# On-Premise RAG App with data preprocessing pipeline

## 1. Functional Requirements

### 1.1 Frontend (Web UI)

- Development of a user-friendly web interface (e.g., with **Streamlit or React**).
- **Upload functionality** for documents (PDF) with acceptable speed.
- Display of a **chat interface** for user interaction.
- Provision of a **search function** to quickly find relevant information from uploaded documents.
- Implementation of a **preview function** to highlight relevant sections in uploaded documents.
- REUSE CURRENT ELEMENTS FROM PREVIOUS PROJECT

### 1.2 Document Processing (File Processing)

- **Extraction of relevant pages** from uploaded PDFs.
- **OCR analysis** to extract text from images or scanned documents.
- Storage of extracted texts including **positioning information** (bounding boxes) for later UI highlighting.

### 1.3 Data Preprocessing

- Conversion of extracted data into a model-compliant format for subsequent analysis.
- **Entity Recognition (NER)** to identify key terms (e.g., "Concrete Type," "Supplier").
- Storage of structured information in an appropriate database.

### 1.4 Data Storage

- Storage of extracted texts, metadata, and structured entities.
- **Option 1: Graph Database (e.g., Neo4j)** for managing project, document, and entity relationships.
- **Option 2: Vector Store (e.g., ChromaDB)** for semantic search based on embeddings.

### 1.5 Question-Answer System

- Capability for **semantic search** within a project.
- **Retrieval module** that extracts relevant text sections from the database.
- Optional: Integration of an **LLM** to generate answers based on extracted document information (e.g., Llama or Hugging Face models).
- Linking of **answers with original sources** and marking relevant passages.

### 1.6 User Interaction

- Provision of a **question-answer interface** for user queries. → existing one
- Session-based storage of conversations (UUID approach for sessions).

- Show references in PDF → see examples I provided in Git repositories

### 1.7 Deployment & Infrastructure

- Local execution on an **on-premise GPU server**.

---

# 2. System Architecture

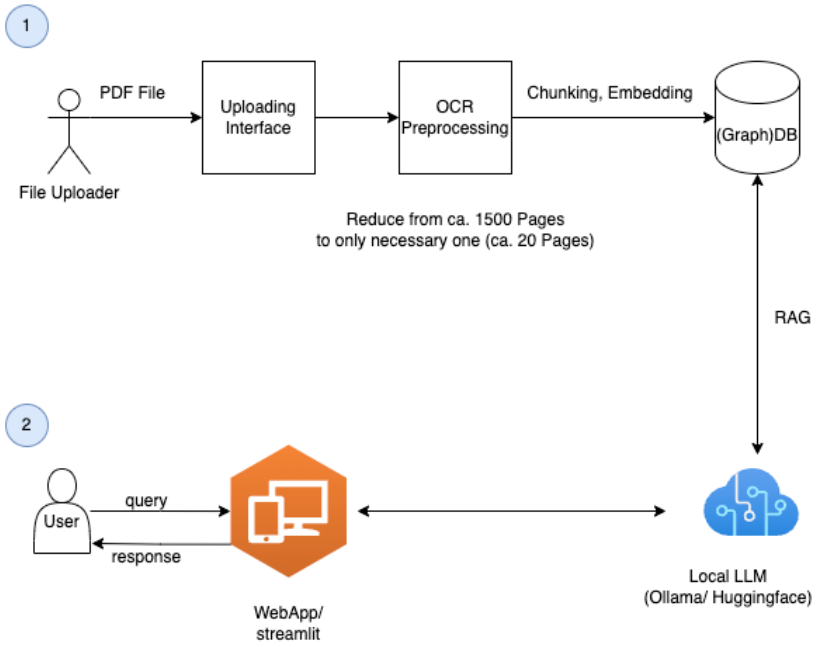
## 2.1 Process Workflow

1. **Upload** → PDF files are uploaded to the system.
2. **Preprocessing Step** → Identification of relevant pages, OCR processing, entity recognition, and storage of structured information.
3. **Database Storage** → Storage of extracted content in the **Vector Store or a GraphDB**.
4. **Frontend Integration** → The actual user interaction begins only after storage, as the UI accesses these structured data.
5. **Further development based on existing work** → Frontend work can build upon and extend the existing **Streamlit** demonstrator, leveraging already implemented functionalities.

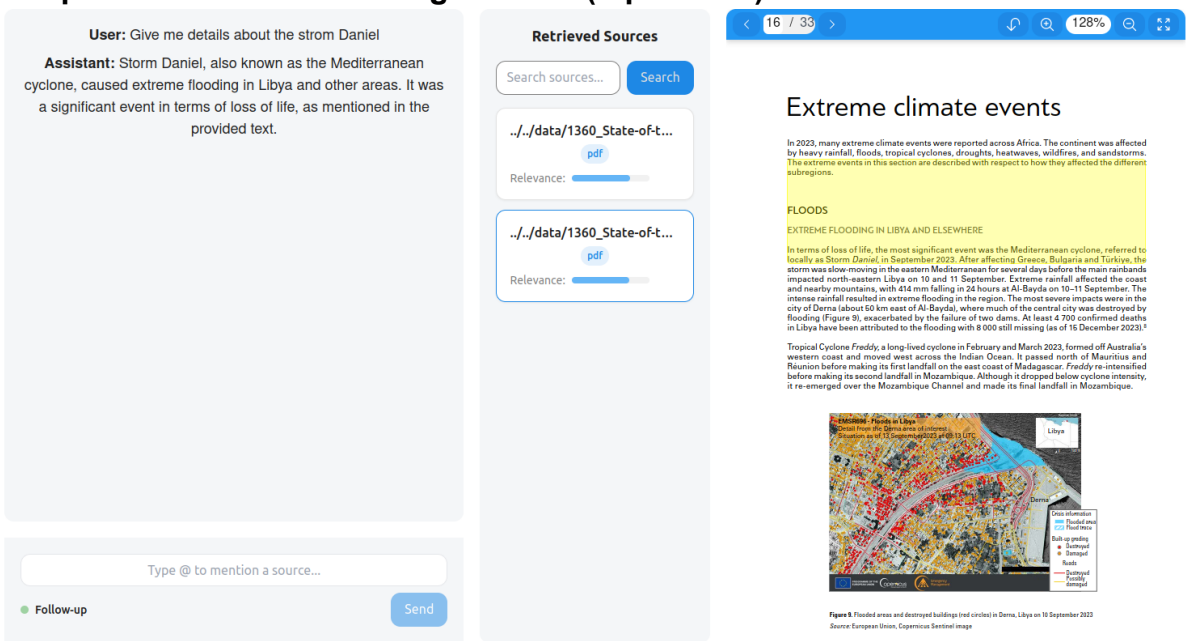
## 2.2 Database Options

- **Graph Database** for relationships between projects, documents, and entities.
- **Vector Store** for semantic similarity search.
- **What are advantages/ disadvantages?**

# Architecture




## Comparison Feature with Original PDF (if possible)



# References

I really like this Repo, (especially the view of the retrieved sources) so please check weather we can use components of it!

https://github.com/Renumics/lexio/tree/main