MiniRAG Assistant

Este proyecto permite hacer RAG (Retrieval-Augmented Generation) usando Chroma en Docker, embeddings de HuggingFace y consultas a un LLM de OpenAI.

IMPORTANTE: Antes de ejecutar el proyecto, asegúrate de tener **Docker Desktop instalado y abierto**, y que el servicio de Chroma esté corriendo (`docker compose up -d`).

Tabla de Contenidos

---

1) Preparar el proyecto

Estructura de proyecto

```
mkdir mini-rag-assistant `
; cd mini-rag-assistant `
; mkdir docs `
; New-Item app.py -ItemType File `
; New-Item indexer.py -ItemType File `
; New-Item verify_chroma.py -ItemType File `
; New-Item inspect_chroma.py -ItemType File `
; New-Item reset_chroma.py -ItemType File `
; New-Item requirements.txt -ItemType File `
; New-Item .env -ItemType File `
; New-Item docker-compose.yml -ItemType File
```

---

2) Docker: Chroma remoto

```
services:
  chroma:
    image: chromadb/chroma:latest
    container_name: chroma
    environment:
      CHROMA_SERVER_HOST: 0.0.0.0
      CHROMA_SERVER_HTTP_PORT: 8000
      ALLOW_RESET: "true"
    ports:
      - "8000:8000"
    volumes:
      - ./chroma_data:/chroma/.chroma
    restart: unless-stopped
```

```
docker compose up -d
```

---

3) Entorno Python

```
py -3.11 -m venv venv
.\venv\Scripts\Activate
```

```
chromadb==1.0.20
langchain==0.3.27
langchain-core==0.3.74
langchain-community==0.3.27
langchain-chroma==0.2.5
langchain-huggingface==0.3.1
langchain-openai==0.3.32
langchain-text-splitters==0.3.9
streamlit==1.37.1
python-dotenv==1.0.1
pypdf==4.2.0
sentence-transformers==2.6.1
```

```
pip install -r requirements.txt
```

---

4) Variables de entorno

```
OPENAI_API_KEY=tu_api_key_sin_comillas
```

---

5) Coloca tus documentos

Crea TXT de prueba en la carpeta `docs/`.

---

6) Indexador (Chroma remoto + HF embeddings)

```
# indexer.py completo
" + `import os
import time
import chromadb
from chromadb.config import Settings
from dotenv import load_dotenv
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.document_loaders import TextLoader, PyPDFLoader

load_dotenv()

HOST = "localhost"
PORT = 8000
COLLECTION = "mini_rag"
DOCS_DIR = "docs"

def load_docs(docs_dir: str):
    if not os.path.exists(docs_dir):
        print(f"❌ No existe la carpeta '{docs_dir}'.")
        return []

    files = [f for f in os.listdir(docs_dir) if f.lower().endswith((".txt",
".pdf"))]
    if not files:
        print(f"❌ No hay .txt/.pdf en '{docs_dir}'.")
        return []

    print(f"☣️ Archivos en '{docs_dir}': {files}")
    docs = []
    for fname in files:
        fpath = os.path.join(docs_dir, fname)
        if fname.lower().endswith(".txt"):
            loader = TextLoader(fpath, encoding="utf-8")
        else:
            loader = PyPDFLoader(fpath)
        loaded = loader.load()
        print(f"☪️ '{fname}' → {len(loaded)} documento(s)")
        docs.extend(loaded)
    return docs

def main():
    documents = load_docs(DOCS_DIR)
    if not documents:
        return

    print(f"1️⃣ Dividiendo {len(documents)} documentos en chunks...")
    splitter = RecursiveCharacterTextSplitter(chunk_size=600,
chunk_overlap=80)
    chunks = splitter.split_documents(documents)
    print(f"✅ Total de chunks: {len(chunks)}")
    if not chunks:
        print("❌ No se generaron chunks.")
        return

    print(" 9 Cargando modelo de embeddings (HF MiniLM-L6-v2)...")
    embedder = HuggingFaceEmbeddings(model_name="sentence-transformers/all-
MiniLM-L6-v2")
    test_vec = embedder.embed_query("prueba")
    print(f"🔢 Dimensión de embedding (query): {len(test_vec)}")

    print("💹 Conectando a Chroma remoto...")
    client = chromadb.HttpClient(host=HOST, port=PORT,
settings=Settings(anonymized_telemetry=False))
    print("✅ Conectado")

    print(f"☢️ Preparando colección '{COLLECTION}'...")
    try:
        client.delete_collection(COLLECTION)
        print(" 6 Colección previa eliminada")
    except Exception:
        print("ℹ️ Colección previa no existía, continuamos")

    collection = client.get_or_create_collection(name=COLLECTION)
    print("✅ Colección lista")

    print("♑ Insertando chunks en batch...")
    BATCH = 64
    ids, texts = [], []
    for i, c in enumerate(chunks, 1):
        ids.append(f"doc_{i}")
        texts.append(c.page_content)

        if len(ids) == BATCH or i == len(chunks):
            print(f"   → Batch con {len(ids)} items… (generando embeddings)")
            embs = embedder.embed_documents(texts)
            collection.add(ids=ids, documents=texts, embeddings=embs)
            print(f"   ✅ Upsert de {len(ids)} items")
            ids, texts = [], []

    time.sleep(0.4)
    count = collection.count()
    print(f"‼️ Indexación completa. Documentos en '{COLLECTION}': {count}")
```

```
if __name__ == "__main__":
    main()
```

```
python indexer.py
```

7) Inspección de colección

```
# inspect_chroma.py completo
import chromadb
from chromadb.config import Settings

client = chromadb.HttpClient(host="localhost", port=8000,
settings=Settings(anonymized_telemetry=False))
COLLECTION_NAME = "mini_rag"

try:
    collection = client.get_collection(COLLECTION_NAME)
except Exception as e:
    print(f"❌ No se encontró la colección '{COLLECTION_NAME}'. Error: {e}")
    exit()

print(f"☣️ Inspeccionando colección: {COLLECTION_NAME}")
count = collection.count()
print(f"🈂️ Total de documentos: {count}")

if count == 0:
    print("⚠️ La colección está vacía.")
    exit()

batch_size = 5
for offset in range(0, count, batch_size):
    results = collection.get(include=["documents", "metadatas"],
limit=batch_size, offset=offset)
    for i, doc in enumerate(results["documents"]):
        doc_id = results["ids"][i]
        meta = results["metadatas"][i]
        preview = doc[:100].replace("\n", " ") + "..." if len(doc) > 100 else
doc
        print(f"\n🆔 {doc_id}")
        print(f"☪️ Texto: {preview}")
        print(f"⤴️ Metadatos: {meta}")
```

```
python inspect_chroma.py
```

8) App de Streamlit (RAG con LLM)

```
# app.py completo
import streamlit as st
import chromadb
from chromadb.config import Settings
from dotenv import load_dotenv
from langchain_huggingface import HuggingFaceEmbeddings
from langchain_chroma import Chroma
from langchain_openai import ChatOpenAI
from langchain.chains import RetrievalQA

load_dotenv()
HOST = "localhost"
PORT = 8000
COLLECTION = "mini_rag"

st.set_page_config(page_title="Mini RAG Assistant", page_icon="🟫")
st.title("🟫 Mini RAG Assistant")
st.write("Haz preguntas sobre tus documentos indexados en Chroma remoto.")

query = st.text_input("Tu pregunta:")

if query:
    with st.spinner("Procesando..."):
        st.write(" 🔹 Inicializando embeddings (HF MiniLM-L6-v2)...")
        embeddings = HuggingFaceEmbeddings(model_name="sentence-transformers/
all-MiniLM-L6-v2")
        st.success("✅ Embeddings inicializados")
        st.write(" 🔹 Conectando a Chroma remoto…")
        client = chromadb.HttpClient(host=HOST, port=PORT,
settings=Settings(anonymized_telemetry=False))
        st.success("✅ Conexión con Chroma exitosa")
        st.write(" 🔹 Cargando vectorstore…")
        vectorstore = Chroma(client=client, collection_name=COLLECTION,
embedding_function=embeddings)
        retriever = vectorstore.as_retriever(search_kwargs={"k": 3})
        st.success("✅ Retriever listo")
        st.write(" 🔹 Inicializando LLM…")
        llm = ChatOpenAI(model="gpt-4o-mini", temperature=0)
        st.success("✅ LLM listo")
        st.write(" 🔹 Construyendo chain de RAG…")
        qa_chain = RetrievalQA.from_chain_type(llm=llm, retriever=retriever,
return_source_documents=True)
        st.success("✅ Chain creada")
        st.write(" 🔹 Consultando el modelo…")
        result = qa_chain.invoke({"query": query})
        st.success("✅ Consulta completada")
        st.markdown("### Respuesta")
        st.write(result["result"])
        st.markdown("### ⚪ Documentos fuente")
        src_docs = result.get("source_documents", [])
        if src_docs:
            for i, doc in enumerate(src_docs, 1):
                st.markdown(f"**Documento {i}:**")
                st.write(doc.page_content[:600] + "…")
                st.caption(f"Metadata: {doc.metadata}")
        else:
            st.warning("⚠️ No se encontraron documentos relevantes.")
        st.markdown("### 🈚 Top-K por similitud (debug)")
        sims = vectorstore.similarity_search_with_score(query, k=3)
        for j, (d, score) in enumerate(sims, 1):
            st.write(f"**{j}.** score={score:.4f}")
            st.caption(d.page_content[:200] + "…")
```

```
streamlit run app.py
```

9) Resetear contenido

```
# reset_chroma.py completo
import chromadb
from chromadb.config import Settings
HOST = "localhost"
PORT = 8000
COLLECTION = "mini_rag"
client = chromadb.HttpClient(host=HOST, port=PORT,
settings=Settings(anonymized_telemetry=False))
try:
    client.delete_collection(COLLECTION)
    print(f" 6 Colección '{COLLECTION}' eliminada.")
except Exception as e:
    print("ℹ️ La colección no existía:", e)
# client.reset() # opción B, opcional
```

```
python reset_chroma.py
```

10) Problemas comunes & cómo solucionarlos

- Dimensión de embeddings no coincide (384 vs 1536)
- Streamlit “se desconecta” o cuelga
- ImportError: cannot import name 'Chroma'... tras crear un inspect.py
- No logra borrar carpeta local por Docker