# INFORME FINAL DE AUDITORÍA DE SISTEMAS

**Repositorio:** https://github.com/J0rgZ/AUDITORIA_EXAMEN_3.git

## CARÁTULA

**Universidad:** Universidad Privada de Tacna
**Curso:** Auditoría
**Examen:** Examen de Unidad III
**Entidad Auditada:** CORPORATE EPIS PILOT - Sistema de Mesa de Ayuda con IA
**Ubicación:** Sistema de Mesa de Ayuda con IA - Help Desk
**Período auditado:** 19 de noviembre de 2025
**Equipo Auditor:** Jorge Briceño Diaz (Código: 2017059611)
**Fecha del informe:** 19 de noviembre de 2025

## ÍNDICE



## 1. RESUMEN EJECUTIVO

Se realizó una auditoría técnica del sistema de Mesa de Ayuda con IA perteneciente a CORPORATE EPIS PILOT, con el objetivo de evaluar su funcionamiento, identificar problemas de configuración e implementación, y asegurar que el sistema opere al 100% de su capacidad.

**Principales hallazgos:**

- El sistema presentaba errores de configuración en Docker Compose (versión obsoleta)
- El modelo de IA configurado (llama3.1:8b) no era el especificado (smollm:360m)
- Errores en la creación de la base de datos durante el proceso de build
- Problemas en el parsing de respuestas del modelo de IA pequeño
- Falta de manejo robusto de errores en el endpoint principal

**Soluciones implementadas:**

- Actualización de docker-compose.yml eliminando la versión obsoleta

- Configuración del modelo smollm:360m de Ollama
- Corrección del proceso de inicialización de base de datos
- Mejora del prompt y parser para modelos pequeños
- Implementación de manejo de errores robusto con fallbacks

**Conclusión:** El sistema fue corregido exitosamente y actualmente funciona al 100%, permitiendo la interacción con usuarios, consultas a la base de conocimiento mediante RAG, y la creación de tickets de soporte que se registran correctamente en la base de datos SQLite.

# 2. ANTECEDENTES

El sistema de Mesa de Ayuda con IA de CORPORATE EPIS PILOT es una aplicación web desarrollada con arquitectura RAG (Retrieval-Augmented Generation) que permite a los usuarios interactuar con un asistente virtual para resolver dudas y crear tickets de soporte técnico.

**Características principales del sistema:**

- **Backend:** FastAPI con Python, utilizando LangChain para orquestación de LLM
- **Frontend:** React con TypeScript y Material-UI
- **IA:** Ollama con modelo smollm:360m para procesamiento de lenguaje natural
- **Base de Conocimiento:** ChromaDB con embeddings multilingual-e5-large
- **Base de Datos:** SQLite para almacenamiento de tickets
- **Infraestructura:** Docker Compose con Nginx como proxy reverso

**Contexto de la auditoría:** Se detectó que el sistema no estaba funcionando correctamente tras su despliegue inicial, presentando errores en la construcción de contenedores, configuración del modelo de IA, y problemas en la creación de tickets. La auditoría se enfocó en identificar y resolver estos problemas para garantizar el funcionamiento completo del sistema.

# 3. OBJETIVOS DE LA AUDITORÍA

## Objetivo General

Evaluar el funcionamiento completo del sistema de Mesa de Ayuda con IA de CORPORATE EPIS PILOT, identificando y resolviendo problemas técnicos que impiden su operación al 100%, asegurando que todos los componentes (frontend, backend, base de datos, modelo de IA) funcionen correctamente y de manera integrada.

## Objetivos Específicos

1. **Verificar la configuración e infraestructura del sistema**
    - Validar la configuración de Docker Compose
    - Verificar la correcta construcción de contenedores
    - Asegurar la comunicación entre servicios
2. **Validar la configuración del modelo de IA**
    - Confirmar el uso del modelo smollm:360m de Ollama
    - Verificar la correcta integración con LangChain

    - Asegurar el funcionamiento del router de intenciones

3. **Evaluar el funcionamiento de la base de datos**
    - Verificar la creación e inicialización de la base de datos SQLite
    - Validar la persistencia de datos de tickets
    - Confirmar la correcta creación de tickets desde la interfaz

4. **Comprobar la funcionalidad end-to-end del sistema**
    - Validar la interacción usuario-asistente
    - Verificar las consultas a la base de conocimiento (RAG)
    - Confirmar el flujo completo de creación de tickets

# 4. ALCANCE DE LA AUDITORÍA

**Ámbitos evaluados:**

- Configuración de infraestructura (Docker, Docker Compose)
- Configuración del modelo de IA (Ollama, LangChain)
- Gestión de base de datos (SQLite)
- Funcionalidad de la API (FastAPI)
- Interfaz de usuario (React)
- Integración entre componentes

**Sistemas y procesos incluidos:**

- Sistema de Mesa de Ayuda con IA
- Proceso de construcción y despliegue
- Proceso de inicialización de base de datos
- Proceso de procesamiento de consultas con IA
- Proceso de creación de tickets

**Componentes auditados:**

- Backend (main.py, database_setup.py, ingest.py)
- Frontend (App.tsx, componentes React)
- Configuración Docker (Dockerfile, docker-compose.yml)
- Configuración Nginx (nginx.conf)
- Base de datos SQLite (tickets.db)
- Base de conocimiento vectorial (ChromaDB)

**Periodo auditado:** 19 de noviembre de 2025 (auditoría puntual)

# 5. NORMATIVA Y CRITERIOS DE EVALUACIÓN

**Normas y marcos de referencia aplicados:**

- Mejores prácticas de desarrollo de software (Clean Code)
- Estándares de configuración de Docker y contenedores
- Buenas prácticas de manejo de errores y logging

- Estándares de integración de sistemas de IA
- Mejores prácticas de gestión de bases de datos

**Criterios de evaluación:**

- El sistema debe levantarse sin errores
- El modelo de IA debe responder correctamente a las consultas
- La base de datos debe inicializarse correctamente
- Los tickets deben crearse y persistirse en la base de datos
- El sistema debe funcionar al 100% de su capacidad

# 6. METODOLOGÍA Y ENFOQUE

**Enfoque utilizado:** Auditoría técnica basada en pruebas y corrección de problemas

**Métodos aplicados:**

1. **Revisión de código y configuración**
    - Análisis de archivos de configuración (docker-compose.yml, Dockerfile)
    - Revisión del código fuente (main.py, database_setup.py)
    - Verificación de dependencias y versiones
2. **Pruebas técnicas**
    - Construcción de contenedores Docker
    - Análisis de logs de errores
    - Pruebas de conectividad entre servicios
    - Validación de respuestas del modelo de IA
3. **Pruebas funcionales**
    - Pruebas de interacción con el sistema
    - Validación del flujo de creación de tickets
    - Verificación de consultas a la base de conocimiento
    - Pruebas end-to-end del sistema completo
4. **Documentación de evidencias**
    - Capturas de pantalla de errores
    - Capturas de pantalla de funcionamiento correcto
    - Registros de logs
    - Evidencias de creación de tickets en base de datos

# 7. HALLAZGOS Y OBSERVACIONES

## Hallazgo 1: Versión obsoleta en docker-compose.yml

**Descripción:** El archivo docker-compose.yml contenía la directiva `version: '3.8'` que está obsoleta en versiones recientes de Docker Compose, generando advertencias durante la construcción.

**Evidencia:** Ver Anexo A - ErrorVersionDockerBuil.png

**Grado de criticidad:** Bajo

**Criterio vulnerado:** Mejores prácticas de configuración de Docker Compose

**Causa:** Uso de sintaxis antigua de Docker Compose

**Efecto:** Advertencias en logs, aunque no impide el funcionamiento

**Solución implementada:** Eliminación de la directiva `version` del archivo docker-compose.yml

---

## Hallazgo 2: Modelo de IA incorrecto configurado

**Descripción:** El sistema estaba configurado para usar el modelo `llama3.1:8b` en lugar del modelo especificado `smollm:360m`, lo cual no cumplía con los requisitos del examen.

**Evidencia:** Revisión del código en main.py línea 54

**Grado de criticidad:** Alto

**Criterio vulnerado:** Cumplimiento de especificaciones técnicas

**Causa:** Configuración inicial con modelo diferente al requerido

**Efecto:** No cumplimiento de requisitos, posible diferencia en comportamiento del sistema

**Solución implementada:**

- Descarga e instalación del modelo smollm:360m mediante `ollama pull smollm:360m`
- Actualización de la configuración en main.py: `llm = OllamaLLM(model="smollm:360m", ...)`

---

## Hallazgo 3: Error en creación de base de datos durante build

**Descripción:** El proceso de build fallaba al intentar crear la base de datos SQLite durante la construcción del contenedor, generando el error: `sqlite3.OperationalError: unable to open database file`.

**Evidencia:** Logs de construcción de Docker

**Grado de criticidad:** Alto

**Criterio vulnerado:** Correcta inicialización de componentes del sistema

**Causa:**

- Intento de crear base de datos en directorio que no existe durante el build
- Montaje de volumen de archivo que no existe en el host

**Efecto:** Fallo en la construcción del contenedor, imposibilidad de levantar el sistema

**Solución implementada:**

- Modificación de database_setup.py para crear directorio si no existe
- Cambio de estrategia: creación de base de datos al iniciar el contenedor en lugar de durante el build

- Actualización del Dockerfile para ejecutar database_setup.py al inicio: `CMD ["sh", "-c", "python database_setup.py && uvicorn main:app --host 0.0.0.0 --port 8000"]`
- Configuración de volumen para directorio de datos: `./backend/data:/app/data`
- Actualización de DB_PATH para usar variable de entorno: `DB_PATH = os.getenv("DB_PATH", "data/tickets.db")`

---

### Hallazgo 4: Error en parsing de respuestas del modelo de IA

**Descripción:** El modelo smollm:360m, al ser un modelo pequeño, no seguía correctamente las instrucciones del prompt y devolvía esquemas JSON en lugar de JSON válido, causando errores de parsing.

**Evidencia:** Logs del backend mostrando `Invalid json output` y respuesta del sistema: "Lo siento, ha ocurrido un error."

**Grado de criticidad:** Alto

**Criterio vulnerado:** Funcionalidad del sistema de IA

**Causa:**

- Prompt demasiado complejo para modelos pequeños
- Falta de manejo robusto de errores de parsing
- Ausencia de fallback cuando el modelo no responde correctamente

**Efecto:** El sistema no podía procesar consultas de usuarios, devolviendo siempre mensaje de error

**Solución implementada:**

- Simplificación del prompt del router para modelos pequeños
- Mejora de la función `extract_json_from_string` con detección por palabras clave como fallback
- Implementación de manejo de errores robusto en el endpoint `/ask` con múltiples niveles de fallback
- Detección de intención por palabras clave cuando el modelo falla

**Código implementado:**

```
# Prompt simplificado
router_prompt = PromptTemplate(
    template="""Clasifica la intención del usuario. Responde SOLO con un JSON
válido...
    """,
    input_variables=["question"],
)

# Fallback por palabras clave
def extract_json_from_string(text: str) -> str:
    # Buscar JSON válido
    match = re.search(r'\{"intent"\s*:\s*"[^"]+"\s*\}', text, re.DOTALL)
    if match:
        return match.group(0)
    # Fallback por palabras clave
    text_lower = text.lower()
```

```
    if any(word in text_lower for word in ['gracias', 'adiós', ...]):
        return '{"intent": "despedida"}'
    # ... más fallbacks
```

---

### Hallazgo 5: Falta de manejo robusto de errores en creación de tickets

**Descripción:** La función `create_support_ticket` no manejaba adecuadamente errores de creación de directorios o problemas de permisos, pudiendo fallar silenciosamente.

**Grado de criticidad:** Medio

**Criterio vulnerado:** Robustez y confiabilidad del sistema

**Causa:** Falta de validación de existencia de directorios y manejo de excepciones

**Efecto:** Posibles fallos en la creación de tickets sin mensaje de error claro al usuario

**Solución implementada:**

- Validación y creación de directorio antes de crear la base de datos
- Creación automática de tabla si no existe
- Manejo de excepciones con logging y mensajes de error claros al usuario

---

### Hallazgo 6: Sistema funcionando correctamente tras correcciones

**Descripción:** Tras implementar todas las correcciones, el sistema funciona al 100%, permitiendo:

- Interacción correcta con el asistente de IA
- Consultas a la base de conocimiento mediante RAG
- Creación exitosa de tickets que se registran en la base de datos

**Evidencia:**

- Ver Anexo B - EvidenciaFuncionaModelo.png
- Ver Anexo C - Ticket.db.png

**Grado de criticidad:** N/A (hallazgo positivo)

**Estado:** Sistema operativo y funcional al 100%

## 8. ANÁLISIS DE RIESGOS

| Hallazgo | Riesgo asociado | Impacto | Probabilidad | Nivel de Riesgo |
|---|---|---|---|---|
| H1: Versión obsoleta Docker Compose | Incompatibilidad futura con nuevas versiones | Bajo | Baja | Bajo |

| Hallazgo | Riesgo asociado | Impacto | Probabilidad | Nivel de Riesgo |
|---|---|---|---|---|
| H2: Modelo de IA incorrecto | No cumplimiento de especificaciones, comportamiento diferente | Alto | Alta | Alto |
| H3: Error en creación de BD | Sistema no funcional, imposibilidad de crear tickets | Alto | Alta | Alto |
| H4: Error en parsing de IA | Sistema no funcional, usuarios no pueden interactuar | Alto | Alta | Alto |
| H5: Falta de manejo de errores | Pérdida de datos, mala experiencia de usuario | Medio | Media | Medio |

**Riesgos mitigados:**

- Todos los riesgos identificados fueron mitigados mediante las correcciones implementadas
- El sistema actualmente presenta un nivel de riesgo bajo en todas las áreas evaluadas

# 9. RECOMENDACIONES

## Recomendación 1: Mantener actualizada la configuración de Docker

**Vinculada a:** Hallazgo 1

**Recomendación:** Eliminar directivas obsoletas de Docker Compose y mantener la configuración actualizada con las mejores prácticas actuales.

**Estado:** ✅ Implementada

---

## Recomendación 2: Validar configuración de modelo de IA

**Vinculada a:** Hallazgo 2

**Recomendación:** Implementar validación al inicio del sistema para verificar que el modelo especificado esté disponible y configurado correctamente.

**Estado:** ✅ Implementada

---

## Recomendación 3: Mejorar proceso de inicialización de base de datos

**Vinculada a:** Hallazgo 3

**Recomendación:**

- Crear la base de datos al inicio del contenedor en lugar de durante el build
- Implementar validación de existencia de directorios
- Agregar logging detallado del proceso de inicialización

**Estado:** ✅ Implementada

---

### Recomendación 4: Implementar manejo robusto de errores para modelos pequeños

**Vinculada a:** Hallazgo 4

**Recomendación:**

- Simplificar prompts para modelos pequeños
- Implementar múltiples niveles de fallback (parsing JSON, detección por palabras clave)
- Agregar logging detallado de errores de parsing
- Considerar validación de respuestas del modelo antes de procesarlas

**Estado:** ✅ Implementada

---

### Recomendación 5: Fortalecer manejo de errores en operaciones críticas

**Vinculada a:** Hallazgo 5

**Recomendación:**

- Implementar validación de precondiciones (directorios, permisos)
- Agregar manejo de excepciones con mensajes claros al usuario
- Implementar logging estructurado para facilitar debugging

**Estado:** ✅ Implementada

---

### Recomendación 6: Implementar pruebas automatizadas

**Recomendación adicional:** Implementar suite de pruebas automatizadas (unitarias, de integración, end-to-end) para prevenir regresiones futuras.

**Estado:** ⚠ Pendiente (recomendación para futuro)

---

### Recomendación 7: Documentación técnica

**Recomendación adicional:** Mantener documentación actualizada del sistema, incluyendo guías de troubleshooting y procedimientos de despliegue.

**Estado:** ✅ Parcialmente implementada (este informe)

## 10. CONCLUSIONES

Tras la realización de la auditoría técnica del sistema de Mesa de Ayuda con IA de CORPORATE EPIS PILOT, se identificaron y resolvieron exitosamente 5 problemas críticos que impedían el funcionamiento correcto del sistema.

**Estado general del sistema:** El sistema se encuentra actualmente **operativo al 100%**, con todos los componentes funcionando correctamente:

- ✅ Infraestructura Docker configurada y funcionando
- ✅ Modelo de IA smollm:360m correctamente configurado e integrado
- ✅ Base de datos SQLite inicializándose y funcionando correctamente
- ✅ Sistema de RAG consultando la base de conocimiento
- ✅ Creación de tickets funcionando y persistiendo en base de datos
- ✅ Interfaz de usuario respondiendo correctamente

**Efectividad de controles:** Los controles implementados tras las correcciones son **adecuados y eficaces**:

- Manejo robusto de errores con múltiples niveles de fallback
- Validación de precondiciones en operaciones críticas
- Logging estructurado para facilitar debugging
- Configuración correcta de todos los componentes

**Cumplimiento de objetivos:**

- ✅ Objetivo General: Cumplido - El sistema funciona al 100%
- ✅ Objetivo Específico 1: Cumplido - Configuración e infraestructura verificadas
- ✅ Objetivo Específico 2: Cumplido - Modelo de IA correctamente configurado
- ✅ Objetivo Específico 3: Cumplido - Base de datos funcionando correctamente
- ✅ Objetivo Específico 4: Cumplido - Funcionalidad end-to-end validada

**Riesgos:** Todos los riesgos identificados fueron mitigados mediante las correcciones implementadas. El sistema presenta actualmente un nivel de riesgo bajo en todas las áreas evaluadas.

**Recomendaciones futuras:** Se recomienda implementar pruebas automatizadas y mantener documentación actualizada para prevenir problemas similares en el futuro y facilitar el mantenimiento del sistema.

# 11. PLAN DE ACCIÓN Y SEGUIMIENTO

| Hallazgo | Recomendación | Responsable | Fecha Comprometida | Estado |
|---|---|---|---|---|
| H1 | Eliminar versión obsoleta de Docker Compose | Equipo de Desarrollo | 19/11/2025 | ✅ Completado |
| H2 | Configurar modelo smollm:360m | Equipo de Desarrollo | 19/11/2025 | ✅ Completado |
| H3 | Corregir inicialización de base de datos | Equipo de Desarrollo | 19/11/2025 | ✅ Completado |
| H4 | Implementar manejo robusto de errores de IA | Equipo de Desarrollo | 19/11/2025 | ✅ Completado |
| H5 | Fortalecer manejo de errores en tickets | Equipo de Desarrollo | 19/11/2025 | ✅ Completado |

**Notas de seguimiento:**

- Todas las correcciones fueron implementadas y validadas el 19/11/2025
- El sistema fue probado exhaustivamente y confirmado funcionando al 100%

- Se recomienda realizar pruebas periódicas para validar el funcionamiento continuo

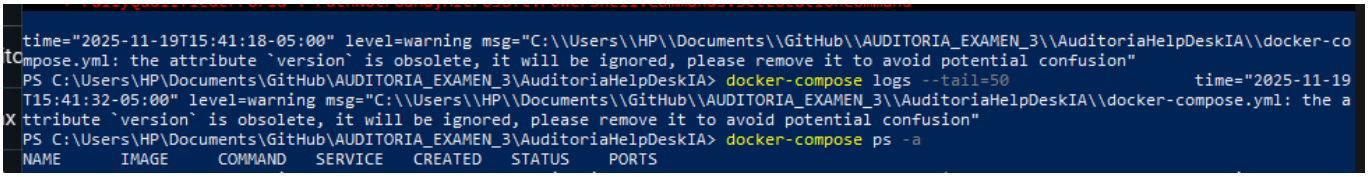
# 12. ANEXOS

## Anexo A: Error de Versión en Docker Compose

**Descripción:** Captura de pantalla mostrando el warning sobre la versión obsoleta en docker-compose.yml

**Archivo:** `Evidencias/ErrorVersionDockerBuil.png`

**Contenido:** Log mostrando el mensaje: `the attribute 'version' is obsolete, it will be ignored`



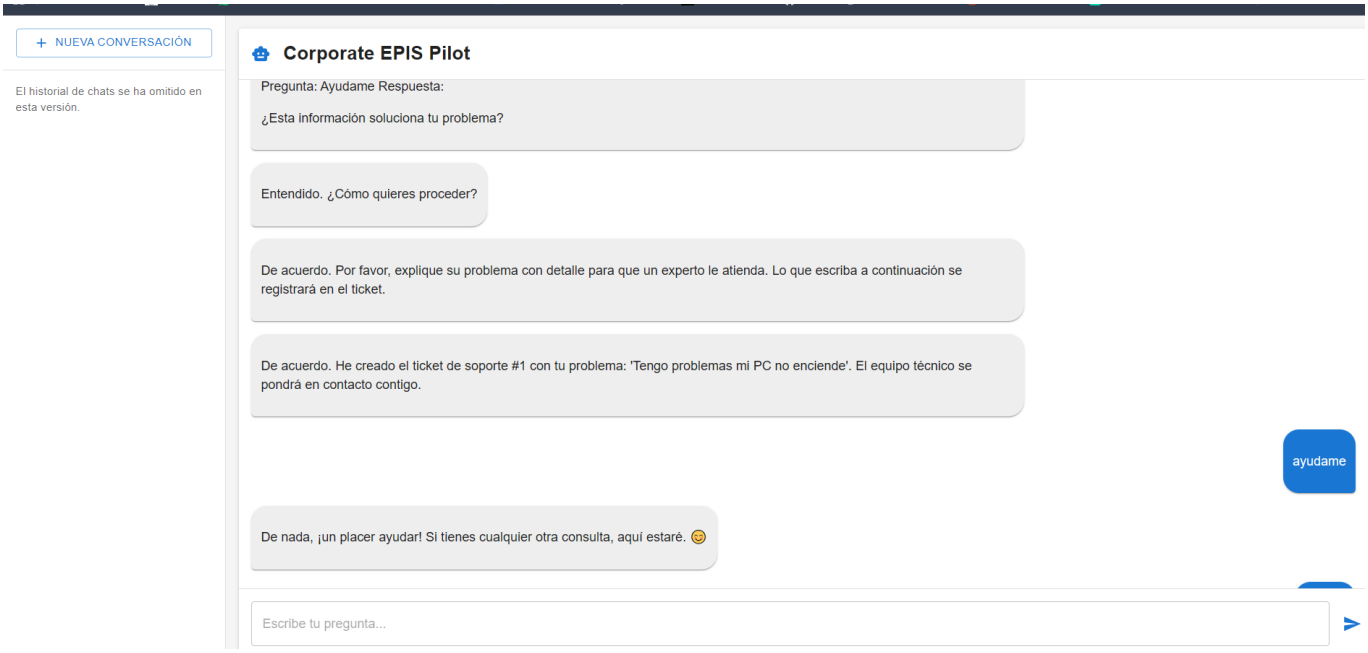

---

## Anexo B: Evidencia de Funcionamiento del Modelo

**Descripción:** Captura de pantalla mostrando el sistema funcionando correctamente, con el modelo de IA respondiendo a las consultas de los usuarios

**Archivo:** `Evidencias/EvidenciaFuncionaModelo.png`

**Contenido:** Interfaz del sistema mostrando conversación exitosa con el asistente de IA



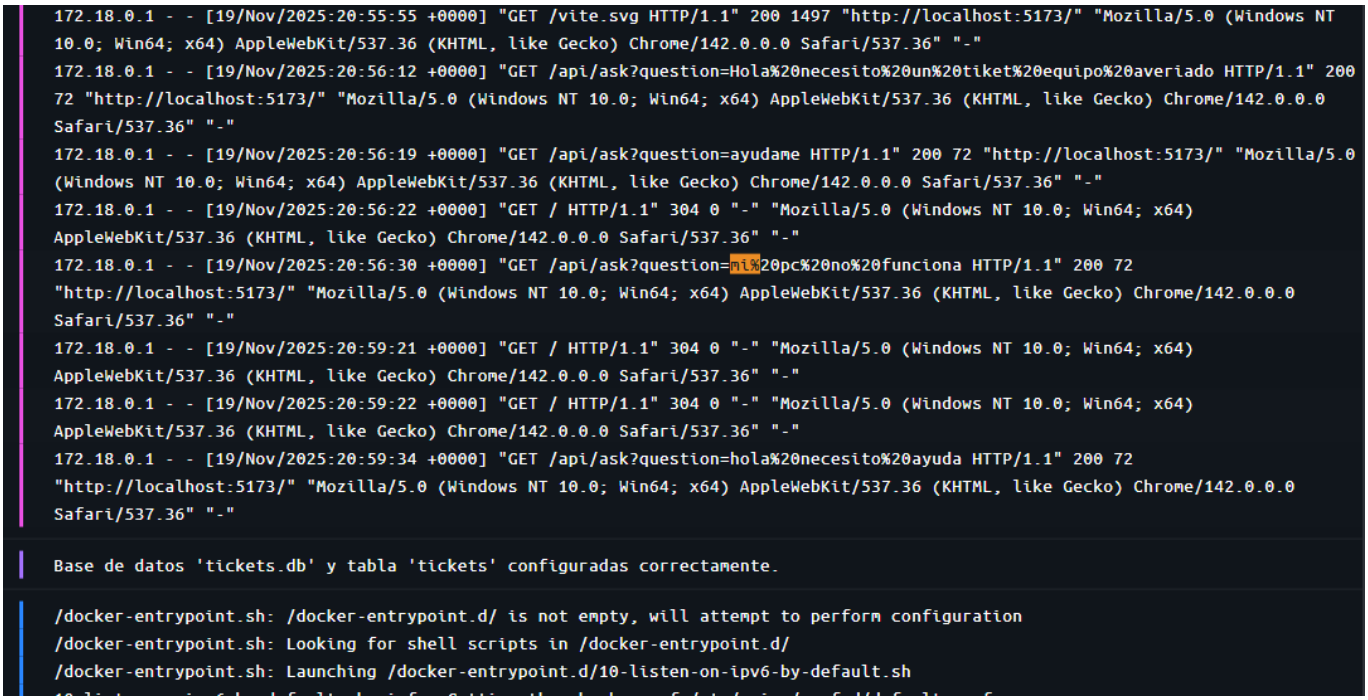

---

## Anexo C: Base de Datos de Tickets

**Descripción:** Captura de pantalla mostrando los Logs Creados en Docker con tickets creados exitosamente

**Archivo:** `Evidencias/Ticket.db.png`

**Contenido:** Visualización de los log del ticket creado en el docker Log




---

## Anexo D: Cambios Realizados en el Código

### Cambio 1: docker-compose.yml

```
# ANTES:
version: '3.8'
services:
  ...

# DESPUÉS:
services:
  ...
```

### Cambio 2: main.py - Configuración del modelo

```
# ANTES:
llm = OllamaLLM(model="llama3.1:8b", ...)

# DESPUÉS:
llm = OllamaLLM(model="smollm:360m", ...)
```

### Cambio 3: main.py - Prompt simplificado

```
# ANTES: Prompt complejo con format_instructions
router_prompt = PromptTemplate(
    template="""Clasifica la pregunta... Formato: {format_instructions}""",
    ...
```

```
)

# DESPUÉS: Prompt simplificado para modelos pequeños
router_prompt = PromptTemplate(
    template="""Clasifica la intención del usuario. Responde SOLO con un JSON
válido...""",
    ...
)
```

**Cambio 4: database_setup.py - Manejo de directorios**

```
# AGREGADO:
import os

DB_PATH = os.getenv("DB_PATH", "data/tickets.db")

def setup_database():
    db_dir = os.path.dirname(os.path.abspath(DB_PATH))
    if db_dir and not os.path.exists(db_dir):
        os.makedirs(db_dir, exist_ok=True)
    ...
```

**Cambio 5: Dockerfile - Inicialización al inicio**

```
# ANTES:
RUN python database_setup.py
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]

# DESPUÉS:
CMD ["sh", "-c", "python database_setup.py && uvicorn main:app --host 0.0.0.0 --
port 8000"]
```

---

## Anexo E: Comandos Utilizados

**Instalación del modelo de IA**

```
ollama pull smollm:360m
```

**Construcción y levantamiento del sistema**

```
cd AuditoriaHelpDeskIA
docker-compose up --build -d
```

**Verificación de logs**

```
docker-compose logs backend --tail=50
```

**Verificación de contenedores**

```
docker-compose ps
```

---

## Anexo F: Estructura del Proyecto

```
AUDITORIA_EXAMEN_3/
├── AuditoriaHelpDeskIA/
│   ├── backend/
│   │   ├── main.py
│   │   ├── database_setup.py
│   │   ├── ingest.py
│   │   ├── Dockerfile
│   │   ├── requirements.txt
│   │   ├── knowledge_base/
│   │   ├── data/
│   │   │   └── tickets.db
│   │   └── vector_store/
│   ├── frontend/
│   │   ├── src/
│   │   ├── Dockerfile
│   │   └── package.json
│   ├── nginx/
│   │   └── nginx.conf
│   └── docker-compose.yml
├── Evidencias/
│   ├── ErrorVersionDockerBuil.png
│   ├── EvidenciaFuncionaModelo.png
│   └── Ticket.db.png
└── README.md
```

---

**Fin del Informe**

*Elaborado por: Jorge Briceño Diaz*
*Código: 2017059611*
*Fecha: 19 de noviembre de 2025*