I am working as a shadow product manager and building a portfolio of products for my interview, for the uploaded project tasks, i have completed till Day 5, now help break down DAY 6 in easy steps with easy explanation as well and codes

Day 6 focuses on creating a local vector database and embedding the de-identified note chunks so retrieval will work on Day 7. Here is a clear, step-by-step breakdown with simple explanations and ready-to-run Python scripts, aligned exactly with the Day 6 objectives and artifacts in the uploaded plan.[1]

What to do today

Create chunking + embedding + vector DB persistence so de-identified notes from Day 5 can be searched semantically, producing indexer.py and a local stored index.[1]

Folder setup

- Use existing folders from earlier days: /app, /data/synthetic_notes, /data/outputs, /secure_store.[1]
- Create a new folder for the vector DB, for example /data/vector_store (auto-created by the script).[1]

Step 1: Install required libraries

- Two solid choices: FAISS (fast) or Chroma (easy persistence) as per the plan; embeddings via sentence-transformers/all-MiniLM-L6-v2 or Bio_ClinicalBERT for clinical flavor.[1]
- Commands:
  - pip install faiss-cpu sentence-transformers chromadb tqdm pyyaml.[1]

Step 2: Choose the vector store

- Option A (recommended for simplicity): Chroma persists to disk automatically (good for demos).[1]
- Option B: FAISS for speed; persistence requires saving index files and metadata; both are supported below.[1]

Step 3: Prepare inputs

- Use de-identified text outputs from Day 5 as the only source for indexing to stay HIPAA-aligned; do not index original PHI.[1]
- Ensure /data/outputs contains de-identified .txt files or JSON with de-identified text fields; the script supports .txt and .json.[1]

Step 4: Chunking strategy

- Chunk to 200–400 tokens with a small overlap (e.g., 40–60) to preserve context across boundaries, matching the Day 6 guidance.[1]
- Simple approach: word-based chunking with an approximate token size; this is sufficient for the MVP.[1]

Step 5: Embedding model

- Default: sentence-transformers/all-MiniLM-L6-v2 (fast, 384-dim).[1]
- Optional: swap to Bio_ClinicalBERT embeddings if desired for domain semantics later (slower, extra setup).[1]

Step 6: Run indexer.py to build the DB

- The script:
  - Loads de-identified notes from /data/outputs.
  - Chunks each note into 200–400 token-like segments.
  - Computes embeddings.
  - Writes to Chroma (disk) or FAISS files with metadata like note_id and section.[1]

Step 7: Verify

- After running, confirm a persisted database exists (e.g., /data/vector_store/chroma.sqlite for Chroma or .faiss + .pkl files for FAISS).[1]
- Keep this for Day 7 retriever + RAG pipeline.[1]

indexer.py (drop-in, end-to-end)

- Place this in app/indexer.py and run: python app/indexer.py —input_dir ./data/outputs —db_type chroma —persist_dir ./data/vector_store.[1]

```
# app/indexer.py
# Day 6: Vector store & embeddings
# Usage examples:
#   python app/indexer.py --input_dir ./data/outputs --db_type chroma --persist_dir ./dat
#   python app/indexer.py --input_dir ./data/outputs --db_type faiss  --persist_dir ./dat

import os
import json
```

```
import argparse
from pathlib import Path
from typing import List, Dict, Tuple
from tqdm import tqdm

# Embeddings
from sentence_transformers import SentenceTransformer

# Vector stores
# Chroma
import chromadb
from chromadb.config import Settings as ChromaSettings

# FAISS
import faiss
import pickle

DEFAULT_CHUNK_TOKENS = 300
DEFAULT_OVERLAP_TOKENS = 50

def read_note_files(input_dir: str) -> List[Dict]:
    """
    Reads de-identified notes from .txt or .json in input_dir.
    Expects .json to have a 'text' field containing de-identified content.
    Returns list of dicts: {id, text, section?}
    """
    items = []
    p = Path(input_dir)
    if not p.exists():
        raise FileNotFoundError(f"Input dir not found: {input_dir}")

    for fp in p.glob("**/*"):
        if fp.is_dir():
            continue
        if fp.suffix.lower() == ".txt":
            text = fp.read_text(encoding="utf-8", errors="ignore").strip()
            if text:
                items.append({"id": fp.stem, "text": text, "section": None})
        elif fp.suffix.lower() == ".json":
            try:
                obj = json.loads(fp.read_text(encoding="utf-8", errors="ignore"))
                text = obj.get("text") or obj.get("deidentified_text") or ""
                section = obj.get("section")
                if text:
                    items.append({"id": fp.stem, "text": text.strip(), "section": section
            except Exception:
                # Skip malformed
                continue
    return items

def approx_tokenize(text: str) -> List[str]:
    """
    Approximate tokenization by splitting on whitespace.
    For MVP this is fine; can replace with tiktoken later.
    """
    return text.split()
```

```
def detokenize(tokens: List[str]) -> str:
    return " ".join(tokens)

def chunk_text(text: str, chunk_tokens: int, overlap_tokens: int) -> List[str]:
    """
    Simple sliding window chunking.
    """
    tokens = approx_tokenize(text)
    chunks = []
    i = 0
    n = len(tokens)
    while i < n:
        j = min(i + chunk_tokens, n)
        chunk = detokenize(tokens[i:j])
        if chunk.strip():
            chunks.append(chunk)
        if j == n:
            break
        i = j - overlap_tokens
        if i < 0:
            i = 0
    return chunks

def embed_texts(model: SentenceTransformer, texts: List[str]):
    return model.encode(texts, show_progress_bar=False, convert_to_numpy=True, normalize_

def build_chroma(persist_dir: str, collection_name: str = "notes"):
    client = chromadb.PersistentClient(
        path=persist_dir,
        settings=ChromaSettings(allow_reset=True)
    )
    if collection_name in [c.name for c in client.list_collections()]:
        coll = client.get_collection(collection_name)
    else:
        coll = client.create_collection(collection_name)
    return client, coll

def save_faiss(index, vectors_meta: List[Dict], persist_dir: str):
    os.makedirs(persist_dir, exist_ok=True)
    faiss_path = os.path.join(persist_dir, "index.faiss")
    meta_path = os.path.join(persist_dir, "meta.pkl")
    faiss.write_index(index, faiss_path)
    with open(meta_path, "wb") as f:
        pickle.dump(vectors_meta, f)

def load_faiss(persist_dir: str):
    faiss_path = os.path.join(persist_dir, "index.faiss")
    meta_path = os.path.join(persist_dir, "meta.pkl")
    if os.path.exists(faiss_path) and os.path.exists(meta_path):
        index = faiss.read_index(faiss_path)
        with open(meta_path, "rb") as f:
            meta = pickle.load(f)
        return index, meta
    return None, []
```

```
def main():
    parser = argparse.ArgumentParser(description="Day 6: Build local vector DB from de-id
    parser.add_argument("--input_dir", required=True, help="Directory with de-identified
    parser.add_argument("--persist_dir", default="./data/vector_store", help="Where to pe
    parser.add_argument("--db_type", choices=["chroma", "faiss"], default="chroma", help=
    parser.add_argument("--model_name", default="sentence-transformers/all-MiniLM-L6-v2",
    parser.add_argument("--chunk_tokens", type=int, default=DEFAULT_CHUNK_TOKENS, help="A
    parser.add_argument("--overlap_tokens", type=int, default=DEFAULT_OVERLAP_TOKENS, hel
    parser.add_argument("--collection", default="notes", help="Collection name (Chroma).'
    args = parser.parse_args()

    notes = read_note_files(args.input_dir)
    if not notes:
        print(f"No de-identified notes found in {args.input_dir}. Ensure Day 5 outputs ex
        return

    print(f"Loaded {len(notes)} de-identified notes from {args.input_dir}")
    os.makedirs(args.persist_dir, exist_ok=True)

    print(f"Loading embedding model: {args.model_name}")
    model = SentenceTransformer(args.model_name)

    all_chunk_texts = []
    all_chunk_ids = []
    all_metadata = []

    print("Chunking notes...")
    for note in tqdm(notes):
        chunks = chunk_text(note["text"], args.chunk_tokens, args.overlap_tokens)
        for idx, ch in enumerate(chunks):
            cid = f"{note['id']}::chunk_{idx}"
            all_chunk_texts.append(ch)
            all_chunk_ids.append(cid)
            all_metadata.append({
                "note_id": note["id"],
                "chunk_index": idx,
                "section": note.get("section")
            })

    print(f"Total chunks: {len(all_chunk_texts)}")

    print("Embedding chunks...")
    vectors = embed_texts(model, all_chunk_texts)

    if args.db_type == "chroma":
        print("Building Chroma persistent collection...")
        client, coll = build_chroma(args.persist_dir, args.collection)

        # Upsert in manageable batches
        batch = 512
        for i in tqdm(range(0, len(all_chunk_texts), batch)):
            j = min(i + batch, len(all_chunk_texts))
            coll.upsert(
                ids=all_chunk_ids[i:j],
                embeddings=vectors[i:j].tolist(),
                documents=all_chunk_texts[i:j],
```

```
                metadatas=all_metadata[i:j],
            )
        print(f"Chroma collection '{args.collection}' persisted at {args.persist_dir}")

    elif args.db_type == "faiss":
        print("Building FAISS index...")
        d = vectors.shape[^1_1]
        index = faiss.IndexFlatIP(d)  # normalized vectors → use inner product as cosine
        # Try to load existing
        existing_index, existing_meta = load_faiss(args.persist_dir)
        if existing_index is not None:
            print("Appending to existing FAISS index...")
            index = existing_index
            vectors_meta = existing_meta
        else:
            vectors_meta = []
        index.add(vectors)
        vectors_meta.extend([
            {
                "id": all_chunk_ids[k],
                "text": all_chunk_texts[k],
                "meta": all_metadata[k]
            } for k in range(len(all_chunk_texts))
        ])
        save_faiss(index, vectors_meta, args.persist_dir)
        print(f"FAISS index persisted at {args.persist_dir}")

    print("Done.")

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

How to run

- For Chroma (recommended):
  - python app/indexer.py —input_dir ./data/outputs —db_type chroma —persist_dir ./data/vector_store —collection notes.[1]
- For FAISS:
  - python app/indexer.py —input_dir ./data/outputs —db_type faiss —persist_dir ./data/vector_store_faiss.[1]

What this produces

- An artifact indexer.py and a persisted local vector DB containing chunk embeddings with metadata (note_id, section), exactly as required by Day 6.[1]

Tips and next steps

- Keep chunk_tokens between 200–400 and overlap 40–60 to match Day 6 guidance for retrieval quality tomorrow.[1]
- Tomorrow (Day 7), wire a retriever using this DB (Haystack or LangChain) and feed top_k results into a generator for structured summaries.[1]

⁂

1. COMPLETE-WEEK-BY-WEEK-TASKS.docx