Ficha Técnica – Asistente RAG para CleanPro

**Asignatura:** ISY0101 – Ingeniería de Soluciones con IA

**Estudiante:**

* Benjamin Osses
* Nicola Osses

1. Nombre y breve descripción de la organización

CleanPro es una empresa de servicios de limpieza industrial con operaciones en distintas sucursales de Santiago. Para esta evaluación se simulan datos de la bodega de Las Condes (inventario) y de la planificación de turnos de septiembre (personal).

1. Identificación y descripción del problema/desafío

Actualmente, la empresa enfrenta dificultades en:

* Acceder rápidamente al stock actualizado de insumos de limpieza.
* Consultar de forma simple los turnos del personal.
* Evitar errores por falta de reposición o confusión en asignación de turnos.

Esto genera ineficiencias y retrasa la toma de decisiones en terreno.

1. Objetivos de la intervención

* Implementar un asistente inteligente que responda preguntas en lenguaje natural sobre inventario y turnos.
* Integrar un pipeline RAG que garantice respuestas basadas en evidencia (CSV).
* Entregar acceso vía API (FastAPI) y frontend web (Streamlit).
* Mejorar la precisión y transparencia en la gestión de datos internos.

1. Datos disponibles

* inventory\_las\_condes.csv: inventario de insumos de limpieza en la sucursal Las Condes.
* turnos\_septiembre.csv: planificación de turnos de trabajadores para septiembre.

Ambos se cargan en un vectorstore ChromaDB para consulta con embeddings.

1. Restricciones o requerimientos

* Solo se dispone de datos de una sucursal (Las Condes) y un mes de turnos (septiembre).
* La evidencia es simulada y limitada; el asistente debe indicar cuando no existe información suficiente.
* Uso exclusivo de .env para proteger credenciales (OPENAI\_API\_KEY).
* El archivo .env debe ser creado por el usuario en la raíz del proyecto antes de ejecutar la ingesta o levantar la API.

1. Motivación para usar LLM + RAG

* Los LLMs permiten consultas en lenguaje natural.
* El RAG garantiza que las respuestas estén basadas en los CSV internos y no en conocimiento general.
* La combinación mejora la precisión, evita alucinaciones y entrega fuentes trazables.

1. Diseño e implementación del pipeline RAG
2. Ingesta de datos (ingest\_all.py) → carga CSV → chunking → embeddings con text-embedding-3-small → almacenamiento en ChromaDB.
3. Backend FastAPI (app.py) → expone endpoint /query.
4. Cadena RAG (rag\_chain.py) →
   * Recupera chunks relevantes desde Chroma.
   * Pasa la pregunta y contexto al LLM (gpt-4o).
   * Devuelve respuesta + evidencia.
5. Frontend Streamlit (app\_streamlit.py) → interfaz de chat estilo WhatsApp.
6. Arquitectura de la solución

Usuario (pregunta)

↓

Streamlit (frontend) → FastAPI (backend)

↓ ↓

Pregunta JSON RAG Chain (LangChain)

↓

Retriever (ChromaDB + Embeddings)

↓

LLM (OpenAI GPT-4o)

↓

Respuesta + Evidencia

↑

Usuario ve chat con historial

1. Herramientas utilizadas

* Python 3.13
* LangChain (community + openai)
* ChromaDB (vectorstore)
* FastAPI + Uvicorn (API REST)
* Streamlit (frontend)
* dotenv (manejo de credenciales)
* PowerShell (testing con Invoke-RestMethod y test.ps1)

1. Decisiones de diseño

* RAG vs Fine-tuning → se eligió RAG porque los datos cambian constantemente y es más eficiente actualizar el vectorstore que entrenar un modelo.
* ChromaDB local → almacenamiento simple y persistente en .chroma\_db.
* FastAPI → API rápida y escalable.
* Streamlit → interfaz ligera y fácil de usar para los usuarios no técnicos.
* Prompt en español + evidencia obligatoria → asegura relevancia y confiabilidad.

1. Uso del sistema
2. # Ingesta: python src/ingestion/ingest\_all.py
3. # Levantar API: uvicorn src.api.app:app --reload --port 8000
4. Luego desde otra terminal powershell:

* Para hacer la consulta en el mismo powershell:

$body = @{ q = "¿Cuánto cloro queda en la bodega de Las Condes?" } | ConvertTo-Json -Compress

Invoke-RestMethod -Uri "http://127.0.0.1:8000/query" -Method Post - ContentType "application/json; charset=utf-8" -Body ([System.Text.Encoding]::UTF8.GetBytes($body))

* Para usar streamlit: streamlit run app\_streamlit.py

1. Ejemplos de preguntas

* Inventario → ¿Qué stock de guantes hay en la bodega de Las Condes?
* Turnos → ¿Quién tiene turno mañana en oficinas administrativas el 2025-09-25?
* Casos fuera de evidencia → ¿Qué productos hay en Providencia?

1. Conclusión

El asistente CleanPro demuestra cómo un pipeline RAG puede resolver necesidades organizacionales reales con datos limitados, entregando respuestas claras, basadas en evidencia y accesibles tanto desde API como desde un frontend simple.