Private Chatbot — Ready-to-run Code Files

This document contains all the files you asked for. Save each section into the filename shown in the header (e.g., `ingest.py`, `index_builder.py`, etc.).

---

requirements.txt

```
streamlit
sentence-transformers
faiss-cpu
pdfplumber
python-docx
torch
transformers
regex
tqdm
nltk
```

Notes: If you don't want local generation remove `transformers` and keep only the retriever-related requirements.

---

ingest.py

```python
from pathlib import Path
from typing import List, Dict
import pdfplumber
import docx


def extract_text_from_pdf(path: Path) -> str:
    texts = []
    with pdfplumber.open(path) as pdf:
        for p in pdf.pages:
            texts.append(p.extract_text() or "")
    return "\n".join(texts)


def extract_text_from_docx(path: Path) -> str:
    doc = docx.Document(path)
    return "\n".join(p.text for p in doc.paragraphs)
```

```
# Simple character-based chunker with overlap
def chunk_text(text: str, chunk_size: int = 800, overlap: int = 200) ->
List[Dict]:
    chunks = []
    start = 0
    n = len(text)
    i = 0
    while start < n:
        end = min(start + chunk_size, n)
        chunk = text[start:end].strip()
        if chunk:
            chunks.append({"id": i, "text": chunk, "start": start, "end": end})
            i += 1
        start += chunk_size - overlap
    return chunks


def ingest_folder(folder_path: str) -> List[Dict]:
    p = Path(folder_path)
    all_chunks = []
    for file in p.iterdir():
        if file.suffix.lower() == '.pdf':
            text = extract_text_from_pdf(file)
        elif file.suffix.lower() == '.docx':
            text = extract_text_from_docx(file)
        else:
            continue
        chunks = chunk_text(text)
        for c in chunks:
            c['source'] = file.name
        all_chunks.extend(chunks)
    return all_chunks
```

---

index_builder.py

```
from sentence_transformers import SentenceTransformer
import numpy as np
import faiss
import pickle

MODEL_NAME = "all-MiniLM-L6-v2"
```

```
def build_index(chunks, model_name=MODEL_NAME, index_path="faiss_index.bin",
meta_path="meta.pkl"):
    model = SentenceTransformer(model_name)
    texts = [c['text'] for c in chunks]
    embeddings = model.encode(texts, show_progress_bar=True,
convert_to_numpy=True)

    # faiss expects float32
    embeddings = embeddings.astype('float32')
    dim = embeddings.shape[1]

    index = faiss.IndexFlatL2(dim)
    index.add(embeddings)

    faiss.write_index(index, index_path)
    with open(meta_path, 'wb') as f:
        pickle.dump(chunks, f)

    print(f"Saved FAISS index to {index_path} and metadata to {meta_path}")


if __name__ == '__main__':
    import argparse
    from ingest import ingest_folder

    parser = argparse.ArgumentParser()
    parser.add_argument('--folder', required=True, help='Folder with PDFs/DOCX
to ingest')
    parser.add_argument('--index', default='faiss_index.bin')
    parser.add_argument('--meta', default='meta.pkl')
    args = parser.parse_args()

    chunks = ingest_folder(args.folder)
    build_index(chunks, index_path=args.index, meta_path=args.meta)
```

search.py

```
from sentence_transformers import SentenceTransformer
import faiss
import pickle

MODEL_NAME = "all-MiniLM-L6-v2"

class Retriever:
    def __init__(self, index_path='faiss_index.bin', meta_path='meta.pkl',
model_name=MODEL_NAME):
        self.model = SentenceTransformer(model_name)
        self.index = faiss.read_index(index_path)
        with open(meta_path, 'rb') as f:
            self.meta = pickle.load(f)

    def retrieve(self, query: str, top_k: int = 5):
        q_emb = self.model.encode([query], convert_to_numpy=True)
        q_emb = q_emb.astype('float32')
        D, I = self.index.search(q_emb, top_k)
        results = []
        for idx, dist in zip(I[0], D[0]):
            meta = self.meta[idx]
            results.append({"score": float(dist), "text": meta['text'], "meta":
meta})
        return results


if __name__ == '__main__':
    r = Retriever()
    q = input('Query: ')
    res = r.retrieve(q, top_k=5)
    for i, r in enumerate(res, 1):
        print('---')
        print(i, r['score'], r['meta'].get('source'))
        print(r['text'])
```

local_gen.py (optional — only if you want a local generation step)

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import torch

MODEL = "distilgpt2"  # small example; replace with your local model
DEVICE = 'cuda' if torch.cuda.is_available() else 'cpu'


def generate_answer(context: str, question: str, model_name=MODEL,
max_new_tokens: int = 150):
    tokenizer = AutoTokenizer.from_pretrained(model_name)
    model = AutoModelForCausalLM.from_pretrained(model_name).to(DEVICE)

    prompt = f"Context:\n{context}\n\nQuestion: {question}\nAnswer:"
    inputs = tokenizer(prompt, return_tensors='pt').to(DEVICE)
    outputs = model.generate(**inputs, max_new_tokens=max_new_tokens,
do_sample=False)
    text = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return text[len(prompt):].strip()
```

Notes: `from_pretrained` will download weights the first time. Place the model in a local cache to avoid repeated downloads.

---

streamlit_app.py

```
import streamlit as st
import tempfile
import os
from ingest import extract_text_from_pdf, extract_text_from_docx, chunk_text
from index_builder import build_index
from search import Retriever

INDEX_PATH = 'faiss_index.bin'
META_PATH = 'meta.pkl'

st.set_page_config(page_title='Private Document Chatbot')
st.title('Private Document Chatbot (Local)')

st.markdown('Upload PDF or DOCX files. All indexing/search stays local.')

uploaded_files = st.file_uploader('Upload PDF or DOCX',
accept_multiple_files=True)
if uploaded_files:
    all_chunks = []
    for uf in uploaded_files:
        suffix = uf.name.split('.')[-1].lower()
        with tempfile.NamedTemporaryFile(delete=False, suffix='.' + suffix) as
tmp:
            tmp.write(uf.getbuffer())
            tmp_path = tmp.name
        if suffix in ['pdf']:
            text = extract_text_from_pdf(tmp_path)
        elif suffix in ['docx']:
            text = extract_text_from_docx(tmp_path)
        else:
            st.warning(f"Unsupported file type: {suffix}")
            continue
        chunks = chunk_text(text)
        for c in chunks:
            c['source'] = uf.name
        all_chunks.extend(chunks)

    if all_chunks:
        st.info('Building index — this may take a moment')
        build_index(all_chunks, index_path=INDEX_PATH, meta_path=META_PATH)
        st.success('Index built and saved locally.')

# Load existing index if present
if os.path.exists(INDEX_PATH) and os.path.exists(META_PATH):
    try:
        retriever = Retriever(INDEX_PATH, META_PATH)
    except Exception as e:
        st.error(f"Failed to load index: {e}")
        retriever = None

    if retriever:
        query = st.text_input('Ask a question about your documents:')
        top_k = st.slider('Top K', min_value=1, max_value=10, value=5)

        if st.button('Search') and query:
            results = retriever.retrieve(query, top_k=top_k)
            st.write('Top passages:')
            for i, r in enumerate(results, 1):
                st.markdown(f"**Result {i} — score {r['score']:.4f} — source:
{r['meta'].get('source','unknown')}**")
                st.write(r['text'])

            if st.checkbox('Generate concise answer (local LLM)'):
                try:
                    from local_gen import generate_answer
                    context = "\n\n".join([r['text'] for r in results])
                    with st.spinner('Generating...'):
                        answer = generate_answer(context, query)
                        st.subheader('Generated Answer')
                        st.write(answer)
                except Exception as e:
                    st.error(f"Generation failed: {e}")

else:
    st.info('No index found yet. Upload files above to create an index.')
```

How to run

1. Create a virtual environment and install dependencies:

```
python -m venv venv
source venv/bin/activate   # linux/mac
venv\Scripts\activate     # windows
pip install -r requirements.txt
```

1. Index a folder (optional) or use Streamlit uploader:

```
python index_builder.py --folder ./my_docs
```

1. Run Streamlit app:

```
streamlit run streamlit_app.py
```

Notes & caveats

- The first run of `sentence-transformers` or `transformers` will download model weights — you need internet for the initial download. Once the model is cached, the app is offline.
- For scanned PDFs you need OCR (Tesseract) — not included by default.
- For large corpora consider FAISS IVF and persisting embeddings on disk.

If you'd like, I can now: - package these into a ZIP in the sandbox so you can download them directly, or - adapt the app to strictly **avoid** any model downloads by using a pre-downloaded local model path, or - remove the local LLM pieces and optimize for speed and small memory footprint.

Tell me which of those you'd like next and I'll prepare it.