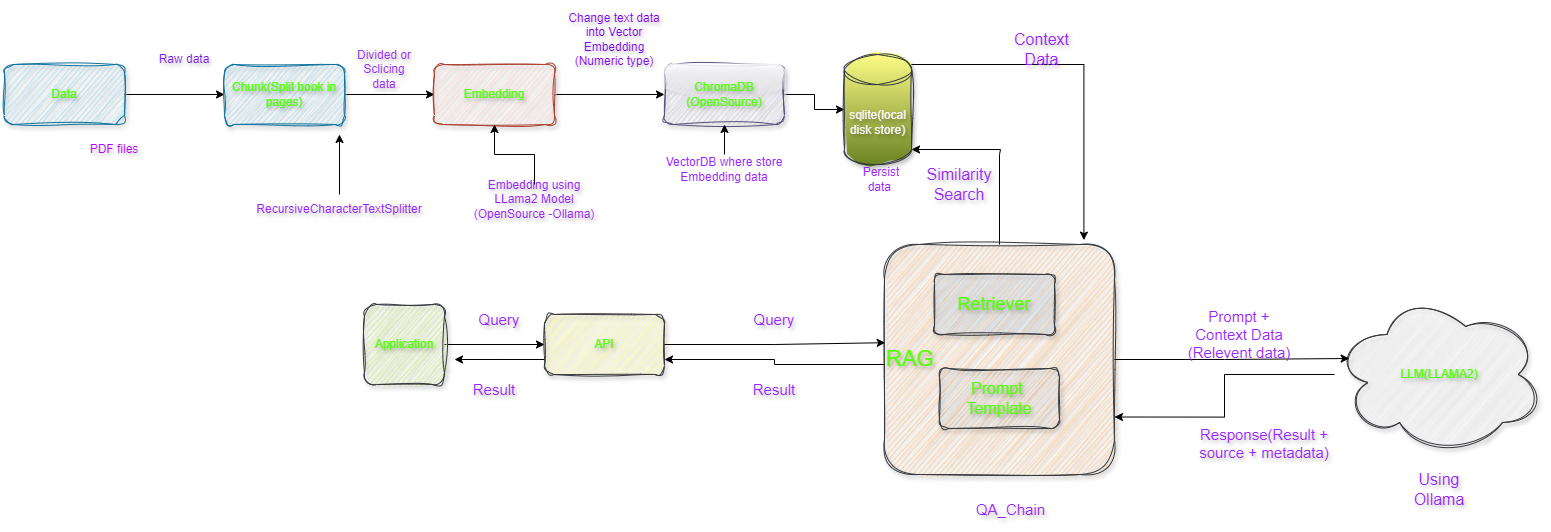
**Overview of the chatbot architecture.**

****

**Architecture of Chatbot for PDF**

**Steps For Building Chatbot**

I’d be glad to clarify the steps involved in creating a PDF chatbot using the provided using the provided code snippet:

1. **Getting And Loading the PDF File:**

* You’ll need a PDF document stored in your project’s docs directory.
* The code utilizes **langchain.** Directory to locate all PDF files within that directory using the **glob= ‘./pdf’** pattern.
* The **loader\_cls=pypdfloader** argument specifies that the pypdfloader Class from the Langchain Library should be used for loading the PDF content.

1. **Splitting the PDF into Chunks:**

* The **RecursiveCharacterTextSplitter** function takes the loaded PDF data and splits it into smaller, manageable segments.
* This could involve dividing the document by pages, sections, or other logical groupings based on your specific needs**.**

1. **Embedding Text Chunks into Numerical Representation:**

* The **OllamaEmbeddings(model='llama2')** function creates embeddings for each text chunk.
* Embeddings are essentially numerical representations that capture the meaning and context of the text. This allows the retrieval and question answering models to efficiently work with the document's content.
* The **llama2** model in this case suggests the use of a pre-trained language model (LLM) for generating these embeddings.

1. **Saving Embeddings in a Vector Database:**

* The **Chromadb(OpenSource VectorDatabase** function is used to create a vector database **(ChromaDB)** to store the generated embeddings. This enables efficient retrieval and searching of similar content later on.
* **Chroma.from\_documents** assembles the embeddings and corresponding text chunks into a document format for storage.
* **persist\_directory** specifies the directory where the database will be stored on your local disk.
* **vectordb.persist()** permanently saves the database contents.

1. **Creating Retrievers and LLM Model:**

* A retriever object is created for each text chunk. Retrievers are responsible for finding relevant information from the corpus (in this case, the embeddings database) based on a given query.
* The **retriever.as\_retriever()** function likely prepares the retriever object for use in the question answering system.
* LLM **model Ollama(base\_url='http:/localhost:11434',model="llama2")** indicates the use of the **Ollama** LLM, probably running locally at http://localhost:11434. The **model="llama2"** part specifies the specific LLM variant employed.

1. **Building the Retrieval-Augmented Generation(RAG) Chain:**

* The from **langchain.chains** import **RetrievalQA** function imports the **RetrievalQA** class, which is designed for building RAG chains.
* RAG **(Retrieval-Augmented Generation)** combines retrieval from a knowledge base **(here, the ChromaDB)** with LLM generation to provide more informative and contextually relevant answers.
* **qa\_gen=RetrievalQA.from\_chain\_type(llm=llm\_llama, chain\_type="stuff", retriever=retriever, return\_source\_documents=True)** creates a **qa\_gen** object that embodies the RAG chain.
* **llm=llm\_llama** assigns the previously defined LLM model to the chain.**chain\_type="stuff**" (the exact meaning depends on the LangChain library implementation) likely indicates the type of chain being created (possibly a generic retrieval and generation chain).
* **retriever=retriever** specifies the retriever object to be used within the chain.
* **return\_source\_documents=True** instructs the chain to return the source documents (text chunks) that were most relevant to the query in the response.

1. **Interacting with the Chatbot (Querying):**

* Once the qa\_gen object is constructed, you can interact with the chatbot by providing a question.
* **qa\_gen(question)['result']** extracts the answer generated by the RAG chain for the given question. This answer would likely combine information retrieved from the **ChromaDB** embeddings with text generation from the LLM model.
* The **return\_source\_documents=True** setting in the **qa\_gen** creation might also provide details about the relevant text chunks that contributed to the answer.

**Explanation of how RAG, Vectordb, Embedding, and LLM frameworks are utilized.**

**RAG(Retriever Augmented Generation)**

**VectorDB**

**Embedding**

**LLM Framework**

**Instructions for setting up the environment and running the chatbot.**

**Evaluation Criteria:**

**Code Link:** - https://github.com/GENRATECODE/pdf\_chatbot/blob/main/README.md