Documentación RAG-assistant.

Etapa 0 (Previa): Configuración del entorno

Objetivo: Garantizar que todas las dependencias funcionen correctamente desde el inicio.
Contenido:

- Instalación de pyzbar (wrapper de ZBar para lectura de códigos) + opencv-python
- Configuración de drivers de cámara (evitar fallos en tiempo real)
- Script de verificación rápida (test_cam.py) para confirmar acceso a la webcam

Etapa 1:

Objetivo: Realizar un sistema RAG

Medios:

- NotebookLM
- Antigravity

Cuando voy a estar satisfecho: Cuando hable de los documentos proporcionados.

Etapa 2:

Que me de una interfaz minimalista, que sea fácil de acceder a todo para una mayor comodidad y fluidez para que el usuario promedio se siente agusto.

Fase 1: Arquitectura de Ingesta y Recuperación.

• Extracción y Chunking: Implementa la lógica de rag-engineer utilizando Docling para una extracción de tablas con 97.9% de precisión. Aplica un chunking recursivo de 512 tokens con un solapamiento del 10-20% para preservar el contexto semántico.

• Almacenamiento y Búsqueda: Utiliza vector-database-engineer para configurar un almacén de vectores local (como ChromaDB o Qdrant). Implementa una búsqueda híbrida que combine vectores semánticos con búsqueda léxica (BM25), fusionados mediante el algoritmo Reciprocal Rank Fusion (RRF) con un valor de k=60.

• Generación Blindada (Grounding): Configura un System Prompt estricto: 'Responde ÚNICAMENTE basándote en los documentos proporcionados. Si no está en las fuentes, indica que no tienes información suficiente'. Activa la lógica de Corrective RAG (CRAG) para evaluar la calidad del contexto antes de generar la respuesta

Fase 2: Interfaz de Usuario

• Usa frontend-developer para crear una interfaz en React con soporte Drag & Drop para archivos PDF, DOCX y MD, asegurando que cada respuesta incluya citaciones directas al documento fuente.

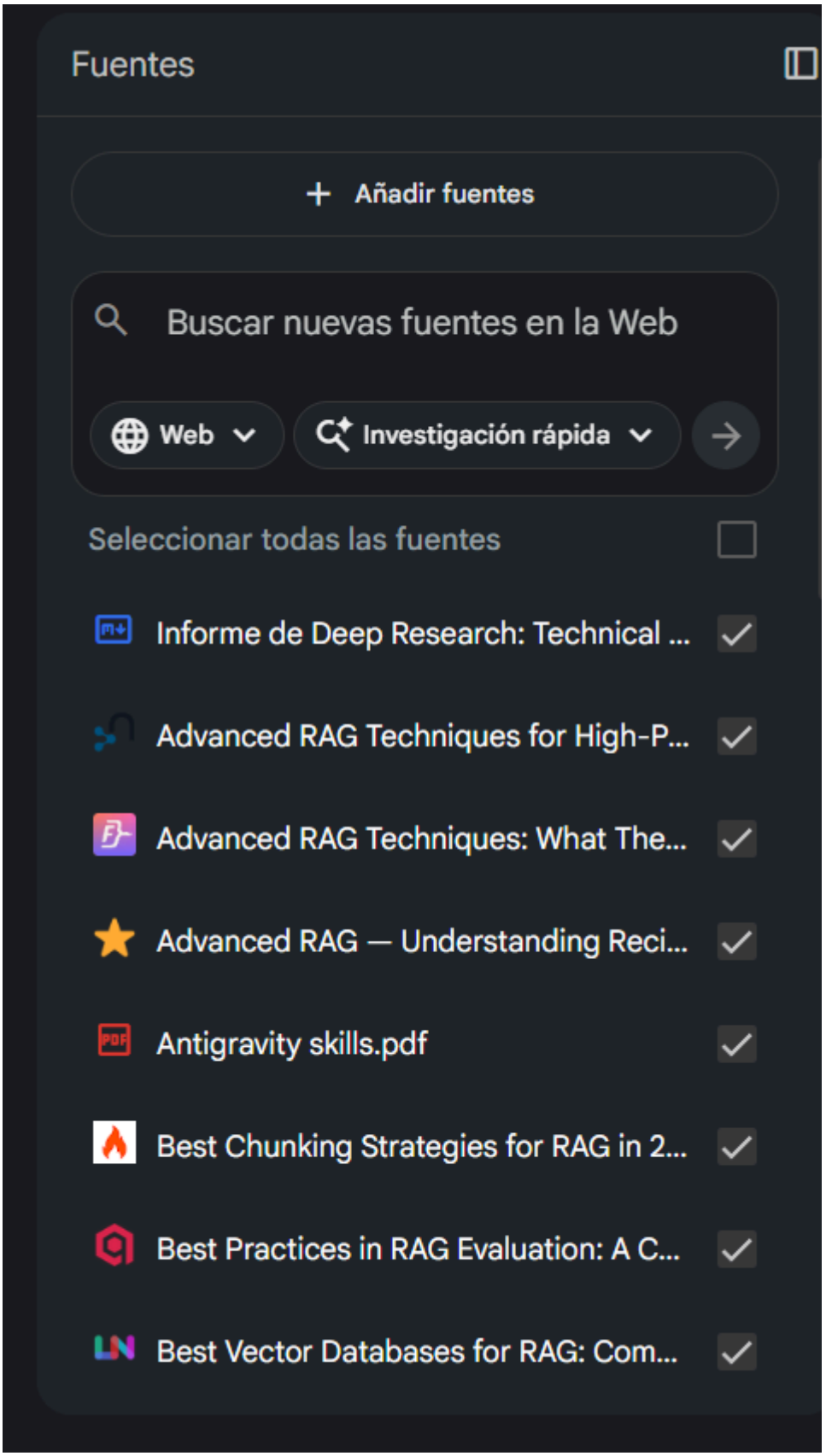
Fase 3: Documentación y Exposición de Posibilidades (Fase Crítica)

• Documentación de la Forma: Invoca a docs-architect y c4-architecture-c4-architecture para generar un README.md y diagramas técnicos que expliquen la estructura del proyecto (ingesta -> vectorización -> recuperación -> generación).

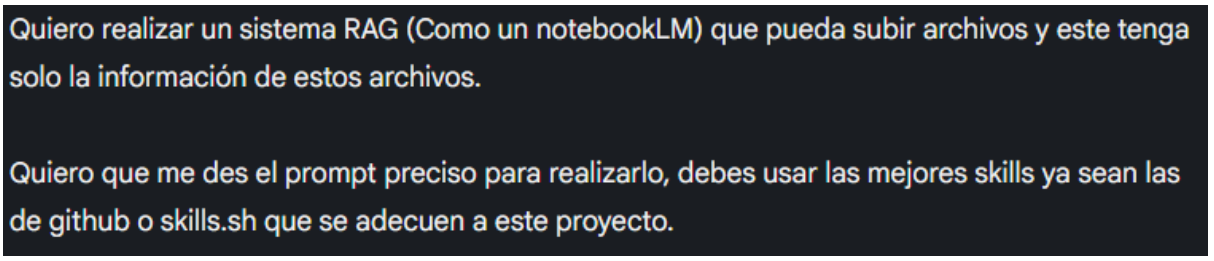
• Reporte de Posibilidades: Solicita al agente que elabore un informe de 'Posibilidades y Escalabilidad' que exponga cómo el sistema puede evolucionar hacia GraphRAG para conectar entidades complejas, o integrar agentes autónomos para tareas de razonamiento multi-paso.

Historial:

1. He buscado información con notebook LM para comprender que es un RAG (Retriveral Augmented Generation)

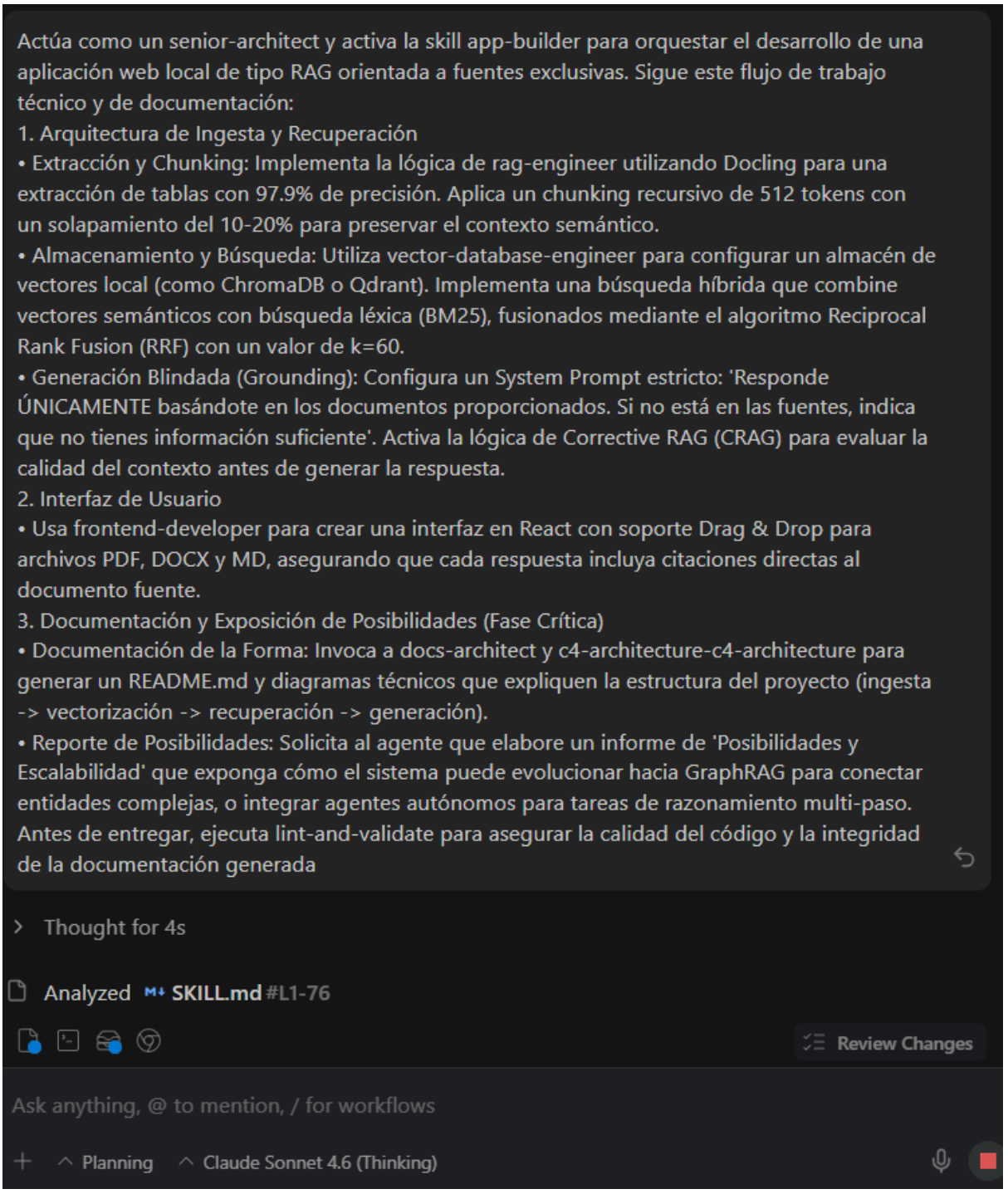


2. Proceder a crear el prompt:

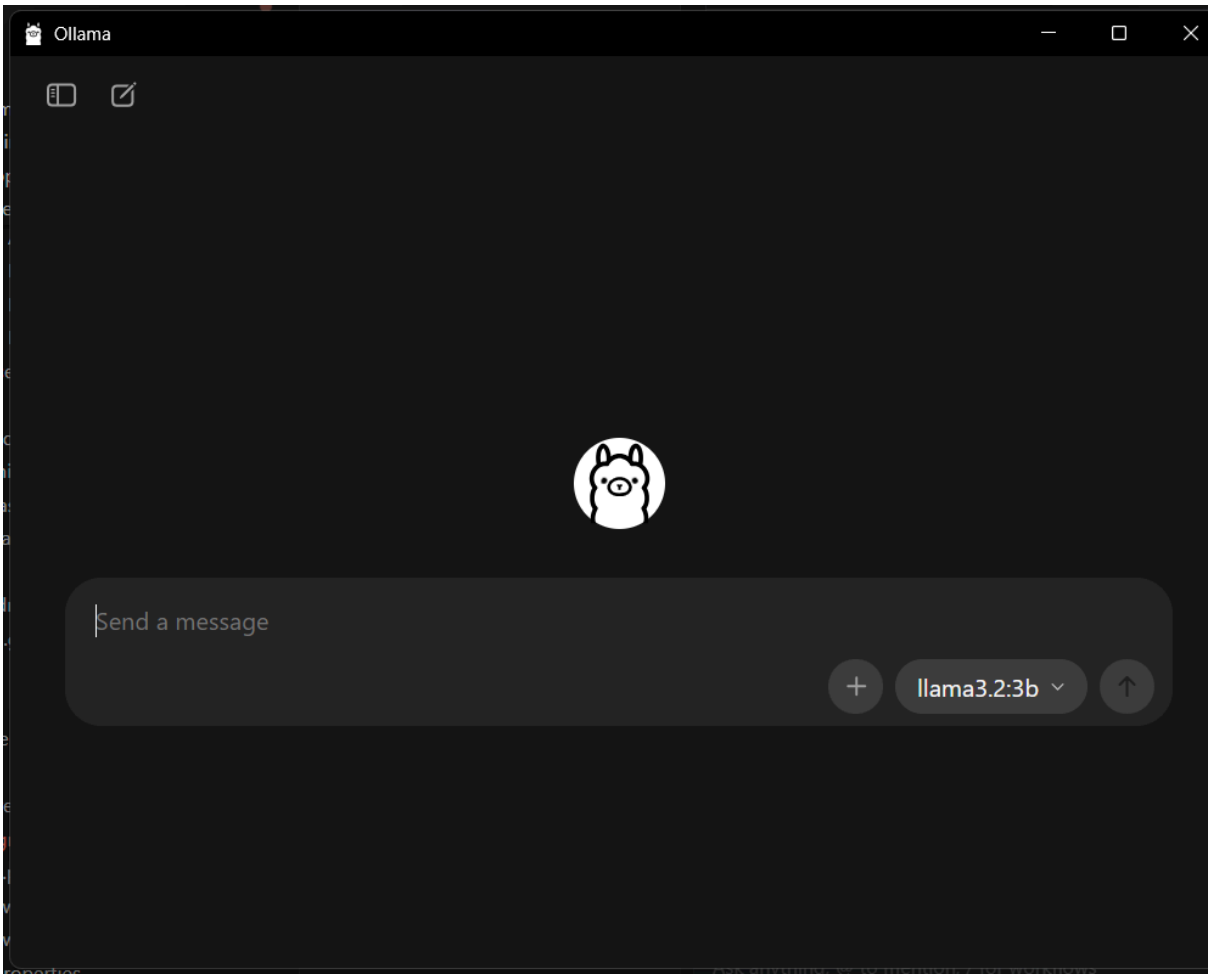


3. Confirmar el prompt:
   Actúa como un senior-architect y activa la skill app-builder para orquestar el desarrollo de una aplicación web local de tipo RAG orientada a fuentes exclusivas. Sigue este flujo de trabajo técnico y de documentación:
   1. Arquitectura de Ingesta y Recuperación
   • Extracción y Chunking: Implementa la lógica de rag-engineer utilizando Docling para una extracción de tablas con 97.9% de precisión. Aplica un chunking recursivo de 512 tokens con un solapamiento del 10-20% para preservar el contexto semántico.
   • Almacenamiento y Búsqueda: Utiliza vector-database-engineer para configurar un almacén de vectores local (como ChromaDB o Qdrant). Implementa una búsqueda híbrida que combine vectores semánticos con búsqueda léxica (BM25), fusionados mediante el algoritmo Reciprocal Rank Fusion (RRF) con un valor de k=60.
   • Generación Blindada (Grounding): Configura un System Prompt estricto: 'Responde ÚNICAMENTE basándote en los documentos proporcionados. Si no está en las fuentes, indica que no tienes información suficiente'. Activa la lógica de Corrective RAG (CRAG) para evaluar la calidad del contexto antes de generar la respuesta.
   2. Interfaz de Usuario
   • Usa frontend-developer para crear una interfaz en React con soporte Drag & Drop para archivos PDF, DOCX y MD, asegurando que cada respuesta incluya citaciones directas al documento fuente.
   3. Documentación y Exposición de Posibilidades (Fase Crítica)
   • Documentación de la Forma: Invoca a docs-architect y c4-architecture-c4-architecture para generar un README.md y diagramas técnicos que expliquen la estructura del proyecto (ingesta -> vectorización -> recuperación -> generación).
   • Reporte de Posibilidades: Solicita al agente que elabore un informe de 'Posibilidades y Escalabilidad' que exponga cómo el sistema puede evolucionar hacia GraphRAG para conectar entidades complejas, o integrar agentes autónomos para tareas de razonamiento multi-paso.
   Antes de entregar, ejecuta lint-and-validate para asegurar la calidad del código y la integridad de la documentación generada

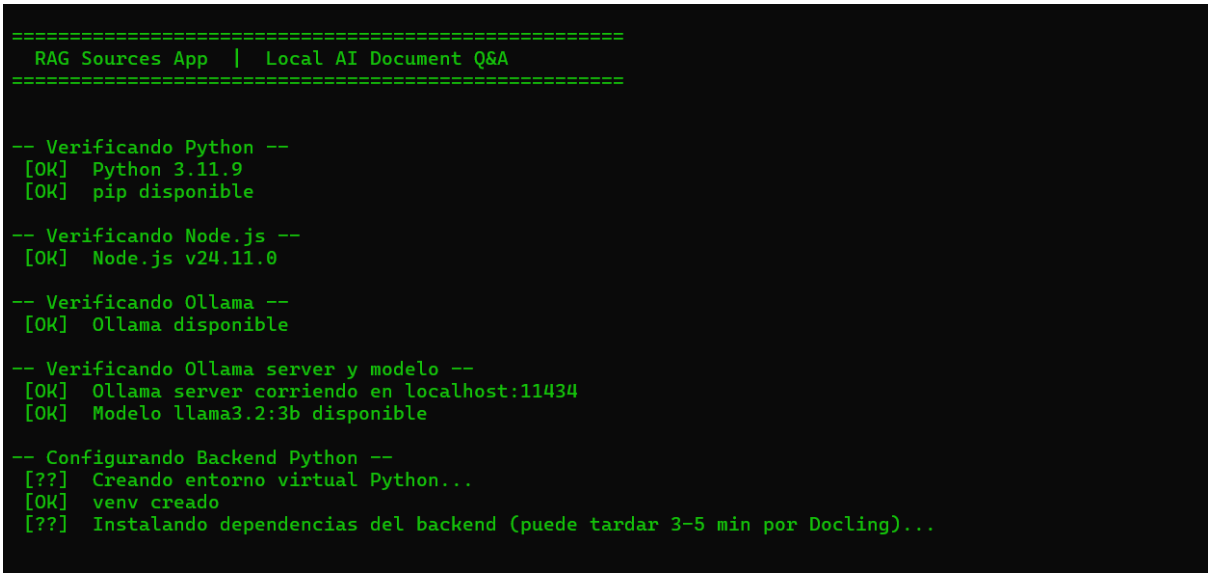
4. Subir el propmt a nuestro agente favorito, en mi caso, Antigravity:



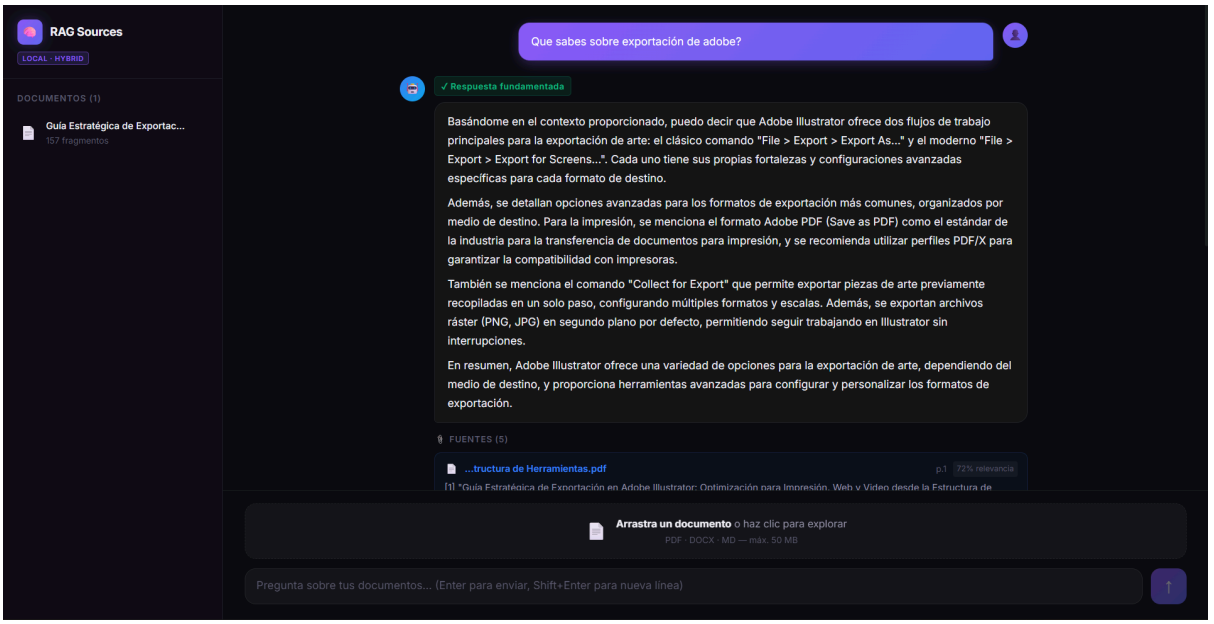
5. Descargar Ollama:



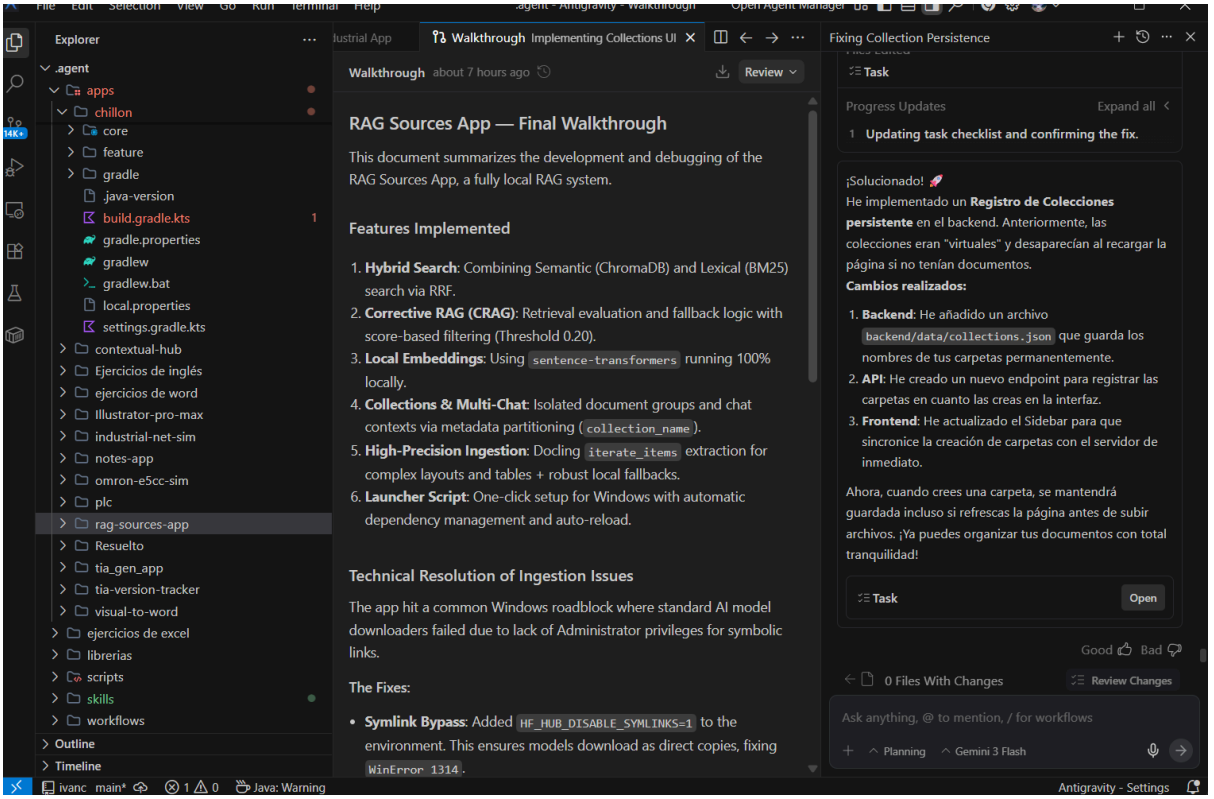
6. Aceptar todo lo que nos diga…
7. Pedirle que nos haga un start.bat para un inicio cómodo.
8. Ejecutar el start.bat y que este se revise / descargue todas las dependencias necesarias para este proyecto.



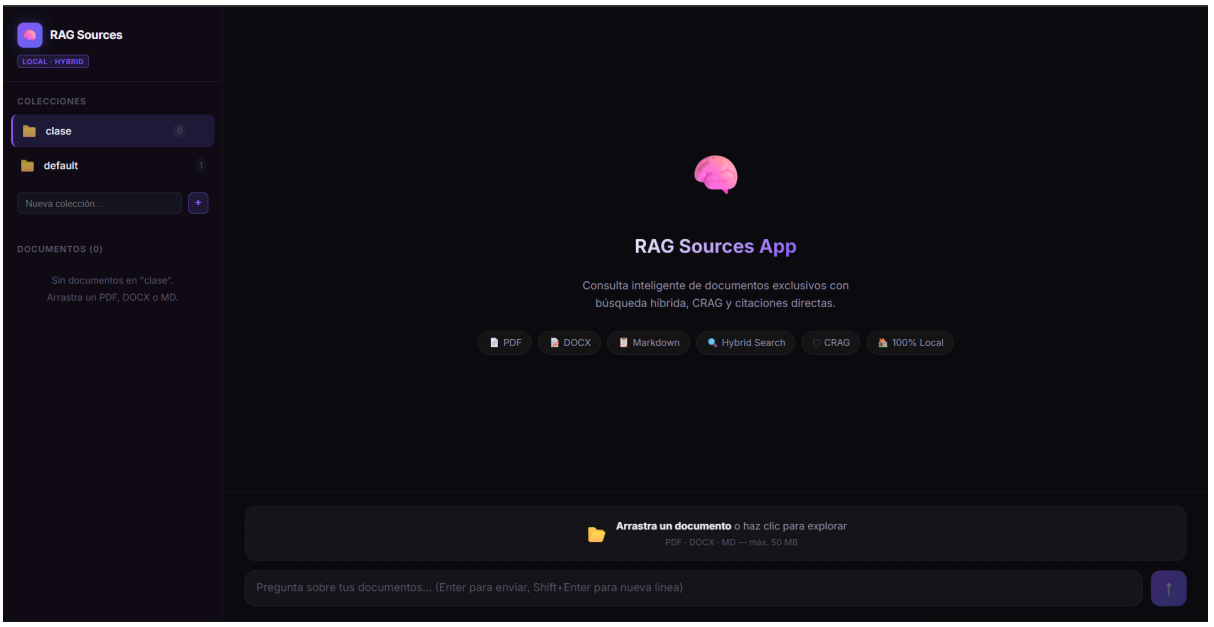
9. Comprobamos que funcione subiendo un archivo y preguntando sobre este:



10. Añadimos una función de crear carpetas para varios chats para evitar que se mezcle la información proporcionada.



11. Revisamos que es funcional.



12. Podemos dar ya el proyecto por finalizado, ya cumple con todos nuestros requisitos.