High-Level Design (HLD): Healthcare PDF Hub (Modular)

1) Purpose & Goals

Purpose. A Streamlit application that lets users upload, browse, and retrieve knowledge from healthcare PDFs (Medical Documents, Medicine Details, Hospital Details). It adds lightweight RAG (retrieve-and-generate) using FAISS + sentence-transformer embeddings and an LLM (Euri AI wrapper) to answer questions grounded in the uploaded content.

Primary goals

- Organize PDFs by domain with quick preview/download.
- Read PDFs from local “resource folders” (Windows paths with relative fallbacks).
- Build in-memory search over uploaded PDFs (chunk → embed → FAISS).
- Ask questions and get answers that cite the retrieved text.
- Ship as a modular codebase that’s easy to extend.

Non-goals (current version)

- No multi-user tenancy or persistent database.
- No server-side file persistence (uploads live in session only).
- No OCR for scanned PDFs (image-only pages won’t extract text).
- No enterprise auth/SSO.

---

2) Users & Use Cases

- **Clinician/Pharmacist:** Search guidance leaflets, dosage notes, hospital brochures.
- **Patient/Patient Advocate:** Organize personal medical reports; ask clarifying questions.
- **Ops/Admin:** Maintain curated “resource folders” and export bundles (ZIP).

Key flows:

1. Upload PDFs → Add to Library → Preview/Download.
2. Build FAISS index from uploaded files → Query (prompt) → Answer with citations.
3. Browse local resource folders and one-click **Download ALL (ZIP)** per folder.

---

3) System Context & Architecture

```
+------------------+      +-------------------+       +---------------------------+
|  User Browser   | <----> |   Streamlit App   |  -----> |  Euri AI Chat Model API   |
| (single-session) |      |  (Python runtime) |       | (via euriai.langchain)    |
+------------------+      +----+---------+----+       +---------------------------+
                             |      \
                             |       \
                             |        \ Embeddings (CPU)
                             |         \
                             v          v
                  +----------------+  +------------------+
                  |   pypdf      |  | sentence-      |
                  |  text extract |  | transformers    |
                  +--------+-------+  +---------+--------+
                           |                   |
                           v                   v
                     In-memory text          +--------+
                     chunking (LangChain) ---> | FAISS  |
                                             | index  |
                                             +--------+
```

Key external libs

- streamlit (UI), pypdf (text extraction), sentence-transformers + faiss-cpu (vector store), langchain(-community, -text-splitters) (splitters & vectorstore wrapper), euriai.langchain (LLM wrapper), python-dotenv (env).

---

4) Module View

Top-level layout

Healthcare-PDF-Hub-Modular/

```
├– app.py
├– requirements.txt
├– README.md
└ src/
  └ healthcare_pdf_hub/
    ├– __init__.py
    ├– catalogs.py      # medicine catalog/brands, hospitals list
    ├– config.py        # resolves resource folders (env → abs → rel)
    ├– ui/
    │  └ components.py    # reusable Streamlit UI pieces
    └ utils/
      ├– pdf_utils.py     # pypdf helpers, preview HTML, folder listing, ZIP
      ├– chat_model.py    # Euri AI wrapper (create & invoke)
      └ faiss_utils.py   # create_faiss_index, retrive_relevant_docs
```

Key components

- **app.py**
  - Sets page, tabs, and orchestrates flows for each tab.
  - Initializes chat model via .env (EURI_API_KEY) with @st.cache_resource.
  - Uses session_state (uploads, per-tab last batch, vectorstores) to manage in-memory state.
- **utils/pdf_utils.py**
  - extract_text_from_pdf(bytes), get_page_count, pdf_preview_html, list_pdfs_from_folder(Path), make_zip_from_items.
- **utils/faiss_utils.py**
  - create_faiss_index(texts) → FAISS.from_texts with HuggingFaceEmbeddings (e.g., all-MiniLM-L6-v2).
  - retrive_relevant_docs(vectorstore, query, k=4).
- **utils/chat_model.py**

  - get_chat_model(api_key) → euriai.langchain.create_chat_model.
  - ask_chat_model(chat_model, question) → .invoke().
- **catalogs.py**
  - MEDICINE_CATALOG, MEDICINE_BRANDS, HOSPITALS_2025.
- **config.py**
  - choose_resource_dirs() uses env vars HPDFHUB_MEDICAL_DIR, HPDFHUB_MEDICINE_DIR, HPDFHUB_HOSPITAL_DIR; otherwise absolute Windows defaults; then relative fallbacks.
- **ui/components.py**
  - process_uploads(files, bucket_key) stores bytes in st.session_state.uploads.
  - render_bucket_table(bucket) shows a small table + preview expander.

---

5) Data Design

Session state keys

```
st.session_state = {
 "uploads": {
   "medical": [ {name, size, pages, uploaded_at, data}, ... ],
   "medicine": [ ... ],
   "hospital": [ ... ]
 },
 "medical_last_batch":   [ {name, data}, ... ],
 "medicine_last_batch":  [ {name, data}, ... ],
 "hospital_last_batch":  [ {name, data}, ... ],


 "medical_vectorstore":  <FAISS>,
 "medicine_vectorstore": <FAISS>,
 "hospital_vectorstore": <FAISS>,
 # optional brief notes if retained in some tabs
 "hosp_notes": { "<Hospital — City>": "<text>" },
 "med_notes":  { "<MedicineName>": "<text>" },
}
```

In-memory vector index

- Texts are **chunked** with RecursiveCharacterTextSplitter (default: 1,000 chars, 200 overlap).
- Embeddings via HuggingFaceEmbeddings, model sentence-transformers/all-MiniLM-L6-v2 (configurable).
- FAISS index held in memory; not persisted by default.

---

6) Core Flows (Sequence)

A) Upload & Library

1. User selects PDFs with st.file_uploader.
2. Click **Add to Library** → process_uploads() stores bytes in session_state.uploads[tab].
3. Also cache the **last batch** for that tab (e.g., medicine_last_batch).

B) Build Index & Ask

1. User enters **prompt** (disabled until Library has docs) or selects entity (medicine/hospital).
2. Extraction: extract_text_from_pdf for each file in last batch (or all library items).
3. Chunking: splitter.split_text.
4. Embedding + Index: create_faiss_index(chunks) → store in session_state[tab_vectorstore].
5. Retrieval: retrive_relevant_docs(vectorstore, prompt_or_selection).
6. Compose **system prompt** with context; call ask_chat_model.
7. Render answer; optionally show retrieved snippets.

C) Resource Folders

1. Read PDFs from configured directories via list_pdfs_from_folder.

2. Offer **per-file downloads** and a **Download ALL (ZIP)** button per folder (make_zip_from_items).

---

7) Functional Requirements

- **FR1**: Upload PDFs per domain and show them in a Library table.
- **FR2**: Preview PDFs inline and download individually.
- **FR3**: Read PDFs from local resource directories, list, and download (including ZIP-all).
- **FR4**: Build a searchable index from uploaded PDFs and run queries.
- **FR5**: Answer user questions with LLM using retrieved context; keep answers conservative and cite sources (as available in retrieved chunks).
- **FR6**: Disable prompts until docs are added to Library.

---

8) Non-Functional Requirements

- **Performance**:
  - Reasonable on CPU (MiniLM embeddings).
  - Chunking runs client-triggered; document size dependent.
- **Scalability**:
  - Single session; in-memory state; suitable for desktop/single user or small teams behind a shared session.
- **Reliability**:
  - Handle empty/invalid PDFs; show warnings.
- **Security/Privacy**:
  - API key from .env.
  - No server-side persistence of uploads by default; sensitive medical data stays in session memory unless user changes code to persist.
  - **Do not log extracted text** to console in production.
- **Compliance**:
  - App is informational; not a medical device. Answers stress safety disclaimers.

- **Observability**:
  - Streamlit logs and minimal UI alerts; optional future: structured logging.

---

9) Configuration

- **Environment variables**
  - EURI_API_KEY (required) — for chat model.
  - HPDFHUB_MEDICAL_DIR, HPDFHUB_MEDICINE_DIR, HPDFHUB_HOSPITAL_DIR (optional) — override resource folders.
- **Requirements**
  - streamlit, pypdf, python-dotenv,
  - langchain, langchain-community, langchain-text-splitters,
  - sentence-transformers, faiss-cpu, torch (CPU or CUDA),
  - euriai (or the package that provides euriai.langchain).

---

10) Error Handling & Edge Cases

- **Scanned PDFs**: pypdf may return empty text → show warning; optional roadmap: OCR (e.g., pdf2image + pytesseract).
- **Model init failure**: show visible error if EURI_API_KEY missing or SDK unavailable.
- **Large files**: Streamlit upload limit (default 200MB/file). Chunk size tunable to balance recall vs. speed.
- **Package conflicts**: Pin compatible versions (Torch ↔ Transformers ↔ Sentence-Transformers ↔ Accelerate).

---

11) Security Considerations

- Do not store or transmit PHI beyond the user’s control; avoid server persistence unless secured.
- Keep .env out of version control.
- Sanitize/limit what is sent to the LLM (only retrieved context, not entire documents).

- Consider content filtering and PII redaction for production.

---

12) Extensibility Roadmap

- **Persistence**: Save FAISS indexes & metadata to disk (e.g., vectorstore.save_local()).
- **OCR**: Add OCR fallback path with a toggle per tab.
- **Source citations**: Attach metadata={"source": file_name, "page": n} to chunks; surface in answers.
- **Better RAG**: Use MultiQueryRetriever / Hybrid (BM25 + dense) retrieval.
- **Multi-user**: User auth; isolate session stores; server-side storage (S3/GCS/Azure).
- **Analytics**: Query history, feedback thumbs (store locally or in a DB).
- **Prompting**: Centralize prompt templates with guardrails.

---

13) Deployment

- **Local/dev**:
  - pip install -r requirements.txt
  - streamlit run app.py
- **Server**:
  - Reverse proxy (Nginx) → Streamlit.
  - Secure environment variables.
  - Optional: mount volumes for resource folders and persistent indexes.

---

14) Testing Strategy

- **Unit**: pdf_utils.extract_text_from_pdf, make_zip_from_items, faiss_utils.create_faiss_index.
- **Integration**: End-to-end on small sample PDFs per tab; verify retrieval returns expected chunks.
- **UX**: Prompt disabled until Library has docs; “Download ALL” includes all files.

15) Open Risks / Assumptions

- Embedding/model downloads on first run (network/time). Consider packaging or pre-warming.
- Session memory growth with many/large PDFs. Option: cap, or paginate Library.
- Legal/clinical risk: reinforce disclaimers; never present guidance as medical advice.

16) Quick Data Flow (per tab)

1. **Upload → Add to Library** → store {name, data, meta} in st.session_state.uploads[tab] and *_last_batch.
2. **Submit/Run Search** → extract text (pypdf) → chunk (LangChain) → embed (MiniLM) → index (FAISS) → retrieve (K docs).
3. Compose **LLM prompt** with retrieved context + user question → ask_chat_model → UI renders **Answer** + (optional) retrieved snippets.