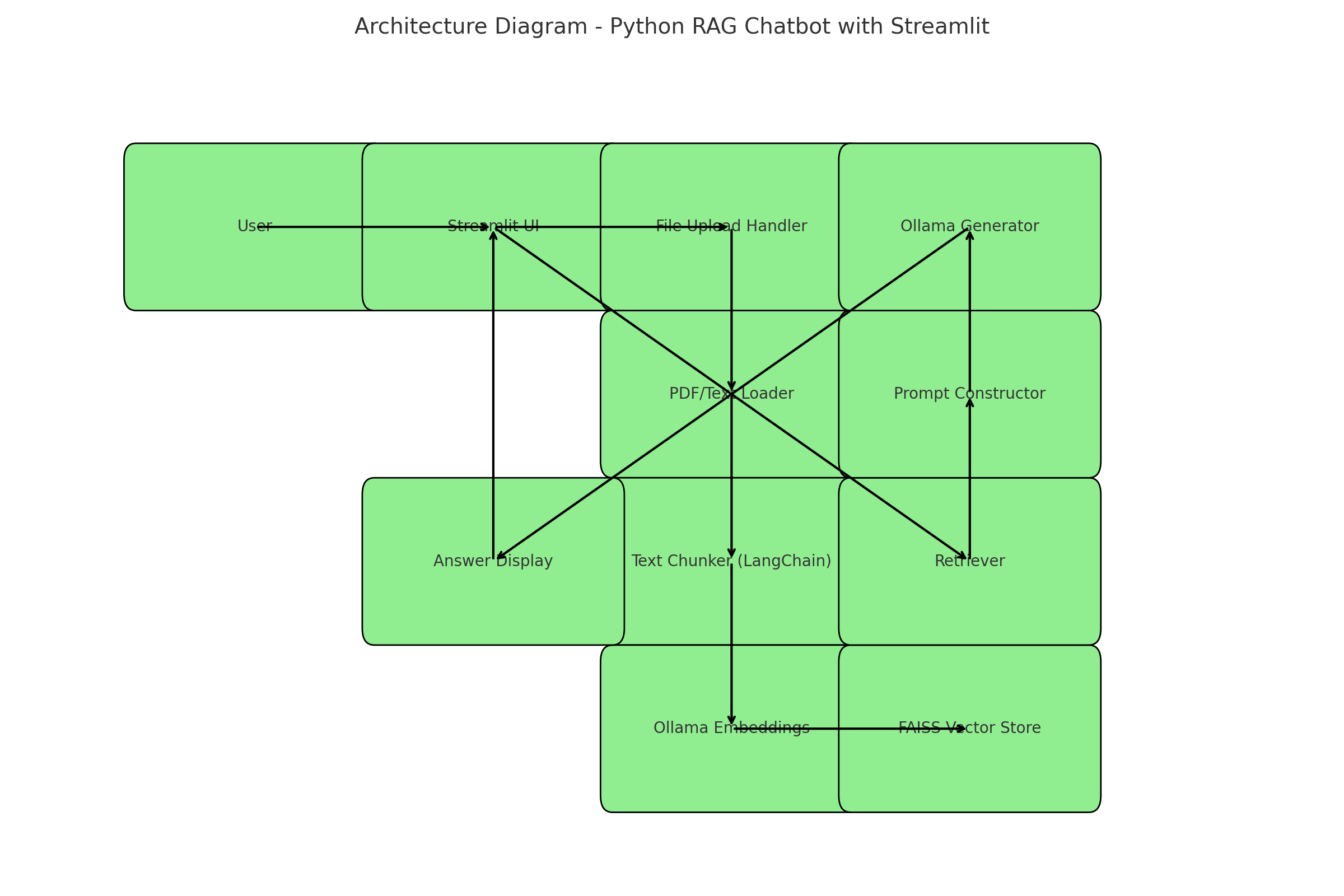
Python RAG Chatbot with Streamlit - Documentation

# 1. Overview

This document provides a detailed explanation of the Python-based RAG (Retrieval-Augmented Generation) chatbot. The application enables users to upload PDFs or text files, generates vector embeddings using a local LLM via Ollama, and performs semantic search with a user-friendly Streamlit UI.

# 2. Architecture Flow Diagram

Below is the architecture diagram for the Python RAG chatbot implementation.



# 3. Code Structure and Key Components

The application consists of the following main components:

1. Streamlit UI (`app.py`): Handles file upload, query input, and displaying results.  
2. RAG Pipeline (`rag\_pipeline.py`): Responsible for loading documents, chunking text, embedding generation, storing vectors, and retrieving relevant context for a given question.  
3. FAISS: In-memory vector database used to store and search document embeddings.  
4. Ollama: Local LLM server providing embedding and generation APIs.

# 4. Libraries and Tools Used

- \*\*Streamlit\*\*: For building a simple and reactive web UI.  
- \*\*LangChain\*\*: Provides document loading, text splitting, and integration with Ollama embeddings.  
- \*\*FAISS\*\*: Vector similarity search library from Facebook AI Research, used for storing and retrieving text chunks.  
- \*\*PyPDF\*\*: For parsing PDF content into text.  
- \*\*Ollama\*\*: Local deployment tool for LLMs such as LLaMA 3, providing `/embeddings` and `/generate` HTTP APIs.  
- \*\*ChromaDB\*\* (optional): Can be used as an alternative vector store backend.

# 5. Application Flow

1. User uploads a PDF or text document via Streamlit interface.  
2. The text is extracted (using PyPDF for PDFs).  
3. Text is split into smaller overlapping chunks using LangChain.  
4. Each chunk is converted into a vector using the Ollama `/embeddings` endpoint.  
5. Chunks and vectors are stored in FAISS for fast retrieval.  
6. When a user asks a question, it is embedded into a vector using the same embedding model.  
7. FAISS retrieves the most relevant chunks based on cosine similarity.  
8. Retrieved chunks are added to the prompt for the `/generate` API call to Ollama.  
9. Final answer is shown in the Streamlit interface.

# 6. Sample UI Code Snippet

Below is a sample from the Streamlit application code (app.py):

st.title("📚 RAG Chatbot with Ollama")  
query = st.text\_input("Ask me anything from the docs...")  
if query:  
 response = get\_answer(query)  
 st.success(response)

# 7. Future Enhancements

- Enable file upload for real-time document updates.  
- Persist vector store across sessions using ChromaDB or SQLite.  
- Support for markdown rendering and follow-up questions.  
- UI improvements and chat history support.