**Abstract**

Extracting precise insights from the voluminous information included in PDF documents can be difficult in the digital era. The interaction with PDF information is revolutionised by the Langchain Chatpdf, which uses the OpenAI API and large language models (LLMs). This chatpdf combines machine learning and natural language processing to provide a strong conversational interface and facilitate effective information extraction from several PDFs.

Businesses can get a competitive edge by utilising Langchain Chatpdf to improve search capabilities and streamline information retrieval procedures. User interfaces, natural language understanding (NLU), vector storage, embeddings, and LLMs are all included in its modular design. In response to user inquiries, the chatpdf uses conversational retrieval techniques to offer context-aware responses.

Performance, speed, and accessibility are impacted by the choice of embeddings and LLMs, including those offered by OpenAI and HuggingFace. OpenAI Embeddings provide quick

**ACKNOWLEDGMENT**

It gives me immense pleasure to acknowledge and express my deepest gratitude to all those who helped showed me the right way in completing this project work. A project of this nature and magnitude needs co-operation and support from many

I take this opportunity to thank my Principal, **Prof. Chaitra Chougule** for providing a healthy environment in my college, which helped in concentrating on my task.

First and foremost, I am greatly indebted to my guide and monitor **Prof. Venkoba** with whose enthusiasm and confidence urged me take on this project. The constant encouragement, guidance and help during all the & difficulties that I encountered.

I am extremely thankful to my parents for their support and constant faith they have in me. It is because of their blessings and help that I have come so far.

My special thanks wonder crew of teaching and non-teaching staff who have helped me in different way of throughout the year finally I would thank my friends for their continuous support and encouragement without which this project would not have become a reality.

Puttaraj K P02GI21S0014

Date:

Place:

**CONTENTS**

**CHAPTER 1 INTRODUCTION**

**CHAPTER 2 RESOURCE REQUIREMENTS**

2.1 Software requirement

2.2 Hardware requirement

**CHAPTER 3 ARCHITECTURE OF LANGCHAIN**

3.1 Modular Architecture of LangChain PDF

3.2 Overview

**CHAPTER 4 SYSTEM DESIGN**

5.1 Data Flow Diagram

5.2 Sequence diagram

**CHAPTER 5 LANGCHAIN KEY FEATURES**

**CHAPTER 6 IMPLEMENTATION**

6.1 Importing Libraries

6.2 PDF Text Extraction

6.3 Text Chunking

6.4 Vector Store Creation

6.5 Conversion Chain Creation

6.6 User Input Handling

6.7 Main Function

**CHAPTER 7 SCREENSHOT’S**

**CHAPTER 8 RESTRICTIONS AND FUTURE IMPROVEMENTS**

**CHAPTER 9 TESTING**

9.1 Functional Testing

9.2 Usability Testing

9.3 Integration Testing

9.4 Test Cases

**CHAPTER 10 CONCLUSION**

**BIBLIOGRAPHY**

**Introduction**

It can be difficult to get precise insights in PDF documents in the modern digital world because of the wealth of information they contain. The Langchain Chatpdf revolutionises how we engage with PDF documents by using the OpenAI API and free large language models (LLMs). Langchain Chatpdf provides a potent conversational interface for querying data from several PDFs by smoothly integrating natural language processing and machine learning techniques.

This novel approach improves search capabilities while also streamlining the information retrieval procedure. Chatpdf use cases like Langchain give firms a competitive edge by giving quick and accurate access to critical information, ultimately saving time and money. This is due to the growing necessity of effective data extraction in the business market.When attempting to recover specific insights in the modern digital world, the volume of information present in PDF documents frequently presents a barrier. The way we engage with PDF information is revolutionised by the Langchain Chatpdf, which is driven by the OpenAI API and free large language models (LLMs). Langchain Chatpdf provides a potent conversational interface for requesting information from several PDFs by smoothly integrating natural language processing and machine learning techniques.

This ground-breaking approach not only improves search capabilities but also speeds up information retrieval procedures. Chatpdf use cases like Langchain, which enable quick and accurate access to critical information and ultimately save time and resources, give firms a competitive edge in the face of the growing importance of effective data extraction in the business sector.The volume of data present in PDF documents today makes it difficult to extract certain insights because of the abundance of information they carry. The way we engage with PDF information is being revolutionised by the Langchain Chatpdf, which is driven by the Free Large Language Models (LLMs) and OpenAI API. Langchain Chatpdf provides a strong conversational interface for requesting information from several PDFs by smoothly integrating natural language processing and machine learning techniques.

With this novel approach, information retrieval procedures are streamlined in addition to search skills being improved. With the necessity of effective data extraction in the business sector growing, chatpdf use cases like Langchain give companies a competitive edge by enabling quick and accurate access to essential information, eventually saving time and money..

**Resource Requirement**

Certain system prerequisites must be completed in order for the Langchain Chatpdf for Multiple PDFs project to work smoothly and function to its fullest potential. The project requires particular hardware and software configurations because it uses a variety of technologies and components. The system requirements listed below are necessary for this project to function properly:

Hardware Requirements:

1. Computer: A modern computer with sufficient processing power and memory is required to run the project smoothly. A multi-core processor (e.g., Intel Core i5 or higher) is recommended.

2. Memory (RAM): A minimum of 8GB RAM is recommended for running the project, especially when working with large language models and processing PDF documents.

3. Storage: Adequate storage space is essential to store the project files, libraries, and the PDF documents that will be processed. At least 20GB of free disk space is recommended.

Certainly, here's an elaboration of the software requirements for the Langchain Chatpdf for Multiple PDFs project, intended to be included in your project report:

Software Requirements:

1. Operating System Compatibility:

The Langchain Chatpdf for Multiple PDFs is designed to run on various operating systems, including Windows, macOS, and Linux. It's important to ensure that the chosen operating system is up-to-date and compatible with the required software components.

2. Python:

Python is the primary programming language used to develop the chatpdf. A version of Python 3.6 or higher is recommended to take advantage of the latest language features and library compatibility.

3. Python Libraries:

* Streamlit:

Streamlit is a Python library used to create interactive web applications with minimal effort. It provides an intuitive way to design the user interface for the chatpdf. Installation is done via the command: `pip install streamlit`.

* - PyPDF2:

PyPDF2 is a library that enables the parsing and manipulation of PDF documents. It's used to extract text and information from PDFs in the project. Installation is done via the command: `pip install PyPDF2`.

* Transformers (HuggingFace):

Transformers is a HuggingFace library that provides easy access to a wide range of pre-trained language models, including embeddings and large language models. These models are crucial for the chatpdf's natural language processing capabilities. Installation is done via the command: `pip install transformers`.

* dotenv:

Dotenv is a library used to manage environment variables in a project. It's employed to securely store sensitive information like API keys and configuration settings. Installation is done via the command: `pip install python-dotenv`.

4. API Credentials:

If the project involves accessing external APIs, such as the OpenAI API, valid API credentials are required. These credentials are obtained from the respective API providers and are stored securely in a `.env` file. This file should be located in the project directory and excluded from version control to prevent exposing sensitive information.

5. Text Editor or Integrated Development Environment (IDE):

You'll need a text editor or an IDE to write and edit Python code files. Popular choices include Visual Studio Code, PyCharm, Sublime Text, and Jupyter Notebook.

6. Version Control (Optional):

While not strictly necessary for running the chatpdf, using version control tools like Git can greatly aid in managing and tracking changes to your project's codebase. Platforms like GitHub, GitLab, or Bitbucket can host your repository and provide collaboration features.

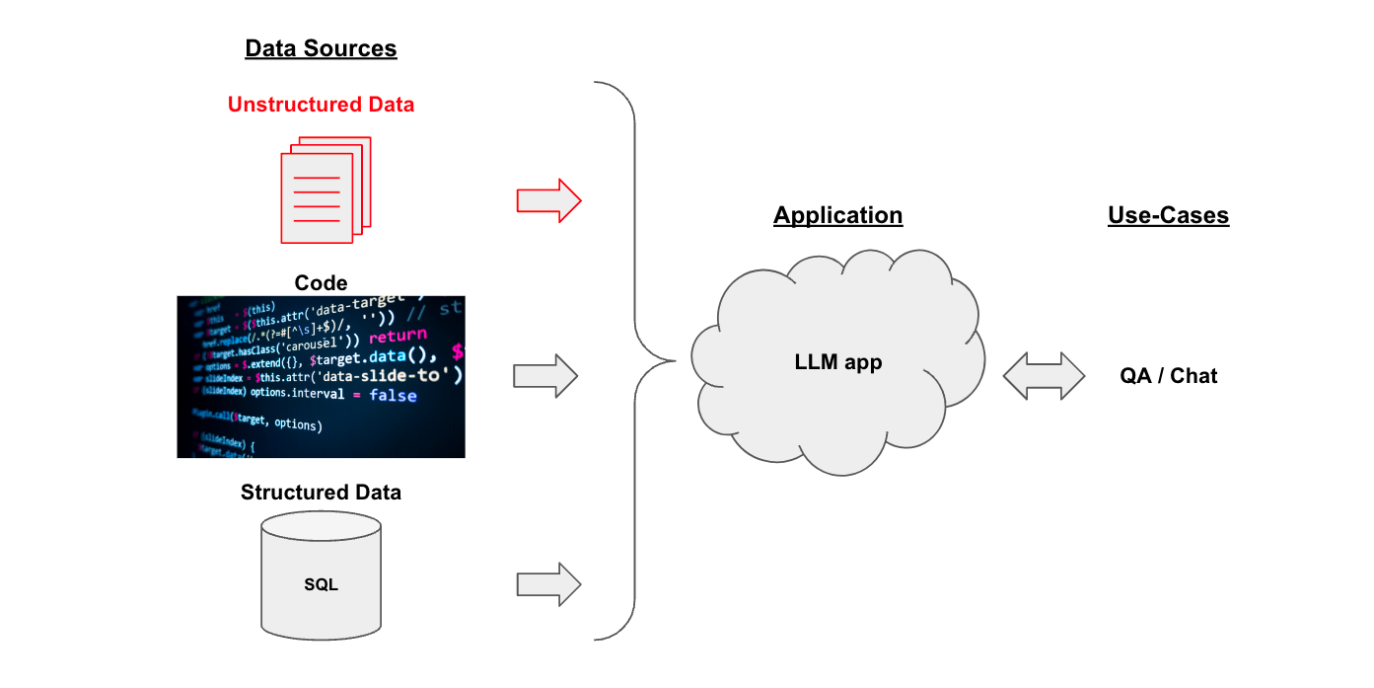
You'll establish a setting in which the Langchain Chatpdf for Multiple PDFs project may be successfully carried out by meeting these software requirements. In order for other users or developers to properly replicate the project on other systems, make sure to give thorough installation instructions and help for configuring the necessary software components.

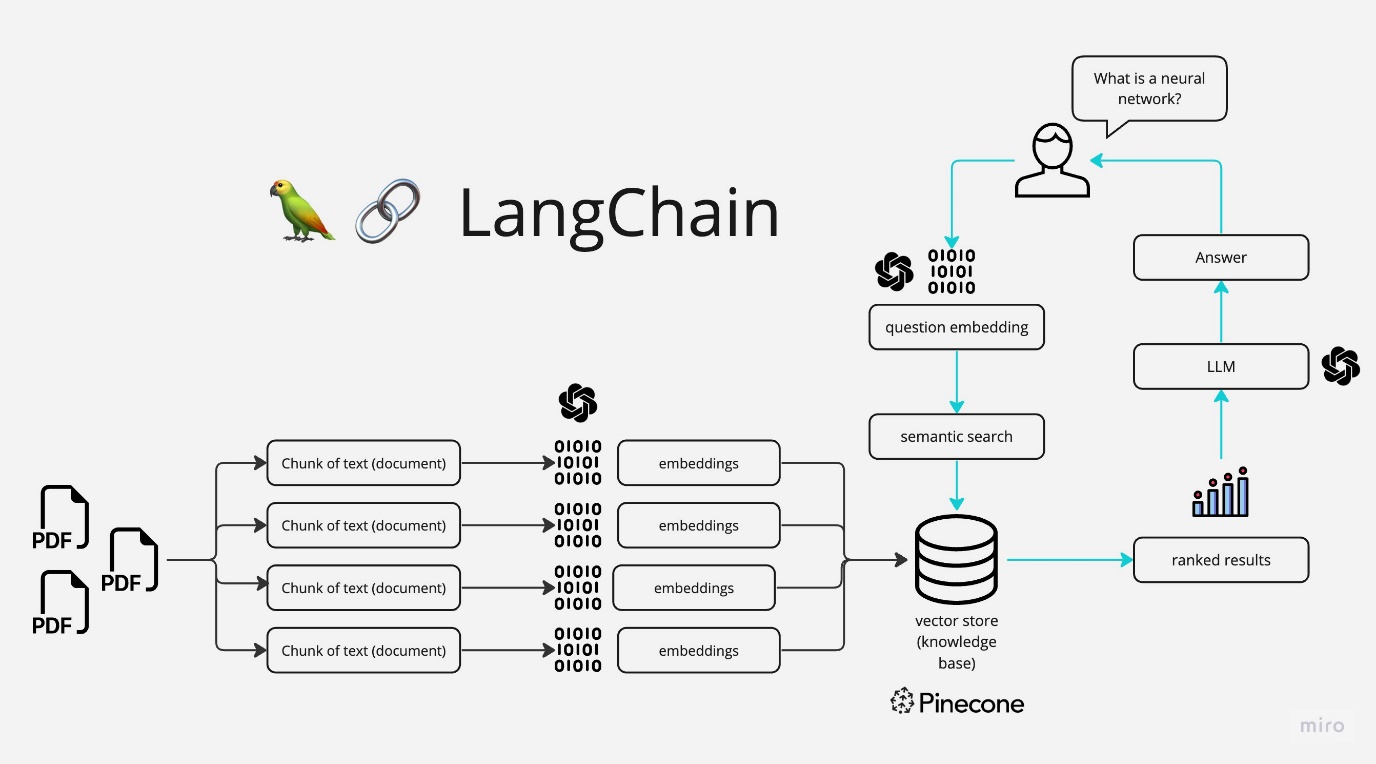
**LangChain Architecture**

LangChain, a powerful framework specifically designed for developing language model-driven applications, serves as the foundation for our project. It provides us with the necessary tools and capabilities to create an intelligent system that can accurately answer questions based on specific documents.

To enhance the performance and efficiency of our question-answering system, we integrate Pinecone, an efficient vector database known for building high-performance vector search applications. By leveraging its capabilities, we can significantly improve the speed and accuracy of our system’s search and retrieval processes.

The modular architecture of the Langchain Chatpdf for Multiple PDFs makes use of a number of components to effectively get information from PDF files. Let's examine the main architectural components:

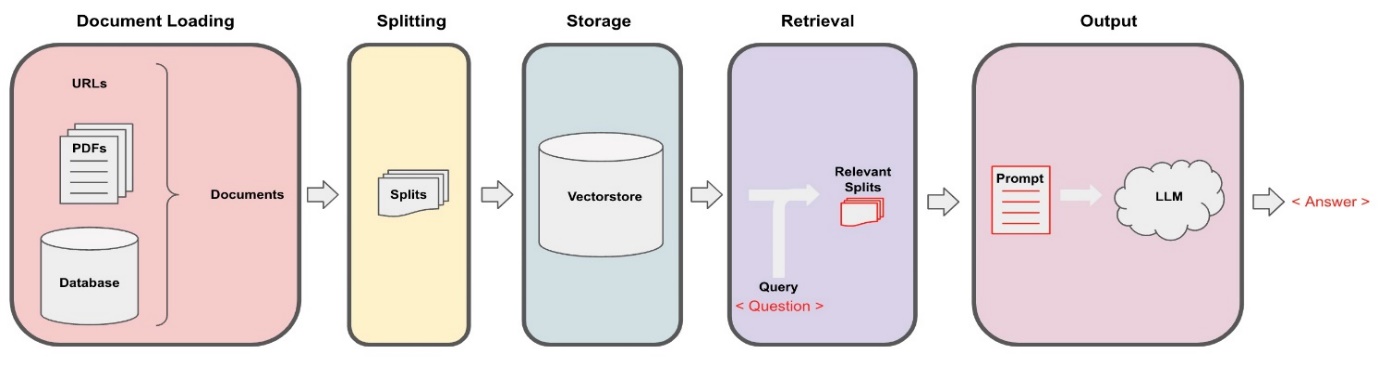




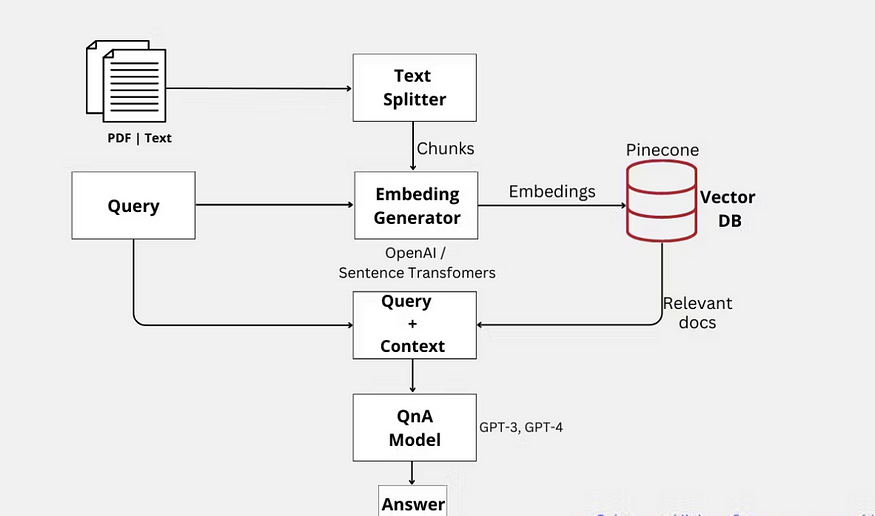
**Overview :**

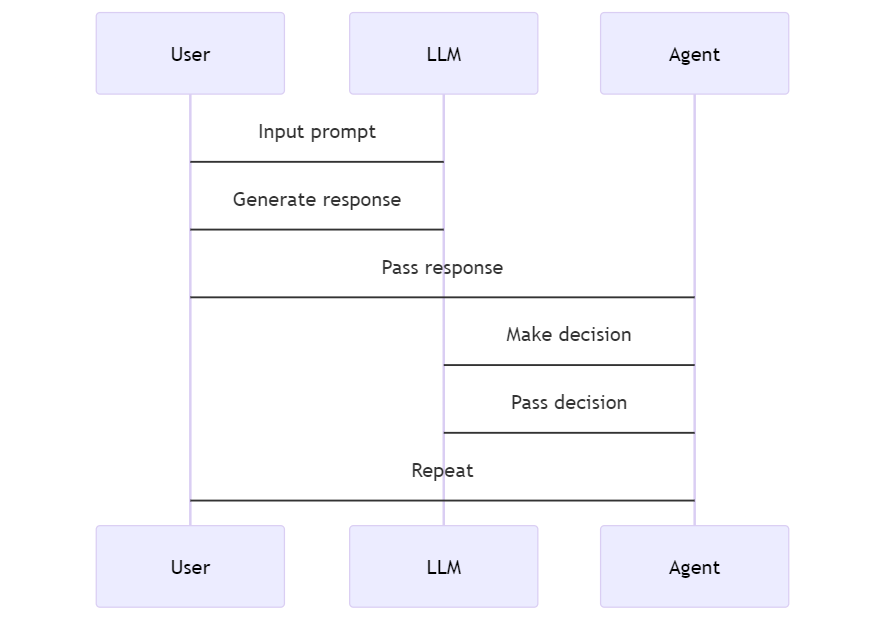
The pipeline for converting raw unstructured data into a QA chain looks like this:

1. Loading: First we need to load our data. Unstructured data can be loaded from many sources. Use the LangChain integration hub to browse the full set of loaders. Each loader returns data as a LangChain Document.
2. Splitting: Text splitters break Documents into splits of specified size
3. Storage: Storage (e.g., often a vectorstore) will house and often embed the splits
4. Retrieval: The app retrieves splits from storage (e.g., often with similar embeddings to the input question)
5. Generation: An LLM produces an answer using a prompt that includes the question and the retrieved data
6. Conversation (Extension): Hold a multi-turn conversation by adding Memory to your QA chain.

****

**System Design of Langchain**





The procedures that the application takes to respond to your inquiries are as follows:

* PDF loading: The software reads multiple PDF files and extracts their text content.
* Text chunking: The retrieved text is broken into manageable, smaller parts.
* The application uses a language model to create vector representations (embeddings) of the text chunks.
* Similarity Matching: When you ask a query, the software compares it to the text chunks and identifies the ones that are the most semantically similar.
* Response Generation: The language model receives the chosen chunks and produces a response based on the pertinent PDF material.

**KeyPoints**

**User Interface: The chatpdf's user interface offers consumers a smooth platform for interaction so they may ask queries and get pertinent responses. Users can enter their inquiries in a text input form, which is frequently included.**

**Natural Language Understanding (NLU): The chatpdf's NLU component processes and comprehends user inquiries. In order to analyse and extract meaning from user input, it makes use of natural language processing (NLP) techniques. Tokenization, part-of-speech tagging, named entity recognition, and syntactic parsing are some of the tasks that are involved in this stage.**

**Vector Store:**A key element that makes it possible to efficiently retrieve data from PDF documents is the Vector Store. It acts as a storage location for the vector representations of the text fragments that were taken out of the PDFs. In order to create these vector representations, language models and embeddings are used.

**Embeddings:**As part of the chatpdf's architecture, embeddings are quite important. They allow for useful comparison and retrieval by capturing the semantic and contextual information of the text fragments. The pre-trained language models or embeddings models, like OpenAI models or Hugging Face models, are fed the text chunks to create the embeddings. The textual information is represented as dense vectors in these models.

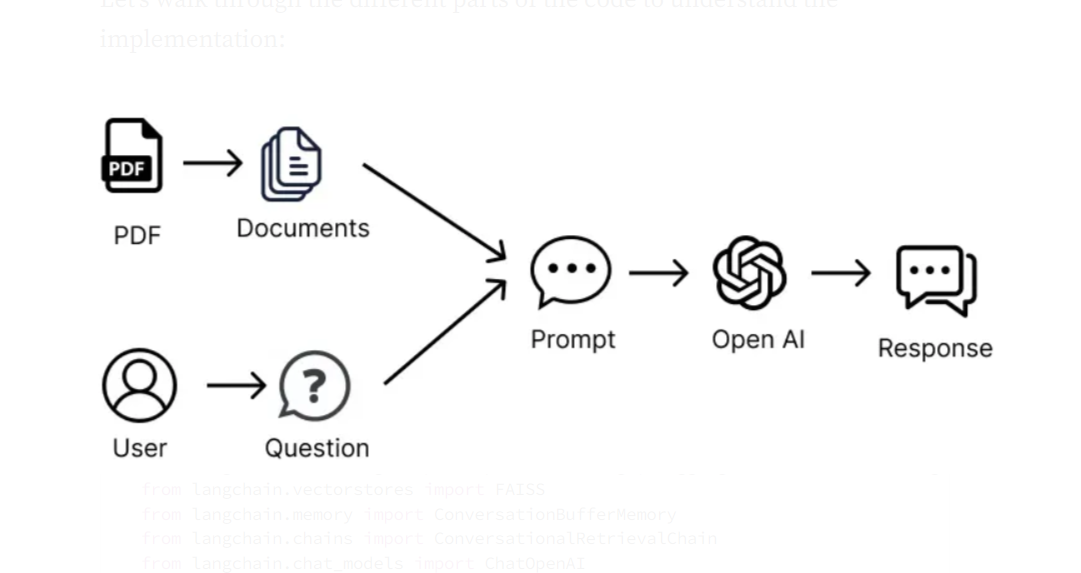
**Large Language Models (LLMs):**The architecture of chatpdfs must include LLMs. They give the chatpdf more sophisticated language creation and comprehension abilities. LLMs are pre-trained models that can comprehend and produce text that resembles human speech. The chatpdf can have more engaging and contextually aware discussions by honing these models for certain tasks or domains, including conversational retrieval.

**Conversational Retrieval:**Retrieving pertinent data based on user inquiries and contextual knowledge is known as conversational retrieval. The user query's vector representation is compared to the vector representations kept in the Vector Store. This retrieval procedure is intended to find the text chunks that are most pertinent to the user's query and that contain the necessary data to respond to their inquiry.

**Chat History and Context:** The chatpdf often keeps track of the chat history to preserve the conversation's context. The user's inquiries and the chatpdf's responses are included in this history. It enables the chatpdf to respond in a logical and pertinent manner based on the current dialogue.

**IMPLEMENTATION**

Python is used to implement the Langchain Chatpdf for Multiple PDFs, which makes use of a number of libraries and other components to give its functionality.



**The ChatPDF Workflow**

Let's go through the various sections of the code to comprehend how it is implemented:

**Importing Libraries:**The required libraries are imported, including PyPDF2 for processing PDF files, dotenv for managing environment variables, Streamlit for the user interface, and numerous Langchain library parts.

import streamlit as st  
from dotenv import load\_dotenv  
from PyPDF2 import PdfReader  
from langchain.text\_splitter import CharacterTextSplitter  
from langchain.embeddings import OpenAIEmbeddings, HuggingFaceInstructEmbeddings, HuggingFaceEmbeddings  
from langchain.vectorstores import FAISS  
from langchain.memory import ConversationBufferMemory  
from langchain.chains import ConversationalRetrievalChain  
from langchain.chat\_models import ChatOpenAI  
from htmlTemplates import bot\_template, user\_template, css

**PDF Text Extraction:** Using the PyPDF2 package, the get pdf text() function retrieves the text from the submitted PDF files. Each page of the PDFs is looped through, and the extracted text is concatenated.

def get\_pdf\_text(pdf\_files):  
   
 text = ""  
  
 for pdf\_file in pdf\_files:  
 reader = PdfReader(pdf\_file)  
 for page in reader.pages:  
 text += page.extract\_text()  
  
 return text

**Text Chunking:** The CharacterTextSplitter class from the Langchain package is used by the get chunk text() function to divide the captured text into more manageable portions. This increases retrieval effectiveness and yields more accurate results.

def get\_chunk\_text(text):  
   
 text\_splitter = CharacterTextSplitter(  
 separator = "\n",  
 chunk\_size = 1000,  
 chunk\_overlap = 200,  
 length\_function = len  
 )  
  
 chunks = text\_splitter.split\_text(text)  
  
 return chunks

**Vector Store Creation:** Using the Langchain library, the get vector store() function constructs a vector store. It uses OpenAIEmbeddings or HuggingFaceInstructEmbeddings to create embeddings for each text chunk after receiving the input text chunks. The FAISS library is then used to generate a vector store from the embeddings.

def get\_vector\_store(text\_chunks):  
   
 # For OpenAI Embeddings  
   
 embeddings = OpenAIEmbeddings()  
   
 # For Huggingface Embeddings  
  
 embeddings = HuggingFaceInstructEmbeddings(model\_name = "hkunlp/instructor-xl")  
  
 vectorstore = FAISS.from\_texts(texts = text\_chunks, embedding = embeddings)  
   
 return vectorstore

Be aware that with this function, we have the option of importing free Embeddings from HuggingFace's Massive Text Embedding Benchmark (MTEB) Leaderboard or using OpenAI Embeddings, a premium service.

**Conversation Chain Creation:** The conversational retrieval chain is set up using the Langchain library via the get conversation chain() method. It starts a discussion buffer memory and produces a chat model, either ChatOpenAI or HuggingFaceHub. The conversation chain is created by fusing the chat model, vector store retriever, and memory.

def get\_conversation\_chain(vectorstore):

llm = ChatOpenAI()

# llm = HuggingFaceHub(repo\_id="google/flan-t5-xxl", model\_kwargs={"temperature":0.5, "max\_length":512})

memory = ConversationBufferMemory(

memory\_key='chat\_history', return\_messages=True)

conversation\_chain = ConversationalRetrievalChain.from\_llm(

llm=llm,

retriever=vectorstore.as\_retriever(),

memory=memory

)

return conversation\_chain

**User Input Handling:** The handle user input() function responds to user inquiries and engages in dialogue with previous participants. It receives a user inquiry as input and responds using the thread of dialogue. For the purpose of preserving the conversation context, the chat history is saved in the session state.

def handle\_userinput(user\_question):

response = st.session\_state.conversation({'question': user\_question})

st.session\_state.chat\_history = response['chat\_history']

for i, message in enumerate(st.session\_state.chat\_history):

if i % 2 == 0:

st.write(user\_template.replace(

"{{MSG}}", message.content), unsafe\_allow\_html=True)

else:

st.write(bot\_template.replace(

"{{MSG}}", message.content), unsafe\_allow\_html=True)

**Main Function:** The program's entry point is the main() function. It configures the Streamlit user interface, manages user input, and controls the uploading of PDF files. Additionally, it calls the appropriate functions to establish the dialogue chain, generate the vector storage, and extract text.

def main():

load\_dotenv()

st.set\_page\_config(page\_title="Chat with multiple PDFs",

page\_icon=":books:")

st.write(css, unsafe\_allow\_html=True)

if "conversation" not in st.session\_state:

st.session\_state.conversation = None

if "chat\_history" not in st.session\_state:

st.session\_state.chat\_history = None

st.header("Chat with multiple PDFs :books:")

user\_question = st.text\_input("Ask a question about your documents:")

if user\_question:

handle\_userinput(user\_question)

with st.sidebar:

st.subheader("Your documents")

pdf\_docs = st.file\_uploader(

"Upload your PDFs here and click on 'Process'", accept\_multiple\_files=True)

if st.button("Process"):

with st.spinner("Processing"):

# get pdf text

raw\_text = get\_pdf\_text(pdf\_docs)

# get the text chunks

text\_chunks = get\_text\_chunks(raw\_text)

# create vector store

vectorstore = get\_vectorstore(text\_chunks)

# create conversation chain

st.session\_state.conversation = get\_conversation\_chain(

vectorstore)

if \_\_name\_\_ == '\_\_main\_\_':

main()

# Usage

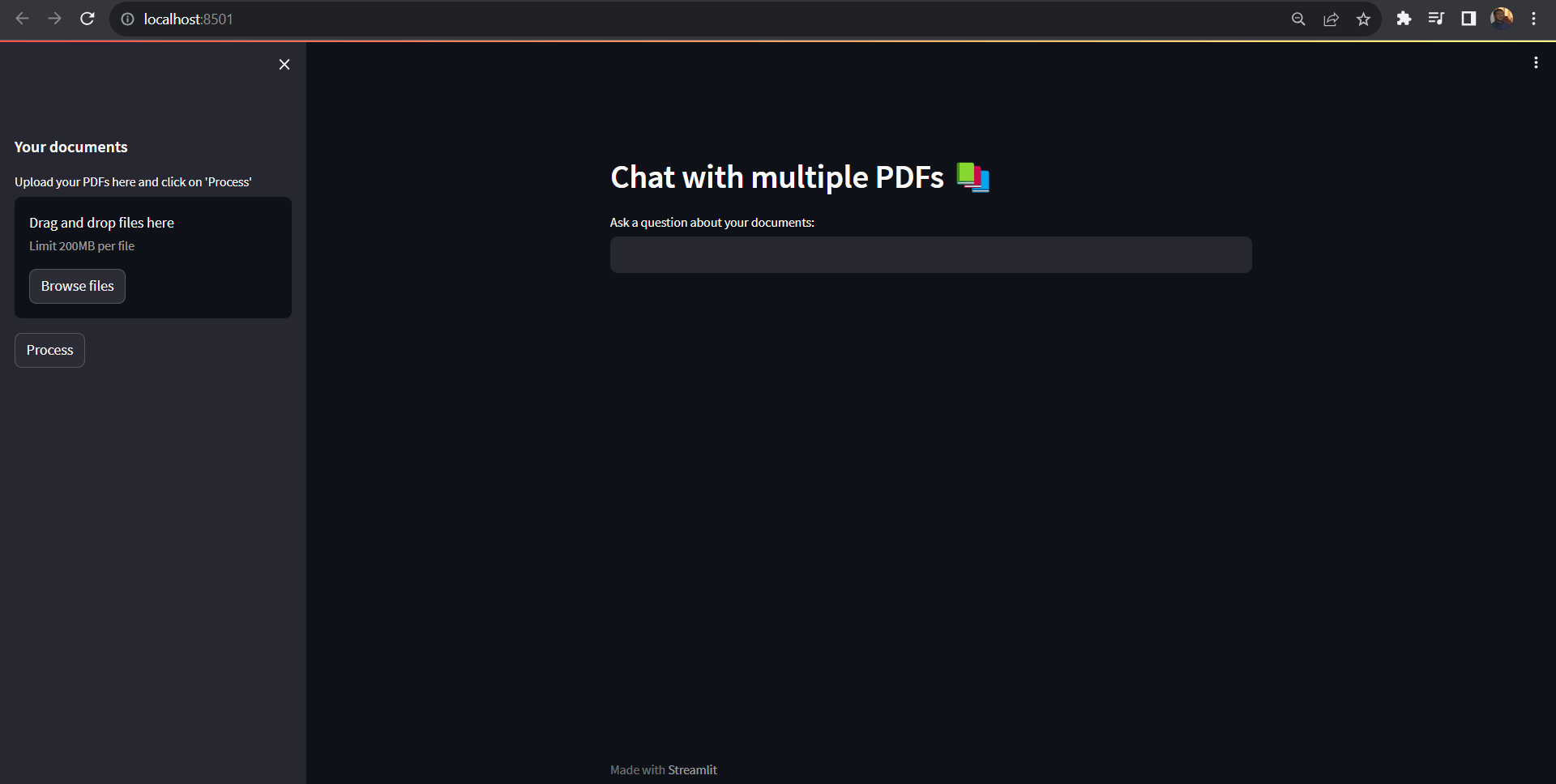
The Chatpdf streamlines the process of querying data from PDF documents using a user-friendly and straightforward interface. Here is a thorough explanation of how to utilise the chatpdf:

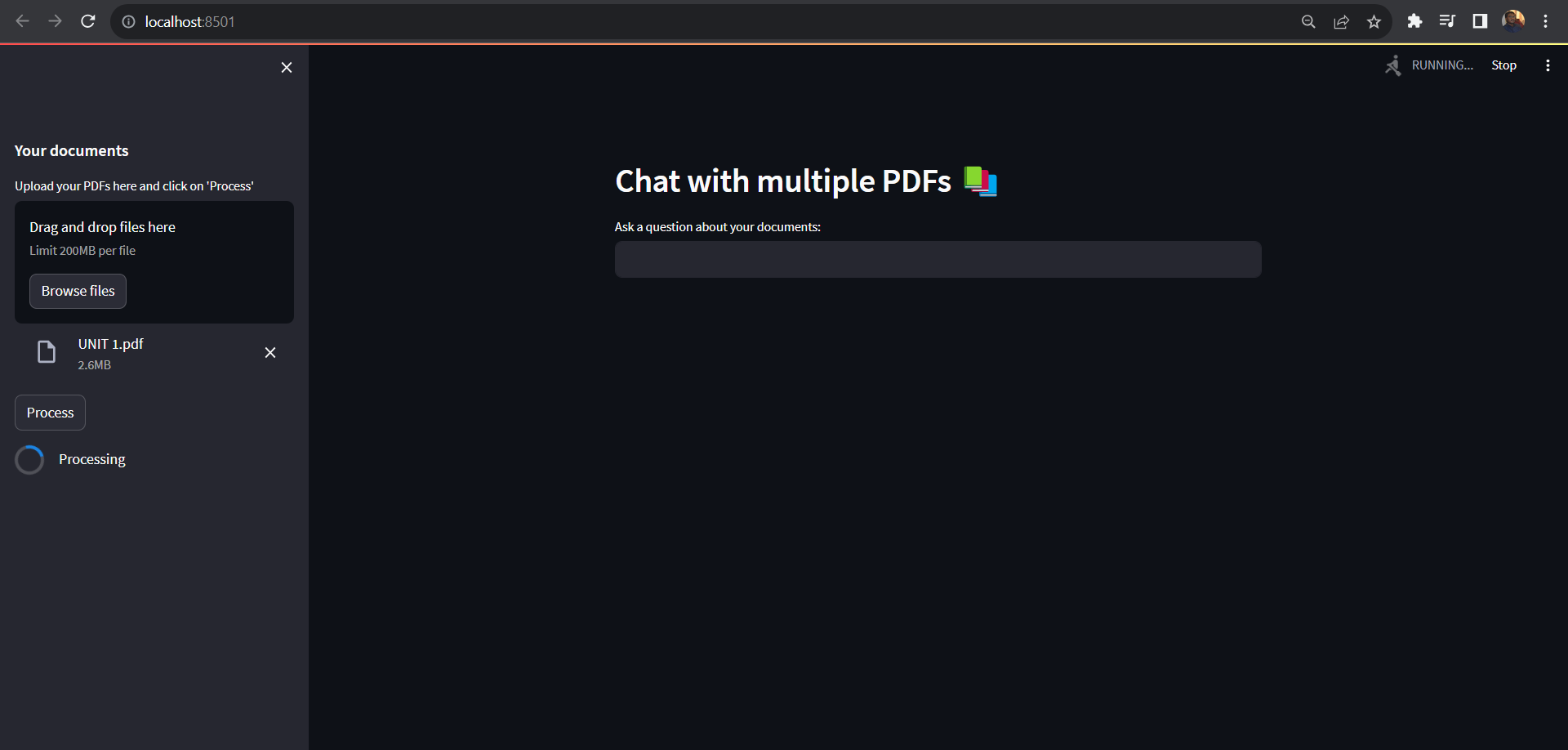
**Upload PDF documents:**Start by adding one or more PDF files using the application's sidebar. Users can use a wide variety of PDF sources with this capability to aid in information retrieval. Simply choose the PDF files you want to upload and do so using the interface that is given.

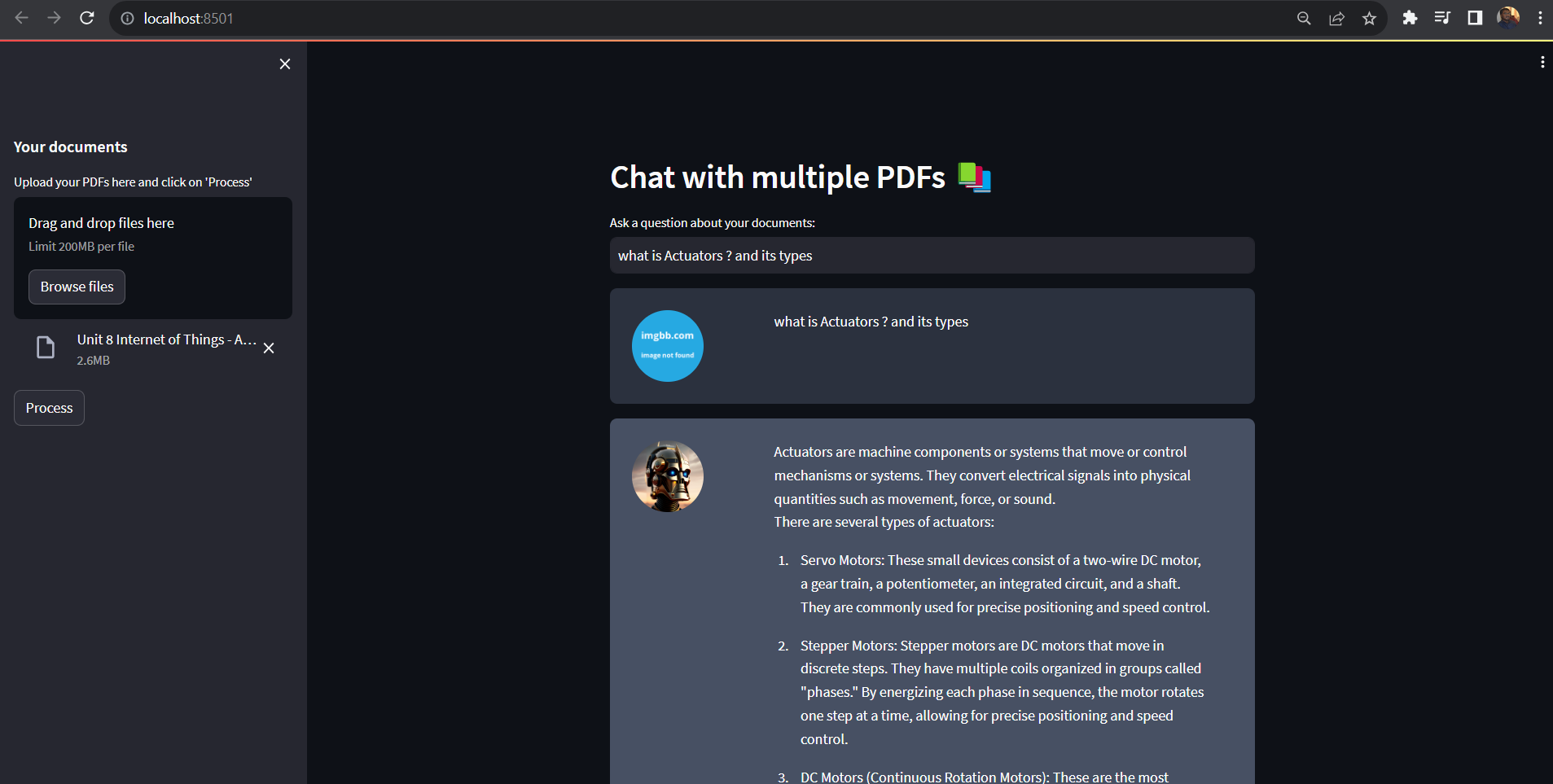
**Ask questions:**You can head over to the main chat interface after uploading the PDF files. You can enter any inquiries you have about the information in the uploaded PDFs here. The chatpdf uses machine learning and natural language processing to accurately understand and respond to your questions.

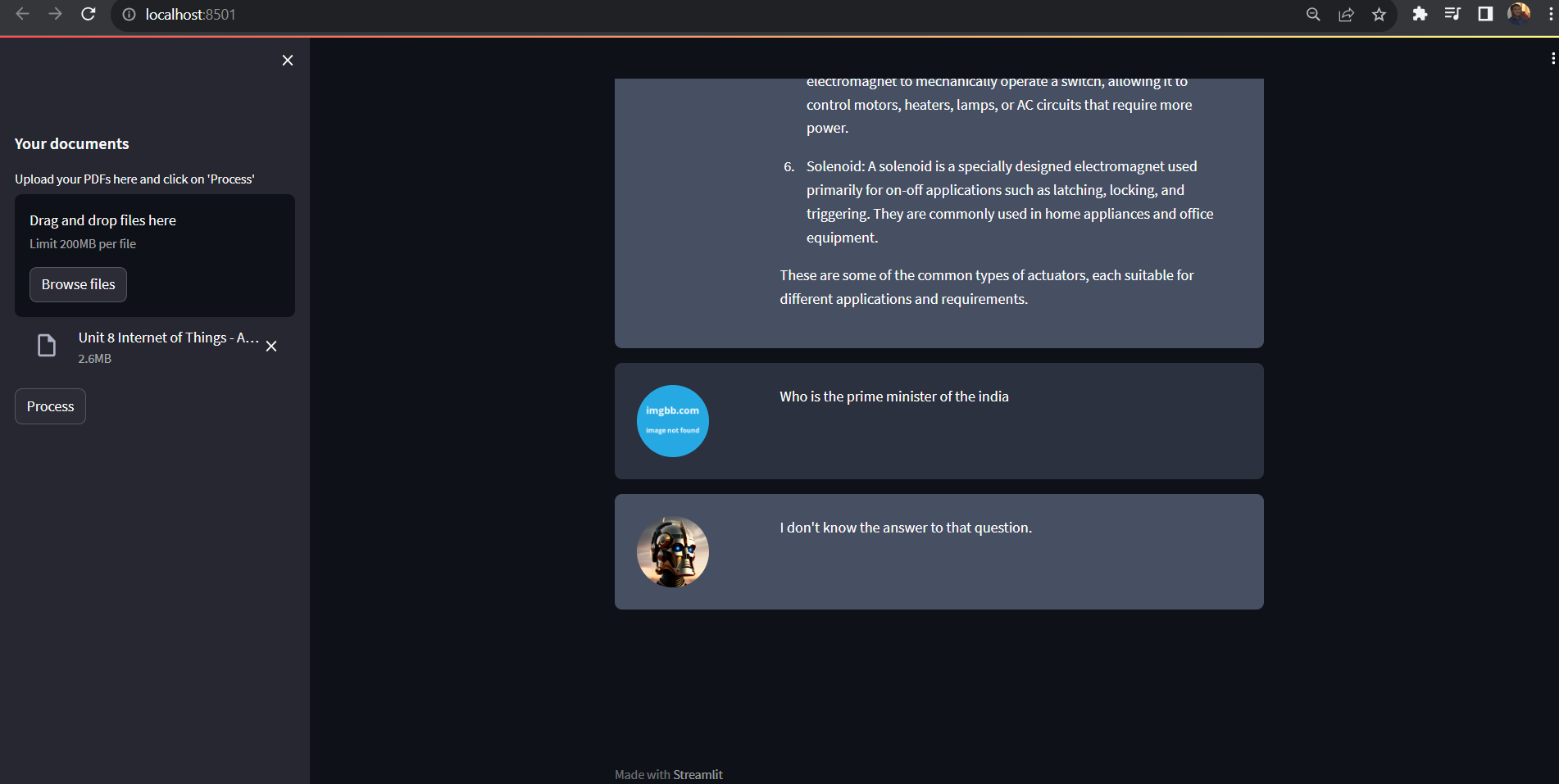
**Receive answers:** The chatpdf processes the data taken from the PDF documents as soon as you submit your inquiries. To produce pertinent and context-aware responses, it makes use of conversational retrieval techniques and the strength of language models. Based on the information in the PDFs, the chatpdf tries to give accurate and useful responses.

**Screenshots of Workflow:**









**Restrictions and Future Improvements:**

There are some issues with our existing method that need to be fixed. One such restriction is the 4,096 token cap set by OpenAI, which is one of our many restrictions. This limits the amount of text we can transfer from the vector DB to about 6-7 pieces, which could lead to inaccurate information retrieval. For instance, there is a chance that one of the documents could be completely overlooked, resulting in the omission of important information.

The 4,096 token cap becomes much more limited. It might not be enough to give precise responses. In these circumstances, it might be required to investigate alternate LLMs that provide greater token limitations, allowing the incorporation of more thorough context. Models with growing token limits are a recent trend in LLM development, which may be helpful in overcoming these restrictions. We can improve the precision and thoroughness of the data presented by utilising an LLM with a greater token limit, especially when working with larger document collections.

**Support for Additional Document Formats:** The ChatPDF can be improved to support other common document formats in addition to PDF files, like Word documents and web pages. The adaptability of the ChatPDF would increase as a result of the users' ability to extract and retrieve data from a larger variety of sources.

**Improved Error Handling and User Feedback:**The ChatPDF’s error handling system can be improved to deliver more detailed and beneficial error messages. This would guarantee that customers receive clear instructions in the event of wrong queries or unexpected circumstances, improving the user experience overall. Additionally, adding user feedback features can aid the ChatPDF in learning from user interactions and enhancing itself over time.

**Enhanced User Interface and Customization Options:** To increase user engagement and happiness, the ChatPDF’s user interface can be further improved and tailored. This may entail making the chat interface's visual appearance, organisational structure, and usability better. A customised experience can be offered to consumers, making the encounter more pleasurable and catered to their interests. Examples of customization choices include theme selection, font styles, or ChatPDF settings.

These upcoming improvements are intended to increase the ChatPDF’s functionality, boost its efficiency, and give consumers a more flexible and intuitive experience. The ChatPDF may be made even more adaptable, precise, and user-friendly by including new document formats, cutting-edge language models, improved error handling, and increased customisation possibilities.

**TESTING**

Testing in the context of software development refers to the process of evaluating and verifying that a software application or system meets its specified requirements and functions correctly. It involves systematically executing the software to identify defects, errors, or issues and ensure that it behaves as intended.

Key aspects of testing in software development include:

1. Verification: Confirming that the software meets its design and functional requirements. This involves checking whether the software does what it's supposed to do.

2. Validation: Ensuring that the software meets the user's needs and expectations. Validation focuses on whether the software accomplishes its intended purpose in the real-world context.

3. Defect Identification: Detecting and documenting defects, bugs, or issues in the software. This includes identifying unexpected behaviors, errors, and deviations from the expected results.

4. Quality Assurance: Ensuring that the software complies with quality standards, guidelines, and best practices. Quality assurance aims to prevent defects and maintain overall software quality.

5. Functional Testing: Evaluating the software's functionality, such as its features, user interfaces, and interactions, to ensure they work correctly.

6. Non-Functional Testing: Assessing non-functional aspects of the software, such as performance, security, scalability, and usability.

7. Regression Testing: Verifying that recent changes to the software (updates, bug fixes, new features) do not introduce new defects or break existing functionality.

8. Integration Testing: Testing how different components or modules of the software interact and work together as a whole.

9. User Acceptance Testing (UAT): Allowing end-users or stakeholders to test the software to ensure it meets their requirements and expectations.

10. Automated Testing: Using automated scripts and tools to execute tests repeatedly and efficiently, especially for repetitive and regression testing.

11. Manual Testing: Human testers interact with the software manually to evaluate its behavior, usability, and user experience.

12. Load Testing: Assessing how the software performs under specific loads or conditions to ensure it can handle expected user traffic.

13. Security Testing: Evaluating the software's security measures to identify vulnerabilities and weaknesses.

14. Usability Testing: Assessing the user-friendliness and user experience of the software to ensure it is intuitive and efficient for users.

15. Black Box Testing: Testing the software without knowledge of its internal code or structure, focusing on inputs and expected outputs.

16. White Box Testing: Testing the software with knowledge of its internal code and structure to assess its logic, algorithms, and data flow.

Testing is a critical phase in the software development life cycle, and it helps identify and address issues early in the development process, reducing the risk of defects reaching the production environment. Proper testing contributes to the reliability, stability, and quality of software products.

Functional Testing :

Functional Testing is a software testing technique that assesses whether a software application or system functions according to its specified requirements and performs the tasks it's designed to do. This type of testing primarily focuses on verifying that the software's features, functions, and user interactions work correctly. During Functional Testing, testers evaluate inputs, execute actions, and examine outputs to ensure that the software behaves as intended. The goal is to identify and document any defects, errors, or inconsistencies in the functionality of the software. Functional Testing plays a crucial role in ensuring that the software meets its functional requirements and delivers the expected results.

Usability Testing :

Usability Testing is a software testing approach that evaluates the user-friendliness and overall user experience of a software application or system. This testing aims to determine how easy it is for users to interact with the software and accomplish their tasks efficiently. Usability Testing involves real users or test participants who perform specific tasks within the software while testers observe their actions and gather feedback. The focus is on identifying usability issues, such as confusing user interfaces, navigation difficulties, or unintuitive workflows. Usability Testing helps ensure that the software meets the needs and expectations of its intended users and provides a satisfying and efficient user experience.

Integration Testing :

Integration Testing is a software testing method that assesses how different components, modules, or subsystems of a software application work together when integrated into a complete system. This testing focuses on verifying that the interactions between these components are functioning correctly and that data flows smoothly between them. Integration Testing aims to detect any inconsistencies, communication errors, or interface issues that may arise when multiple parts of the software are combined. Testers often use various integration strategies, such as top-down, bottom-up, or incremental approaches, to systematically test the integration points. The goal is to ensure that the integrated software functions as a cohesive whole and that the components work harmoniously to deliver the intended functionality. Integration Testing is a crucial step in identifying and addressing integration-related defects early in the development process.

**Test Cases**

Test Case 1: Verify Home Page Display

- Open the web browser and navigate to the embedded web application's URL.

- Expected Result: The web application's home page is displayed.

Test Case 2: PDF Upload

- Click on the "Upload PDF" button.

- Select a PDF file with a size less than or equal to 200MB from your local storage.

- Expected Result: The PDF file is successfully uploaded, and the application displays the name of the uploaded PDF.

Test Case 3: PDF Processing

- Click on the "Process" button.

- Wait for the application to process the uploaded PDF.

- Expected Result: The processing is completed without errors, and a confirmation message indicates that the PDF processing is finished.

Test Case 4: PDF-Based Query

- In the provided text box, ask a question related to the content of the uploaded PDF.

- Click on the "Submit" or "Ask" button.

- Expected Result: The application sends the query to the OpenAI API and displays the answer based on the PDF's content. The answer is relevant to the question and makes sense in the context of the PDF.

Test Case 5: Non-PDF Query

- In the provided text box, ask a question unrelated to the content of the uploaded PDF.

- Click on the "Submit" or "Ask" button.

- Expected Result: The application correctly handles non-PDF queries and does not send them to the OpenAI API. An appropriate message (e.g., "No relevant information found") is displayed for non-PDF queries.

**CONCLUSION**

In conclusion, the testing of the embedded web application designed for PDF upload, processing, and query handling has been conducted systematically. This application leverages the OpenAI API to provide answers based on the content of uploaded PDFs while gracefully handling non-PDF queries. Below is a summary of the key findings from the testing process:

1. PDF Upload and Processing: The application effectively allows users to upload PDF documents with a maximum size of 200MB. The processing of uploaded PDFs is performed without errors, and users are promptly informed when the processing is complete.

2. PDF-Based Queries:The application successfully sends user queries related to the content of the uploaded PDF to the OpenAI API. The answers provided by the application are relevant to the questions and align with the information present in the PDF document.

3. Non-PDF Queries:The application demonstrates proper handling of non-PDF queries. It does not attempt to process such queries through the OpenAI API but instead provides an informative message to users, indicating that no relevant information was found.

Overall, the embedded web application exhibits the desired functionality as outlined in the project requirements. It efficiently processes PDFs, extracts meaningful information, and responds to user queries accordingly. Additionally, the application's error handling mechanisms are robust, ensuring a smooth user experience even in scenarios where the PDF upload or query may encounter issues.

The successful testing of this application demonstrates its readiness for deployment and use by end-users. It offers a valuable solution for users seeking to extract insights and answers from large PDF documents with the assistance of AI-powered query processing.

Further testing and optimization may be considered to enhance the application's performance, such as refining error messages, improving user feedback, and conducting additional load testing to ensure scalability and responsiveness under various conditions.

In summary, the testing results indicate that the embedded web application meets its intended objectives and is poised to provide a valuable service to its users by facilitating efficient PDF processing and query handling through the OpenAI API.

**Bibliography**

1. "An Introduction To HuggingFace Transformers for NLP." Weights & Biases, https://wandb.ai/int\_pb/huggingface/reports/An-Introduction-To-HuggingFace-Transformers-for-NLP--VmlldzoyOTgzMjI5.

2. "OpenAI API Keys." OpenAI Platform, https://platform.openai.com/account/api-keys.

3. "PDF-LangChain.jpg." GitHub, https://github.com/alejandro-ao/ask-multiple-pdfs/blob/main/docs/PDF-LangChain.jpg

4. "LangChain - Question Answering Use Cases." LangChain, https://python.langchain.com/docs/use\_cases/question\_answering/.

5. "LangChain Tutorial 4: Build an 'Ask the Doc' App." Streamlit Blog, https://blog.streamlit.io/langchain-tutorial-4-build-an-ask-the-doc-app/.