. Introducción **2**
- 1.1. Formulación del problema . . . 3
- 1.2. Descripción del problema . . . 3
  - 1.2.1. Caso de estudio: una panificadora del Valle del Cauca . . . 4
  - 1.2.2. La brecha entre las PYMEs y el software disponible . . . 4
  - 1.2.3. Qué aporta Pymetory frente a este problema . . . 5
- 1.3. Objetivos . . . 6
  - 1.3.1. Objetivo general . . . 6
  - 1.3.2. Objetivos específicos . . . 6
- 1.4. Resultados esperados . . . 7
- 1.5. Metodología . . . 7
  - 1.5.1. Tipo de Investigación . . . 7
  - 1.5.2. Enfoque Metodológico: SCRUM . . . 7
  - 1.5.3. Cómo se organizó el trabajo . . . 8
- 1.6. Información sobre los capítulos . . . 9

2. Justificación y alcance del proyecto **10**
- 2.1. Justificación . . . 10
  - 2.1.1. Justificación Académica y Profesional . . . 10
  - 2.1.2. Beneficiarios del Proyecto . . . 10
- 2.2. Alcance del Proyecto y Niveles de Analítica . . . 11
  - 2.2.1. Funcionalidades Operativas del Sistema . . . 11
  - 2.2.2. Módulo de Consulta Inteligente LLM (Aporte Diferencial) . . . 11
  - 2.2.3. Límites del Proyecto . . . 11

3. Marco referencial **12**
- 3.1. Marco Teórico y Estado del Arte . . . 12
  - 3.1.1. Marco Teórico . . . 12
  - 3.1.2. Principios de Gestión de Inventarios . . . 13
  - 3.1.3. Arquitectura RAG (Retrieval-Augmented Generation) . . . 13
  - 3.1.4. Framework Laravel y el Patrón MVC . . . 17
  - 3.1.5. React e Inertia.js: Single Page Applications sin API REST . . . 17
  - 3.1.6. Marco Tecnológico . . . 18
  - 3.1.7. Estado del Arte . . . 18
  - 3.1.8. Análisis de Soluciones Académicas . . . 19
  - 3.1.9. Tecnologías de Despliegue en la Nube . . . 19
- 3.2. Marco Conceptual . . . 19

3.2.1. Glosario . . . 19
3.2.2. Contexto del caso de estudio . . . 22

4. Metodología 23
4.1. Tipo de Investigación . . . 23
4.2. Verificación de resultados . . . 23
4.3. Enfoque Metodológico (SCRUM) . . . 23
4.4. Fases del Proyecto . . . 24
4.5. Planificación de Sprints . . . 25
4.5.1. Plan frente a ejecución . . . 25
4.6. Herramientas de Desarrollo y Colaboración . . . 26
4.7. Consideraciones Éticas . . . 26

5. Cronograma de Hitos 28

6. Presupuesto 30

7. Impacto Ambiental 31

8. Levantamiento de requerimientos 32
8.1. Caso de estudio y participante . . . 32
8.2. Diseño y ejecución de la entrevista . . . 32
8.3. Respuestas obtenidas . . . 34
8.3.1. Ingreso de materia prima y unidad de medida (P1) . . . 34
8.3.2. Reabastecimiento (P2) . . . 34
8.3.3. Reporte de consumo (P3) . . . 34
8.3.4. Conciliación entre sistema y bodega (P4) . . . 34
8.3.5. Vencimiento y rotación (P5) . . . 35
8.3.6. Ubicación de productos (P6) . . . 35
8.3.7. Consultas deseadas al sistema (P7) . . . 35
8.3.8. Acceso a la bodega (P8) . . . 35
8.3.9. Problemas en el conteo de fin de mes (P9) . . . 35
8.3.10. Recepción con novedades (P10) . . . 36
8.4. Problema evidenciado . . . 36
8.5. Requerimientos funcionales . . . 36
8.6. Requerimientos no funcionales . . . 39
8.7. Fuera del alcance . . . 39
8.8. Limitaciones del levantamiento . . . 39

9. Diseño 40
9.1. De los requerimientos a los componentes . . . 40
9.2. Arquitectura del sistema . . . 41
9.3. Selección de herramientas . . . 41
9.4. Modelo de datos . . . 43
9.4.1. Diseño del Kardex . . . 43
9.5. Diseño del módulo de consulta en lenguaje natural . . . 43
9.5.1. Proveedores de modelo de lenguaje . . . 45
9.6. Diseño de la interfaz de usuario . . . 45

10. Implementación **46**
10.1. Organización del código . . . 46
10.2. Patrones de diseño . . . 47
10.2.1. Strategy por configuración: proveedores del asistente . . . 47
10.2.2. Chain of Responsibility: control de acceso . . . 47
10.2.3. Active Record: reglas del lote en el propio modelo . . . 48
10.2.4. Transaction Script: despacho FEFO . . . 48
10.3. Proyección de reabastecimiento . . . 49
10.4. Problemas enfrentados y cómo se resolvieron . . . 49
10.5. Infraestructura . . . 51
10.6. Despliegue . . . 52

11. Pruebas **53**
11.1. Estrategia . . . 53
11.2. Pruebas unitarias y de integración . . . 54
11.3. Pruebas de extremo a extremo . . . 55
11.4. Defectos encontrados . . . 55
11.5. Verificación en producción . . . 55
11.6. Precisión del asistente . . . 56
11.6.1. Método . . . 56
11.6.2. Resultados . . . 57
11.7. Evaluación heurística . . . 57
11.8. Accesibilidad . . . 58
11.8.1. Revisión automática . . . 58
11.8.2. Revisión manual . . . 58
11.9. Pruebas con usuario . . . 59
11.10Cumplimiento del indicador del objetivo específico 4 . . . 59

12. Metodología de gestión del proyecto **60**
12.1. Enfoque de trabajo . . . 60
12.2. Tablero en GitHub Projects . . . 60
12.3. Flujo de una tarea . . . 61
12.4. Criterios de aceptación . . . 61
12.5. Definition of Ready y Definition of Done . . . 62

13. Trabajos Futuros **63**
13.1. Requerimientos del caso de estudio pendientes . . . 63
13.2. Infraestructura y despliegue . . . 63
13.3. Inteligencia artificial y RAG . . . 64
13.4. Funcionalidades de negocio . . . 64
13.5. Experiencia de usuario, seguridad e investigación . . . 64
13.6. Hoja de ruta priorizada . . . 65

14. Conclusiones **66**
14.1. Conclusiones . . . 66
14.1.1. Cumplimiento de los objetivos . . . 66
14.1.2. Reglas de negocio de la dirección . . . 67
14.1.3. Hallazgos . . . 67

14.1.4. Lecciones aprendidas 67

15.Bibliografía **68**

Anexos **71**
Diagramas UML del sistema 71
Modelo relacional 77
Diagrama de flujo del proceso FEFO 79
Manual de usuario 80

Lista de Figuras

5.1. Diagrama de Gantt con el aporte diferencial en IA resaltado. 29

9.1. Arquitectura en capas de Pymetory y flujo de una solicitud. 41

10.1. Infraestructura de producción de Pymetory. 52

12.1. Tablero del proyecto en GitHub Projects (vista *Priority board*, 25 de septiembre de 2026). 61

A.1. Diagrama de casos de uso del sistema Pymetory. El Operario registra lotes, despacha por FEFO, consulta al asistente y genera reportes; el Administrador hereda esos casos y además registra materiales, concilia, gestiona usuarios y bodegas y configura el modelo de lenguaje. 72

A.2. Diagrama de secuencia del registro de entrada de un lote. Muestra la interacción entre el Operario, el frontend React+Inertia.js, el InventoryController y MySQL, incluyendo la transacción atómica que garantiza la creación simultánea del lote y su movimiento de Kardex. 73

A.3. Diagrama de secuencia de una consulta al módulo RAG. El flujo muestra la clasificación de intención, la construcción del contexto a partir de la base de datos, el envío del prompt enriquecido al LLM y el almacenamiento de la respuesta en el historial de chat. 74

A.4. Diagrama de secuencia del despacho aplicando el algoritmo FEFO. Ilustra cómo el sistema ordena los lotes por fecha de vencimiento, descuenta del más próximo primero, marca lotes agotados como `consumed` y registra cada movimiento en el Kardex inmutable dentro de una transacción atómica. 74

A.5. Secuencia de la conciliación: el administrador registra el conteo físico y el sistema corrige el lote dejando un movimiento de ajuste en el Kardex. 75

A.6. Diagrama de componentes del sistema Pymetory. Muestra la organización modular en cuatro capas: frontend React, backend Laravel con sus controladores y middlewares, persistencia MySQL, y el módulo de inteligencia artificial con el RAG Context Builder que conecta la base de datos con los proveedores de modelos de lenguaje (Ollama local, Ollama Cloud y APIs compatibles). 76

A.7. Diagrama de clases simplificado con las cinco entidades principales del sistema: User, Bodega, Material, Lote (entidad central FEFO) y Movimiento (Kardex inmutable). Se incluye también ChatHistory para la persistencia de consultas al módulo RAG. 77

A.8. Modelo relacional de Pymetory. 78

A.9. Diagrama de actividad del proceso FEFO completo: desde el registro de un lote hasta su despacho, incluyendo monitoreo de vencimientos, alertas críticas y el algoritmo de selección por fecha de expiración. 79

Lista de tablas

1.1. Barreras para la adopción de software de inventario en PYMEs colombianas. 5
1.2. Relación entre los objetivos específicos y los resultados esperados 7
1.3. Estructura de capítulos. 9

3.1. Comparativa de estrategias de rotación de inventario. 14
3.2. Comparativa entre chat genérico con LLM y arquitectura RAG. 16
3.3. Comparativa de enfoques de frontend para aplicaciones web. 18
3.4. Tabla comparativa de soluciones de inventario. 19

4.1. Planificación de sprints del anteproyecto (casos de uso con la numeración del Anexo). . . 25
4.2. Ejecución real según el repositorio. 26
4.3. Herramientas de desarrollo y colaboración. 26

5.1. Planificación detallada por semanas, entregables y dependencias técnicas. 28

6.1. Valoración académica del esfuerzo de ingeniería. 30
6.2. Costos operativos anuales estimados. 30

8.1. Preguntas en bruto de la entrevista. 33
8.2. Preguntas técnicas depuradas. 33
8.3. Requerimientos funcionales de Pymetory. 37
8.4. Requerimientos no funcionales derivados de la entrevista. 39

9.1. Correspondencia entre requerimientos funcionales y componentes. 40
9.2. Herramientas elegidas, alternativas y justificación. 42
9.3. Tablas del dominio de Pymetory. 43
9.4. Intenciones reconocidas por el clasificador del asistente. 44

10.1. Organización del código fuente de Pymetory. 46
10.2. Problemas del desarrollo y su solución. 49

11.1. Pruebas PHPUnit por área. 54
11.2. Pruebas de extremo a extremo. 56
11.3. Precisión del asistente: resultado definitivo (corte del 10 de octubre de 2026). 57
11.4. Hallazgos de la evaluación heurística por severidad. 58
11.5. Estado del indicador del objetivo específico 4. 59

12.1. Ejemplos de criterios de aceptación y la prueba que los verifica. 62

13.1. Hoja de ruta priorizada de trabajos futuros. 65

Resumen

El presente trabajo de grado aborda el diseño y desarrollo de Pymetory, un sistema web para la gestión de inventario de materia prima orientado a pequeñas y medianas empresas (PYMEs) [1, 2] que actualmente enfrentan deficiencias en el control de insumos debido a la carencia de herramientas digitales especializadas. El proyecto parte de una entrevista de levantamiento de requisitos con la administradora de una panificadora del Valle del Cauca, cuyas necesidades reales —diferencias entre el inventario contable y el físico, conciliación manual, reabastecimiento sin datos de consumo y trazabilidad por lote exigida por la normativa sanitaria— guiaron los requerimientos del sistema junto con las cuatro reglas de negocio establecidas por la dirección del trabajo de grado.

El sistema se implementó con software de código abierto: Laravel 13 (PHP 8.3) como framework backend bajo el patrón Modelo-Vista-Controlador (MVC), React 19 con TypeScript como capa de presentación, Inertia.js 2.0 como puente de comunicación entre backend y frontend, Tailwind CSS v4 para los estilos, y MySQL 8.0 como sistema gestor de base de datos relacional. El aporte principal es un módulo de consulta inteligente basado en Large Language Models (LLM) con arquitectura RAG (Retrieval-Augmented Generation), que permite a los usuarios interactuar con el sistema mediante lenguaje natural en español colombiano y recibir respuestas construidas a partir de los datos reales del inventario, lo que reduce el riesgo de que el modelo de lenguaje invente cifras. El módulo RAG clasifica las consultas en 14 categorías de intención y recupera contexto pertinente desde las tablas de la base de datos antes de enviar el prompt al LLM; en una batería de 50 consultas verificadas contra la base de datos respondió correctamente 49 (34 antes de las correcciones).

El sistema implementa el estándar FEFO (First Expired, First Out) como regla obligatoria de despacho, un Kardex digital inmutable con trazabilidad completa de cada movimiento, control de acceso basado en roles (RBAC) con perfiles de Admin y Operario, módulo de conciliación de inventario físico contra lógico, generación de reportes en PDF, CSV y Excel, etiquetas QR para identificación de lotes, un dashboard con indicadores clave en tiempo real y una proyección de reabastecimiento con punto de reorden calculada desde el Kardex. El proyecto se gestionó con un enfoque ágil inspirado en SCRUM, adaptado a un único desarrollador, sobre un tablero de GitHub Projects. El sistema cuenta con 178 pruebas PHPUnit (177 aprobadas y una omitida porque depende de un proveedor externo) y 61 pruebas de extremo a extremo con Playwright, todas aprobadas; la evaluación heurística de Nielsen corrigió 13 de 14 hallazgos, la auditoría WCAG 2.1 AA no encontró incumplimientos automáticos, y la prueba con usuario, con una participante, obtuvo un SUS de 92,5. El sistema se desplegó en la capa gratuita de Oracle Cloud Infrastructure y está disponible en `https://app.pymetory.com`.

El proyecto se enmarca en la modalidad de Trabajo de Investigación [3]. Como resultado, se obtuvo una solución de software funcional, desplegada, verificada con pruebas automatizadas y de código abierto, que además de resolver un problema práctico, documenta la arquitectura y la evaluación de un asistente RAG sobre datos estructurados de inventario.

Palabras clave: Gestión de inventarios, sistema web, Laravel, PYMEs, FEFO, Kardex digital, RAG (Retrieval-Augmented Generation), Large Language Models, investigación aplicada, logística de bodegas.

Capítulo 1

Introducción

La gestión del inventario de materia prima es un factor determinante para la sostenibilidad operativa de las pequeñas y medianas empresas (PYMEs) [4, 1]. En particular, las bodegas encargadas del almacenamiento y distribución de insumos enfrentan diariamente desafíos como el desabastecimiento, la sobreacumulación, pérdidas por vencimiento o deterioro, y falta de trazabilidad en los movimientos de entrada y salida de mercancía. Las PYMEs colombianas, reguladas por la Ley 590 de 2000 y la Ley 1581 de 2012 en materia de protección de datos [5], requieren herramientas que no solo automaticen procesos sino que garanticen la integridad y confidencialidad de la información.

A pesar de los avances tecnológicos y la oferta de soluciones empresariales robustas, muchas empresas pequeñas continúan administrando sus inventarios de forma manual, utilizando cuadernos, hojas de cálculo o registros poco estructurados. Esto genera una alta dependencia del conocimiento tácito del personal, errores humanos frecuentes y una limitada capacidad para tomar decisiones basadas en datos históricos. Esta realidad se corroboró en la entrevista de levantamiento de requisitos con una panificadora del Valle del Cauca, donde se evidenció que el inventario contable no coincide con la bodega, que la conciliación se hace a mano una vez al mes y que los pedidos dependen de la experiencia de la administradora, lo que ya ha ocasionado faltantes de insumos.

Ante esta problemática, surge la necesidad de diseñar una solución tecnológica que no solo digitalice estos procesos, sino que también sea accesible, personalizable, de bajo costo y que pueda integrarse fácilmente a las dinámicas de operación de una bodega promedio. La propuesta del presente trabajo de grado, denominada Pymetory, consiste en el desarrollo de un sistema web para la gestión de inventario de materia prima, centrado en registrar entradas y salidas bajo el estándar FEFO (First Expired, First Out), mantener actualizado el estado del stock, garantizar la trazabilidad completa mediante un Kardex digital inmutable y generar reportes analíticos en tiempo real para la toma de decisiones gerenciales.

Pymetory se distingue de los sistemas de inventario convencionales por la integración de un módulo de consulta inteligente basado en Large Language Models (LLM) con arquitectura RAG (Retrieval-Augmented Generation). Este módulo permite que los usuarios interactúen con el sistema mediante lenguaje natural en español colombiano, realizando preguntas como “¿cuánta harina me queda?”, “¿qué está por vencer?” o “¿dónde está guardado el azúcar?”, y recibiendo respuestas basadas en los datos reales del inventario, no en información genérica. Esta capacidad busca facilitar el acceso a la información a usuarios sin conocimientos técnicos.

El sistema fue desarrollado bajo principios de accesibilidad y facilidad de uso, aprovechando un stack tecnológico moderno y de código abierto: Laravel 13 [6] como framework backend con arquitectura MVC, React 19 con TypeScript como librería de interfaz de usuario, Inertia.js 2.0 [7] como puente de comunicación sin API REST, Tailwind CSS v4 para los estilos, MySQL 8.0 como sistema gestor de base de datos, y modelos de lenguaje abiertos servidos con Ollama como motor del módulo RAG [8] (en la versión

final, gpt-oss:120b en Ollama Cloud, con respaldos en la nube y en el propio servidor). El ecosistema de modelos locales incluye Ollama [9] como servidor de inferencia y modelos de HuggingFace [10]. El proyecto se gestionó con un enfoque ágil inspirado en SCRUM, adaptado a un único desarrollador, sobre un tablero de GitHub Projects [11], y se desplegó en la capa gratuita de Oracle Cloud Infrastructure con el dominio propio `https://app.pymetory.com`.

La estructura de este documento sigue los lineamientos de la Resolución núm. 137 de la Facultad de Ingeniería de la Universidad del Valle [3] y la plantilla indicada por la dirección del trabajo.

1.1. Formulación del problema

¿Cómo desarrollar un sistema web que, además de digitalizar el control de inventarios, incorpore un asistente inteligente capaz de responder preguntas en lenguaje natural, para mejorar la calidad de la información con la que una PYME del sector panificador gestiona su bodega —conciliación, reabastecimiento y trazabilidad por lote— y apoyar decisiones basadas en datos reales, dejando atrás la dependencia del cuaderno y la memoria?

1.2. Descripción del problema

Las pequeñas y medianas empresas (PYMEs) son el motor de la economía colombiana: representan el 99,5 % de las empresas formales del país y generan cerca del 40 % del Producto Interno Bruto (PIB).[1] Sin embargo, existe una brecha importante en su nivel de digitalización. Mientras que el 63 % de las empresas medianas ya ha avanzado en su transformación digital, solo el 42 % de las microempresas ha dado ese paso.[12]

Una de las áreas más rezagadas es el control de la bodega. Muchas PYMEs todavía registran sus entradas y salidas de materia prima en cuadernos, libros de contabilidad o, en el mejor de los casos, hojas de cálculo que cada quien maneja a su manera.[1] Estos métodos manuales tienen un costo real: errores de conteo, registros duplicados o perdidos, y una incapacidad casi total de saber qué pasó con un insumo, cuándo y por qué. La situación es más grave si se considera que apenas el 40 % de las microempresas colombianas usa internet en sus operaciones diarias.[1]

Esta forma de trabajar produce tres problemas con costo directo para la empresa [13]:

- **Pérdidas por ineficiencia**: Productos que se vencen o se dañan antes de usarse (planteamiento del anteproyecto que la entrevista no confirmó en la empresa del caso), falta de insumos justo cuando más se necesitan y compras de emergencia a precios elevados porque nadie anticipó la escasez.
- **Falta de visibilidad**: El dueño o administrador no sabe en tiempo real qué tiene en su bodega. Sin esa información, planificar compras se convierte en un juego de adivinanza.
- **Decisiones a ciegas**: Sin datos históricos ni reportes confiables, las decisiones estratégicas (¿cuánto comprar?, ¿cada cuánto?, ¿a qué proveedor?) se toman por intuición y experiencia, no por evidencia.

Aunque existen sistemas de gestión de inventarios en el mercado —como Odoo, Zoho Inventory o Stockpile— la mayoría resultan inalcanzables para una PYME colombiana. Sus precios de suscripción, la complejidad de sus interfaces y la necesidad de personal técnico para operarlos los convierten en herramientas diseñadas para grandes empresas, no para panaderías, ferreterías o talleres.[1]

1.2.1. Caso de estudio: una panificadora del Valle del Cauca

Para entender el problema desde la operación real y no solo desde la literatura, se realizó un levantamiento de requisitos con una panificadora de tamaño pequeño a mediano del municipio de La Victoria (Valle del Cauca), que elabora productos de panadería a escala industrial y compra de forma continua harinas, mejoradores, levaduras, grasas, semillas, sal, azúcar y material de empaque. Por respeto a la empresa y a las personas involucradas, en este documento no se registran sus nombres.

El levantamiento consistió en una entrevista semiestructurada de aproximadamente 41 minutos con la persona a cargo de la administración de la empresa, organizada en diez preguntas en lenguaje cotidiano y un grupo de preguntas técnicas depuradas. El detalle completo de la entrevista y de los requerimientos que se derivaron de ella se presenta en el Capítulo 8.

La entrevista mostró que el problema de esta empresa no es la pérdida de materia prima por vencimiento —la rotación es rápida y, según la administradora, no se presentan mermas por caducidad— sino la calidad de la información con la que se gestiona la bodega:

1. **El inventario digital no coincide con la bodega**: El sistema contable se alimenta a partir de las facturas y del formato diario de producción, sin verificación física. La confusión entre referencias casi idénticas al digitar las facturas y los consumos que no se desglosan generan diferencias que solo aparecen en el cierre de mes.

2. **La conciliación es manual**: Una vez al mes, la administradora compara en una hoja de cálculo el conteo físico con el saldo del sistema contable y corrige las diferencias con movimientos manuales que no dejan constancia de quién los hizo ni por qué.

3. **El reabastecimiento depende de la experiencia de una persona**: No hay stock mínimo ni punto de reorden; los pedidos se deciden recorriendo la bodega. La empresa ya ha tenido faltantes de insumos que obligaron a comprarlos en el mercado local a un precio mayor.

4. **La trazabilidad por lote es una exigencia sanitaria**: Aunque no haya mermas, ante una inspección del INVIMA la empresa debe demostrar qué lotes de materia prima usó y cuándo vencían [14].

Estos problemas no son exclusivos de esta empresa: coinciden con las dificultades de digitalización que las fuentes citadas describen para las PYMEs colombianas, y son consistentes con la pertinencia de una herramienta de gestión de inventario orientada a este segmento.

1.2.2. La brecha entre las PYMEs y el software disponible

La Tabla 1.1 resume las barreras que separan a las PYMEs colombianas de las soluciones de inventario que existen hoy en el mercado.

Tabla 1.1: Barreras para la adopción de software de inventario en PYMEs colombianas.

Barrera	**Descripción**
Costo de licenciamiento	Suscripciones mensuales por usuario, un gasto recurrente difícil de justificar
Complejidad técnica	Interfaces diseñadas para grandes empresas con procesos estandarizados
Falta de contexto local	Software diseñado en otros países sin adaptación al español colombiano ni a las regulaciones locales
Curva de aprendizaje	Tiempo y costo de capacitar al personal en el uso de la herramienta
Dependencia de infraestructura	Requisitos de hardware, servidores o conectividad permanente

A estas barreras se suma que, en las versiones revisadas en julio de 2025, ninguna de esas soluciones permitía preguntar en lenguaje natural cosas como "¿qué debo despachar primero?" o "¿cuánta harina me queda?" sin tener que navegar por menús complejos o aprender filtros avanzados. El operario de bodega necesita respuestas, no capacitación en software.

1.2.3. Qué aporta Pymetory frente a este problema

Frente al panorama descrito, Pymetory se diseñó para cerrar cinco brechas concretas:

- **Kardex digital automático**: Reemplaza los registros manuales por un sistema que documenta cada movimiento automáticamente, guardando quién lo hizo, cuándo, de qué tipo fue, qué cantidad movió y por qué se realizó.
- **Despacho inteligente (FEFO)**: El sistema toma siempre primero el lote con la fecha de vencimiento más próxima, formalizando la práctica de "gastar primero lo que ya estaba en bodega" que hoy depende de la memoria del personal.
- **Consultas en español colombiano**: El usuario puede preguntar "¿cuánta harina me queda?", "¿qué está por vencer?" o "¿quién hizo el último despacho?" y recibir una respuesta basada en los datos reales de *su* inventario.
- **Alertas automáticas**: El sistema avisa cuando un lote está próximo a vencer o cuando el stock de un material cae por debajo del nivel mínimo, sin depender de que alguien se acuerde de revisar.
- **Reportes en un clic**: Informes de inventario, movimientos, FEFO, consumo y valorización en formatos PDF, CSV y Excel, que apoyan el cierre y la conciliación del inventario.

1.3. Objetivos

1.3.1. Objetivo general

Implementar un sistema web con arquitectura moderna y un módulo de consulta inteligente, que ayude a las PYMEs colombianas a optimizar el control de su bodega, reducir pérdidas y tomar mejores decisiones —dejando atrás los procesos manuales, la falta de trazabilidad y la ausencia de reportes.

1.3.2. Objetivos específicos

Cada objetivo incluye su indicador de cumplimiento para que el lector pueda verificar si se alcanzó lo propuesto.

1. **Entender las necesidades reales de una PYME.** Analizar cómo funciona la bodega de una panificadora real del Valle del Cauca, mediante entrevistas guiadas con el personal, para identificar y documentar al menos 9 casos de uso, 10 reglas de negocio y los criterios que debe cumplir el sistema.

   Indicador: Documento de requisitos (estándar IEEE 830) con 9 casos de uso validados por la empresa caso de estudio y aprobados por el director del trabajo de grado.

2. **Diseñar el plano del sistema.** Crear el modelo de la base de datos (mínimo 10 tablas), los diagramas UML de los 4 procesos más importantes (entrada, despacho, conciliación y consulta inteligente) y al menos 18 prototipos visuales (mockups) en Figma que muestren cómo se verá cada pantalla. Incluir el diseño del módulo de consulta inteligente con su clasificador de preguntas en 10 categorías.

   Indicador: Documento de diseño con modelo entidad-relación de 10 tablas, diagramas UML para 4 procesos críticos, 18 mockups en Figma aprobados por el director y especificación del motor RAG con 10 categorías de clasificación de intención.

3. **Construir el sistema por etapas.** Desarrollar los 9 módulos del sistema usando la metodología ágil SCRUM en ciclos de dos semanas. Implementar la gestión de materiales y lotes, el despacho automático por fecha de vencimiento (FEFO), el historial inmutable de movimientos (Kardex), los roles de usuario (Admin y Operario), la conciliación de inventario, los reportes en PDF y CSV, las etiquetas QR para identificar lotes y el asistente de consultas en lenguaje natural conectado a la base de datos real.

   Indicador: Sistema web funcional con 9 módulos operativos, repositorio en GitHub con al menos 50 commits y el sistema desplegado en al menos un servidor en la nube.

4. **Verificar que todo funcione correctamente.** Ejecutar un plan de pruebas con 13 conjuntos de casos que cubran: pruebas automáticas de los 9 casos de uso (con la herramienta Playwright), 7 escenarios de despacho FEFO, 50 consultas en español para medir la precisión del asistente inteligente, evaluación de usabilidad con los 10 principios de Jakob Nielsen y una auditoría de accesibilidad según el nivel AA de las pautas WCAG 2.1. El sistema debe alcanzar al menos un 95 % de pruebas exitosas.

   Indicador: Documento de resultados con 111 casos ejecutados, tasa de éxito global $\geq 95\,\%$, informe de evaluación heurística y auditoría de accesibilidad WCAG AA.

1.4. Resultados esperados

Tabla 1.2: Relación entre los objetivos específicos y los resultados esperados

Objetivo específico	Resultado esperado
1. Analizar los requerimientos reales de una PYME del sector alimentos.	Un documento de requisitos con criterios de aceptación y las conclusiones de las entrevistas con la panificadora caso de estudio.
2. Diseñar la arquitectura del sistema y la experiencia de usuario.	Un documento técnico con el modelo Entidad-Relación, diagramas UML de los procesos clave y los 18 mockups (prototipos visuales) de la interfaz de usuario.
3. Desarrollar los módulos del sistema, incluyendo el asistente de consultas inteligentes.	El código fuente completo del sistema, versionado en Git, listo para funcionar y con el módulo de consultas en lenguaje natural integrado.
4. Validar que el sistema funciona, es usable y es accesible.	Un informe de pruebas con los resultados de las pruebas funcionales, de usabilidad (heurísticas de Nielsen) y de accesibilidad (WCAG AA).

1.5. Metodología

1.5.1. Tipo de Investigación

Este proyecto es de **investigación aplicada**: toma el conocimiento de la ingeniería de sistemas y lo usa para resolver un problema real del entorno empresarial colombiano. También es **investigación proyectiva**, porque propone y construye una solución de software funcional. La evaluación aborda dos preguntas: con qué precisión responde el asistente a preguntas que no se usaron para ajustarlo, y si esa precisión depende del modelo de lenguaje. El Capítulo 11 responde la primera solo con una estimación optimista, porque los conjuntos de prueba dejaron de ser independientes, y la segunda con que la batería no detecta diferencias entre los modelos probados. El proyecto sigue los lineamientos de la Resolución núm. 137 de la Facultad de Ingeniería de la Universidad del Valle para trabajos de grado en la modalidad de Trabajo de Investigación [3].

1.5.2. Enfoque Metodológico: SCRUM

Para gestionar el desarrollo se planteó **SCRUM**, una metodología ágil que organiza el proyecto en ciclos cortos llamados *sprints*. Se eligió porque, al construir software para un usuario real, no es posible anticipar todas las necesidades desde el inicio, y los ciclos cortos permiten mostrar avances y ajustar el rumbo con frecuencia.

SCRUM define tres roles principales. El plan los adaptó de la siguiente manera:

- **Product Owner**: Rol que representa al cliente y define qué debe hacer el sistema. En Pymetory,

esta función la cumple el director del trabajo de grado (Prof. Héctor Fabio Ocampo), quien definió las reglas de negocio del proyecto y revisa los avances en las reuniones de seguimiento.

- **Equipo de Desarrollo**: El autor del proyecto (Germán David Murillas Mondragón), que construye el sistema de manera unipersonal.
- **Scrum Master**: Rol que facilita el proceso y elimina obstáculos. También lo asume el autor.

1.5.3. Cómo se organizó el trabajo

El plan dividió el proyecto en 6 sprints de dos semanas cada uno —12 semanas de desarrollo activo— precedidos por 4 semanas de levantamiento de requisitos y diseño. Cada sprint contemplaba cuatro momentos:

- **Product Backlog (lista de pendientes)**: Listado priorizado de las funcionalidades del sistema, gestionado en GitHub Projects.
- **Sprint Planning (planeación del ciclo)**: Al inicio de cada sprint se eligen las tareas del backlog para las dos semanas siguientes.
- **Sprint Review (revisión del avance)**: Al final de cada sprint se muestra al director un avance de software funcionando, para recibir retroalimentación y ajustar prioridades.
- **Sprint Retrospective (lecciones aprendidas)**: Después de la revisión se identifica qué funcionó, qué mejorar y qué acción tomar en el siguiente sprint.

El plan de sprints se detalla en el Capítulo 4. Con un solo desarrollador, en la práctica el trabajo se gestionó como un flujo continuo sobre un tablero Kanban en GitHub Projects, con criterios de aceptación y una *Definition of Done*; esa ejecución real se describe en el Capítulo 12.

1.6. Información sobre los capítulos

La Tabla 1.3 describe brevemente cada capítulo del documento e indica qué objetivo específico aborda. Los capítulos 1 a 7 parten de la propuesta aprobada en el anteproyecto, con los ajustes que se señalan en cada caso; los capítulos 8 a 14 documentan la ejecución del trabajo en el orden acordado con la dirección del trabajo de grado.

Tabla 1.3: Estructura de capítulos.

Capítulo	**Descripción**	**Objetivo específico**
Capítulo 1: Introducción	Problema, objetivos, resultados esperados y metodología.	Ninguno
Capítulo 2: Justificación y alcance	Por qué se desarrolla el proyecto y qué incluye y excluye.	Ninguno
Capítulo 3: Marco referencial	Conceptos, tecnologías y trabajos previos en los que se apoya el proyecto.	Ninguno
Capítulo 4: Metodología	Tipo de investigación, enfoque ágil y fases de trabajo.	Ninguno
Capítulos 5 a 7: Cronograma, presupuesto e impacto ambiental	Planeación, costos e impacto del proyecto.	Ninguno
Capítulo 8: Levantamiento de requerimientos	Entrevista con la empresa del caso de estudio, problema evidenciado y requerimientos funcionales y no funcionales.	1
Capítulo 9: Diseño	Arquitectura, selección de herramientas, modelo de datos, diseño del asistente y de la interfaz.	2
Capítulo 10: Implementación	Patrones de diseño en el código, problemas enfrentados, infraestructura y despliegue.	3
Capítulo 11: Pruebas	Pruebas unitarias, de integración, de extremo a extremo y con usuario, con sus resultados.	4
Capítulo 12: Metodología de gestión del proyecto	Manejo de tareas en GitHub Projects, criterios de aceptación, *Definition of Ready* y *Definition of Done*.	3
Capítulo 13: Trabajos futuros	Mejoras y líneas de continuación identificadas.	Ninguno
Capítulo 14: Conclusiones	Cumplimiento de cada objetivo con su indicador.	1 a 4
Anexos	Diagramas UML, modelo entidad relación, flujo FEFO y manual de usuario.	2 y 3

Capítulo 2

Justificación y alcance del proyecto

2.1. Justificación

Desde la ingeniería, este proyecto ataca una brecha concreta: las PYMEs necesitan controlar su inventario pero las herramientas que existen no están pensadas para ellas. La solución propuesta es un sistema web construido con software de código abierto, sin licencias de pago y operable sin personal técnico dedicado.

El sistema centraliza la información de la bodega en un solo lugar, automatiza las tareas repetitivas que hoy se hacen a mano y permite consultar datos en tiempo real desde cualquier dispositivo con internet. Así se reducen los riesgos del registro manual: errores de conteo, registros perdidos, datos desactualizados.

El módulo de consulta inteligente basado en Large Language Models (LLM) —modelos avanzados de lenguaje entrenados con grandes volúmenes de texto— representa tanto un desafío técnico como una oportunidad. Su implementación explora el uso de APIs especializadas para interpretar preguntas en lenguaje natural, extraer la información relevante de la base de datos y responder con datos reales del inventario, no con información genérica.

2.1.1. Justificación Académica y Profesional

Este proyecto se alinea con la modalidad de Trabajo de Investigación de la Escuela de Ingeniería de Sistemas y Computación. Va más allá de aplicar conocimientos técnicos: es un estudio de caso documentado sobre cómo construir software para empresas con poca experiencia digital, usando metodologías ágiles de desarrollo.

Los resultados no son solo un sistema que funciona. Son también un modelo de arquitectura documentado y un conjunto de funcionalidades mínimas viables (MVP) que otros estudiantes y desarrolladores pueden tomar como punto de partida para crear soluciones similares para este sector empresarial, que es vital para la economía colombiana.

2.1.2. Beneficiarios del Proyecto

- **Pequeñas y medianas empresas (PYMEs):** Podrían obtener una herramienta de bajo costo para digitalizar y optimizar uno de sus procesos logísticos más críticos.
- **Responsables de bodega y operarios:** Contarán con una interfaz sencilla para registrar y consultar información de manera rápida y precisa, reduciendo la carga de trabajo manual y los errores.
- **Administradores y dueños de negocio:** Tendrán acceso a reportes y alertas que facilitan una toma de decisiones informada y estratégica.

- **Las comunidades académica y de desarrollo:** El proyecto y su documentación sirven como caso de estudio y como modelo replicable de un gestor de inventario con tecnologías modernas y de un asistente conversacional que facilita el acceso a la información a usuarios no técnicos.

2.2. Alcance del Proyecto y Niveles de Analítica

El sistema propuesto va más allá del simple registro digital. Integra un asistente basado en Inteligencia Artificial y una proyección estadística del consumo: además de mostrar lo que pasó, estima lo que puede pasar.

2.2.1. Funcionalidades Operativas del Sistema

El sistema integra por diseño las siguientes capacidades de gestión:

- **Actualización del stock:** Cálculo automático y en tiempo real del inventario disponible por cada insumo registrado.
- **Generación de alertas:** Notificaciones visuales que aparecen cuando un producto alcanza su nivel mínimo, antes de que se agote.
- **Historial de movimientos (Kardex):** Registro cronológico detallado de todas las entradas y salidas, para garantizar que se pueda reconstruir el historial completo de cualquier material.
- **Reportes automatizados:** Generación de informes analíticos exportables en formatos PDF, Excel y CSV, listos para compartir con contabilidad o gerencia.
- **Interfaz web responsive:** Diseño que se adapta automáticamente a computadores, tablets y celulares, para poder consultar el inventario desde cualquier lugar.

2.2.2. Módulo de Consulta Inteligente LLM (Aporte Diferencial)

El aporte diferencial es un asistente con un modelo de lenguaje de gran escala (LLM) que atiende dos tipos de preguntas:

1. **Analítica Descriptiva y Diagnóstica:** El usuario puede preguntar en español colombiano —sin saber SQL, sin aprender filtros— cosas como "¿cuánta harina me queda?", "¿qué movimientos hubo esta semana?" o "¿por qué bajó tanto el stock de azúcar?".
2. **Analítica Predictiva (Valor Agregado):** El módulo procesa el historial del Kardex para hacer *forecasting* —previsión de la demanda futura. Esto permite predecir fechas probables de desabastecimiento (*stock-out*) y sugerir cuándo reabastecer, basándose en el cálculo dinámico del Punto de Reorden (ROP), el nivel de inventario en el que conviene hacer un nuevo pedido. Es una proyección estadística (promedio móvil de cuatro semanas, Sección 10.3), no un modelo de IA, y no se validó contra el consumo real; el asistente solo la consulta y la redacta.

2.2.3. Límites del Proyecto

Para que el proyecto sea viable dentro del tiempo académico disponible, se definen límites claros: el sistema gestiona una sola bodega física, los productos se registran manualmente (sin escáneres), no incluye facturación automatizada ni integración contable con terceros, y el LLM utiliza una arquitectura pre-entrenada accesible mediante API. Todo lo demás está dentro del alcance. El sistema construido amplía dos: maneja varias bodegas (RF-16) y lee con la cámara las etiquetas QR de cada lote (RF-13).

Capítulo 3

Marco referencial

Este capítulo explica los conceptos, teorías y definiciones necesarias para entender cómo se construyó Pymetory. Está organizado en tres bloques.

El **Marco Teórico** expone los fundamentos que sustentan el sistema: desde los principios de gestión de inventarios y las estrategias de rotación de productos (FEFO, FIFO, LIFO) hasta la arquitectura RAG que hace posible el módulo de consulta inteligente, pasando por las tecnologías de desarrollo web que se usaron (Laravel con patrón MVC, React con Inertia.js).

El **Estado del Arte** analiza las soluciones comerciales y académicas que ya existen, identificando qué hacen bien y qué brechas deja sin cubrir este proyecto. Esto permite ubicar qué vacíos concretos busca cubrir Pymetory.

El **Marco Conceptual** presenta un glosario de términos técnicos y de negocio, contextualizados, cuando aplica, en el caso de estudio de una panificadora del Valle del Cauca. Si en algún momento aparece un acrónimo o concepto que no queda claro, este glosario es el lugar para buscarlo.

3.1. Marco Teórico y Estado del Arte

3.1.1. Marco Teórico

Un sistema de gestión de inventario es más que una base de datos donde se anota qué entra y qué sale. Es una herramienta de optimización logística. Según Ballou [13], gestionar bien un inventario significa encontrar el equilibrio entre dos fuerzas opuestas: mantener suficiente producto para no parar la operación y no tener tanto que el dinero quede congelado en estanterías.

El sistema propuesto busca ayudar a las PYMEs a acercarse a ese equilibrio, dándoles la visibilidad que hoy no tienen: saber en tiempo real qué hay, qué está por acabarse y qué está por vencerse. Sin esa información, cualquier decisión de compra es a ciegas.

El módulo de consulta inteligente se apoya en el Procesamiento de Lenguaje Natural (PLN), la rama de la inteligencia artificial que enseña a las computadoras a entender cómo hablamos los humanos —no solo palabras sueltas, sino intenciones y contexto. Los Large Language Models (LLMs) son la evolución más reciente del PLN: modelos entrenados con cantidades masivas de texto, capaces de entender una pregunta en español colombiano y responder con información pertinente [15]. En lugar de obligar al usuario a aprender filtros y menús, el LLM permite preguntar como si se hablara con un compañero.

La usabilidad —qué tan fácil es usar un sistema sin equivocarse— es otro pilar del diseño. Nos basamos en los 10 principios heurísticos de Jakob Nielsen [16], que son reglas prácticas para diseñar interfaces que no confundan al usuario: mostrar siempre en qué estado está el sistema, hablar el lenguaje del usuario (no el del programador), prevenir errores antes de que ocurran y ofrecer salidas claras cuando algo sale mal.

La accesibilidad web complementa la usabilidad. Se adoptaron las Pautas de Accesibilidad para el Contenido Web (WCAG) [17], buscando cumplir el nivel AA: el estándar recomendado para que una página web pueda ser usada por personas con dificultades visuales, motoras o cognitivas. Esto incluye contraste suficiente, navegación sin mouse y etiquetas que los lectores de pantalla puedan interpretar.

3.1.2. Principios de Gestión de Inventarios

La Importancia de la Trazabilidad en la Logística

La trazabilidad es la capacidad de seguirle el rastro a un producto desde que entra a la bodega hasta que sale o se consume. Es como la hoja de vida de cada material: quién lo recibió, cuándo, dónde se guardó, cuándo se usó, quién lo despachó. Sin trazabilidad, cuando algo falla —un lote vencido, una merma inexplicable— es imposible saber qué pasó. Implementar un Kardex digital que garantice trazabilidad completa es una de las cuatro reglas inmutables de este proyecto.

Modelos de Control de Inventario: EOQ y Punto de Reorden (ROP)

Dos conceptos clásicos de la teoría de inventarios guían el diseño del sistema. La Cantidad Económica de Pedido (EOQ, por *Economic Order Quantity*) responde a la pregunta "¿cuánto debo pedir cada vez para minimizar costos?". El Punto de Reorden (ROP, por *Reorder Point*) responde a "¿en qué momento debo hacer el pedido para no quedarme sin stock?". En su forma básica, $ROP = d \times L + SS$, donde d es la demanda diaria, L el tiempo de entrega y SS el inventario de seguridad [13].

Pymetory no calcula el EOQ, pero conserva en el Kardex los consumos y costos que se necesitarían para hacerlo. El ROP sí se calcula: con el consumo diario promedio de cada insumo y los días de entrega de su proveedor, el sistema indica en qué nivel de existencia conviene pedir y hasta qué fecha alcanza lo que hay (Sección 10.3).

Métodos de Valoración de Inventario: PEPS y UEPS

Valorar un inventario es ponerle precio a lo que hay guardado. Los métodos más comunes son PEPS (Primeras en Entrar, Primeras en Salir) y UEPS (Últimas en Entrar, Primeras en Salir). PEPS asume que lo que se compró primero se consume primero, lo cual refleja el flujo físico real en la mayoría de los negocios. Pymetory valora cada lote por su costo de compra; el método contable de la empresa se mantiene en su sistema contable.

Estrategias de Rotación de Inventario: FEFO, FIFO y LIFO

Rotar el inventario es decidir en qué orden se despachan los productos. Hay tres estrategias principales. La Tabla 3.1 las compara.

¿Por qué FEFO para este proyecto? FEFO se acordó con el director del trabajo de grado como regla de negocio no negociable del sistema. La entrevista con la empresa del caso de estudio (Capítulo 8) mostró que allí existe una rotación informal por orden de llegada —la administradora instruye al personal para que se consuma primero lo que ya estaba en bodega— y que la trazabilidad de lotes y fechas de vencimiento es una exigencia sanitaria ante entes como el INVIMA [14]. Formalizar FEFO en el software elimina la dependencia de esa instrucción verbal.

3.1.3. Arquitectura RAG (Retrieval-Augmented Generation)

El módulo de consulta inteligente de Pymetory se construye sobre un paradigma llamado RAG — Generación Aumentada por Recuperación, por sus siglas en inglés. Es una arquitectura que combina dos

Tabla 3.1: Comparativa de estrategias de rotación de inventario.

Criterio	**FEFO (First Expired, First Out)**	**FIFO (First In, First Out)**	**LIFO (Last In, First Out)**
Definición	El producto con la fecha de vencimiento más próxima sale primero	El producto que ingresó primero al inventario sale primero	El producto que ingresó más recientemente sale primero
Criterio de ordenamiento	Fecha de vencimiento o caducidad	Fecha de ingreso al almacén	Fecha de ingreso al almacén (inversa)
Aplicación principal	Industria alimenticia, farmacéutica, química (productos perecederos)	Manufactura general, productos no perecederos, retail	Materiales no perecederos apilables (minerales, granos a granel)
Ventaja principal	Minimiza las pérdidas por vencimiento y deterioro	Refleja el flujo físico natural en la mayoría de las bodegas	Reduce costos de manipulación cuando los productos se apilan
Desventaja principal	Requiere registro y seguimiento de fechas de vencimiento por lote	No considera la caducidad; puede resultar en pérdidas por vencimiento si hay productos perecederos	Puede resultar en obsolescencia de los productos más antiguos
Complejidad de implementación	Alta: requiere identificar cada lote con su fecha de vencimiento y ordenar por ese campo	Media: requiere registrar la fecha de ingreso de cada lote y ordenar cronológicamente	Baja: el último que entra es el primero que sale, generalmente por disposición física
Ejemplo en panificadora	Despachar primero la harina que vence en 5 días, aunque haya ingresado después que un lote que vence en 30 días	Despachar la harina que se compró primero, sin importar que el lote más nuevo venza antes	Despachar la harina que acaba de llegar, dejando la más antigua al fondo de la estantería (no recomendado)

cosas: la capacidad de un modelo de lenguaje para redactar respuestas fluidas y la capacidad de un sistema de bases de datos para encontrar información precisa.

El Problema de las Alucinaciones en los LLM

Los Large Language Models como GPT-4, Claude o Llama se entrenan con cantidades masivas de texto y desarrollan la capacidad de generar respuestas coherentes. Pero tienen una limitación crítica: su conocimiento está congelado en el momento del entrenamiento. No saben qué entró o salió de una bodega hoy. Cuando se les pregunta sobre datos que desconocen, estos modelos tienden a "alucinar": generan respuestas que suenan convincentes pero que son factualmente incorrectas [15].

En un sistema de inventario, una alucinación puede ser costosa. Si un usuario pregunta "¿cuánta harina me queda?" y el modelo responde "50 kg" cuando en realidad hay 5 kg, la empresa podría dejar de comprar y sufrir un desabastecimiento. Para reducir este riesgo se necesita un mecanismo que ancle las respuestas del modelo a datos reales. Esa es exactamente la función del patrón RAG.

Componentes de la Arquitectura RAG

La arquitectura RAG, introducida por Lewis et al. (2020) [8], opera en tres etapas cada vez que un usuario hace una consulta:

1. **Recuperación (Retrieval):** El sistema consulta la base de datos MySQL de Pymetory —las tablas de materiales, lotes, bodegas, movimientos y configuraciones— y extrae únicamente la información pertinente a la pregunta. Esta etapa es crítica: de ella depende que el modelo reciba datos relevantes, no ruido.

2. **Aumento (Augmentation):** Los datos recuperados se organizan en un formato textual que el modelo puede procesar y se inyectan en el prompt —la instrucción que se le da al modelo— junto con indicaciones precisas de cómo responder. La clave está en ser selectivo: no se envía toda la base de datos, sino solo lo relevante.

3. **Generación (Generation):** Con el contexto real inyectado, el modelo genera una respuesta. Como recibe los datos reales, el riesgo de que invente cifras baja, aunque no desaparece. Si el modelo no encuentra los datos que necesita, se le instruye para responder que no tiene esa información, en lugar de inventarla.

Ventajas del Enfoque RAG sobre un Chat Convencional

La Tabla 3.2 contrasta implementar un chat genérico con un LLM versus implementar la arquitectura RAG.

El Mecanismo de Degradación Elegante (Fallback)

Una decisión de diseño fundamental fue asumir que el servicio de IA externo puede fallar. Depender de una API de terceros introduce un riesgo: si la clave no está configurada, si hay un corte de red, o si se agotan los créditos, el módulo de consulta inteligente quedaría inoperativo justo cuando se necesita.

Para mitigar esto, se implementó un mecanismo de **degradación elegante** (*graceful degradation*). Cuando la llamada al LLM falla, el sistema responde en "modo texto": muestra los datos recuperados de la base de datos en un formato legible, sin la capa de generación de lenguaje natural. La respuesta es menos conversacional, pero el usuario obtiene la información. Este principio —la información del inventario debe estar disponible incluso sin IA— se acordó con el director del proyecto.

Tabla 3.2: Comparativa entre chat genérico con LLM y arquitectura RAG.

Criterio	**Chat LLM Convencional**	**Arquitectura RAG (Pymetory)**
Fuente de datos	Conocimiento del entrenamiento del modelo (estático)	Base de datos del inventario en tiempo real
Actualización	No se actualiza; el modelo no sabe qué entró o salió hoy	Cada consulta dispara una lectura de la base de datos
Riesgo de alucinación	Alto cuando se pregunta por datos específicos no incluidos en el entrenamiento	Bajo, aunque no nulo; el contexto inyectado orienta al modelo a responder con los datos proporcionados
Personalización	Genérica; responde sobre "inventarios en general"	Específica; responde sobre el inventario de la PYME concreta
Auditabilidad	Baja; no hay trazabilidad entre la respuesta y una fuente de datos verificable	Media; se registran la pregunta, la respuesta y el modelo que la generó
Dependencia externa	Total; sin LLM no hay respuesta	Parcial; con mecanismo de fallback, el sistema puede responder con los datos crudos de la base de datos

3.1.4. Framework Laravel y el Patrón MVC

Laravel es un framework de código abierto para desarrollar aplicaciones web en PHP, creado por Taylor Otwell en 2011. Se eligió por cuatro razones: su madurez, su licencia MIT que permite uso comercial gratuito, su ecosistema de herramientas integradas (ORM, migraciones, middleware) y su adopción del patrón MVC.

El Patrón Modelo-Vista-Controlador (MVC)

El patrón MVC, formalizado por Trygve Reenskaug en 1979, propone separar el código en tres capas que no se mezclan:

- **Modelo (Model):** Representa los datos y la lógica de negocio. En Laravel se implementa con Eloquent, un ORM (Object-Relational Mapper) que permite manipular la base de datos usando objetos de PHP, no consultas SQL escritas a mano. Por ejemplo, el modelo `Lote` contiene el método `despacharFefo()`, por el que pasa todo despacho en orden de vencimiento.
- **Vista (View):** Es lo que el usuario ve y toca. En Pymetory, las vistas son componentes React 19 conectados al backend mediante Inertia.js 2.0, que actúa como puente. Esto permite construir una interfaz moderna (Single Page Application) sin la complejidad de una API REST separada.
- **Controlador (Controller):** Recibe las solicitudes del usuario, aplica la lógica de negocio y retorna una respuesta. Cada módulo tiene su controlador: `InventoryController` para materiales y lotes, `ConsumptionController` para despachos, `ChatLLMController` para consultas inteligentes.

La separación en capas trae beneficios prácticos: se puede rediseñar la interfaz sin tocar la lógica de negocio y probar cada capa por separado.

3.1.5. React e Inertia.js: Single Page Applications sin API REST

Usar React 19 con Inertia.js 2.0 como capa de presentación fue una decisión arquitectónica que redujo significativamente la complejidad del sistema.

El Problema de las SPAs Tradicionales

En una SPA (Single Page Application) convencional, el frontend se comunica con el backend mediante una API REST. Esto implica definir y documentar endpoints, implementar serialización JSON en ambos lados, manejar estados de carga y error en cada componente, construir un cliente HTTP con interceptores de autenticación, y mantener sincronizados los tipos de TypeScript con los de PHP. Para un proyecto con un cronograma académico y recursos unipersonales, esta complejidad no se justificaba.

Inertia.js como Puente entre Backend y Frontend

Inertia.js resuelve este problema con un principio simple: en lugar de que el frontend haga llamadas HTTP a endpoints JSON, el servidor de Laravel comparte datos directamente con los componentes React como props. El flujo es:

1. El usuario navega a una página (ej. el dashboard).
2. Laravel procesa la solicitud, consulta la base de datos y prepara los datos.
3. En lugar de devolver HTML, el controlador retorna `Inertia::render('Dashboard', [props])`.

4. Inertia.js, en el navegador, renderiza el componente React `Dashboard.tsx` con esos datos.

5. Las navegaciones siguientes no recargan la página completa; solo se intercambia el componente y los datos.

Este enfoque ofrece la fluidez de una SPA sin los costos de una API REST: sin endpoints adicionales, sin serialización manual, sin duplicación de tipos.

La interfaz se construyó con React 19 y TypeScript, que agrega tipos a JavaScript: documentan la forma de los datos y permiten que el editor señale errores como pasar un texto donde se esperaba un número.

Comparativa de Enfoques de Frontend

Tabla 3.3: Comparativa de enfoques de frontend para aplicaciones web.

Criterio	**Laravel Blade (MPA)**	**React + API REST (SPA)**	**React + Inertia.js (Adoptado)**
Experiencia de usuario	Recargas de página completas en cada navegación	Navegación fluida, sin recargas	Navegación fluida, sin recargas
Complejidad de desarrollo	Baja: todo en Laravel	Alta: dos proyectos separados, API REST, serialización, CORS	Media: backend y frontend integrados, sin API REST
Renderizado	En el servidor (SSR)	En el cliente (CSR)	Híbrido: primer render en servidor, navegaciones en cliente
Manejo de estado	No aplica (sin estado en cliente)	Complejo: loading, error, cache	Simplificado: los datos llegan como props
SEO	Óptimo (HTML completo)	Problemático (requiere SSR adicional)	Bueno (primer render con datos)
Curva de aprendizaje	Baja (PHP + HTML)	Alta (API REST, React, estado)	Media (React + props de servidor)

3.1.6. Marco Tecnológico

El stack completo —Laravel, PHP, MySQL, React, Inertia.js y Tailwind CSS—, las alternativas consideradas y la razón de cada elección se presentan en la Sección 9.3.

3.1.7. Estado del Arte

Al analizar las soluciones existentes se observa un patrón. Plataformas como Odoo [18], Zoho Inventory [19] y Stockpile [20] son robustas en gestión de inventarios, pero, en las versiones revisadas (julio de 2025), ninguna permitía interactuar con el sistema mediante lenguaje natural. El usuario debe aprender la lógica de la herramienta, en lugar de que la herramienta se adapte a la lógica del usuario. Desde entonces, Zoho

Inventory incorporó un asistente (Zia) que consulta existencias y órdenes en lenguaje natural [21], de modo que la diferencia de este trabajo no es la conversación en sí, sino su costo y su foco en FEFO y trazabilidad por lote para una PYME pequeña. La propuesta busca cerrar esa brecha de costo y de foco.

Tabla 3.4: Tabla comparativa de soluciones de inventario.

Plataforma	**Ventajas**	**Desventajas para PYMEs**
Odoo [18]	Escalable, personalizable, comunidad activa.	Configuración compleja, curva de aprendizaje alta.
Zoho Inventory [19]	Interfaz amigable, integración con otras apps.	Plan gratuito muy limitado, costos recurrentes.
Stockpile [20]	Gratuito, sencillo de usar.	Funcionalidades muy básicas, sin reportes avanzados, sin consulta inteligente.
Solución Propuesta	Código abierto, personalizable, enfocado en PYMEs, módulo LLM incluido.	Alcance limitado a inventario, requiere instalación inicial.

3.1.8. Análisis de Soluciones Académicas

Se revisaron trabajos de grado en repositorios universitarios colombianos (no fue una revisión sistemática) y se encontró un interés recurrente en resolver la gestión de inventarios para PYMEs. Bedoya (2022) en la UNAD [22] y Bautista y Hernández (2022) en la UPB [23] proponen aplicaciones para administrar inventarios. A nivel de maestría, Acosta (2020) integra metodologías como Lean Manufacturing [24]. Ninguno de estos tres trabajos integra un LLM para la interacción con el usuario; Pymetory explora esa posibilidad.

3.1.9. Tecnologías de Despliegue en la Nube

El sistema se desplegó en la capa gratuita de Oracle Cloud Infrastructure, en una instancia ARM con 4 OCPU y 24 GB de RAM que permite ejecutar modelos de lenguaje locales con Ollama, y se publica en `https://app.pymetory.com` mediante un túnel de Cloudflare que entrega HTTPS sin exponer a internet el puerto de la aplicación (Capítulo 10).

3.2. Marco Conceptual

3.2.1. Glosario

Esta sección reúne los términos que aparecen a lo largo del documento. Cada definición se contextualiza, cuando aplica, en el caso de estudio de una panificadora del Valle del Cauca. Los términos están agrupados en dos áreas: logística y negocios, y computación e ingeniería de software.

Área de Empresas y Logística

- **Kardex:** Registro detallado y cronológico de todos los movimientos (entradas y salidas) de un producto específico [13]. Cada línea documenta el tipo de operación, la cantidad, el usuario responsable, la fecha exacta y la razón. En Pymetory, el Kardex es de solo inserción en la aplicación: ningún usuario, ni el administrador, puede editar ni borrar un movimiento, y los errores se corrigen con un

ajuste. En la empresa del caso, el consumo se registra en el sistema contable sin verificación física ni constancia de quién corrige (Capítulo 8).

- **Stock:** Cantidad de un producto disponible en el inventario [13]. En la empresa del caso de estudio, la existencia real solo se conoce recorriendo la bodega; Pymetory calcula el stock en tiempo real sumando las cantidades de todos los lotes activos de cada material.
- **Trazabilidad:** Capacidad de seguir el rastro de un producto desde su entrada hasta su salida [13]. En el caso de estudio, la trazabilidad por lote y vencimiento es una exigencia sanitaria que el sistema contable de la empresa no cubre. Pymetory garantiza trazabilidad completa vinculando cada movimiento del Kardex con el lote, el material, la bodega y el usuario responsable.
- **ERP (Enterprise Resource Planning):** Sistemas de planificación de recursos empresariales — paquetes de software que integran y automatizan todos los procesos de una empresa [13]. Soluciones como Odoo o SAP Business One ofrecen funcionalidades de inventario, pero su costo y complejidad las hacen inaccesibles para la mayoría de las PYMEs colombianas.
- **Stakeholder:** Cualquier persona o grupo que tiene un interés en el resultado de un proyecto. En SCRUM, el Product Owner representa los intereses de los stakeholders [25]. Para Pymetory, los principales fueron el director del trabajo de grado (Prof. Héctor Fabio Ocampo), la panificadora caso de estudio y el propio autor.
- **Desabastecimiento:** Situación en la que la demanda supera el inventario disponible, lo que puede detener la producción y generar pérdidas [13]. La empresa del caso ha tenido faltantes (por ejemplo, de sal) que obligaron a compras locales más costosas (Capítulo 8).
- **Mermas:** Reducciones en el inventario por deterioro, vencimiento, daño, robo o errores de registro [13]. En la empresa del caso de estudio la rotación es rápida y no se reportan mermas por caducidad; aun así, las mermas por errores de registro se manifiestan como diferencias entre el sistema contable y la bodega.
- **PYME (Pequeña y Mediana Empresa):** Empresas que por su facturación, empleados y activos se sitúan entre las microempresas y las grandes corporaciones; en Colombia las clasifica la Ley 590 de 2000 [4]. Las PYMEs representan el 99,5 % de las empresas formales del país [1]. La empresa del caso de estudio es una panificadora de tamaño pequeño a mediano.
- **FEFO (First Expired, First Out):** Estrategia de rotación que establece que los productos con fecha de vencimiento más próxima deben despacharse primero. A diferencia de FIFO (First In, First Out), que ordena por fecha de llegada, FEFO ordena por fecha de caducidad. En Pymetory, FEFO es una regla inmutable: todo despacho pasa por el método `Lote::despacharFefo()` (Capítulo 10).

Área de Computación e Ingeniería de Software

- **Laravel:** Framework de código abierto para desarrollo web en PHP, basado en el patrón MVC y con licencia MIT [26]. Pymetory usó Laravel 13 aprovechando su ORM Eloquent, sistema de migraciones, middlewares e integración con Inertia.js 2.0 para comunicarse con el frontend React.
- **MVC (Modelo-Vista-Controlador):** Patrón de arquitectura que separa el código en tres capas: los datos (Modelo), lo que ve el usuario (Vista) y la lógica que conecta ambas (Controlador) [27]. En Pymetory, los modelos son clases de Eloquent como `Lote` y `Movimiento`; las vistas son componentes React como `Dashboard.tsx`; los controladores como `InventoryController` orquestan el flujo.

- **Responsive (Diseño Adaptativo):** Capacidad de una interfaz para adaptarse a distintos tamaños de pantalla [28]. Pymetory se diseñó con enfoque mobile-first usando Tailwind CSS v4, y se probó en pantallas de escritorio (1366 px) y de celular (390 px).
- **SCRUM:** Metodología ágil para gestionar proyectos de software en ciclos iterativos llamados Sprints [25]. El anteproyecto planificó SCRUM con sprints de dos semanas; con un solo desarrollador se ejecutó como flujo continuo en un tablero Kanban de GitHub Projects (Capítulo 12).
- **Accesibilidad Web:** Disciplina que busca que una página web pueda ser usada por el máximo número de personas, independientemente de sus capacidades [17]. La interfaz de Pymetory se auditó contra el nivel AA de las WCAG 2.1, verificando contraste, navegación por teclado y etiquetas para lectores de pantalla.
- **Principios de Nielsen:** Diez reglas heurísticas para diseñar interfaces intuitivas, propuestas por Jakob Nielsen [16]. La implementación final de Pymetory se evaluó contra estas 10 heurísticas: se registraron 14 hallazgos y se corrigieron 13 (Capítulo 11).
- **Usabilidad:** Qué tan fácil es para un usuario lograr un objetivo usando el sistema, de manera efectiva, eficiente y satisfactoria [16]. Pymetory priorizó la usabilidad para operarios de bodega y administradores de PYMEs, usando lenguaje del dominio logístico colombiano y minimizando la curva de aprendizaje.
- **Large Language Model (LLM):** Modelos de lenguaje entrenados con grandes volúmenes de texto, capaces de entender y generar lenguaje natural [15]. El modelo principal del módulo RAG de Pymetory es gpt-oss:120b, servido con Ollama Cloud; el sistema admite además modelos locales en el propio servidor.
- **RAG (Retrieval-Augmented Generation):** Arquitectura de IA que combina la generación de lenguaje natural con la recuperación de información desde bases de datos externas. En Pymetory, el RAG opera en tres pasos: clasificar la intención de la consulta, recuperar contexto desde MySQL y generar una respuesta anclada a datos reales.
- **API (Application Programming Interface):** Conjunto de definiciones y protocolos que permite que diferentes programas se comuniquen entre sí [29]. Pymetory se conecta a los modelos de lenguaje mediante la API de Ollama y APIs compatibles con la de OpenAI.
- **Base de Datos (MySQL):** Sistema de gestión de bases de datos relacional de código abierto [30]. Pymetory utilizó MySQL 8.0 con transacciones ACID que garantizan la integridad del Kardex.
- **Wireframes:** Esqueletos visuales de una interfaz que muestran la disposición de los elementos sin detalles estéticos [29]. En Pymetory evolucionaron a 18 mockups de alta fidelidad en Figma.
- **UML (Unified Modeling Language):** Lenguaje estándar para modelar sistemas de software mediante diagramas [31]. Incluye diagramas de casos de uso, clases, secuencia y otros.
- **Sprint:** Período fijo de tiempo (1 a 4 semanas) en el que un equipo SCRUM completa un incremento de producto [25].
- **Product Backlog:** Lista priorizada de todo lo que se desea incluir en el producto, gestionada por el Product Owner [25].
- **Sprint Backlog:** Elementos del Product Backlog seleccionados para completar durante un Sprint, junto con el plan para lograrlo [25].

- **Incremento de Software:** Suma de todos los elementos completados durante un Sprint más los incrementos anteriores. Debe ser funcional y potencialmente entregable [25].
- **Git/GitHub:** Git es un sistema de control de versiones que permite a varios desarrolladores colaborar en el mismo código. GitHub es la plataforma web que aloja repositorios Git [32]. El código de Pymetory se gestionó en GitHub bajo el usuario `germanmurillas`.
- **Framework:** Estructura de trabajo predefinida que proporciona herramientas y convenciones para acelerar el desarrollo. Laravel es un ejemplo [26].
- **ORM (Object-Relational Mapper):** Técnica que permite interactuar con una base de datos usando objetos del lenguaje de programación en lugar de SQL. Eloquent es el ORM de Laravel [27].
- **Tailwind CSS:** Framework de CSS basado en utilidades que permite diseñar interfaces directamente en el marcado HTML/JSX [33]. Pymetory usó Tailwind CSS v4 para implementar seis temas visuales.
- **Inertia.js y SPA:** Inertia.js es una biblioteca que une un backend Laravel con un frontend React para construir una *Single Page Application* (SPA), que actualiza el contenido sin recargar la página completa, sin necesidad de una API REST (Sección 3.1.5).
- **GSAP (GreenSock Animation Platform):** Biblioteca de JavaScript para crear animaciones web de alto rendimiento. En Pymetory se usó para micro-interacciones, transiciones y animaciones de entrada.
- **Playwright:** Framework de automatización de pruebas desarrollado por Microsoft que simula interacciones reales de usuarios en navegadores [34]. Pymetory utilizó Playwright 1.59 para sus pruebas de extremo a extremo (Capítulo 11).
- **Middleware:** Capa de software que se ejecuta antes de que una solicitud llegue al controlador. Pymetory usa el de autenticación de Laravel y uno propio, `CheckRole`, que verifica permisos según el rol.
- **RBAC (Role-Based Access Control):** Modelo de control de acceso donde los permisos se asignan a roles y los usuarios heredan los permisos de su rol. Pymetory define dos roles: Admin (acceso completo) y Operario (acceso restringido a funciones diarias).

3.2.2. Contexto del caso de estudio

El caso de estudio es una panificadora de tamaño pequeño a mediano del Valle del Cauca, cuyo inventario de materia prima se lleva en kilogramos en un sistema contable alimentado desde las facturas y el reporte diario de producción. Los requisitos se levantaron con una entrevista semiestructurada a la administradora; las preguntas, las respuestas, el problema y los requerimientos derivados se presentan en el Capítulo 8.

Capítulo 4

Metodología

4.1. Tipo de Investigación

Este proyecto se enmarca en la **investigación aplicada**, ya que su fin es utilizar el conocimiento de la ingeniería de sistemas para desarrollar una solución tecnológica a un problema práctico y concreto del entorno empresarial. Se apoya también en la **investigación proyectiva**, al proponer y construir un artefacto de software funcional. El proyecto sigue los lineamientos de la Resolución núm. 137 de la Facultad de Ingeniería de la Universidad del Valle para trabajos de grado en la modalidad de Trabajo de Investigación [3].

4.2. Verificación de resultados

Para no apoyar las conclusiones en una sola fuente de evidencia, cada afirmación sobre el sistema se contrasta con fuentes independientes, en el sentido de la triangulación metodológica [35]:

Pruebas automatizadas. Pruebas unitarias y de integración con PHPUnit, que se ejecutan en cada cambio que llega a la rama principal, y pruebas de extremo a extremo con Playwright, que se ejecutan localmente antes de cada entrega (Capítulo 11).

Análisis estático. PHPStan con Larastan en el nivel 5 sobre `app/`, `routes/` y `config/`. En la medición del 26 de septiembre de 2026 reportó 46 hallazgos en 10 archivos; 25 de ellos corresponden a propiedades de los modelos Eloquent que el analizador no reconoce porque se definen en la base de datos.

Mediciones contra la base de datos. Las respuestas del asistente se comparan con el resultado de una consulta directa a la base de datos de producción (batería de consultas, Capítulo 11).

Evaluación de usabilidad y accesibilidad. Evaluación heurística con los 10 principios de Nielsen y auditoría WCAG 2.1 AA, además de la prueba con usuario.

Cada cifra del Capítulo 11 indica de cuál de estas fuentes proviene.

4.3. Enfoque Metodológico (SCRUM)

Para la gestión del proyecto se planteó una metodología de desarrollo ágil basada en el marco de trabajo SCRUM. Se eligió este enfoque por su flexibilidad, su capacidad para gestionar la incertidumbre y su énfasis en la entrega de valor de manera iterativa e incremental.

Aunque SCRUM define roles como Product Owner, Scrum Master y Equipo de Desarrollo, para este proyecto se adaptaron de la siguiente manera: para obtener una retroalimentación crítica y asegurar que el desarrollo se alineara con los objetivos, el Director de Trabajo de Grado (Prof. Héctor Fabio Ocampo) actuó como el principal *stakeholder* y su perspectiva en las reuniones de seguimiento cumplió una función análoga a la del Product Owner, validando los incrementos de software y ajustando prioridades según los objetivos académicos y técnicos del proyecto.

El autor (Germán David Murillas Mondragón) asumió, de manera unipersonal, las responsabilidades tanto del Equipo de Desarrollo como las del Scrum Master, facilitando el proceso y eliminando impedimentos.

El plan organizó el trabajo en sprints, ciclos iterativos de dos semanas, con los eventos del marco. Cómo se adaptaron en la práctica a un único desarrollador se describe en el Capítulo 12.

- **Product Backlog:** Listado priorizado de las funcionalidades del sistema en GitHub Projects, derivado de los casos de uso y de los requerimientos levantados con la panificadora caso de estudio (Capítulo 8). Las historias se priorizaron según el valor para el usuario y las dependencias técnicas. La gestión del proyecto se apoyó en GitHub Projects [11] como herramienta de seguimiento.
- **Sprint Planning:** Al inicio de cada sprint se seleccionan tareas del Product Backlog para conformar el *Sprint Backlog*, se estima su esfuerzo y se descomponen en tareas técnicas.
- **Sprint Review:** Al finalizar cada sprint se presenta el incremento al director para recibir retroalimentación y ajustar prioridades. Las reuniones con el director y sus acuerdos quedaron registrados en `https://reuniones.pymetory.com`.
- **Sprint Retrospective:** Después de la revisión se identifican mejoras al proceso: qué funcionó, qué se puede mejorar y qué acción concreta se toma en el siguiente sprint.

4.4. Fases del Proyecto

El desarrollo del proyecto se estructuró en las siguientes fases:

- **Levantamiento de Requerimientos:** Identificación de necesidades y elaboración del product backlog con las funcionalidades priorizadas.
- **Diseño del Sistema:** Diseño de la arquitectura MVC, modelado de la base de datos y prototipado de las interfaces de usuario (wireframes).
- **Desarrollo e Implementación:** Configuración del entorno y codificación iterativa de los módulos (sprints), con control de versiones mediante Git/GitHub.
- **Implementación del Módulo LLM:** Configuración y puesta en marcha del módulo de consulta inteligente basado en LLM, incluyendo la integración de APIs y el desarrollo de la lógica para la interpretación de lenguaje natural y la generación de respuestas.
- **Pruebas y Validación:** Ejecución de pruebas unitarias y funcionales en cada tarea, según la definición de terminado (Capítulo 12). Esta fase incluyó una evaluación heurística formal basada en los 10 principios de Jakob Nielsen para medir la usabilidad de la interfaz. Asimismo, se realizó una auditoría de accesibilidad utilizando herramientas automáticas y revisión manual para verificar el cumplimiento de los criterios del nivel AA de las WCAG 2.1.
- **Documentación y Despliegue:** Redacción concurrente del documento de trabajo de grado y preparación para la sustentación final.

4.5. Planificación de Sprints

El anteproyecto planificó seis sprints de dos semanas, precedidos por cuatro semanas de levantamiento de requisitos y diseño. La Tabla 4.1 reproduce ese plan.

Tabla 4.1: Planificación de sprints del anteproyecto (casos de uso con la numeración del Anexo).

Sprint	**Semanas**	**Objetivo**	**Entregables**	**Casos de uso**
Fase 0	Sem 1–4	Requisitos y diseño	Documento de requisitos, modelo de datos, 18 mockups y backlog priorizado	UC1–UC9
Sprint 1	Sem 5–6	Entorno y autenticación	Proyecto base, roles, datos de prueba e inicio de sesión	Base
Sprint 2	Sem 7–8	Inventario y lotes	Materiales, lotes, bodegas, búsqueda y tablero	UC1, UC2, UC9
Sprint 3	Sem 9–10	FEFO, conciliación y reportes	Despacho FEFO, conciliación, reportes PDF/CSV y etiquetas QR	UC3, UC5, UC6
Sprint 4	Sem 11–12	Asistente RAG	Clasificador, contexto desde la base, historial y modo texto	UC4, UC8
Sprint 5	Sem 13–14	Interfaz y ajustes	Componentes según los mockups, temas, ajustes y usuarios	UC7
Sprint 6	Sem 15–16	Pruebas, despliegue y documentación	Pruebas automatizadas, despliegue en la nube y documentación	UC1–UC9

4.5.1. Plan frente a ejecución

El anteproyecto contiene dos calendarios: esta tabla, con seis sprints de dos semanas, y el cronograma general de 36 semanas del Capítulo 5, que asigna cuatro semanas a cada etapa. La ejecución no siguió ninguno al pie de la letra. El registro verificable de lo que se hizo y cuándo son los *issues* del tablero de GitHub Projects y el historial de *commits* del repositorio, que se describen en el Capítulo 12. Las diferencias principales con el plan son:

- **Modelo de lenguaje.** En lugar de una API de pago (el plan mencionaba GPT-4o-mini), el asistente usa como principal un modelo gratuito de Ollama Cloud, con respaldos en la nube y en el propio servidor, seleccionables desde Ajustes.
- **Despliegue.** El sistema quedó en la capa gratuita de Oracle Cloud, publicado con dominio propio mediante un túnel de Cloudflare.
- **Interfaz.** Los prototipos de Figma se tradujeron a componentes React; la interfaz se rediseñó después para uso en el celular y con un sistema de temas visuales.
- **Verificación.** Las pruebas automatizadas, la batería de consultas al asistente, la evaluación heurística y la auditoría de accesibilidad se completaron al cierre (Capítulo 11); la prueba con usuario, el 8 y el 9 de octubre.

La Tabla 4.2 resume la ejecución real según el historial del repositorio hasta el corte de las pruebas.

Tabla 4.2: Ejecución real según el repositorio (rama principal, hasta el 2 de octubre de 2026; 205 *commits*).

Periodo	**Commits**	**Hitos (commit)**
Feb.–mar. 2026	3	Creación del repositorio (`17f17ba`); sin *commits* en marzo.
Abr. 2026	15	Tablero con datos reales, primer chat RAG, tablero Kanban y primer despliegue (`999f87c`, `1ce059c`).
May. 2026	45	Reportes PDF y CSV, historial del chat y RAG con clasificación de consultas (`7a8981d`, `f015af2`).
Jun. 2026	38	Nginx, ajustes, integración continua con pruebas (`1ec431b`, `33f07a7`).
Jul.–ago. 2026	2	Pausa: solo ajustes visuales (`e874959`).
Sep. 2026	82	Documento por capítulos verificado contra el código, reglas de despacho y seguridad, Reabastecimiento (`e0e1078`, `9d0efb8`, `e17e6ea`).
Oct. 2026 (1–2)	20	Costos por consulta, comparación de modelos y claves de API seguras (`6ddad01`, `bf8cd8d`).

4.6. Herramientas de Desarrollo y Colaboración

La Tabla 4.3 resume las herramientas utilizadas durante el desarrollo del proyecto.

Tabla 4.3: Herramientas de desarrollo y colaboración.

Categoría	**Herramienta**	**Propósito**
Control de versiones	Git + GitHub	Repositorio público, historial de commits y despliegue automático desde la rama principal
Gestión de proyecto	GitHub Projects	Tablero Kanban con backlog, tareas en curso y terminadas
IDE	Visual Studio Code + Cursor	Edición de código con soporte para PHP, TypeScript, React y Tailwind
Diseño UI	Figma	Prototipo de alta fidelidad de los módulos
Comunicación	Google Meet + WhatsApp	Reuniones con el director
Base de datos	MySQL 8 y SQLite	MySQL en producción; SQLite en desarrollo y pruebas
Pruebas	PHPUnit + Playwright 1.59	Pruebas unitarias y de integración; pruebas de extremo a extremo en Chromium (escritorio y celular)
Documentación	Overleaf (LaTeX)	Redacción del documento de trabajo de grado
IA asistentes	OpenCode Go, Claude Code, Gemini Antigravity	Asistencia en generación de código, depuración y documentación
Despliegue	SSH + SCP + Tmux	Transferencia de archivos y administración de servidores remotos

4.7. Consideraciones Éticas

El desarrollo de este proyecto se rigió por principios de integridad académica y profesional:

- **Confidencialidad de datos empresariales**: La empresa del caso de estudio no se identifica en el documento. Los datos de demostración del sistema son ficticios. La evaluación con usuario (Capítulo 11) usó una instancia aparte, de acceso restringido, con el catálogo real de insumos que entregó la administradora y vencimientos estimados; esos datos no se publican. La empresa autorizó el uso de sus procesos operativos como referencia para el levantamiento de requisitos, pero los datos específicos

de su operación comercial no se divulgan. Las consultas al modelo en la nube envían la pregunta y el contexto filtrado, que puede incluir nombres de usuario en los movimientos; el administrador puede elegir un modelo local.

- **Uso de inteligencia artificial como herramienta**: Durante el desarrollo, el autor utilizó asistentes de IA (OpenCode Go, Claude Code, Gemini Antigravity) como herramientas de productividad para generación de código, depuración, y redacción y revisión de este documento. El autor revisó y validó el código antes de incluirlo y verificó las cifras del documento contra el repositorio y la base de datos.
- **Licencias de software**: Las tecnologías del sistema —Laravel, React, Inertia.js, Tailwind CSS, MySQL, Playwright— son de código abierto; GSAP es de uso gratuito bajo su propia licencia. El modelo principal y el local no tienen costo (capa gratuita de Ollama Cloud y Ollama en el servidor); la comparación de modelos y el respaldo en la nube usan una suscripción del autor a OpenCode Go. La infraestructura también es de capa gratuita (Oracle Cloud y Cloudflare).
- **Cumplimiento normativo**: El proyecto se acoge a la Resolución núm. 137 de la Facultad de Ingeniería de la Universidad del Valle, que establece los lineamientos para trabajos de grado en la modalidad de Trabajo de Investigación.

Capítulo 5

Cronograma de Hitos

El anteproyecto distribuyó el trabajo en 36 semanas (9 meses) bajo el marco SCRUM. Para verificar cada hito —los puntos de control que marcan el avance— se presenta la distribución de actividades, entregables y dependencias entre fases; la ejecución real se registra en el Capítulo 12.

Fase / Sprint	**Semanas**	**Entregable Tangible**	**Dep.**
Fase 1: Análisis de Requisitos	1 - 4	Documento SRS y Product Backlog	N/A
Fase 2: Diseño Arquitectónico	5 - 8	Modelo Entidad-Relación y Mockups	Fase 1
Sprint 1: Seguridad y Core	9 - 12	Módulo de Autenticación y Roles	Fase 2
Sprint 2: Gestión de Catálogo	13 - 16	CRUD de Productos y Categorías	Sprint 1
Sprint 3: Operaciones Logísticas	17 - 20	Lógica de Movimientos y Kardex	Sprint 2
Sprint 4: IA Predictiva	**21 - 24**	**Módulo de Consulta LLM funcional**	**Sprint 3**
Sprint 5: Reportería y Alertas	25 - 28	Subsistema de Reportes y Notificaciones	Sprint 4
Fase 3: Calidad y Validación	29 - 32	Informe de Pruebas y Métricas de IA	Sprint 5
Fase 4: Cierre y Documentación	33 - 36	Documento de Tesis y Aplicación Final	Fase 3

Tabla 5.1: Planificación detallada por semanas, entregables y dependencias técnicas.

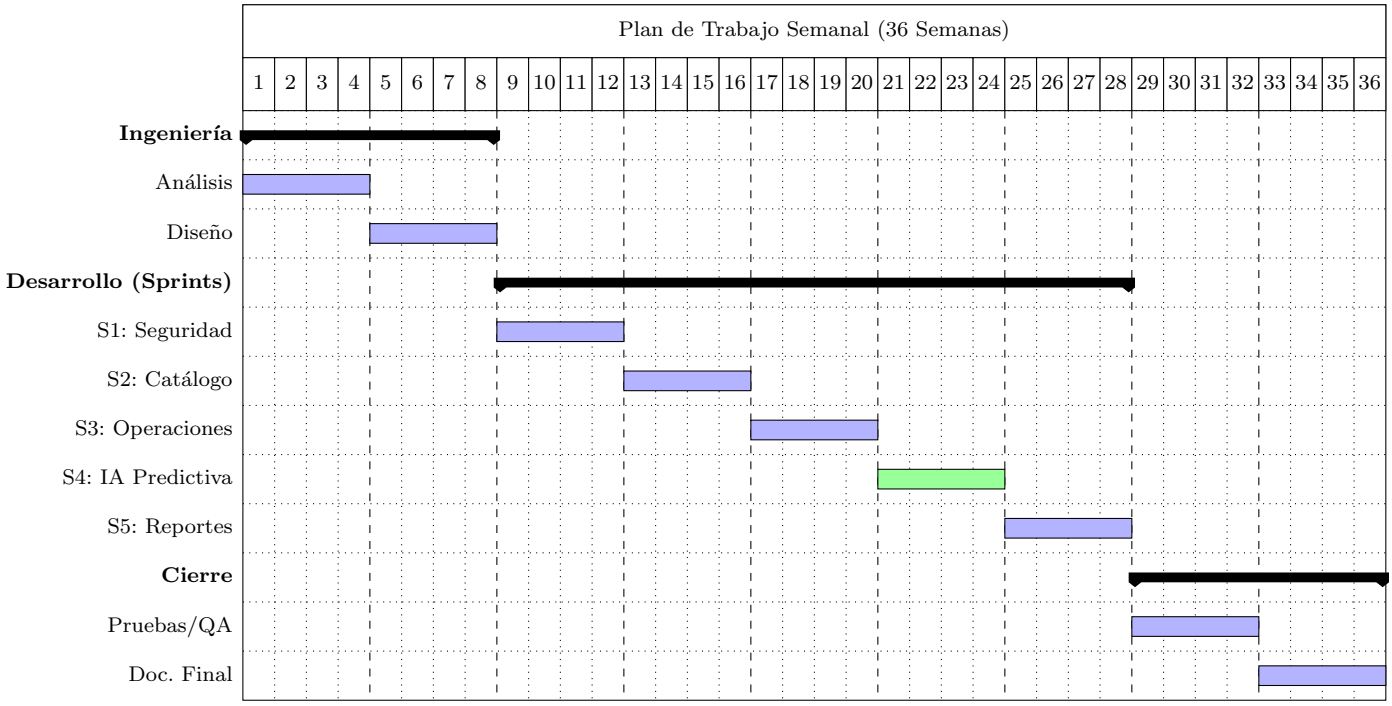


Figura 5.1: Diagrama de Gantt con el aporte diferencial en IA resaltado.

Capítulo 6

Presupuesto

Este proyecto se ejecuta como trabajo de grado universitario. La inversión principal no es dinero sino tiempo y conocimiento. El presupuesto que se presenta a continuación es una **valoración académica del esfuerzo de ingeniería** —es decir, cuánto costaría este trabajo si se contratara a un ingeniero para hacerlo, usando como referencia el salario mínimo del sector.

Todas las herramientas utilizadas son de código abierto (licencia MIT, Apache o similar), lo que significa que no hay costos de licenciamiento. El computador, la conexión a internet y el software de desarrollo son propiedad del autor. En resumen: el proyecto no tuvo costos de licencias ni de infraestructura; los gastos menores del autor (el dominio y una suscripción a OpenCode Go, usada para comparar modelos y como respaldo en la nube) no se incluyen.

Tabla 6.1: Valoración académica del esfuerzo de ingeniería.

Actividad	**Horas Estimadas**	**Valor por Hora (Referencial)**	**Costo Estimado**
Análisis y Diseño	55 horas	$15.000 COP	$825.000
Desarrollo del Sistema	90 horas	$15.000 COP	$1.350.000
Pruebas y Validación	20 horas	$15.000 COP	$300.000
Documentación del Proyecto	35 horas	$15.000 COP	$525.000
Total Estimado	**200 horas**		**$3.000.000 COP**

Las horas son la estimación del anteproyecto, y el valor por hora es una tarifa supuesta, sin fuente salarial.

Estimación de costos de despliegue y mantenimiento anual

La Tabla 6.2 estima lo que costaría mantener el sistema en producción durante un año si una PYME contratara hosting comercial; en este proyecto se utiliza la capa gratuita de Oracle Cloud [36], que no tiene costo.

Tabla 6.2: Costos operativos anuales estimados.

Concepto	**Costo Anual Estimado (COP)**	**Fuente**
Registro de Dominio (.com.co)	$59.000 + IVA	Colombia Hosting [37]
Hosting Básico en la Nube	$300.000	Colombia Hosting [38]
Total Anual Estimado	**~$359.000 + IVA**	

Capítulo 7

Impacto Ambiental

El principal impacto ambiental de este proyecto es **positivo y se deriva de la desmaterialización**. Desmaterializar significa reemplazar objetos físicos —papel, tinta, archivadores, cuadernos— por equivalentes digitales. Al digitalizar la gestión de inventarios, el sistema reduce la necesidad de registros en papel, impresiones de reportes y archivadores físicos.

Como orden de magnitud, una bodega podría dejar de imprimir algunas resmas de papel al año; si muchas PYMEs adoptaran una herramienta similar, el efecto acumulado sería mayor.

El impacto negativo es mínimo. Proviene del consumo eléctrico del computador usado durante el desarrollo y del servidor donde se despliega el sistema. Se estimó, como orden de magnitud y sin medición, un consumo de 600 kWh durante todo el proyecto, equivalente a unas 0.24 toneladas de dióxido de carbono (CO2). Este impacto se mitiga usando software eficiente y optimizando el código para que el servidor trabaje menos tiempo.

La estimación de la huella de carbono se calcula con el factor de conversión oficial del Sistema Interconectado Nacional de Colombia para 2021, establecido en 0.403 toneladas de CO2 equivalente por cada megavatio-hora (MWh) consumido [39]. El cálculo es directo:

$$600\,\text{kWh} \times \frac{0{,}403\,\text{t CO2eq}}{1000\,\text{kWh}} \approx 0{,}24\,\text{toneladas de CO2eq.}$$

A futuro, este impacto podría reducirse aún más eligiendo un proveedor de hosting que use energías renovables.

Capítulo 8

Levantamiento de requerimientos

Este capítulo documenta cómo se obtuvieron los requerimientos de Pymetory: a quién se entrevistó, qué se preguntó, qué se respondió, qué problema quedó en evidencia y qué requerimientos funcionales y no funcionales se derivaron de ello. Al final de cada requerimiento se indica su estado en la versión entregada del sistema, de manera que el lector pueda distinguir lo que se construyó de lo que quedó planteado para trabajos futuros.

8.1. Caso de estudio y participante

El caso de estudio es una panificadora de tamaño pequeño a mediano ubicada en el municipio de La Victoria (Valle del Cauca), en adelante *la empresa*. El anteproyecto la ubicó en Santiago de Cali y llamó «encargado de bodega» a la entrevistada, que es su administradora; la empresa está en La Victoria. La empresa elabora productos de panadería a escala industrial y compra de forma continua materias primas como harinas, mejoradores, levaduras, grasas, semillas, sal, azúcar y material de empaque.

La participante fue la persona que ocupa el cargo de administración de la empresa, en adelante *la administradora*. Entre sus responsabilidades están recibir las materias primas, solicitar los pedidos a los proveedores, ingresar al sistema contable el consumo diario reportado por producción y realizar el cierre de inventario al final de cada mes.

Criterio de anonimización. Para no comprometer a personas ni entidades, en este documento no se registran el nombre de la empresa, el de la administradora ni el de las demás personas mencionadas durante la entrevista; se usan únicamente sus cargos (la administradora, la líder de producción, la persona encargada de digitar las facturas). Las marcas comerciales de los insumos se reemplazan por descripciones genéricas.

8.2. Diseño y ejecución de la entrevista

El 4 de marzo de 2026 se realizó una entrevista semiestructurada [29, 35], en una sola sesión con una duración aproximada de 41 minutos, que fue grabada en audio. El entrevistador planteó las preguntas "como si no supiera nada" del proceso, para que la participante describiera la operación real y no una versión idealizada.

La entrevista se organizó en dos bloques:

1. **Preguntas en bruto.** Diez preguntas formuladas en lenguaje cotidiano, orientadas a que la participante contara cómo funciona la bodega en el día a día (Tabla 8.1).

2. **Preguntas técnicas depuradas.** Reformulaciones técnicas de algunas preguntas en bruto, con las que se contrastaron las primeras respuestas y se exploró qué esperaría la participante de un sistema de software (Tabla 8.2).

La grabación se transcribió de forma automática. Las cifras mencionadas por la participante se presentan como aproximadas, porque corresponden a ejemplos dados de memoria durante la conversación y no a registros contables verificados.

Tabla 8.1: Preguntas en bruto de la entrevista.

N.º	**Pregunta**
P1	¿Cómo anotan lo que llega de los proveedores y qué datos guardan de eso?
P2	¿Qué hacen cuando ven que un producto se va a acabar para que la producción no se detenga?
P3	¿Quién se encarga de decir cuánto material se gastó al final del turno?
P4	¿Cómo revisan si lo que dice el computador es lo mismo que hay guardado en la bodega?
P5	¿Qué pasa cuando un producto está muy cerca de vencerse? ¿Cómo lo identifican?
P6	¿Cómo encuentran un producto rápido cuando hay mucha mercancía amontonada?
P7	¿Qué cosas le gustaría preguntarle al sistema si pudiera darle respuestas inteligentes?
P8	¿Quiénes son las personas que pueden entrar a la bodega y sacar materiales?
P9	¿Cuál es el problema más común que tienen cuando cuentan la mercancía al final de mes?
P10	¿Cómo registran si un producto llegó dañado o si no era lo que habían pedido?

Tabla 8.2: Preguntas técnicas depuradas.

Deriva de	**Pregunta técnica**
P1	¿Qué metadatos son críticos para garantizar la trazabilidad de la materia prima en el ingreso?
P2	¿Cómo se definen los parámetros del stock mínimo y los disparadores de alertas de reabastecimiento?
P4	¿Mediante qué mecanismos de auditoría se pueden conciliar las existencias físicas con las del inventario del programa contable?
P5	¿Cómo se gestiona la rotación de inventario para minimizar mermas por caducidad?
P7	¿Qué consultas de trazabilidad y reportes analíticos se deben resolver mediante lenguaje natural?
P9	¿Qué factores causan la degradación de la integridad de los datos en el sistema actual?

8.3. Respuestas obtenidas

A continuación se resume lo que respondió la administradora, agrupado por tema. Entre comillas se incluyen fragmentos textuales de la transcripción cuando aportan precisión.

8.3.1. Ingreso de materia prima y unidad de medida (P1)

Las materias primas se reciben con factura del proveedor. Llegan en distintas presentaciones: las harinas, los mejoradores y las semillas en bultos (las harinas, de 50 kg netos), y productos como la grasa y la levadura en cajas. Sin importar la presentación, **todo el inventario se lleva en kilos**: si llegan 100 cajas de levadura de 10 kg cada una, lo que se ingresa al sistema contable son 1.000 kg. Los gramos solo aparecen en el pesaje de producción y en el conteo de remanentes de báscula durante el cierre mensual; el peso de la caja y del plástico no se cuenta.

Para la parte administrativa, los datos que importan en el ingreso son la cantidad, el número de factura (electrónica) y el proveedor. Al presentarle la pregunta depurada, la administradora añadió que, en un sistema de software, sería conveniente registrar también la fecha de vencimiento, el lote y la ficha de calidad del producto, no tanto por el consumo sino porque es un requisito reglamentario: "el día de mañana se presenta algún problema, alguna intoxicación, y nos viene a revisar el INVIMA o algún ente regulador, y es lo primero que nos va a pedir: el manejo de los lotes, el manejo de las fechas de vencimiento de nuestras materias primas".

8.3.2. Reabastecimiento (P2)

No existe un stock mínimo formal. La administradora revisa físicamente la bodega y, con base en su experiencia, decide cuándo pedir. Reconoció que "en esa parte sí tenemos falencias": en ocasiones se han quedado sin un insumo (mencionó la sal) y han tenido que comprarlo en el mercado local, a un precio mayor que el del proveedor. En sus palabras, el proceso es "demasiado humano, demasiado artesanal".

Como ejemplo de cómo razona hoy el punto de pedido, explicó que un tipo de semilla que llega desde Bogotá tarda aproximadamente 8 días después del pedido, mientras que otro insumo que llega por Cartagena tarda entre 20 y 25 días; por eso, cuando le quedan unos 15 bultos, ya debería estar pidiendo. Señaló que un pedido de 48 bultos le dura aproximadamente dos meses. Expresó el deseo de que el software "me diga: tienes tantos bultos que te van a alcanzar para tantos días", calculado a partir de la estadística de consumo.

8.3.3. Reporte de consumo (P3)

La líder de producción diligencia diariamente un formato en papel con las mezclas del día y la materia prima utilizada; con ese formato, la administradora ingresa el consumo al sistema contable. Debido a diferencias que detectó entre el sistema y la bodega, la administradora pidió que al pie del formato se especificara, día a día, cuánto se consumió de cada producto, incluyendo cuánto se destinó a la esponja (prefermento), cuánto al espolvoreo y cuánto a la producción propiamente dicha.

8.3.4. Conciliación entre sistema y bodega (P4)

La conciliación es manual. Al cierre de cada mes, la administradora registra en una hoja de Excel el conteo físico y lo compara con lo que reporta el sistema contable. Explicó que desde que se incorporó el proceso de esponja (hace aproximadamente tres años) aparecen diferencias: por ejemplo, cuando el programa de producción indica unos 45 bultos de harina, en la realidad puede haber una diferencia de 2 a 3 bultos, lo que a su vez descuadra los demás insumos asociados.

8.3.5. Vencimiento y rotación (P5)

La administradora afirmó que **no tienen pérdidas por vencimiento**: se compra lo que se va a consumir en menos de un mes, la materia prima llega de forma constante y, en general, lo comprado se consume en 15 a 20 días. Aun así, instruye al personal para que se consuma primero lo que ya estaba en bodega (una rotación informal por llegada, que el sistema formaliza como FEFO) y considera importante revisar lotes y fechas por razones sanitarias.

8.3.6. Ubicación de productos (P6)

La bodega mantiene un volumen de materia prima fácil de identificar y de consumo rápido, por lo que ubicar un producto no es un problema. La administradora recorre la bodega y el cuarto de pesaje dos o tres veces por semana para verificar existencias y evitar pedir en exceso.

8.3.7. Consultas deseadas al sistema (P7)

Lo primero que pidió fueron **estadísticas de consumo diarias, semanales y mensuales**. Ilustró la idea con un ejemplo: que el sistema pueda responder cuántas bolsas de un tipo de empaque quedan hoy respecto a las que había en el último cierre de inventario, y qué porcentaje se consumió en ese periodo. En la pregunta depurada añadió consultas del tipo "cuántos bultos me quedan" y "cuántos bultos necesito para la producción de cada producto", según el histórico semanal, quincenal o mensual. También planteó como idea deseable que, a partir de lo que producción reporta como consumido de harina en el día, el sistema estime cuánto se consumió de las demás materias primas y cuánto debería quedar.

8.3.8. Acceso a la bodega (P8)

Solo pueden sacar materiales la jefe, la líder de producción y las personas que ellas designan; por ejemplo, la persona de pesaje retira las cantidades que la líder le indica, incluyendo de qué marca o de qué referencia tomar el insumo.

8.3.9. Problemas en el conteo de fin de mes (P9)

La administradora identificó dos fuentes principales de error:

- **Estibado no estandarizado.** Una misma referencia se apila con distinto número de cajas por nivel (por ejemplo, 14 o 18 cajas de levadura), lo que obliga a mover las estibas para contar. Lo mismo ocurre con los bultos de bolsas, que pueden venir por 2.500 o por 3.000 unidades. Propuso como solución un protocolo de bodegaje que fije la forma de estibar cada referencia.
- **Cruce entre productos similares.** Dos mejoradores de la misma marca, con empaques casi idénticos y presentaciones de aproximadamente 20 y 25 kg, se confunden al ingresar las facturas. La persona que digita las facturas lo hace "de factura a factura", sin verificar físicamente en bodega. En el ejemplo que mostró durante la entrevista, el conteo físico de uno de los mejoradores era de unos 100 kg mientras el sistema mostraba un saldo negativo (alrededor de -76 kg), y el otro tenía unos 75 kg físicos frente a 123 kg en el sistema. La diferencia se corrige manualmente con movimientos de entrada y salida en el sistema contable.

Durante esta parte de la conversación se discutió la posibilidad de identificar cada producto con un código leído desde un teléfono celular. La administradora consideró inviable pegar una etiqueta a cada bulto (por el volumen recibido, que puede ser del orden de 400 bultos de harina, y porque la harina hace

que las etiquetas se desprendan), y se concluyó que lo viable sería aprovechar los códigos de barras que ya traen los productos.

8.3.10. Recepción con novedades (P10)

La mercancía se revisa visualmente al recibirla. Si un bulto llega abierto o dañado, o llega menos cantidad de la facturada, se anota en la factura, se firma la novedad con el transportador y se notifica al proveedor; lo usual es devolver el producto para que lo cambien. La administradora indicó que no han recibido productos distintos a los pedidos.

8.4. Problema evidenciado

La entrevista permitió precisar el problema real de la empresa, que no coincide con el supuesto más frecuente para el sector. La empresa **no pierde materia prima por vencimiento**, gracias a una rotación rápida; su problema es de **información**:

1. **El inventario digital no refleja la bodega.** El sistema contable se alimenta desde las facturas y desde el formato de producción, sin verificación física. Los cruces entre referencias similares y los consumos no desglosados (esponja, espolvoreo) generan diferencias que solo se descubren en el cierre mensual.

2. **La conciliación es manual y tardía.** Se hace una vez al mes en Excel, se corrige con movimientos manuales y no deja registro de quién hizo la corrección ni por qué.

3. **El reabastecimiento depende de la memoria de una persona.** No hay stock mínimo, ni punto de reorden, ni estadística de consumo; ya se han presentado faltantes que obligan a compras de emergencia más costosas.

4. **La trazabilidad por lote y vencimiento es una exigencia regulatoria no resuelta.** Aunque no hay mermas, la empresa debe poder responder ante una inspección sanitaria qué lote de qué insumo se usó y cuándo vencía.

Por este hallazgo, la formulación del problema (Sección 1.1) se ajustó respecto al anteproyecto, que se centraba en las pérdidas por vencimiento.

8.5. Requerimientos funcionales

La Tabla 8.3 lista los requerimientos funcionales definidos para Pymetory. La columna *Origen* indica la pregunta de la entrevista o la regla de negocio de la dirección del trabajo de grado de la que proviene cada requerimiento; los requerimientos marcados como *Sistema* surgieron de las necesidades propias de operar la aplicación. La columna *Estado* refleja lo verificado en el código fuente de la versión entregada: *Implementado*, *Parcial* o *No implementado*.

Tabla 8.3: Requerimientos funcionales de Pymetory.

ID	Requerimiento	Origen	Estado
RF-01	Registrar materiales en un catálogo con código único, nombre, unidad de medida y stock mínimo.	P1, P2	Implementado. La unidad por defecto es el kilogramo.
RF-02	Registrar el ingreso de lotes con número de lote, cantidad, costo unitario, bodega de destino y fecha de vencimiento obligatoria.	P1 (depurada)	Implementado. En la revisión final se agregó el ingreso de lotes para insumos ya registrados y se corrigió la recepción de órdenes, que fijaba el vencimiento en lugar de pedirlo (Capítulo 10).
RF-03	Asociar a cada ingreso el proveedor, el número de factura y la ficha de calidad del insumo.	P1 (depurada)	Parcial. Los proveedores se gestionan en las órdenes de compra; factura y ficha de calidad no tienen un campo dedicado.
RF-04	Despachar materia prima aplicando FEFO: el sistema toma primero el lote activo con fecha de vencimiento más próxima.	P5, regla de la dirección	Implementado.
RF-05	Registrar en un Kardex inmutable cada movimiento de inventario con usuario, fecha y hora, tipo, cantidad y motivo.	P4, P9, regla de la dirección	Implementado.
RF-06	Restringir las operaciones según el rol del usuario (administrador u operario).	P8, regla de la dirección	Implementado.
RF-07	Generar alertas cuando la existencia de un material cae por debajo de su stock mínimo.	P2	Implementado (revisión programada cada hora; las alertas se consultan en la vista Alertas).
RF-08	Generar alertas de lotes próximos a vencer según un umbral configurable.	P5	Implementado (umbral por defecto: 15 días).
RF-09	Calcular un punto de reorden a partir del consumo promedio y del tiempo de entrega de cada proveedor, e indicar para cuántos días alcanza la existencia actual.	P2 (depurada)	Implementado en la vista Reabastecimiento y en el asistente: consumo diario promedio, cobertura, fecha de agotamiento y punto de reorden.

ID	Requerimiento	Origen	Estado
RF-10	Consultar estadísticas de consumo por periodo (diario, semanal, mensual).	P7	Implementado. Gráfico por día, semana o mes en Reabastecimiento; el asistente responde el consumo de un periodo y lo compara con el anterior.
RF-11	Conciliar el inventario físico con el del sistema mediante un ajuste justificado que queda registrado en el Kardex.	P4 (depurada)	Implementado (ajuste restringido al administrador).
RF-12	Consultar el inventario en lenguaje natural con respuestas construidas a partir de los datos reales de la base de datos (RAG).	P7, regla de la dirección	Implementado. Respondió correctamente 49 de 50 consultas de la batería (Capítulo 11).
RF-13	Identificar materiales y lotes mediante códigos (QR y código de barras) impresos en etiquetas y leídos con la cámara de un dispositivo.	P9	Implementado con etiquetas generadas por el sistema y, desde el 9 de octubre, con el código de barras de fábrica (EAN-13) de los productos registrados.
RF-14	Registrar novedades de recepción (producto dañado o cantidad incompleta).	P10	Parcial. El lote admite el estado “en cuarentena” y la recepción de órdenes de compra registra la cantidad recibida; no existe un formulario específico de no conformidad.
RF-15	Desglosar el consumo diario por destino de producción (esponja, espolvoreo, línea de producto).	P3, P4	No implementado. El consumo se registra con un motivo general (producción, venta, desperdicio o ajuste).
RF-16	Gestionar bodegas con su capacidad y transferir existencias entre bodegas, dejando registro en el Kardex.	Sistema	Implementado. El administrador crea y edita bodegas con una imagen de fondo (archivo subido o enlace) guardada en la base de datos; cada bodega muestra sus lotes, insumos y valor reales.
RF-17	Generar reportes de inventario, movimientos, FEFO, consumo y valorización en PDF, CSV y Excel.	Sistema, P7	Implementado.

ID	Requerimiento	Origen	Estado
RF-18	Administrar usuarios y sus roles.	Sistema, P8	Implementado.

8.6. Requerimientos no funcionales

Tabla 8.4: Requerimientos no funcionales derivados de la entrevista.

ID	Requerimiento	Origen	Estado
RNF-01	El inventario se expresa en una unidad base (kilogramos), independiente de la presentación comercial (bulto o caja).	P1	Implementado: el ingreso y el consumo convierten la presentación (p. ej., bulto de 50 kg) a kilos.
RNF-02	La trazabilidad por lote y fecha de vencimiento debe poder consultarse en cualquier momento, como soporte ante una inspección sanitaria.	P1 (depurada), P5	Implementado mediante el Kardex y los reportes.
RNF-03	La identificación de productos no debe exigir etiquetar cada bulto individualmente.	P9	Implementado. Las etiquetas se generan por lote.
RNF-04	La interfaz y el asistente deben operar en español y ser utilizables por personal sin formación técnica.	Toda la entrevista	Implementado. Su evaluación se presenta en el capítulo de pruebas.

8.7. Fuera del alcance

Durante la entrevista surgieron ideas que se dejaron explícitamente fuera del alcance de este trabajo:

- Un mapa de la bodega en tiempo real con cada bulto como un punto (etiquetas RFID o NFC), descartado por el costo de etiquetar cada unidad y las condiciones de la bodega.
- El protocolo de estibado propuesto por la administradora: es una medida organizativa, no un requerimiento de software, aunque reduciría el error de conteo de la P9.

8.8. Limitaciones del levantamiento

- Se entrevistó a una sola persona, la administradora; lo relativo a la líder de producción y a quien digita las facturas se conoció a través de ella. Con la empresa hubo tres sesiones (4 de marzo, 8 y 9 de octubre de 2026), no las ocho con la empresa y la dirección que menciona el anteproyecto.
- La transcripción es automática, las cifras son aproximadas y no se analizaron documentos de la empresa.
- Las preguntas depuradas sugieren términos (lote, mermas) que la participante pudo adoptar; el texto distingue sus respuestas de las dadas a las preguntas en bruto.

Capítulo 9

Diseño

Este capítulo explica cómo se diseñó Pymetory a partir de los requerimientos del Capítulo 8: la arquitectura general del sistema, las herramientas elegidas y las razones de cada elección, el modelo de datos, el diseño del módulo de consulta en lenguaje natural y el diseño de la interfaz de usuario.

9.1. De los requerimientos a los componentes

Cada requerimiento funcional implementado se asignó a uno o varios componentes del sistema. La Tabla 9.1 muestra esa correspondencia; los nombres de clases y comandos son los del código fuente.

Tabla 9.1: Correspondencia entre requerimientos funcionales y componentes.

RF	**Requerimiento (resumen)**	**Componentes**
RF-01, RF-02	Catálogo de materiales e ingreso de lotes	`InventoryController`, modelos `Material` y `Lote`
RF-04	Despacho FEFO	`ConsumptionController`, *scope* `Lote::despachables`
RF-05	Kardex	Modelo `Movimiento`, tabla `movimientos`
RF-06, RF-18	Roles y usuarios	*Middleware* `CheckRole`, `UserController`, `AuthController`
RF-07, RF-08	Alertas de stock bajo y de vencimiento	Comandos `alerts:low-stock` y `alerts:fefo`, programados cada hora
RF-09, RF-10	Punto de reorden y consumo por periodo	Servicio `ProyeccionInventario`, `ProyeccionController`, vista Reabastecimiento, intenciones `forecast` y `consumption` del asistente
RF-17	Reportes	`ReportController`, `InventoryController` (exportación PDF, CSV y Excel)
RF-11	Conciliación	`InventoryController::adjust`
RF-12	Consulta en lenguaje natural	`ChatLLMController`
RF-13	Etiquetas y escaneo	`LabelController`, `QRScanController`
RF-14	Recepción de órdenes de compra	`PurchaseOrderController`
RF-16	Bodegas y transferencias	`InventoryController`, `TransferController`

9.2. Arquitectura del sistema

Pymetory es una aplicación web monolítica organizada según el patrón Modelo-Vista-Controlador (MVC) que propone el framework Laravel [26]. La capa de presentación se construye con React y se conecta con los controladores de Laravel mediante Inertia.js [7]: el controlador responde con el nombre de una página y los datos que necesita, e Inertia se encarga de renderizar el componente React correspondiente en el navegador. De esta forma la aplicación se comporta como una aplicación de una sola página (SPA) sin que sea necesario diseñar y mantener una API REST separada para las vistas principales. Algunas funciones puntuales (usuarios, etiquetas, órdenes de compra, proveedores de modelos de lenguaje) sí exponen rutas JSON bajo el prefijo `/api`, que consumen los componentes React con Axios.

La Figura 9.1 resume las capas y el recorrido de una solicitud.

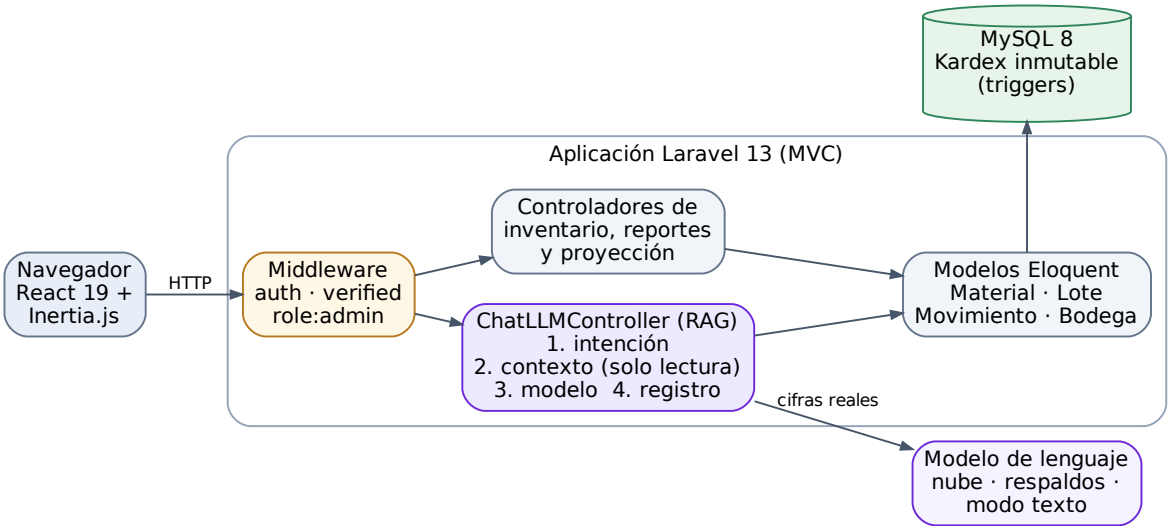


Figura 9.1: Arquitectura en capas de Pymetory y flujo de una solicitud.

Dos decisiones de diseño se derivan directamente de las reglas de la dirección del trabajo de grado:

- **El módulo de lenguaje natural es de solo lectura sobre el inventario.** El `ChatLLMController` consulta las tablas del inventario para construir el contexto, pero no crea, modifica ni elimina lotes o movimientos; solo escribe en su propio historial de conversación. Así, un error del modelo de lenguaje puede producir una respuesta equivocada, pero no puede alterar las existencias.
- **El control de acceso se decide en el servidor.** Las rutas administrativas se protegen con el *middleware* `role:admin`; la interfaz oculta opciones según el rol, pero la autorización real ocurre en Laravel, antes de llegar al controlador.

9.3. Selección de herramientas

La selección de herramientas respondió a cuatro criterios propios del proyecto: (1) costo de licencia nulo, porque el sistema está pensado para PYMEs que no pueden pagar suscripciones; (2) un solo de-

sarrollador, lo que favorece herramientas que resuelven de fábrica autenticación, migraciones y pruebas; (3) experiencia previa del autor con Laravel, GitHub y Figma; y (4) que el sistema pueda operar con un modelo de lenguaje local si la empresa lo exige; por rendimiento, el modelo principal está en la nube (Capítulo 10). La Tabla 9.2 presenta cada herramienta, las alternativas comparables y la razón de la elección.

Tabla 9.2: Herramientas elegidas, alternativas y justificación.

Herramienta	**Alternativas**	**Por qué se eligió**
Laravel 13 (PHP 8.3)	Django, Node.js/Express, Spring Boot	Incluye ORM, migraciones versionadas, *middleware*, validación, tareas programadas y soporte de pruebas en un solo marco; el autor ya lo conocía. PHP se ejecuta en casi cualquier alojamiento de bajo costo.
React 19 + Inertia.js 2	Blade, Livewire, React con API REST aparte	React permite construir una interfaz interactiva (arrastrar y soltar, escáner, chat) con un ecosistema amplio de componentes; Inertia evita duplicar la capa de API para las vistas y conserva el enrutamiento y la autenticación de Laravel.
TypeScript 5	JavaScript	Documenta la forma de los datos que llegan del servidor y permite que el editor señale errores; la verificación estricta en la integración continua queda como trabajo futuro.
Tailwind CSS 4	Bootstrap, CSS propio	Estilos por utilidades que permiten traducir el prototipo de Figma a componentes sin mantener hojas de estilo separadas por vista, y definir temas mediante variables.
MySQL 8.0	PostgreSQL, SQLite	Base de datos relacional con transacciones ACID e integridad referencial, necesarias para que un despacho FEFO que toca varios lotes quede completo o no quede. Disponible en el servidor de producción sin costo. En pruebas automatizadas se usa SQLite en memoria.
Ollama y proveedores compatibles con la API de OpenAI	Un único proveedor comercial	Ollama permite ejecutar modelos abiertos en el propio servidor, sin costo por consulta, o en Ollama Cloud, que recibe la pregunta y el contexto filtrado, los procesa de forma transitoria y no los usa para entrenar [40]. El diseño admite además proveedores externos configurables desde la aplicación, para elegir según disponibilidad y calidad.
GitHub y GitHub Projects	GitLab, Jira, Trello	Repositorio, historial de cambios, automatización de despliegue y tablero de tareas en una sola plataforma gratuita, que el autor ya usaba.
Figma	Adobe XD, Penpot	Prototipado de alta fidelidad en el navegador, con plan gratuito; el autor ya lo usaba.
Playwright 1.59 y PHPUnit	Cypress, Selenium, Pest	PHPUnit viene integrado con Laravel; Playwright automatiza el navegador real para pruebas de extremo a extremo.
Oracle Cloud (capa Always Free)	AWS, Google Cloud, Azure	Instancia ARM con 4 OCPU y 24 GB de RAM sin costo recurrente [36], suficiente para la aplicación, la base de datos y modelos de lenguaje pequeños.

9.4. Modelo de datos

El esquema se define mediante migraciones de Laravel, de modo que cada cambio en la base de datos queda versionado junto con el código. La Tabla 9.3 agrupa las tablas del dominio de Pymetory.

Tabla 9.3: Tablas del dominio de Pymetory.

Grupo	**Tablas y propósito**
Inventario	`materials` (catálogo: código, nombre, unidad, categoría, stock mínimo), `lotes` (número de lote, cantidad, costo unitario, fecha de vencimiento, estado *active*/*quarantined*/*consumed*, material y bodega), `bodegas` (ubicaciones, capacidad, estado y ruta o enlace de su imagen de fondo), `transferencias` (traslados entre bodegas).
Kardex	`movimientos`: lote, usuario, tipo (entrada o salida), cantidad, motivo, descripción y fecha.
Clasificación	`tags` y `material_tag` (etiquetas de materiales), `custom_field_definitions` (campos personalizados).
Compras	`vendors` (proveedores), `purchase_orders` y `purchase_order_items` (órdenes de compra y cantidades recibidas).
Seguridad y auditoría	`users` (con rol *admin* u *operario*), `audit_log` (cambios administrativos), `api_keys` (claves de proveedores cifradas).
Asistente	`chat_histories` (conversaciones por sesión), `llm_providers` (proveedores de modelos configurados), `llm_evaluaciones` y `llm_precios` (precisión medida y costo de cada modelo), `settings` (parámetros clave-valor, como el umbral FEFO de 15 días).
Operación	`notifications` (alertas), `kanban_items` (tablero de actividades).

9.4.1. Diseño del Kardex

La entidad central del diseño es el lote, no el material: la existencia de un material es la suma de las cantidades de sus lotes activos, y cada lote conoce su fecha de vencimiento. Esto es lo que permite aplicar FEFO. Las operaciones que cambian la cantidad de un lote desde el inventario —ingreso, consumo, despacho por escaneo, transferencia y ajuste de conciliación— registran un movimiento en la tabla `movimientos` con el usuario autenticado, el tipo, la cantidad, el motivo y la marca de tiempo. Las operaciones que tocan más de un lote (por ejemplo, un despacho FEFO que agota un lote y continúa con el siguiente, o una transferencia) se ejecutan dentro de una transacción de base de datos, para que el Kardex nunca quede con movimientos a medias.

Por regla acordada con la dirección, el Kardex es de solo inserción, y se protege en tres niveles: el modelo `Movimiento` rechaza modificar o eliminar un registro; dos disparadores (*triggers*) sobre `movimientos` lo impiden también fuera de la aplicación, y las llaves foráneas hacia `lotes` y `users` impiden borrar un lote o un usuario con historial. Los errores se corrigen con un movimiento de ajuste justificado.

9.5. Diseño del módulo de consulta en lenguaje natural

El asistente sigue la arquitectura de generación aumentada por recuperación (RAG) [8]: antes de llamar al modelo de lenguaje, el sistema recupera de la base de datos la información pertinente y la entrega al modelo como contexto, para que la respuesta se base en el inventario real y no en conocimiento general del modelo. Es una variante estructurada: la recuperación son consultas SQL parametrizadas elegidas según la intención, no una búsqueda semántica. El diseño tiene tres etapas:

1. **Clasificación de la intención** (`intencion`). Un modelo de lenguaje asigna la pregunta a una de trece intenciones o a una general (Tabla 9.4); las expresiones regulares de `classifyQuery` quedan como reserva cuando el modelo no responde o contesta algo fuera de las categorías, y un número de lote se reconoce siempre de forma exacta. Además, las palabras de la pregunta se comparan con los nombres y descripciones de los materiales para filtrar el contexto al material consultado.

2. **Construcción del contexto** (`buildRagContext`). Según la intención, se consulta la base de datos (lotes, movimientos, bodegas, valorización) y se arma un texto con los registros encontrados y el estado de las bodegas. Solo se consideran lotes activos con existencias, el texto empieza con la fecha del día y cada lote indica cuántos días le faltan para vencer, de modo que el modelo pueda responder preguntas como "¿qué vence esta semana?" sin calcular fechas por su cuenta.

3. **Generación de la respuesta**. El contexto, la instrucción de sistema (editable desde la configuración) y la pregunta se envían al proveedor de modelo de lenguaje configurado; la respuesta se muestra al usuario y se guarda en `chat_histories`.

Tabla 9.4: Intenciones reconocidas por el clasificador del asistente.

Intención	**Palabras que la activan en los patrones de reserva**	**Contexto que se recupera**
`stock_check`	cuántos hay, stock, cantidad, existencias	Lotes del material, por cantidad
`critical_alerts`	crítico, por vencer, alerta, urgente	Lotes dentro del umbral FEFO
`expiration`	vence, fecha de vencimiento, expira	Lotes por fecha de vencimiento
`location`	dónde, ubicación, bodega, almacén	Materiales por bodega
`valuation`	valor, precio, costo	Cantidad por costo unitario
`movements`	entró, salió, movimiento, historial, kardex	Movimientos con usuario y motivo
`batch_info`	lote HAR-01-...	Detalle de un lote
`summary`	resumen, panorama, todo el inventario	Resumen del inventario
`conciliation`	conciliar, ajuste, diferencia, descuadre	Datos para conciliación
`quarantine`	cuarentena, retenido, no conforme	Lotes en cuarentena
`low_stock`	mínimo, hace falta pedir, reponer, agotado	Insumos bajo su stock mínimo
`forecast`	cuándo se acaba, para cuántos días alcanza, punto de reorden	Proyección: consumo diario, cobertura y punto de reorden
`consumption`	consumo, cuánto gastamos, por qué bajó	Consumo por periodo y por motivo, comparado con el periodo anterior
`general`	(ninguna de las anteriores)	Lotes en orden FEFO

9.5.1. Proveedores de modelo de lenguaje

El controlador define un mapa de proveedores intercambiables —Ollama en el propio servidor, OpenCode y cualquier API compatible con la de OpenAI— y el administrador elige el proveedor y el modelo desde la configuración, con una palanca que muestra la precisión medida y el costo de cada modelo, y con claves de API almacenadas cifradas. Este diseño evita depender de un único proveedor: si el servicio externo no está disponible se puede cambiar a un modelo local, si el proveedor elegido falla responde un modelo rápido en la nube y, después, uno local, y si ningún modelo responde el sistema muestra en texto plano los datos que recuperó para la pregunta, en lugar de un error. La justificación de esta estructura desde el punto de vista de los patrones de diseño se presenta en el capítulo de Implementación.

9.6. Diseño de la interfaz de usuario

La interfaz se prototipó en Figma; las 18 vistas pueden consultarse en `https://www.figma.com/design/cFXxvgbB9VwW6a8CHlcIjD`. El commit del 14 de abril de 2026 (`38377c4`, "Reconstrucción Figma de Alta Fidelidad (18 vistas)") las trasladó a React; por eso varios componentes conservan el prefijo `Figma` en su nombre (`FigmaTablero`, `FigmaLabelPrint`, `FigmaQRScanner`, entre otros).

Los principios que guiaron el diseño de la interfaz fueron:

- **Navegación por tareas.** La barra lateral agrupa los módulos en tres secciones —Principal (Tablero, Inventario, Buscar), Análisis (Asistente, Reportes, Reabastecimiento, Kardex, Etiquetas, Escáner) y Gestión (Historial QR, Transferencias, Órdenes de compra, Conteo físico, Imprimir etiquetas, Alertas)— y el tablero ofrece accesos rápidos a las acciones más frecuentes (nuevo insumo, escanear, reporte, consumo rápido).
- **Lo urgente primero.** El tablero muestra en la parte superior los indicadores que requieren acción (lotes críticos por vencimiento, existencias, valorización) y, debajo, la actividad reciente del Kardex y la ocupación de las bodegas.
- **Uso en bodega desde el celular.** El diseño es adaptable a pantallas pequeñas: el menú lateral se reemplaza por una barra inferior con las cuatro acciones más usadas (tablero, inventario, escanear y asistente) y un panel con el resto de módulos, y el escáner usa la cámara del dispositivo, para que el personal pueda registrar movimientos sin ir a un computador.
- **Temas visuales.** Los colores se definen como variables de CSS, lo que permitió ofrecer seis temas seleccionables (*midnight-luxe*, *carbon-amber*, *nordic-steel*, *obsidian-teal*, *royal-plum* y *paper-light*), incluido uno claro. Cada tema define un pequeño conjunto de colores base (fondo, panel, borde, texto y acento) y a partir de ellos redefine la paleta completa que usan los componentes; así un tema no puede alterar a otro. Las imágenes, como las de las bodegas, provienen de la base de datos (archivo subido o enlace), no de rutas fijas en el código.

Capítulo 10

Implementación

Este capítulo describe cómo se construyó Pymetory: la organización del código, los patrones de diseño aplicados y las razones de cada uno, los problemas reales que aparecieron durante el desarrollo y cómo se resolvieron, la infraestructura en la que corre el sistema y el proceso de despliegue. La forma de gestionar las tareas del proyecto se presenta aparte, en el Capítulo 12.

10.1. Organización del código

El repositorio sigue la estructura estándar de Laravel, con el frontend en React dentro de `resources/js`. La Tabla 10.1 resume sus partes principales.

Tabla 10.1: Organización del código fuente de Pymetory (archivos al 2 de octubre de 2026, el mismo corte de las pruebas; incluye subcarpetas).

Carpeta	**Archivos**	**Contenido**
`app/Http/Controllers`	35	Controladores: inventario, consumo FEFO, transferencias, reportes, etiquetas, escáner, asistente, usuarios, ajustes, entre otros.
`app/Models`	15	Modelos Eloquent (`Material`, `Lote`, `Movimiento`, `Bodega`, ...).
`app/Console/Commands`	3	Revisiones programadas de alertas FEFO y de stock bajo, y el evaluador del asistente (`rag:evaluar`).
`app/Services`	2	Envío de alertas y proyección de reabastecimiento (`ProyeccionInventario`).
`database/migrations`	36	Definición versionada del esquema de la base de datos.
`resources/js/Pages` y `components`	19 y 48	Páginas Inertia y componentes React de la interfaz.
`tests`	46 clases y 11 archivos	Pruebas PHPUnit y la suite Playwright (Capítulo 11).

10.2. Patrones de diseño

Al construir sobre Laravel, Pymetory hereda patrones que el framework trae resueltos: Modelo-Vista-Controlador, Active Record a través de Eloquent, una cadena de *middleware* y un contenedor de servicios. Las decisiones propias del autor están en otros lugares: poner las reglas de despacho en el modelo del lote, escribir el eslabón de rol de la cadena, organizar el despacho FEFO como un procedimiento transaccional y configurar los proveedores del asistente en una tabla. Esta sección describe esos cuatro lugares [27]; para cada uno se indica la alternativa descartada, la razón de la elección y lo que cambió en el sistema.

10.2.1. Strategy por configuración: proveedores del asistente

Qué es. Una familia de algoritmos intercambiables entre los que se elige en tiempo de ejecución.

Dónde está. El `ChatLLMController` define una tabla de proveedores de modelo de lenguaje, cada uno con su dirección y su clave, y elige uno según la configuración del administrador (`app/Http/Controllers/ChatLLMCont` línea 668):

```
$endpoints = [
    'local'       => ['url' => ollama . '/api/chat', 'native' => true],
    'opencode'    => ['url' => 'https://opencode.ai/zen/v1/...', 'key' => $apiKey],
    'opencode-go' => ['url' => 'https://opencode.ai/zen/go/v1/...', 'key' => $apiKey],
    'external'    => ['url' => 'https://api.openai.com/v1/...', 'key' => $apiKey],
];
$sourceCfg = $endpoints[$llmSource] ?? [];
```

Es una aproximación a Strategy: los proveedores se intercambian por configuración, pero no hay una interfaz común con clases polimórficas, y el controlador distingue el caso local con una condición.

Alternativa descartada: una clase por proveedor con una interfaz común. Es la forma clásica del patrón. Con proveedores que comparten el mismo protocolo, una tabla bastaba; si se agregan proveedores con protocolos distintos, convendrá dar ese paso.

Qué cambió. Cambiar de proveedor principal es una modificación de configuración que no requiere tocar el controlador. Esto permitió pasar de modelos locales a proveedores en la nube y dejar en la versión final un modelo gratuito de Ollama Cloud como principal, con un modelo rápido en la nube y uno local como respaldos.

10.2.2. Chain of Responsibility: control de acceso

Qué es. Una petición pasa por una serie de manejadores, cada uno con una única responsabilidad, que pueden detenerla.

Dónde está. Cada ruta declara su cadena: `auth` (la sesión, de Laravel) y `role:admin`. El eslabón de rol es propio del proyecto (`app/Http/Middleware/CheckRole.php`, línea 19):

```
public function handle(Request $request, Closure $next, string $role): Response
{
    if (!$request->user() || $request->user()->role !== $role) {
        abort(403, 'No tienes permisos de ' . strtoupper($role) . '.');
    }
    return $next($request);
}
```

Alternativa descartada: una verificación única en cada controlador. Es válida con una sola regla de acceso, pero mezcla la seguridad con la lógica de negocio y obliga a repetirla.

Qué cambió. La política de acceso se lee en el archivo de rutas y se corrige en un solo lugar. Cuando en la revisión final se detectó que las rutas de usuarios y claves de API no exigían el rol de administrador, la corrección consistió en añadir `role:admin` al grupo de rutas (Sección 10.4).

10.2.3. Active Record: reglas del lote en el propio modelo

Qué es. Cada objeto representa una fila de la base de datos y contiene su persistencia y sus reglas de dominio.

Dónde está. El modelo `Lote` (`app/Models/Lote.php`) concentra las reglas que dependen de él: si está crítico por vencimiento (`getIsCriticalAttribute`, línea 82, con el umbral configurable de `diasCriticos()`), su valor (`getValorTotalAttribute`, línea 99), qué lotes pueden despacharse (*scope* `despachables`, línea 123) y por qué no puede salir un lote en particular (`motivoNoDespachable`, línea 177), por ejemplo porque está vencido o en cuarentena.

Alternativa descartada: Repository o Data Mapper. Son útiles cuando se necesita independencia del motor de persistencia o hay varias fuentes de datos. Pymetory usa un solo motor (MySQL), y una capa Repository sobre Eloquent solo reenviaría llamadas.

Qué cambió. Cada regla vive en un solo lugar. Cuando se descubrió que el tablero ignoraba el umbral configurado, bastó con cambiar el modelo para que tablero, inventario y alertas quedaran coherentes; y cuando se prohibió despachar lotes vencidos o en cuarentena, todos los puntos de despacho pasaron a usar `despachables`.

10.2.4. Transaction Script: despacho FEFO

Qué es. La lógica de una operación de negocio se organiza como un procedimiento único que la atiende de principio a fin.

Dónde está. `Lote::despacharFefo()` (`app/Models/Lote.php`, línea 141) es el único lugar que despacha en orden de vencimiento (el conteo físico aplica ese orden a los faltantes); lo usan el despacho de la interfaz (`ConsumptionController::consumeFefo()`) y el despacho múltiple. Dentro de una transacción, bloquea los lotes despachables del material, verifica el saldo, descuenta la cantidad lote por lote, marca como consumidos los que se agotan y registra cada salida en el Kardex:

```
return DB::transaction(function () use ($material, $cantidad, $motivo, ...) {
    $lotes = self::where('material_id', $material->id)->despachables()
        ->orderBy('expiration_date', 'asc')->lockForUpdate()->get();
    $disponible = (float) $lotes->sum('quantity');
    if ($disponible + 0.0005 < $cantidad) throw new StockInsuficiente($material,
        $disponible);
    $pendiente = $cantidad;
    foreach ($lotes as $lote) {
        if ($pendiente <= 0.0005) break;
        $sale = min((float) $lote->quantity, $pendiente);
        $lote->quantity -= $sale;
        if ($lote->quantity <= 0.001) { $lote->quantity = 0; $lote->status = 'consumed'; }
        $lote->save();
        Movimiento::create([ /* lote, usuario, salida, cantidad, motivo */ ]);
        $pendiente -= $sale;
    }
});
```

Alternativa descartada: Domain Model. Un modelo de dominio rico conviene cuando las reglas son numerosas e interdependientes. El despacho FEFO es un procedimiento lineal de unas treinta líneas.

El recorrido estuvo copiado en varios controladores; la revisión final lo dejó en un solo método.

Qué cambió. La transacción garantiza atomicidad: un despacho que toca varios lotes queda completo o no queda. Además, el bloqueo de filas (`SELECT ... FOR UPDATE`) evita que dos despachos simultáneos descuenten el mismo saldo.

10.3. Proyección de reabastecimiento

Los requerimientos RF-09 y RF-10 y el nivel de analítica predictiva del alcance se implementaron en el servicio `ProyeccionInventario`, que calcula todo a partir del Kardex, sin datos escritos en el código:

- **Consumo diario.** Promedio móvil simple: lo que salió del insumo por producción, venta, desperdicio o escáner en los últimos 28 días, dividido entre 28. Las transferencias no cuentan porque el insumo solo cambia de bodega, y los ajustes tampoco, porque corrigen el conteo y no reflejan demanda.
- **Cobertura y agotamiento.** La existencia activa dividida entre el consumo diario da los días que alcanza; sumados a la fecha de hoy, la fecha probable de agotamiento.
- **Punto de reorden.** Consumo diario × días de entrega del proveedor + stock mínimo, que hace las veces de inventario de seguridad (no se modela la variabilidad), según el modelo de punto de reorden descrito en la Sección 3.1.2. Los días de entrega los registra el administrador por insumo; si no están, el sistema no inventa el punto de reorden y lo indica. Un insumo está por reponer cuando su existencia llega a ese punto o baja de su mínimo; si no, se muestra la última fecha para pedir a tiempo.
- **Diagnóstico.** Para explicar una variación, el servicio agrupa las entradas y salidas del insumo por motivo en los últimos días y las compara con el periodo anterior de igual duración.

El promedio móvil supone que el consumo de los próximos días será como el de las últimas semanas: no anticipa temporadas ni cambios bruscos y no se validó contra el consumo real de la empresa, lo que se propone como trabajo futuro. La vista Reabastecimiento muestra estos valores y grafica el consumo por día, semana o mes, y el asistente los usa en dos intenciones nuevas (`forecast` y `consumption`). Once pruebas PHPUnit verifican los cálculos con fechas fijas, la exclusión de transferencias y ajustes, y que tres preguntas de las baterías se siguen clasificando igual; la clasificación de las 106 se comprobó una vez al agregar las intenciones.

10.4. Problemas enfrentados y cómo se resolvieron

La Tabla 10.2 reúne los problemas más significativos del desarrollo. Cada uno tiene como evidencia el commit que lo corrigió en el repositorio.

Tabla 10.2: Problemas del desarrollo y su solución.

Problema	Causa	Solución	Commit
El despliegue automático fallaba al actualizar el código en el servidor.	El repositorio del servidor usaba HTTPS sin credenciales y `git pull` pedía usuario.	Se usó un token efímero por ejecución; desde que el repositorio es público, el servidor actualiza sin credenciales.	`91732b6`

Problema	Causa	Solución	Commit
La compilación del frontend fallaba en integración continua.	Un archivo importado por la aplicación (`theme.ts`) no estaba versionado.	Se agregó el archivo al repositorio.	`90da6f5`
El asistente devolvía respuestas vacías con modelos que razonan antes de responder.	El razonamiento consumía todo el límite de tokens y dejaba vacío el contenido.	Se filtra el razonamiento y se reintenta con un límite mayor.	`1c9e162`
El asistente respondía con "la lista completa" del inventario.	Las expresiones regulares no soportaban tildes (UTF-8) y un material inexistente devolvía lotes de otros materiales.	Clasificación compatible con UTF-8 y respuesta explícita de "material no registrado".	`5c7c2ef`
Cualquier visitante podía registrarse como administrador.	El registro público aceptaba el rol enviado en la petición; además, las rutas de usuarios y claves no exigían el rol.	El registro creó solo operarios y las rutas quedaron bajo `role:admin`; en la revisión final se eliminó el registro público.	`a802cbb`, `9d0efb8`
Se podía despachar un lote en cuarentena o vencido, y el despacho por lote o por QR no aplicaba FEFO.	Solo el despacho automático ordenaba por vencimiento; los demás no validaban estado ni fecha.	Una sola regla en el modelo `Lote` para todo despacho; un vencido solo se da de baja como desperdicio.	`9d0efb8`
Borrar un usuario eliminaba su historial del Kardex.	Las llaves foráneas de `movimientos` tenían borrado en cascada.	El modelo rechaza modificar o borrar movimientos y las llaves pasaron a `restrict`.	`f53a7d0`
El inicio de sesión fallaba en producción.	Detrás del túnel de Cloudflare, Laravel generaba redirecciones `http` bloqueadas por el navegador.	Se declararon los proxies de confianza (`trustProxies`).	`f174905`
El tablero mostraba tendencias que no provenían de datos.	Valores de maqueta que quedaron en el controlador.	Tendencias calculadas contra 30 días atrás y rotación desde el Kardex.	`361d0af`
El umbral de vencimiento configurado no se reflejaba en el tablero.	El tablero usaba 15 días fijos; solo alertas y reportes leían el ajuste.	El modelo `Lote` lee el umbral configurado.	`66456a3`
Textos invisibles en algunos temas visuales.	La capa de temas reasignaba colores de utilidades como `bg-white` sin ajustar el texto.	Tokens de color por tema en lugar de colores fijos.	`75f694d`
Al corregir un tema visual se descomponía otro.	Más de cien reglas reasignaban clases de color una por una, varias con `!important`, y competían entre sí.	Cada tema define sus tokens y redefine la paleta completa de Tailwind; se eliminaron las reglas por clase.	`10787fd`
La pantalla de alertas mostraba avisos de ejemplo.	La vista era una maqueta con textos fijos; las alertas reales se guardaban pero nadie las leía.	La vista consulta la tabla `notificaciones`; se agregaron marcado como leída y la no repetición de avisos.	`10787fd`
El formulario de transferencias no listaba lotes.	Los datos enviados a la vista no incluían la bodega ni el número de lote que el formulario filtraba.	Se agregaron esos campos a los datos del inventario.	`10787fd`

Problema	Causa	Solución	Commit
El asistente respondió que ningún lote vencía en la semana cuando sí había.	El contexto incluía lotes ya consumidos, con fechas pasadas, y no indicaba la fecha del día.	Solo lotes con existencias, ventana de días según la pregunta y fecha de hoy en el contexto; prueba de regresión.	`ec76a78`
Recibir una orden de compra creaba el lote con vencimiento de "hoy más un año".	La recepción no pedía los datos del lote: fecha, número y bodega se fijaban en el código.	La recepción exige número de lote, vencimiento y bodega por ítem; se agregó el ingreso de lotes para insumos ya registrados.	`2987679`
El pan terminado aparecía siempre como crítico y la ocupación sumaba kilos con unidades.	Un único umbral de 15 días para productos que duran de 5 a 7 días; capacidad sin unidad.	Umbral FEFO por insumo y unidad de capacidad por bodega; ocupación global como promedio.	`de53e61`
El asistente acertó 34 de 50 consultas en la primera medición.	Contextos sin costos, sin ajustes de conciliación ni totales; lotes no reconocidos por su número real.	Contextos completos y clasificador revisado; la misma batería pasó a 49 de 50 (26-sep; 49 de 50 el 10-oct).	`48f092f`
Las listas largas de lotes por vencer salían cortadas.	El tope de 1 024 tokens de salida se aplicaba también a los modelos en la nube.	El tope solo rige para modelos que corren en el servidor.	`717677d`
"¿Qué se está poniendo viejo?" se respondía como una pregunta de existencias.	Decidían los patrones, que tomaron "hay que sacar" por una consulta de cantidad.	Clasifica primero el modelo; los patrones quedan de reserva.	`0afef8a`
Seis tipos de incumplimiento de WCAG 2.1 AA.	Contraste bajo, botones de solo ícono sin nombre y campos sin etiqueta.	Acento sólido por tema, nombres accesibles y etiquetas; cero incumplimientos automáticos en el tema por defecto.	`fb0e9bb`

Además de los defectos de código, la operación del servidor mostró dos problemas de recursos. Dos procesos de un proyecto ajeno a la tesis, alojado en el mismo servidor, llevaban seis días ocupando dos de los cuatro núcleos, lo que hacía muy lentos los modelos de lenguaje locales; se detuvieron y ese proyecto se retiró del repositorio de la tesis. Y al evaluar modelos, cargar varios a la vez agotó la memoria y dejó la aplicación sin servicio unos seis minutos; desde entonces Ollama tiene un límite de 14 GB y un solo modelo cargado a la vez, de modo que no puede afectar a la base de datos ni a la aplicación.

10.5. Infraestructura

El sistema corre en una instancia de Oracle Cloud de la capa gratuita *Always Free* [36]: procesador ARM de 4 núcleos, 24 GB de memoria y 200 GB de disco. La Figura 10.1 muestra sus componentes:

- **Acceso.** El dominio `app.pymetory.com` se publica mediante un túnel de Cloudflare, que establece la conexión desde el servidor hacia Cloudflare. El servidor no expone puertos de la aplicación a internet y el certificado TLS lo gestiona Cloudflare.
- **Aplicación y datos.** Laravel 13 sobre PHP 8.3, con MySQL 8.0 en el mismo servidor, ejecutado por PHP-FPM detrás de Nginx, que además entrega directamente los archivos estáticos (JavaScript,

estilos e imágenes). Al principio se usaba `artisan serve`, que atiende una petición a la vez; el cambio redujo la carga del inicio de sesión de varios segundos a unos 0,3 segundos.

- **Modelos de lenguaje.** El modelo principal es `gpt-oss:120b-cloud`, de la capa gratuita de Ollama Cloud. Si falla o responde vacío, contesta un modelo rápido en la nube (GLM-5.3-Flash, con la suscripción del autor a OpenCode Go; 2 s con el principal caído) y después uno local (`qwen3.5:9b`); si ninguno responde, el asistente muestra en modo texto los datos que recuperó de la base. En el servidor, el respaldo tardó 56 s con el modelo en frío y 33 s ya cargado, frente a cerca de un segundo del modelo en la nube; por eso es un respaldo y no el modelo principal. Ollama no puede usar más de 14 GB de memoria, y ante falta de memoria el sistema operativo lo detiene a él antes que a la base de datos o a la aplicación.
- **Ensayo de modelos locales.** El 27 de septiembre de 2026, `gemma4:12b` y `qwen3.5:9b` leyeron en el servidor a 7,5 y 13,9 tokens por segundo: unos 4 100 tokens les tomaron 9 y 5 minutos, y `qwen3:14b` no terminó en 15. Sin GPU no es viable entregar todo el inventario; el RAG entrega solo los datos filtrados para cada pregunta y la velocidad la da el modelo en la nube.
- **Tareas programadas.** Un *cron* ejecuta cada minuto el programador de Laravel, que revisa cada hora los lotes por vencer y el stock bajo.
- **Respaldos.** Cada noche se genera un volcado comprimido de la base de datos con `mysqldump`, y antes de cada cambio delicado —como la migración que hizo inmutable el Kardex el 25 de septiembre de 2026— se generó uno adicional. Todas las copias quedan en el mismo servidor (ver Trabajos Futuros).

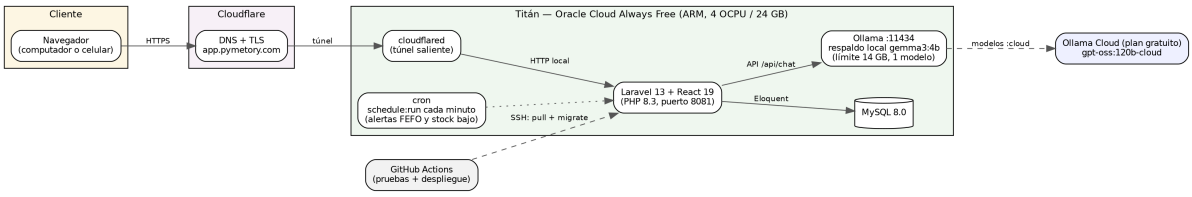


Figura 10.1: Infraestructura de producción de Pymetory.

10.6. Despliegue

El despliegue está automatizado con GitHub Actions (`.github/workflows/deploy.yml`). La estrategia de ramas fue de rama principal (*trunk-based*): los cambios pequeños entraban a `main`, protegida por este flujo, y los grandes se hicieron en ramas aparte (p. ej., `tesis/limite-80-paginas` y `contra-tesis/9oct`), fusionadas después de correr las pruebas. Cada cambio que llega a la rama principal recorre tres etapas:

1. **Revisión.** Verificación de sintaxis de los archivos PHP.
2. **Pruebas.** Instalación de dependencias, compilación del frontend y ejecución de la suite PHPUnit. Si una prueba falla, el despliegue no continúa.
3. **Despliegue.** Conexión por SSH al servidor, actualización del código, instalación de dependencias de producción, migraciones, regeneración de cachés y reinicio del servicio; el frontend compilado desde el mismo commit se sube aparte. Producción usa `APP_DEBUG=false`, para que un error no exponga detalles internos.

Capítulo 11

Pruebas

Este capítulo presenta las pruebas con que se verificó Pymetory: pruebas unitarias y de integración sobre el backend, pruebas de extremo a extremo sobre la interfaz, verificaciones en producción y el protocolo de pruebas con usuario. Las pruebas automatizadas corresponden a las ejecuciones del 2 de octubre de 2026; la medición final del asistente, al 10 de octubre, y las consultas de verificación en producción, al 25 de septiembre.

11.1. Estrategia

Las pruebas se organizaron en seis niveles:

1. **Unitarias y de integración (PHPUnit).** Verifican reglas de negocio, controladores y permisos ejecutando la aplicación contra una base de datos SQLite en memoria que se reconstruye en cada prueba.
2. **De extremo a extremo (Playwright [34]).** Manejan un navegador real contra la aplicación y recorren los casos del plan de pruebas, en escritorio y en celular.
3. **Verificación en producción.** Después de cada despliegue se comprueba en `https://app.pymetory.com` el funcionamiento de los cambios.
4. **Precisión del asistente.** Batería de 50 consultas sobre los datos de demostración en producción (45 insumos y sus lotes, generados a partir de la entrevista), con la respuesta correcta tomada de la base de datos (Sección 11.6).
5. **Usabilidad y accesibilidad.** Evaluación heurística con los principios de Nielsen y auditoría WCAG 2.1 AA (secciones 11.7 y 11.8).
6. **Pruebas con usuario.** Tareas reales ejecutadas por la administradora de la empresa del caso de estudio (Sección 11.9).

Las pruebas PHPUnit se ejecutan automáticamente en cada cambio que llega a la rama principal, y un fallo detiene el despliegue (Capítulo 10).

11.2. Pruebas unitarias y de integración

La suite PHPUnit tiene **178 pruebas en 46 clases**: 177 aprobadas y 1 omitida, con **638 aserciones**. La prueba omitida llama a un proveedor externo de modelos de lenguaje y solo se ejecuta cuando se configura una clave para ello.

En Laravel, la mayoría de estas pruebas son de integración: envían una petición HTTP y recorren rutas, *middleware*, controladores y base de datos. Algunas prueban un componente aislado, como el modelo `Movimiento` (inmutabilidad), el umbral del modelo `Lote` o el clasificador del asistente, y cumplen el papel de pruebas unitarias. La Tabla 11.1 las agrupa por área. La cobertura de líneas, medida con Xdebug el 9 de octubre tras retirar el código muerto, fue del 68 %; lo menos cubierto es el envío de alertas por Telegram y correo (34 %).

Tabla 11.1: Pruebas PHPUnit por área.

Área	**Pruebas**	**Clases y qué verifican**
FEFO	10	`FefoTest`, `ConsumptionFefoTest`, `FefoUmbralTest`, `ExpiredLoteRegressionTest`, `DashboardCriticosConsistencyTest`: orden de despacho por vencimiento, consumo que atraviesa varios lotes, lotes vencidos, umbral general y umbral propio de cada insumo.
Kardex	11	`KardexTest`, `KardexInmutableTest`, `KardexPaginadoTest`: registro de movimientos, rechazo de edición y borrado, protección en la base de datos, entrada al recibir órdenes de compra y paginación.
Seguridad y roles	20	`UserManagementTest`, `ApiKeyTest`, `ApiKeySeguridadTest`, `Auth\AuthenticationTest`, `SettingsTest`: acceso por rol, ausencia de registro público, claves cifradas que solo se envían al servidor oficial de cada proveedor y prueba de conexión sin gastar tokens.
Asistente	44	`ChatLLMTest`, `ChatRagContextTest`, `RespaldoLocalTest`, `HistorialChatTest`, `ConsumoTokensTest`, `ClasificadorYRespaldoTest`, `EspectroModelosTest`, `ChatLLMLiveTest` (omitida): clasificación con el modelo y con patrones de reserva, respaldos, tokens por consulta, tope de salida, palanca de modelos, material inexistente, contexto filtrado, vencimientos de la semana, valorización, conciliación y lotes por su número.
Operación de inventario	27	`InventoryTest`, `InventoryStoreTest`, `TransferTest`, `PurchaseOrderTest`, `QRScanTest`, `LabelTest`, `TagTest`, `BodegaGestionTest`, `CategoriasTest`: categorías sin duplicados; bodegas con imagen (solo administrador) y ocupación sin mezclar unidades; recepción de órdenes con lote, vencimiento y bodega reales, e ingreso de lotes de insumos existentes.

Área	Pruebas	Clases y qué verifican
Reabastecimiento	11	`ProyeccionInventarioTest`: consumo diario, cobertura, fecha de agotamiento y punto de reorden con fechas fijas; exclusión de transferencias y ajustes; consumo por semana; comparación por motivo; y clasificación de las preguntas nuevas sin cambiar la de las baterías.
Reportes	14	`ReportExportTest`: los tipos de reporte en PDF, CSV y Excel (archivo .xlsx válido con las mismas columnas que el CSV) y sus filtros.
Kanban	7	`KanbanTest`, `KanbanReorderTest`, `KanbanProyectoSeederTest` (cada tarea terminada del tablero del proyecto cita su issue o su commit).
Flujos completos	5	`Flujos\*`: autenticación, inventario, exportación, escaneo QR y consulta al asistente de principio a fin.
Alertas, esquema y servicios	8	`NotificacionesTest`, `AlertServiceTest`, `MigrationsSchemaSanityTest` y pruebas base del framework: bandeja de alertas por usuario, marcado como leída y no repetición de avisos.
Reglas de despacho y acceso	21	`ReglasDespachoYSeguridadTest`: lotes en cuarentena o vencidos no se despachan (salvo baja por desperdicio), FEFO también en el despacho por lote y por QR, órdenes de compra por rol, límite de intentos de inicio de sesión y monitor del servidor solo para administradores.
Total	**178**	177 aprobadas, 1 omitida, 638 aserciones.

11.3. Pruebas de extremo a extremo

La suite Playwright tiene **61 pruebas** y las **61 aprobaron**. Se ejecutan en Chromium con una ventana de escritorio de 1366×768 y, para las pruebas móviles, con una pantalla de 390×844.

Además de la suite, el repositorio conserva diez *scripts* de diagnóstico que se ejecutan contra producción. No se cuentan como pruebas porque dependen de los datos vivos del servidor y no son repetibles.

11.4. Defectos encontrados

Las pruebas y la verificación en producción encontraron defectos que se corrigieron antes de cerrar este documento; los más relevantes, con su causa y su commit, están en la Tabla 10.2: el registro público que permitía crear administradores, el borrado en cascada del Kardex y el asistente que devolvía lotes de otros materiales. La última ejecución de extremo a extremo y la revisión módulo por módulo previa al manual encontraron otros cinco, también corregidos: campos sin atributo `name` tras el rediseño del ingreso, bodegas sin identificador en el tablero, avisos de ejemplo en la pantalla de alertas, transferencias sin lotes y lotes ya consumidos en el contexto del asistente (este último, cubierto por una prueba de regresión).

11.5. Verificación en producción

Sobre la versión desplegada se verificó que el operario recibe acceso denegado (HTTP 403) en la gestión de usuarios y de claves de API, que el administrador sí accede, y que la aplicación no muestra información

Tabla 11.2: Pruebas de extremo a extremo.

Conjunto	**Pruebas**	**Qué cubren**
Plan de casos	34	17 casos positivos (ingreso, tablero, inventario, búsqueda, asistente, reportes PDF y CSV, Kardex, etiquetas, escaneo QR, transferencias, órdenes de compra, kanban) y 17 negativos (credenciales inválidas, datos vacíos o duplicados, stock insuficiente, QR corrupto, operario sin permisos, entre otros).
Humo (*smoke*)	5	Ingreso, tablero, kanban, asistente y ajustes responden sin errores.
Estados de la interfaz	5	Los módulos principales se renderizan.
Móvil	4	Tablero e inventario sin desbordes en 390 px.
Tablero, kanban y asistente	10	Indicadores, arrastrar y soltar tarjetas y respuestas del asistente.
Recorrido de sustentación	1	Ingreso, inventario, reporte PDF, kanban y pregunta al asistente con respuesta en kilogramos.
Dominio principal	1	El dominio pymetory.com redirige correctamente.
Reabastecimiento	1	La proyección carga con datos, el filtro de insumos por reponer y el cambio de periodo del gráfico.
Total	**61**	**61 aprobadas.**

interna ante un error. Con el modelo `gpt-oss:120b-cloud` se hicieron tres consultas sobre los datos de demostración (25 de septiembre de 2026): «¿Cuánta harina de trigo hay?» devolvió 6 400,83 kg en 0,7 s, la suma que da una consulta directa a la base (5 810,85 kg panificable más 589,98 kg integral); «¿Qué lotes vencen esta semana?» listó los lotes con su fecha en 1,3 s, y «¿Cuánto queso hay?» respondió que ese material no está registrado, sugiriendo los parecidos, en 1,1 s.

11.6. Precisión del asistente

11.6.1. Método

El pronóstico y el consumo por periodo se verificaron con pruebas unitarias, no con estas consultas. La batería tiene 50 consultas en español repartidas en 11 grupos (existencias, ubicación, vencimiento de un insumo, vencimientos por periodo, lotes críticos, valorización, movimientos, información de un lote, conciliación, resumen e insumos que no existen). Para cada consulta se fijaron de antemano la intención esperada y el criterio de acierto (por ejemplo, en los vencimientos por periodo, mencionar exactamente los lotes de la ventana). La respuesta correcta se calcula con una consulta directa a la base de datos en el momento de preguntar, y la respuesta del asistente se obtiene por el mismo flujo que usa la interfaz. El criterio comprueba que aparezca la cifra esperada, no que el resto de la respuesta carezca de cifras inventadas; eso lo revisó la auditoría manual. La batería se ejecuta con `php artisan rag:evaluar`, que guarda preguntas, respuestas y calificación; las mediciones reportadas están en `docs/evaluaciones`.

Para evitar que las correcciones se ajustaran solo a estas preguntas se usaron dos conjuntos adicionales: uno de validación con 20 consultas sobre otros insumos y con otras redacciones, y una prueba ciega de 10 consultas que se guardó en el repositorio antes de corregir el asistente. Se ejecutó por primera vez el 25 de septiembre, tras la primera corrección, que agregó patrones que aparecen en sus preguntas (por ejemplo, “plata” e “invertida”, commit `48f092f`); por eso su resultado es optimista. Todas las mediciones

se hicieron en producción con el modelo `gpt-oss:120b-cloud`.

Durante la primera medición se encontraron dos fallos del evaluador, no del asistente: no reconocía fechas y códigos escritos con guiones tipográficos ni algunas frases válidas para decir que un insumo no existe. Se corrigió el evaluador y se volvieron a calificar las mismas respuestas guardadas, sin volver a preguntar; esa es la línea base que se reporta.

11.6.2. Resultados

La Tabla 11.3 es el resultado definitivo: la cuarta medición del 10 de octubre de 2026, en producción, con la versión desplegada y el modelo `gpt-oss:120b-cloud`. Las tres anteriores de ese día (95, 98 y 99 de 106) llevaron a tres correcciones del manejo de lotes vencidos (`d103566`, `505f8ec`, `c62ef6a`). Antes de corregir, la batería daba 34 de 50. Las preguntas y sus criterios se fijaron el 25 de septiembre (`af3cde3`, `31a2b75`), antes de corregir el asistente. Las cuatro mediciones del 10 de octubre están en `docs/evaluaciones`.

Tabla 11.3: Precisión del asistente: resultado definitivo (corte del 10 de octubre de 2026).

Conjunto	**Aciertos**	**Observación**
Batería (50 consultas)	49/50 (98 %)	Usada para corregir; es la cifra que cita el Resumen.
Validación (20)	20/20 (100 %)	Otros insumos y redacciones; sus fallos también se revisaron.
Prueba ciega (10)	8/10 (80 %)	Ya no es independiente: sus fallos se revisaron. IC 95 % (Wilson): 49–94 %.
Operario coloquial (18)	17/18	Escritas por el autor imitando a un operario; usadas para corregir.
Operario, prueba ciega (8)	8/8 (100 %)	Ya no es independiente, igual que la anterior. IC 95 %: 68–100 %.
Total	**102/106**	Tiempo mediano de respuesta: 1,1 segundos.

El Resumen cita la batería principal (49 de 50) porque es el indicador que fijó el anteproyecto; el total de los cinco conjuntos es 102 de 106. Las dos pruebas ciegas se escribieron antes de corregir, pero sus fallos se revisaron después, así que ninguna medición sobre ellas es independiente: son una estimación optimista, no una prueba de generalización.

De los cuatro fallos, tres son discutibles (el costo del aceite se dio por litro, el "último movimiento" se respondió con la fecha y el "panorama" con la ocupación) y uno es real (una pregunta de alertas omitió los lotes por vencer). Una auditoría manual de las 106 respuestas del 1 de octubre, no de la medición final, coincidió con el evaluador en 99 (93 %), halló cinco aciertos que eran errores y ninguna cifra inventada en las 20 respuestas con cuentas verificables (`docs/evaluaciones/auditoria-manual-20261009.md`). Otros nueve modelos, con el mismo RAG, acertaron entre 100 y 105 de 106: con esta batería no se detectan diferencias entre modelos (estudio de costos, anexo digital en `https://tesis.pymetory.com/estudio-de-costos.pdf`).

11.7. Evaluación heurística

Se recorrieron las 17 pantallas de la aplicación, en escritorio y en celular, contra las 10 heurísticas de Nielsen [16], y cada hallazgo se verificó en la interfaz y en el código. La severidad se calificó con la escala de Nielsen, de 0 (no es problema) a 4 (catastrófico). La hizo un solo evaluador, con rol de administrador, de

modo que no reemplaza la prueba con usuario. El informe completo, con la pantalla y el elemento de cada hallazgo, está en `docs/evaluaciones/evaluacion-heuristica-nielsen.md`. La vista Reabastecimiento (Sección 10.3) se construyó después de esta evaluación y no forma parte de ella.

Tabla 11.4: Hallazgos de la evaluación heurística por severidad.

Severidad	**Hallazgos**	**Corregidos**
4 — catastrófico	1	1
3 — mayor	8	8
2 — menor	4	4
1 — cosmético	1	0
Total	**14**	**13**

El hallazgo catastrófico era la pantalla de alertas, que mostraba avisos de ejemplo escritos en el código en lugar de las alertas que genera el sistema. Entre los mayores estaban borrar órdenes de compra sin confirmación, mensajes técnicos de error, un escáner que exigía escribir el contenido del QR y unidades fijas en kilogramos. El único pendiente es cosmético: algunas etiquetas usan mayúsculas sostenidas con espaciado amplio en textos pequeños.

11.8. Accesibilidad

11.8.1. Revisión automática

Se auditaron las 17 pantallas en escritorio y en celular (34 vistas) con axe-core [41] y las reglas de WCAG 2.1 niveles A y AA [17]. La primera auditoría, con el tema visual por defecto, encontró seis reglas incumplidas: contraste de color insuficiente (41 elementos), enlaces y botones sin nombre accesible (44), campos de formulario y listas desplegables sin etiqueta (20) y zonas con desplazamiento que no se alcanzaban con el teclado (3). Tras corregirlas, la misma auditoría sobre producción no encontró incumplimientos en ninguno de los seis temas visuales (204 vistas en total). También se auditaron once ventanas y formularios que solo aparecen tras una acción del usuario (ficha de insumo, ingreso de lote, conciliación, creación de insumos, bodegas y órdenes, recepción, filtros, consumo FEFO y menú del celular); sus campos sin etiqueta y un botón con poco contraste se corrigieron. La vista Reabastecimiento, agregada después, se auditó en el entorno local con los seis temas en escritorio y en celular (12 vistas): en el celular su tabla desplazable no se alcanzaba con el teclado; se corrigió y quedó sin incumplimientos.

11.8.2. Revisión manual

Las herramientas automáticas no detectan todos los problemas de accesibilidad, así que se comprobaron por separado cinco criterios sobre todas las pantallas: la aplicación se puede usar sin mouse (inicio de sesión completo con el teclado, criterio 2.1.1); todos los elementos recorridos con la tecla Tab muestran un indicador de foco visible (2.4.7); a 320 píxeles de ancho ninguna pantalla obliga a desplazarse horizontalmente (1.4.10); cada pantalla tiene título (2.4.2) y el documento declara el idioma español (3.1.1). Los cinco se cumplen. Quedan por verificar con un lector de pantalla real el orden de lectura y los anuncios de los cambios dinámicos.

11.9. Pruebas con usuario

Se hizo con la administradora, con consentimiento informado escrito, en dos sesiones grabadas: cinco tareas derivadas de los requerimientos, el cuestionario SUS [42] y preguntas abiertas, con la guía rápida ya incluida en la aplicación. Las tareas constan en los registros del servidor.

Sesión piloto (8 de octubre, datos de demostración). Se completaron el ingreso de un lote, la consulta al asistente y el reporte en PDF, pero no el consumo por FEFO ni el ajuste por conteo físico: la pantalla de consumo enviaba el identificador del lote en lugar del insumo y el servidor respondía con una página de error. Ninguna prueba automática lo detectó; se corrigió y se agregó una prueba de extremo a extremo con las cinco tareas.

Evaluación (9 de octubre, datos reales). En una instancia con el catálogo real, repitió sin errores el ingreso, el consumo por FEFO (con el lote correcto) y el ajuste: las cinco tareas quedaron completadas. Además confirmó por escrito los cuatro puntos de la Sección 8.4.

SUS. Las hojas de la sesión piloto se extraviaron y el cuestionario se respondió por mensaje después de la segunda sesión: 5, 3, 5, 1, 5, 2, 5, 1, 5 y 1, que dan **92,5** sobre 100 (referencia: 68). La única respuesta intermedia fue a «encontré el sistema innecesariamente complejo». Con una sola participante, que ya conocía el sistema, el puntaje es indicativo y no generalizable.

11.10. Cumplimiento del indicador del objetivo específico 4

El anteproyecto fijó para el objetivo 4 un indicador con varios componentes. La Tabla 11.5 muestra su estado a la fecha.

Tabla 11.5: Estado del indicador del objetivo específico 4.

Componente del indicador	**Evidencia**	**Estado**
111 casos de prueba ejecutados	239 pruebas automatizadas (178 PHPUnit y 61 E2E) y 106 consultas al asistente (unidades distintas a las del anteproyecto).	Cumplido
Tasa de éxito global $\geq 95\,\%$	238 de 238 automatizadas aprobadas (una omitida por requerir un servicio externo); en el asistente, la ciega (8/10) queda por debajo.	Cumplido en las automatizadas
Casos de uso automatizados con Playwright	34 casos cubren 7 de 9 casos de uso (UC7 y UC8, solo negativos); UC6 y UC9, con PHPUnit.	Cumplido parcialmente
7 escenarios de despacho FEFO	10 pruebas FEFO, más despacho por lote y por QR (Tabla 11.1).	Cumplido
50 consultas en español para medir la precisión del asistente	Batería de 50: 49 correctas (98 %); prueba ciega: 8 de 10 (Tabla 11.3).	Cumplido
Evaluación heurística de Nielsen	14 hallazgos, 13 corregidos (Tabla 11.4).	Cumplido
Auditoría de accesibilidad WCAG AA	Sin incumplimientos automáticos (34 vistas y Reabastecimiento); cinco criterios manuales cumplidos (Sección 11.8).	Cumplido
Prueba con usuario	Cinco tareas completadas en dos sesiones; SUS de 92,5 (Sección 11.9).	Cumplido

Capítulo 12

Metodología de gestión del proyecto

Este capítulo describe cómo se gestionó el trabajo de desarrollo: la herramienta y el flujo con que se movieron las tareas, los criterios con que una tarea se consideraba lista para empezar y terminada, y los criterios de aceptación de los requerimientos.

12.1. Enfoque de trabajo

El anteproyecto planteó un enfoque ágil inspirado en Scrum (Capítulo 4). En la práctica, al ser un proyecto con un solo desarrollador, las ceremonias de Scrum pensadas para equipos (planeación de *sprint*, reunión diaria, retrospectiva grupal) se simplificaron y el trabajo se gestionó con un tablero Kanban [43]: las tareas avanzan por columnas a medida que cambian de estado, y el ritmo lo marcaron las reuniones de seguimiento con el director del trabajo de grado, en las que se revisaba el avance y se acordaban los siguientes pasos.

12.2. Tablero en GitHub Projects

Las tareas se registraron en GitHub Projects [11], en el proyecto "Gestor de Inventario de Materia Prima con Módulo LLM", vinculado al repositorio del código. El tablero tiene cinco columnas:

- **Backlog**: tareas identificadas que aún no se han preparado para empezar.
- **Ready**: tareas que cumplen la *Definition of Ready* y pueden tomarse.
- **In progress**: tareas en desarrollo.
- **In review**: tareas terminadas en código que se están verificando.
- **Done**: tareas que cumplen la *Definition of Done*.

Al 25 de septiembre de 2026 el proyecto contaba con 130 tareas: 75 en *Done*, 3 en *Ready* y 52 en *Backlog*. Buena parte de las tareas del *Backlog* corresponden a mejoras identificadas al comparar el sistema con productos comerciales, que quedaron fuera del alcance de este trabajo (Capítulo 13). La Figura 12.1 muestra la vista de tablero.

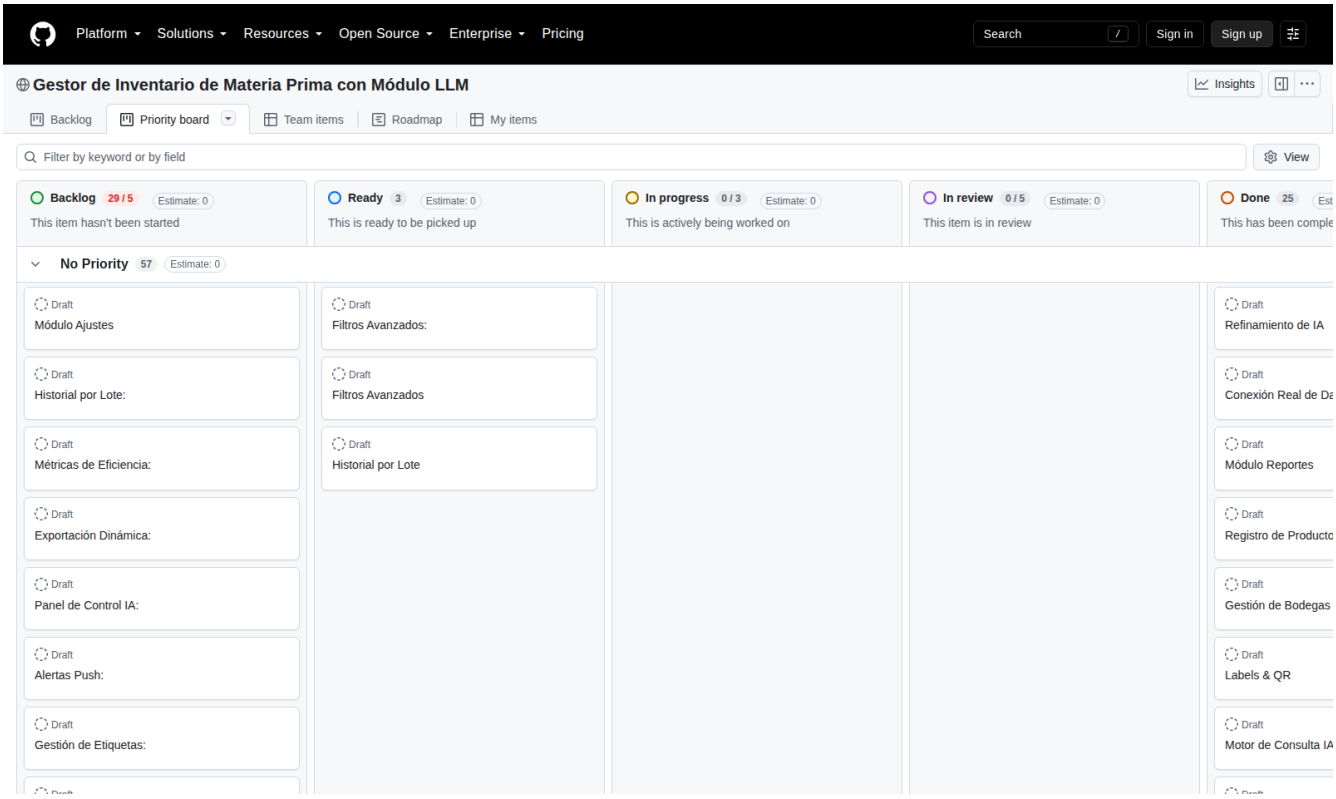


Figura 12.1: Tablero del proyecto en GitHub Projects (vista *Priority board*, 25 de septiembre de 2026).

12.3. Flujo de una tarea

Cada tarea recorrió el mismo camino:

1. Se registra en *Backlog* con un título que describe el resultado esperado; cuando corresponde a un requerimiento, se indica su identificador (por ejemplo, `[RNF-01]`).
2. Cuando cumple la *Definition of Ready*, pasa a *Ready*.
3. Al empezar a trabajarla pasa a *In progress*; el trabajo se hace en el repositorio con commits descriptivos.
4. Con el código terminado pasa a *In review*: se ejecutan las pruebas automatizadas y se verifica manualmente el comportamiento.
5. Cuando cumple la *Definition of Done* pasa a *Done*.

12.4. Criterios de aceptación

Una tarea se aceptaba cuando cumplía los criterios de aceptación del requerimiento que implementaba. Los criterios se expresan en la forma *dado / cuando / entonces*, y los principales están respaldados por una prueba automatizada (Capítulo 11). La Tabla 12.1 muestra ejemplos para las reglas de negocio de la dirección.

Tabla 12.1: Ejemplos de criterios de aceptación y la prueba que los verifica.

RF	Criterio de aceptación	Prueba
RF-04	**Dado** un material con dos lotes activos que vencen en fechas distintas, **cuando** se consume una cantidad, **entonces** se descuenta primero del lote que vence primero.	`ConsumptionFefoTest`, `FefoTest`
RF-05	**Dado** un movimiento registrado en el Kardex, **cuando** alguien intenta modificarlo o eliminarlo, **entonces** el sistema lo rechaza y el registro permanece igual.	`KardexInmutableTest`
RF-06	**Dado** un usuario con rol operario, **cuando** intenta gestionar usuarios o claves de API, **entonces** recibe acceso denegado (HTTP 403).	`UserManagementTest`
RF-08	**Dado** un umbral de días configurado por el administrador, **cuando** un lote vence dentro de ese umbral, **entonces** aparece como crítico en el tablero y en el inventario.	`FefoUmbralTest`
RF-12	**Dado** un material que no está registrado, **cuando** se pregunta por él al asistente, **entonces** la respuesta indica que no está registrado y no muestra lotes de otros materiales.	`ChatRagContextTest`

12.5. Definition of Ready y Definition of Done

La *Definition of Ready* (DoR) establece cuándo una tarea está lista para empezar, y la *Definition of Done* (DoD) cuándo puede darse por terminada. En un equipo, la DoD suele exigir pruebas unitarias y de integración, revisión del código por otra persona y despliegue; aquí había un solo desarrollador, que además probaba, así que no hubo revisión por pares. Los criterios aplicados en la práctica fueron:

Definition of Ready. Una tarea pasa a *Ready* cuando:

- su título describe el resultado esperado, no la actividad;
- está asociada a un requerimiento o a una observación de la dirección;
- tiene al menos un criterio de aceptación verificable;
- no depende de otra tarea sin terminar; y
- es lo bastante pequeña para completarse entre dos reuniones de seguimiento.

Definition of Done. Una tarea pasa a *Done* cuando:

- las pruebas unitarias y de integración (PHPUnit) pasan, incluidas las nuevas que cubren el cambio;
- se hicieron pruebas manuales del comportamiento en el navegador;
- el cambio está en la rama principal del repositorio en GitHub y, desde junio de 2026, el flujo de integración continua terminó sin errores; y
- el cambio está desplegado en producción (`https://app.pymetory.com`) y se verificó que funciona allí.

Desplegar directamente a producción, sin entorno intermedio, no es la práctica habitual; aquí el riesgo se redujo porque la integración continua detiene el despliegue si una prueba falla, se respaldó la base de datos antes de la migración que modificó el Kardex y cada despliegue se verifica en producción antes de cerrar la tarea.

Capítulo 13

Trabajos Futuros

Las propuestas se agrupan en cinco frentes. El primero recoge lo que quedó pendiente del caso de estudio y es el más directamente útil para la empresa; los demás amplían el sistema en infraestructura, inteligencia artificial, funcionalidades de negocio, y experiencia de usuario, seguridad e investigación. En cada caso se indica qué existe hoy, para distinguir lo que falta de lo que ya se construyó.

13.1. Requerimientos del caso de estudio pendientes

Del levantamiento de requerimientos (Capítulo 8) quedaron por completar:

- **Desglose del consumo por destino (RF-15)**: registrar cuánto se usa en esponja, espolvoreo y cada línea de producto, que es la causa de las diferencias que la empresa detecta en el cierre.
- **Factura y ficha de calidad en el ingreso (RF-03)** y un **registro de no conformidades en la recepción (RF-14)**. Hoy el proveedor queda asociado a la orden de compra y el lote admite el estado de cuarentena (aún sin una acción en la interfaz para asignarlo), pero la factura, la ficha y la novedad no tienen un registro propio.
- **Registro de los códigos de barras de fábrica** de todos los insumos: el sistema ya los lee (RF-13), pero en la instancia de evaluación solo se registraron los de dos productos.

13.2. Infraestructura y despliegue

- **Copias de seguridad fuera del servidor**: el volcado nocturno de la base de datos y los respaldos manuales previos a cambios delicados (Capítulo 10) quedan en el mismo servidor, de modo que una falla de la instancia los perdería. Se propone enviarlos a otro almacenamiento de capa gratuita (por ejemplo, OCI Object Storage), incluir los archivos subidos y probar periódicamente la restauración.
- **Integración continua más estricta**: GitHub Actions ya revisa la sintaxis, ejecuta PHPUnit y despliega solo si las pruebas pasan; faltan la suite Playwright contra una instancia temporal y el análisis estático (PHPStan y ESLint) como condición para desplegar.
- **Contenedores Docker**: actualizar el `Dockerfile` y el `docker-compose.yml` del repositorio, que conservan valores de una etapa anterior, y validarlos en la integración continua.
- **Multi-tenencia**: aislar los datos de varias PYMEs en una sola instancia mediante un identificador de inquilino (`tenant_id`).

- **Modelo local afinado**: el respaldo local `qwen3.5:9b` tarda de 33 a 56 segundos por respuesta (Capítulo 10); afinar un modelo más pequeño con consultas del dominio, o usar un servidor con GPU, lo haría más rápido.

13.3. Inteligencia artificial y RAG

- **Consultas con herramientas**: que el modelo pida los datos que necesita mediante funciones de solo lectura sobre la base de datos, con el sistema como filtro de seguridad y una verificación automática de que cada cifra de la respuesta aparezca en los datos consultados.
- **Búsqueda semántica con embeddings**, para responder por significado ("¿qué grasas tenemos?") sin conocer el nombre exacto de cada insumo.
- **Memoria conversacional en la recuperación**: el modelo recibe los dos últimos turnos y, si la pregunta dice "eso", se reutilizan los insumos del turno anterior; falta heredar la intención, para que "¿y de azúcar?" tras "¿cuánta harina queda?" se entienda, y medir conversaciones largas.
- **Mejor pronóstico y detección de anomalías**: la proyección actual es un promedio móvil de cuatro semanas (Sección 10.3), que no anticipa temporadas ni tendencias; se propone probar modelos de series temporales con el historial del Kardex y señalar consumos atípicos, entradas duplicadas o ajustes frecuentes.
- **Evaluación continua del asistente**: programar la batería de `rag:evaluar` (Sección 11.6) y alertar cuando la tasa de acierto baje de un umbral.

13.4. Funcionalidades de negocio

- **Cotizaciones** previas a las órdenes de compra, que hoy ya cubren proveedor, aprobación y recepción con lote, vencimiento y bodega.
- **Catálogo de categorías** propio de cada empresa, para renombrarlas y fusionarlas; hoy la categoría es un texto del insumo.
- **Integración con proveedores** (precios, disponibilidad y orden de reposición al cruzar el punto de reorden).
- **Módulo de producción (BOM)**: recetas que descuenten ingredientes por FEFO al producir y calculen el costo real; resolvería buena parte del RF-15.
- **Contabilidad integrada** y **facturación electrónica DIAN**.

13.5. Experiencia de usuario, seguridad e investigación

- **Registro sin conexión**: la aplicación ya funciona en el celular y su *Service Worker* guarda en caché los archivos estáticos; falta registrar movimientos sin red y sincronizarlos respetando FEFO y la inmutabilidad del Kardex.
- **Notificaciones *push*** del navegador, además de las alertas en la aplicación, Telegram y correo; **identidad visual por empresa** conservando el contraste WCAG 2.1 AA de los seis temas; **tutoriales interactivos** y **consultas por voz**.

- **Accesibilidad**: avanzar al nivel AAA y hacer la revisión con lector de pantalla, que las pruebas automáticas no reemplazan.
- **Seguridad**: autenticación de dos factores para administradores; ampliar `audit_log` a inicios de sesión, intentos fallidos y consultas al asistente; extender el cifrado, que hoy cubre las claves de API, a otros datos reservados.
- **Investigación**: repetir la prueba con usuario (Capítulo 11) en al menos diez PYMEs con la escala SUS; comparar modelos de lenguaje con la misma batería de `rag:evaluar`; medir el tiempo que ahorra el asistente; y documentar el método de desarrollo con asistentes de IA y sus controles.

13.6. Hoja de ruta priorizada

La Tabla 13.1 presenta un orden sugerido. La prioridad combina el impacto esperado para una PYME como la del caso de estudio con la factibilidad, entendida como los recursos, las dependencias técnicas y el tiempo que requiere cada mejora.

Tabla 13.1: Hoja de ruta priorizada de trabajos futuros.

Prioridad	Trabajo Futuro	Impacto	Factibilidad
1	Copias de seguridad fuera del servidor	Alto	Alta
2	Desglose del consumo por destino (RF-15)	Alto	Alta
3	Pronóstico con series temporales y detección de anomalías	Medio	Media
4	Pruebas de extremo a extremo y análisis estático en la integración continua	Medio	Alta
5	Memoria conversacional en la recuperación	Medio	Alta
6	Registro sin conexión en la bodega	Medio	Media
7	Módulo de producción (BOM)	Medio	Media
8	Verificación automática de las cifras de cada respuesta	Medio	Media
9	Arquitectura multi-tenencia	Alto	Baja
10	Integración con facturación electrónica DIAN	Alto	Baja

Capítulo 14

Conclusiones

14.1. Conclusiones

14.1.1. Cumplimiento de los objetivos

- **Objetivo general.** Se construyó y desplegó en `https://app.pymetory.com` un sistema web de gestión de inventario de materia prima para PYMEs, con trazabilidad por lote, despacho FEFO, Kardex inmutable, control de acceso por roles y un asistente que responde en español con los datos reales de la base de datos. Su efecto en la operación de la empresa (conciliación, diferencias de cierre, faltantes) no se midió.

- **Objetivo específico 1 — Entender las necesidades de una PYME (cumplido con ajustes).** La entrevista con la administradora de una panificadora del Valle del Cauca mostró que el problema de la empresa no son las pérdidas por vencimiento, sino un inventario contable que no coincide con la bodega, una conciliación manual y un reabastecimiento sin datos. De ella se derivaron 18 requerimientos funcionales y 4 no funcionales (Capítulo 8); de los funcionales, 15 quedaron implementados, 2 de forma parcial y 1 se propone como trabajo futuro. Se documentaron los 9 casos de uso que pedía el indicador (Anexos). Frente al indicador, hubo una entrevista y no varias, 4 reglas de la dirección en lugar de 10, tablas trazables en lugar de un documento IEEE 830, y no hay constancia de la validación de los casos de uso por la empresa.

- **Objetivo específico 2 — Diseñar el sistema (cumplido).** El diseño (Capítulo 9) incluye la arquitectura, la justificación de cada herramienta, un modelo de datos de 21 tablas del dominio (el indicador pedía al menos 10), diagramas UML de los cuatro procesos críticos —ingreso, despacho FEFO, conciliación y consulta al asistente— y un clasificador de 14 categorías de intención (13 intenciones y una general), frente a las 10 del indicador. Las 18 vistas del prototipo en Figma (enlace en la Sección 9.6) se trasladaron a React (commit `38377c4`, 14 de abril de 2026).

- **Objetivo específico 3 — Construir el sistema (cumplido).** El sistema está en producción con sus módulos operativos, y la rama principal del repositorio público tenía 205 commits el 2 de octubre de 2026, frente a los 50 del indicador. Con la proyección de reabastecimiento, el sistema cubre también el nivel de analítica predictiva del alcance (Sección 10.3). La implementación (Capítulo 10) describe cuatro decisiones de diseño propias, cada una con su alternativa descartada y justificada.

- **Objetivo específico 4 — Verificar el sistema (cumplido parcialmente).** Se ejecutaron 239 pruebas automatizadas (178 PHPUnit y 61 de extremo a extremo), por encima de los 111 casos del indicador, y todas las ejecutadas aprobaron. El asistente respondió correctamente 49 de 50 consultas

de la batería (34 de 50 antes de las correcciones), 8 de 10 y 8 de 8 en dos conjuntos escritos antes de corregir, que ya no son independientes; la evaluación heurística encontró 14 problemas, de los que se corrigieron 13, y la auditoría WCAG 2.1 AA no encontró incumplimientos automáticos en los seis temas visuales. La prueba con usuario halló un defecto, ya corregido, y dio un SUS de 92,5 con una participante (Capítulo 11).

14.1.2. Reglas de negocio de la dirección

- **FEFO.** El consumo descuenta siempre primero el lote activo con el vencimiento más próximo, dentro de una transacción. El umbral de criticidad es configurable y se aplica igual en todas las vistas.
- **Kardex inmutable.** Cada movimiento guarda lote, usuario, tipo, cantidad, motivo y fecha. La aplicación rechaza modificar o eliminar un movimiento, y la base de datos impide borrar un lote o un usuario con historial.
- **Control de acceso por roles.** El *middleware* `CheckRole` protege las rutas administrativas; no hay registro público y las pruebas verifican que un operario reciba acceso denegado.
- **RAG dinámico.** El asistente clasifica cada pregunta, consulta la base de datos y responde a partir de ese contexto; cuando el material no existe, lo dice en lugar de mostrar datos de otros materiales.

14.1.3. Hallazgos

- **El requisito real puede diferir del supuesto.** El planteamiento inicial suponía pérdidas por vencimiento; la entrevista mostró que en la empresa del caso el problema es la calidad de la información. Levantar requerimientos con un usuario real cambió qué se debía priorizar.
- **Lo que el documento afirma debe verificarse en el código.** La revisión final encontró diferencias entre la documentación y el sistema (el Kardex no estaba protegido contra el borrado en cascada y el registro permitía crear administradores); corregirlas fue parte del trabajo de ingeniería.
- **En un servidor sin GPU, los modelos de lenguaje deben elegirse por recursos.** Con cuatro núcleos ARM, el modelo local de respaldo tardó de 33 a 56 segundos y los más grandes, minutos, frente a cerca de un segundo del modelo gratuito en la nube (Capítulo 10). El diseño RAG entrega al modelo solo datos filtrados; la velocidad la da el modelo en la nube, y el servidor propio no necesita GPU.
- **La infraestructura gratuita bastó para la demostración y las mediciones.** El sistema opera en la capa *Always Free* de Oracle Cloud, con un túnel de Cloudflare y modelos gratuitos de Ollama Cloud.

14.1.4. Lecciones aprendidas

- Integrar un modelo de lenguaje en una aplicación de negocio es sobre todo un problema de ingeniería de software: clasificar la intención, construir el contexto, limitar lo que el modelo puede hacer (aquí, solo leer) y prever qué pasa cuando falla.
- Con un solo desarrollador, las pruebas automatizadas en la integración continua son la principal defensa al desplegar directamente a producción.
- Anonimizar a las personas y a la empresa y declarar las cifras aproximadas protege a los participantes sin restarle valor a la evidencia.

Capítulo 15

Bibliografía

[1] BBVA Research, "Una mirada a las Mipymes en Colombia," 2024. Fecha de acceso: 16 de julio de 2025, https://www.bbvaresearch.com/wp-content/uploads/2024/02/202401_MiPymes_Colombia-1.pdf.

[2] Departamento Nacional de Planeación, "Política nacional de ciencia, tecnología e innovación 2022-2031." Política N.º 4069, 2021. Fecha de acceso: 16 de julio de 2025, https://minciencias.gov.co/sites/default/files/upload/paginas/conpes_4069.pdf.

[3] Universidad del Valle, "Por la cual se actualiza el reglamento de Trabajo de Grado para la Facultad de Ingeniería...." Resolución N.º 137, 2023. Fecha de acceso: 16 de julio de 2025, https://eime.univalle.edu.co/images/documentos/programas/docu_curricular/Resolucion_No_137-2023_Reglamento_De_Trabajo_De_Grado_ABR_18.pdf.

[4] Congreso de Colombia, "Por la cual se dictan disposiciones para promover el desarrollo de las micro, pequeñas y medianas empresa." Ley N.º 590, 2000. Fecha de acceso: 16 de julio de 2025, https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=12672.

[5] Congreso de Colombia, "Por la cual se dictan disposiciones generales para la protección de datos personales." Ley N.º 1581, 2012. Fecha de acceso: 16 de julio de 2025, https://www.funcionpublica.gov.co/eva/gestornormativo/norma.php?i=49981.

[6] Taylor Otwell and Laravel Contributors, "Laravel 13.x — The PHP Framework for Web Artisans." Laravel LLC, 2026. Fecha de acceso: 25 de septiembre de 2026, https://laravel.com/docs/13.x.

[7] Inertia.js, "The Modern Monolith — Inertia.js Documentation." Inertia.js, 2025. Fecha de acceso: 7 de mayo de 2026, https://inertiajs.com/.

[8] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." arXiv preprint arXiv:2005.11401, 2020. Fecha de acceso: 7 de mayo de 2026, https://arxiv.org/abs/2005.11401.

[9] Ollama, "Ollama REST API Documentation." Ollama Inc., 2025. Fecha de acceso: 7 de mayo de 2026, https://github.com/ollama/ollama/blob/main/docs/api.md.

[10] Hugging Face, "The AI community building the future." Hugging Face Inc., 2025. Fecha de acceso: 7 de mayo de 2026, https://huggingface.co/.

[11] GitHub Inc., "GitHub Projects — Plan and track work." Microsoft, 2025. Fecha de acceso: 7 de mayo de 2026, https://docs.github.com/en/issues/planning-and-tracking-with-projects.

[12] Innpulsa Colombia, "El 63 % de las empresas medianas se encuentra en los niveles más avanzados de transformación digital, mientras que las microempresas solo alcanzan el 42 %," 2023. Fecha de acceso: 16 de julio de 2025, https://www.innpulsacolombia.com/portfolio/.

[13] R. H. Ballou, *Logística: Administración de la cadena de suministro*. Pearson, 5a ed., 2004.

[14] Ministerio de Salud y Protección Social, "Resolución 2674 de 2013 (requisitos sanitarios de alimentos), arts. 19 y 28." Diario Oficial, 2013. Fecha de acceso: 9 de octubre de 2026, https://normograma.invima.gov.co/compilacion/docs/resolucion_minsaludps_2674_2013.htm.

[15] W. X. Zhao *et al.*, "A Survey of Large Language Models," *arXiv preprint arXiv:2303.18223*, 2023. https://arxiv.org/abs/2303.18223.

[16] J. Nielsen, "10 Usability Heuristics for User Interface Design." Nielsen Norman Group, 1994. Artículo original publicado el 24 de abril de 1994, actualizado. Fecha de acceso: 4 de agosto de 2025, https://www.nngroup.com/articles/ten-usability-heuristics/.

[17] W3C Web Accessibility Initiative (WAI), "Web Content Accessibility Guidelines (WCAG) Overview." World Wide Web Consortium (W3C), 2023. Fecha de acceso: 4 de agosto de 2025, https://www.w3.org/WAI/standards-guidelines/wcag/.

[18] Odoo S.A., "Software empresarial todo en uno," 2025. Accesado: 28 de julio de 2025, https://www.odoo.com/es.

[19] Zoho Corporation, "Software de gestión de inventario," 2025. Accesado: 28 de julio de 2025, https://www.zoho.com/es-xl/inventory/.

[20] Canvus Applications, "Free Online Inventory Management for Small Business," 2025. Accesado: 28 de julio de 2025, https://www.stockpile.com/.

[21] Zoho Corporation, "AI in Zoho Inventory." Zoho, 2026. Fecha de acceso: 9 de octubre de 2026, https://www.zoho.com/us/inventory/features/ai-in-inventory/.

[22] I. J. Bedoya, "Elaboración de un sistema de inventario por medio de la creación de una aplicación, para el almacenamiento y control eficiente," 2022. Trabajo de Grado. https://repository.unad.edu.co/bitstream/handle/10596/56848/ijbedoyal.pdf.

[23] A. P. Bautista and L. F. Hernández, "Propuesta de un Sistema de Gestión de Inventarios Para Maximizar el Efectivo de las Micro y Pequeñas Empresas de Bucaramanga," 2022. Trabajo de Grado. https://repository.upb.edu.co/bitstream/handle/20.500.11912/10036/1914_e_4%20(1)%20(1).pdf.

[24] J. F. Acosta, "Propuesta para la implementación de un sistema de gestión de inventarios para la empresa Vidrios Tempse bajo parámetros de Lean Manufacturing," 2020. Trabajo de Grado (Maestría). https://repository.uamerica.edu.co/bitstreams/7ca7f08f-f05d-4478-8fca-6d4d1738e3d8/download.

[25] K. Schwaber and J. Sutherland, "La Guía de Scrum: La Guía Definitiva de Scrum - Las Reglas del Juego." Scrum.org, Noviembre 2020. Fecha de acceso: 4 de agosto de 2025, https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Spanish-Latin-American.pdf.

[26] Taylor Otwell and Contributors, "Laravel Documentation." Laravel, 2025. Fecha de acceso: 4 de agosto de 2025, https://laravel.com/docs.

[27] M. Fowler, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, 2002.

[28] MDN Web Docs, "Responsive design." Mozilla, 2023. Fecha de acceso: 4 de agosto de 2025, https://developer.mozilla.org/en-US/docs/Learn/CSS/CSS_layout/Responsive_Design.

[29] R. S. Pressman and B. R. Maxim, *Ingeniería de Software: Un Enfoque Práctico*. McGraw-Hill, 7a ed., 2014.

[30] Oracle Corporation, "MySQL Documentation." MySQL, 2025. Fecha de acceso: 4 de agosto de 2025, https://dev.mysql.com/doc/.

[31] Object Management Group (OMG), "OMG Unified Modeling Language (UML) - What is UML?." Object Management Group, 2023. Fecha de acceso: 4 de agosto de 2025, https://www.omg.org/uml/.

[32] S. Chacon and B. Straub, *Pro Git*. Apress, 2a ed., 2014. Disponible en línea: https://git-scm.com/book/es/v2.

[33] Tailwind Labs, "Styling with utility classes – Tailwind CSS." Documentación oficial de Tailwind CSS, 2026. Fecha de acceso: 9 de octubre de 2026, https://tailwindcss.com/docs/styling-with-utility-classes.

[34] Microsoft Corporation, "Playwright — Fast and reliable end-to-end testing for modern web apps." Microsoft, 2025. Fecha de acceso: 7 de mayo de 2026, https://playwright.dev/.

[35] N. K. Denzin and Y. S. Lincoln, eds., *The SAGE Handbook of Qualitative Research*. Thousand Oaks, CA: SAGE Publications, 5 ed., 2018.

[36] Oracle Corporation, "Oracle Cloud Free Tier — Always Free Resources." Oracle Cloud Infrastructure, 2025. Fecha de acceso: 7 de mayo de 2026, https://www.oracle.com/cloud/free/.

[37] Colombia Hosting, "¿Necesitas comprar un dominio web?," 2025. Fecha de acceso: 16 de julio de 2025, https://www.colombiahosting.com.co/registro-dominios.

[38] Colombia Hosting, "Hosting y Dominios en Colombia," 2025. Fecha de acceso: 16 de julio de 2025, https://www.colombiahosting.com.co/.

[39] Unidad de Planeación Minero Energética (UPME), "Factor de emisiones de la red de energía eléctrica en Colombia," 2021. Fecha de acceso: 16 de julio de 2025, https://www1.upme.gov.co/siame/Documents/Calculo-FE-del-SIN/Documento_calculo_Cartilla_Factor_de_emision_2021.pdf.

[40] Ollama, "Privacy Policy." Ollama, 2026. Fecha de acceso: 9 de octubre de 2026, https://ollama.com/privacy.

[41] Deque Systems, "axe-core: Accessibility engine for automated Web UI testing." GitHub, 2026. Fecha de acceso: 26 de septiembre de 2026, https://github.com/dequelabs/axe-core.

[42] J. Brooke, "SUS: A "quick and dirty" usability scale," in *Usability Evaluation in Industry* (P. W. Jordan, B. Thomas, B. A. Weerdmeester, and I. L. McClelland, eds.), pp. 189–194, London: Taylor & Francis, 1996.

[43] D. J. Anderson, "Kanban: Successful Evolutionary Change for Your Technology Business." Blue Hole Press, 2010. ISBN: 978-0984521401.

Anexos

A continuación se presentan los anexos que complementan el trabajo de grado. El código fuente completo del proyecto se encuentra disponible en el repositorio de GitHub: `https://github.com/germanmurillas/gestor-inventario-pymes-llm`. Los anexos digitales (manual de usuario, estudio de costos y conteo físico con su hoja de ejemplo) están versionados en la carpeta `docs/anexos` de ese repositorio, su fuente oficial; `https://tesis.pymetory.com` conserva una copia.

Diagramas UML del sistema

A continuación se presentan los diagramas UML que modelan el sistema Pymetory desde distintas perspectivas: casos de uso, secuencia de operaciones, componentes de software y despliegue de infraestructura.

Diagrama de Casos de Uso

El sistema define dos actores. El Operario realiza la operación diaria y el Administrador, que hereda todos sus casos de uso (generalización), tiene además los de configuración y control. Los permisos corresponden a las rutas protegidas por el *middleware* `CheckRole` en el código.

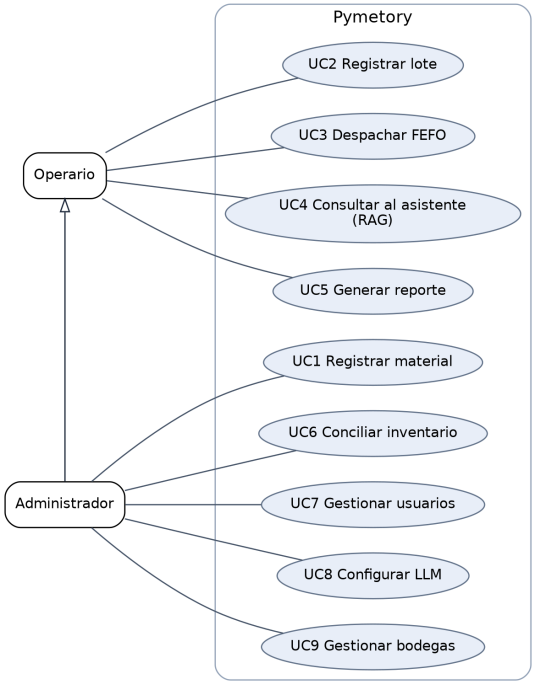


Figura A.1: Diagrama de casos de uso del sistema Pymetory. El Operario registra lotes, despacha por FEFO, consulta al asistente y genera reportes; el Administrador hereda esos casos y además registra materiales, concilia, gestiona usuarios y bodegas y configura el modelo de lenguaje.

Los nueve casos de uso del sistema son:

1. **UC1 — Registrar Material:** El administrador ingresa un nuevo material al catálogo, especificando código único, nombre, unidad de medida y bodega destino. El sistema valida que el código no esté duplicado y crea el registro junto con un lote inicial y su movimiento de entrada en el Kardex.

2. **UC2 — Registrar Lote:** Permite ingresar un nuevo lote de un material existente, especificando número de lote, cantidad, costo unitario, fecha de vencimiento y bodega destino. El sistema actualiza el stock y registra el movimiento.

3. **UC3 — Despachar FEFO:** El operario solicita una cantidad de material para un motivo específico (producción, venta, desperdicio o ajuste). El sistema selecciona automáticamente los lotes más próximos a vencer y descuenta de ellos hasta cubrir la cantidad solicitada.

4. **UC4 — Consultar RAG:** El usuario escribe una pregunta en lenguaje natural sobre el inventario. El sistema clasifica la intención, construye contexto a partir de la base de datos real y envía la consulta al modelo de lenguaje, que responde con datos actualizados del inventario.

5. **UC5 — Generar Reporte:** El operario o el administrador exporta reportes del inventario en PDF, CSV o Excel, con filtros por bodega, material y fechas. Cada exportación queda registrada en la auditoría.

6. **UC6 — Conciliar:** El administrador corrige discrepancias entre el inventario físico y el lógico ingresando la cantidad real verificada en bodega. El sistema registra automáticamente un movimiento de ajuste en el Kardex.

7. **UC7 — Gestionar Usuarios:** El administrador registra nuevos usuarios, asigna roles (admin u operario) y gestiona el control de acceso RBAC del sistema.

8. **UC8 — Configurar LLM:** El administrador ajusta los parámetros del módulo de inteligencia artificial: modelo a utilizar, temperatura de las respuestas, tokens máximos, fuente (local o externa) y clave de API.

9. **UC9 — Gestionar Bodegas:** El administrador crea, edita y monitorea las bodegas del sistema, registrando su capacidad máxima y visualizando el porcentaje de ocupación en tiempo real.

Diagrama de Secuencia — Registro de Entrada

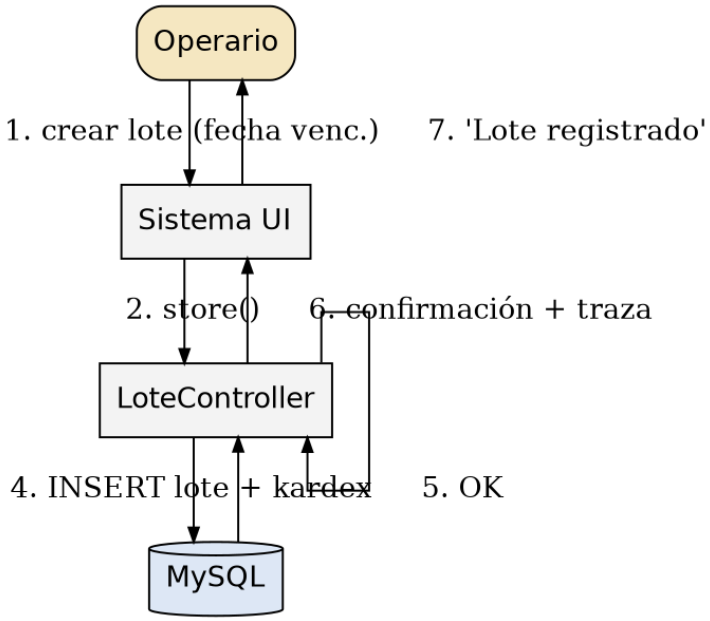


Figura A.2: Diagrama de secuencia del registro de entrada de un lote. Muestra la interacción entre el Operario, el frontend React+Inertia.js, el InventoryController y MySQL, incluyendo la transacción atómica que garantiza la creación simultánea del lote y su movimiento de Kardex.

Diagrama de Secuencia — Consulta RAG

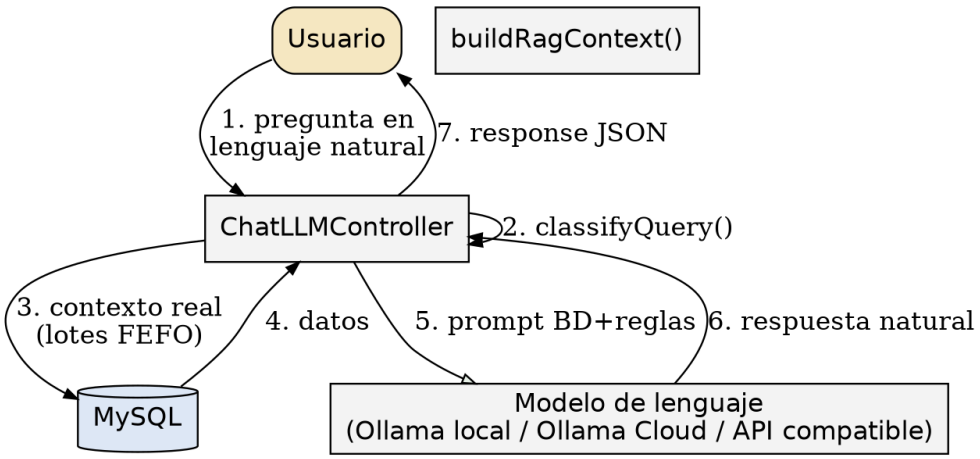


Figura A.3: Diagrama de secuencia de una consulta al módulo RAG. El flujo muestra la clasificación de intención, la construcción del contexto a partir de la base de datos, el envío del prompt enriquecido al LLM y el almacenamiento de la respuesta en el historial de chat.

Diagrama de Secuencia — Despacho FEFO

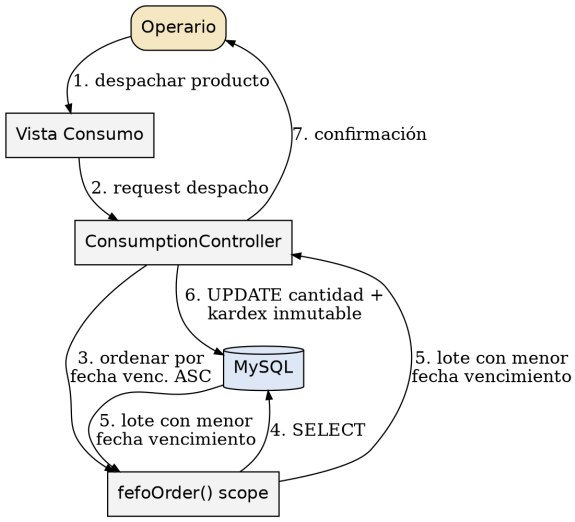


Figura A.4: Diagrama de secuencia del despacho aplicando el algoritmo FEFO. Ilustra cómo el sistema ordena los lotes por fecha de vencimiento, descuenta del más próximo primero, marca lotes agotados como `consumed` y registra cada movimiento en el Kardex inmutable dentro de una transacción atómica.

Diagrama de Secuencia — Conciliación de inventario

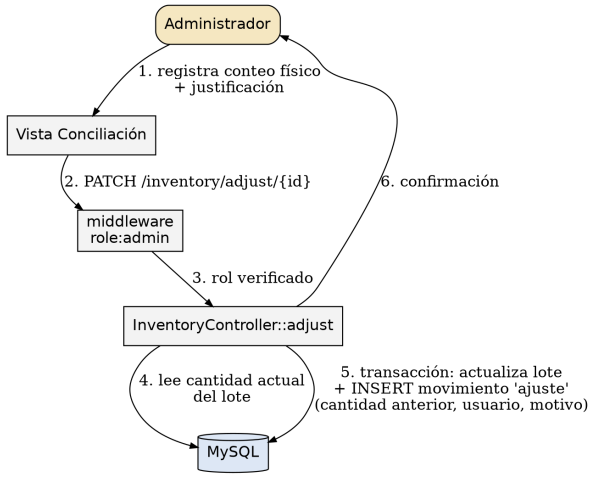


Figura A.5: Secuencia de la conciliación: el administrador registra el conteo físico y el sistema corrige el lote dejando un movimiento de ajuste en el Kardex.

Diagrama de Componentes

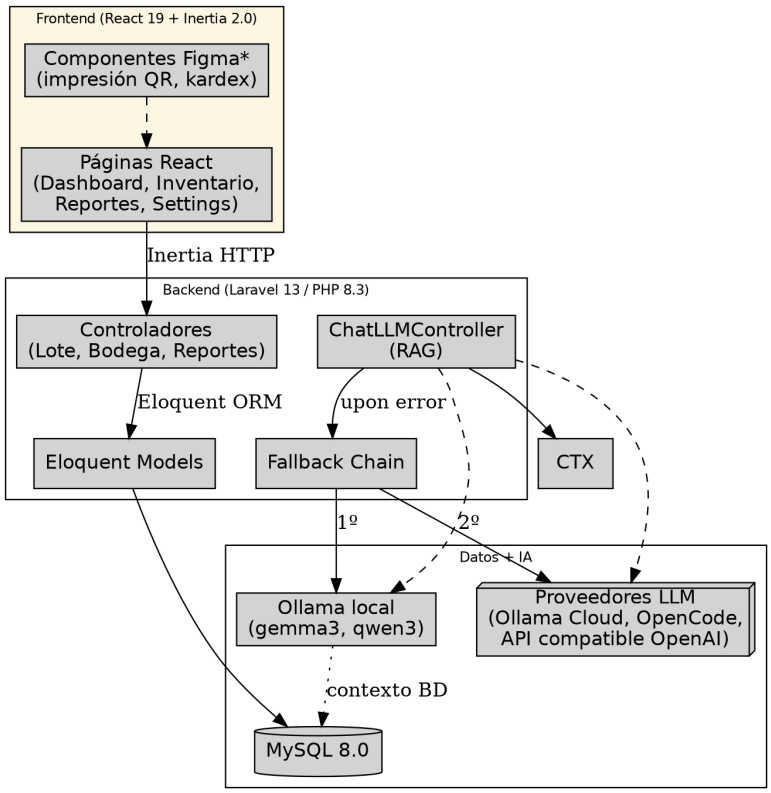


Figura A.6: Diagrama de componentes del sistema Pymetory. Muestra la organización modular en cuatro capas: frontend React, backend Laravel con sus controladores y middlewares, persistencia MySQL, y el módulo de inteligencia artificial con el RAG Context Builder que conecta la base de datos con los proveedores de modelos de lenguaje (Ollama local, Ollama Cloud y APIs compatibles).

Diagrama de Despliegue

El diagrama de despliegue de la infraestructura se presenta en la Figura 10.1 (Capítulo 10).

Diagrama de Clases Simplificado

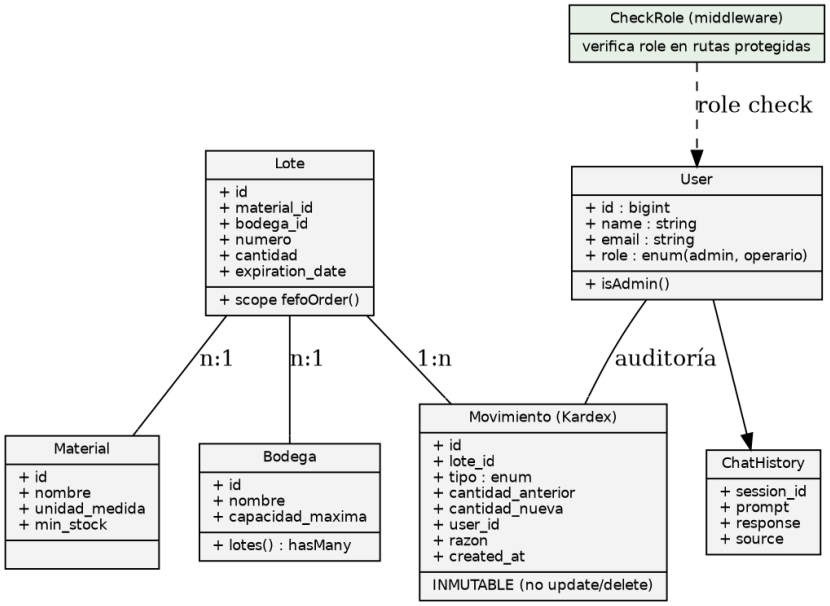


Figura A.7: Diagrama de clases simplificado con las cinco entidades principales del sistema: User, Bodega, Material, Lote (entidad central FEFO) y Movimiento (Kardex inmutable). Se incluye también ChatHistory para la persistencia de consultas al módulo RAG.

Modelo relacional

La Figura A.8 muestra las tablas del dominio de Pymetory y sus llaves foráneas, tal como quedan definidas por las migraciones de Laravel. El anteproyecto lo llamaba modelo entidad-relación; como usa tablas y llaves, y no la notación de entidades y rombos, es un modelo relacional. Se omiten las tablas de infraestructura del framework (caché, colas, sesiones) y las de configuración sin relaciones (`settings`, `api_keys`, `llm_providers`, `llm_evaluaciones`, `llm_precios`, `custom_field_definitions`). La tabla `movimientos`, que implementa el Kardex, aparece resaltada; sus llaves hacia `lotes` y `users` impiden borrar un lote o un usuario que tenga historial.

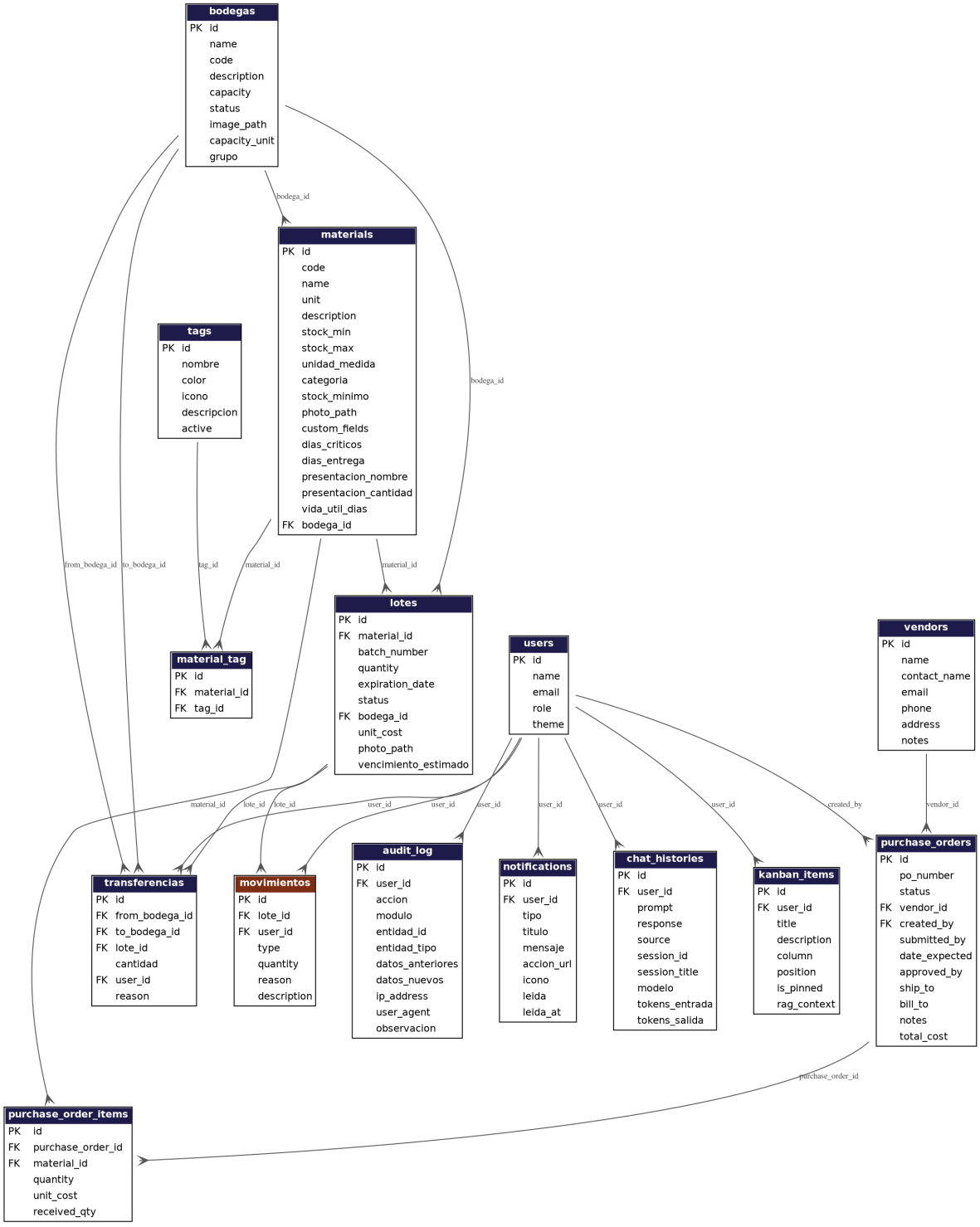


Figura A.8: Modelo relacional de Pymetory.

Diagrama de flujo del proceso FEFO

El siguiente diagrama de actividad modela el ciclo de vida completo de un lote bajo la política FEFO (First Expired, First Out — primero en vencer, primero en salir), desde su registro inicial en bodega hasta su despacho final. El flujo incluye dos puntos de decisión críticos: el monitoreo continuo de fechas de vencimiento con alerta a los 15 días, y el algoritmo de selección que itera sobre los lotes activos ordenados por proximidad de vencimiento, descontando del más urgente primero.

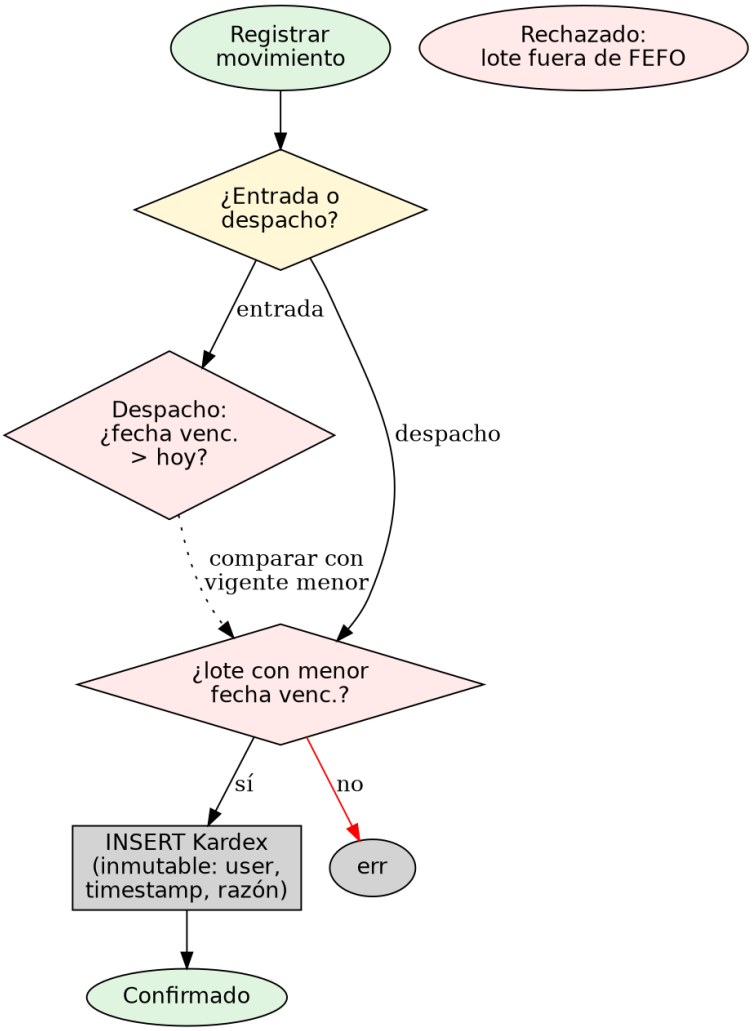


Figura A.9: Diagrama de actividad del proceso FEFO completo: desde el registro de un lote hasta su despacho, incluyendo monitoreo de vencimientos, alertas críticas y el algoritmo de selección por fecha de expiración.

Explicación del flujo FEFO

Al registrar un lote, el sistema lo guarda como activo y, si vence dentro del umbral de días críticos, genera una alerta. En un despacho verifica primero que haya existencia suficiente del material; si no la hay, rechaza la operación indicando la cantidad disponible. Si la hay, descuenta del lote que vence primero, continúa con el siguiente si ese no alcanza y registra cada salida en el Kardex dentro de una sola transacción (Sección 10.2, patrón *Transaction Script*).

Manual de usuario

El manual completo, con capturas de cada pantalla de `https://app.pymetory.com`, se entrega como anexo digital en `https://tesis.pymetory.com/manual-de-usuario.pdf` y se abre desde el menú de la aplicación. A continuación se resume cada módulo. Las funciones marcadas como *solo administrador* no están disponibles para el rol operario.

- **Ingreso y celular:** correo y contraseña asignados por el administrador (las cuentas de administrador solo las crea otro administrador). En el celular, una barra inferior da acceso a Tablero, Inventario, Escanear y Asistente, y **Más** al resto.
- **Tablero:** insumos, lotes, lotes críticos, valorización, últimos movimientos y ocupación de bodegas, con accesos rápidos.
- **Inventario:** en *Insumos* se ven la existencia, los lotes y los avisos de cada insumo, con filtros por bodega y categoría; *Ingreso* crea un insumo con su primer lote, *Registrar ingreso de lote* agrega lotes y *Consumo FEFO* descuenta primero del lote que vence primero. En *Bodegas*, cada tarjeta resume lotes, vencimientos, valor y ocupación.
- **Búsqueda y asistente:** la búsqueda filtra lotes por nombre, código, lote, categoría o bodega; el asistente responde preguntas en español ("¿Cuánta harina de trigo hay?") consultando antes la base de datos, y guarda el historial.
- **Reportes:** inventario, movimientos, FEFO, consumo y valorización, filtrables por bodega, material y fechas, en PDF, CSV o Excel.
- **Reabastecimiento:** consumo diario, cobertura, fecha de agotamiento y punto de reorden de cada insumo (con los días de entrega registrados en sus ajustes) y gráfico de consumo por periodo.
- **Conteo físico** (*solo administrador*): importa la hoja de cálculo del conteo mensual de la bodega (.xlsx o .csv), la compara con el sistema y registra las diferencias como ajustes del Kardex (anexo digital: `https://tesis.pymetory.com/conteo-fisico.pdf`).
- **Log maestro (Kardex):** todos los movimientos con tipo, material, lote, responsable, cantidad y fecha. No se pueden editar ni eliminar; una corrección se registra como un ajuste nuevo con su justificación.
- **Etiquetas, escáner, transferencias y órdenes de compra:** el sistema genera códigos QR y de barras por lote, que el escáner lee con la cámara o con el número escrito; las transferencias registran la salida en la bodega de origen y la entrada en la de destino; al recibir una orden de compra se crean los lotes con su vencimiento, bodega y entrada en el Kardex.
- **Alertas:** stock bajo y lotes próximos a vencer, generadas por una tarea que revisa el inventario cada hora; un aviso sin leer no se repite en 24 horas.
- **Ajustes** (*solo administrador*): umbral FEFO, notificaciones, proveedor y modelo del asistente, claves de API, usuarios y roles, y tema visual (seis paletas).