SFBU Customer Support System - Chatbot From Files

Special Topics: Generative AI-Driven Intelligent Apps Development

Ademilton Marcelo da Cruz Nunes (19679)

Links

Github:
https://github.com/ademiltonnunes/Machine-Learning/tree/main/ChatGPT/Customer%20Support%20System/SFBU%20Customer%20Support%20System%20-%20Chatbot%20From%20Files

Table of Content

- Introduction
- Design
- Design - Question and Answer block
- Design - Upload File Block
- Implementation
- Document Loading
- Splitting documents
- Embedding, Vector db and storage
- Retrieval vector db
- Answering Question
- Conclusion

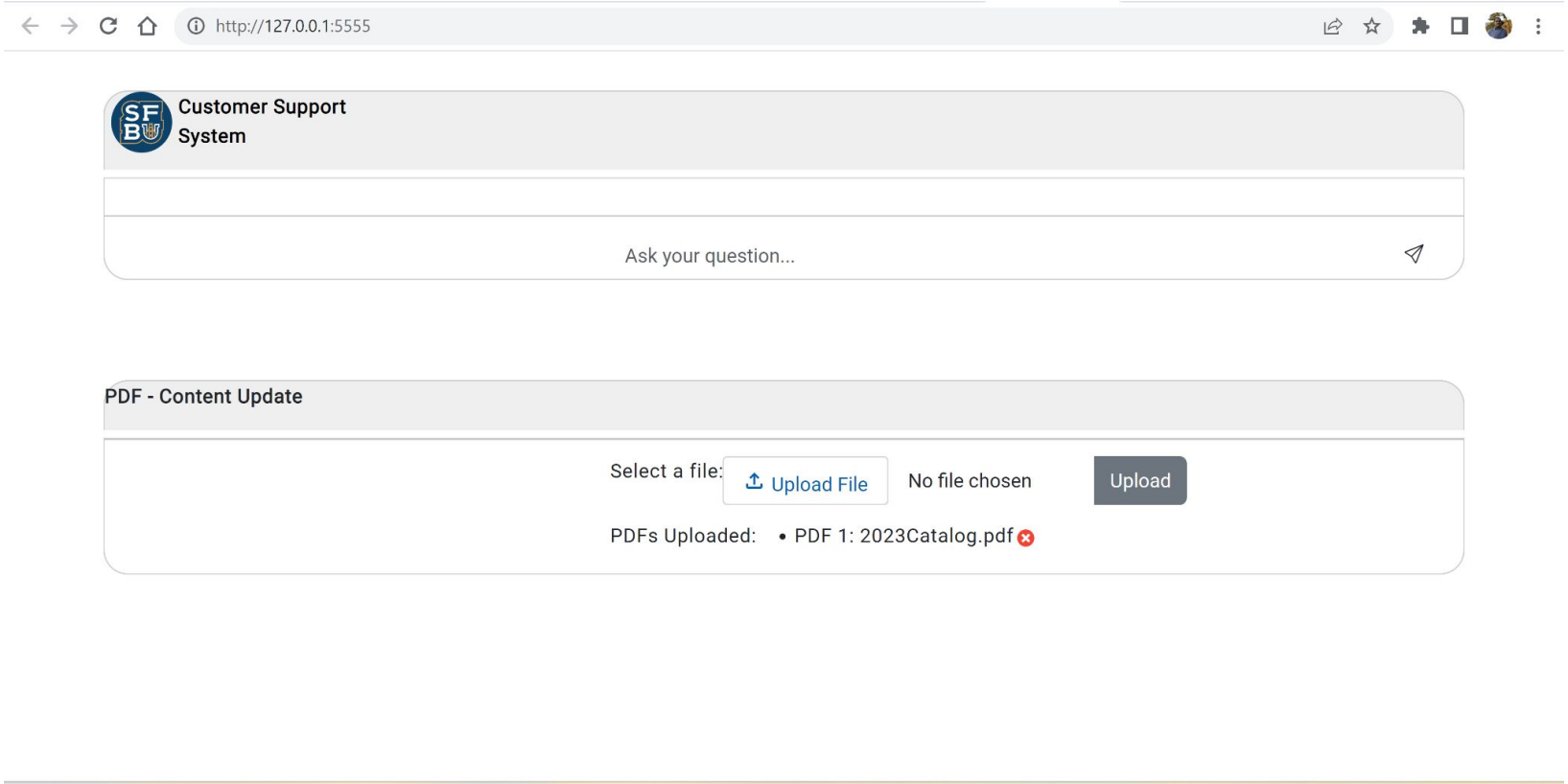
Introduction

This project aims to implement a web application of a customer support system for SFBU that answers customer’s questions based in a loading PDF.

The system was designed as a Flask web application with HTML and CSS user interface. This project will use the ChatGPT OpenAI GPT-3.5 Turbo model.

The system has the following appearance:

Introduction



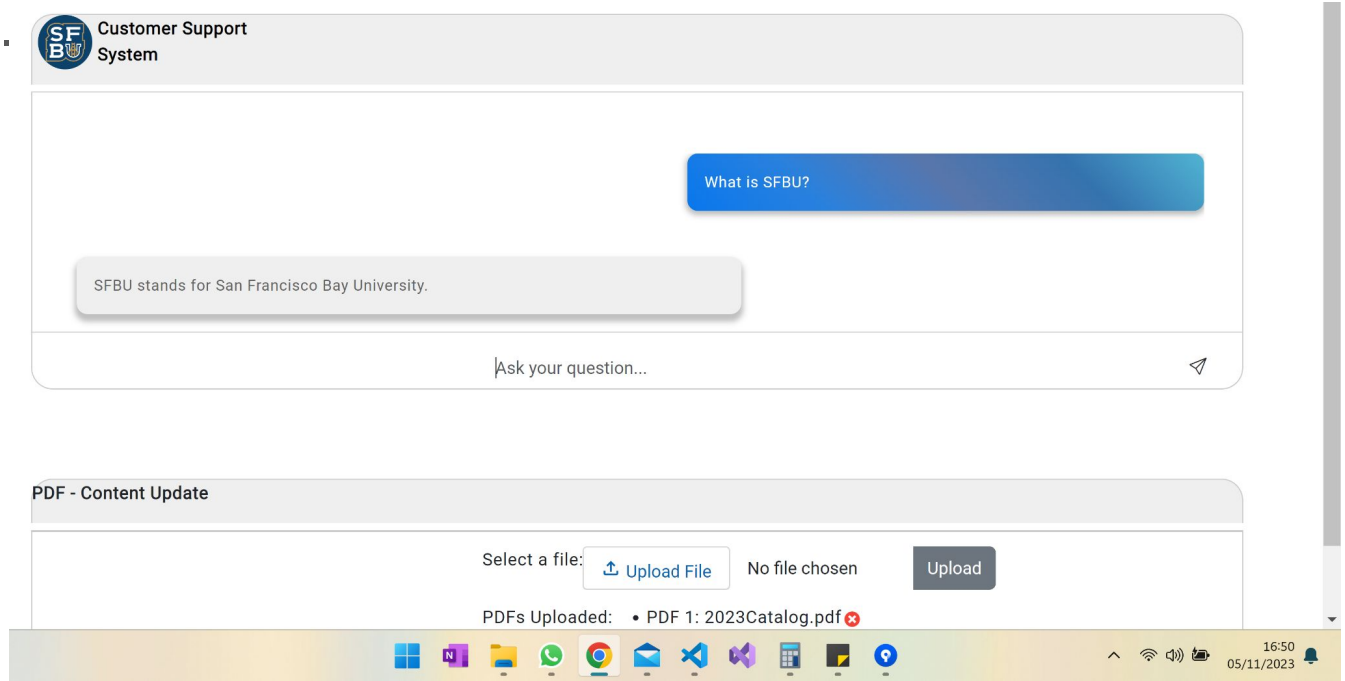
Design

The system is divided in 2 blocks:

- Question and Answer block
- Upload File Block

Design - Question and Answer block

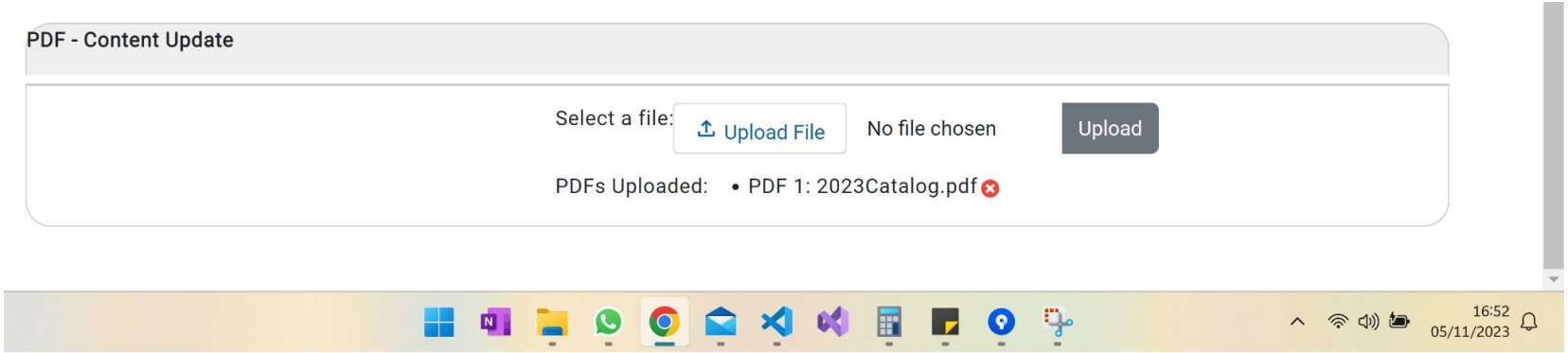
This local is where the customer gets question answered based in the loading PDFs.



Design - Upload File Block

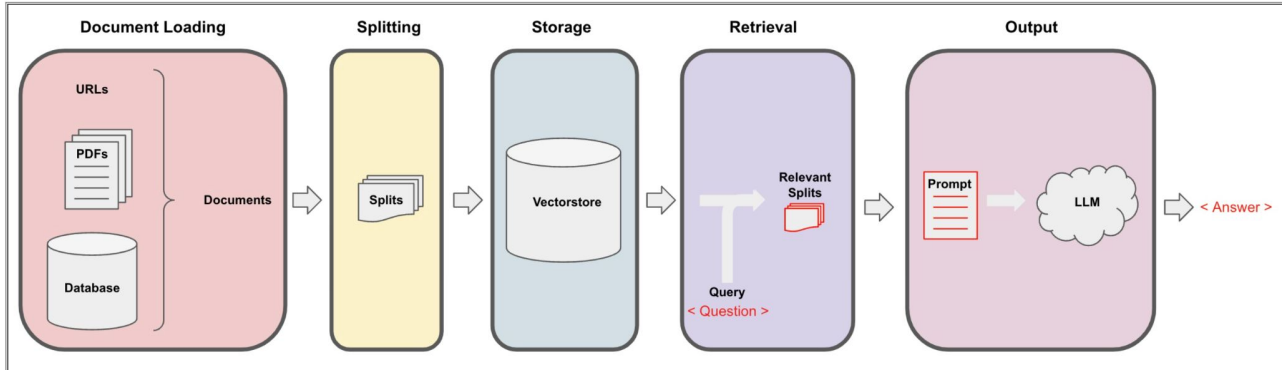
This local is where the customer load files (PDFs) that are used to answer question in the Question and Answer block.

If the user wants to delete a PDF, simply click on option x and the system will reload the PDF to answer questions



Implementation

For the system to be able to answer questions, the customer must first upload at least one PDF. The loading process and answering questions are based on the following processes:



Document Loading

In this project the data comes from PDF, but this can come from other sources such as websites, different databases, YouTube, and etc. In LangChain, there are 80 different types of document loaders.

Regardless of the data source, we must put the data in text format to be fed into the system.

To load PDF files we must import the PyPDFLoader module from langchain:

```python
def __load_PDF(self, pathPdf:str) -> List[Document]:
    pdf = PyPDFLoader(pathPdf)
    return pdf.load()
```

136 ⚠ 0 0 Live Share Spaces: 4 UTF-8 CRLF Python 3.11.5 64-bit

17:04
05/11/2023

Splitting documents

Having the document loaded into the system in text format, we must divide this document into chunks so that they are in the format that the system can use them. We can configure how many chunks we want. Chunks separation is based on semantic syntax. There are some approaches to splitting chunks, in our project we used the RecursiveCharacterTextSplitter approach. We also configured Chunk Overlap, which Controls whether chunks share some data or information with neighboring chunks, influencing the continuity of information.

Splitting documents

In our system we divide the chuks into 1500 sizes, and overlap 150.

```python
def __split_content(self, docs) -> List[Document]:

    text_splitter = RecursiveCharacterTextSplitter(
        chunk_size = 1500,
        chunk_overlap = 150
    )

    return text_splitter.split_documents(docs)
```

136 0 0 Live Share Spaces: 4 UTF-8 CRLF Python 3.11.5 64-bit

17:39
05/11/2023

Embedding, Vector db and storage

Embedding: For each chunk, we generate the embedding indexes. Chunks that are semantically closer have similar indexes.

Vector db: The indexes are stored in a vector store database. This vectorstore has N dimensions, and the chunks that are semantically neighbors are stored in the same dimensions.

Storage: The vectorstore database can be kept in memory or stored locally and can be easily retrieved to be used in the system, without the need to generate a new database.

Embedding, Vector db and storage

```python
def __create_vectorstore(self, chunks) -> None:
    #Create Indexes
    embedding = OpenAIEmbeddings()

    # Remove the direc
    if os.path.exists(persist_directory):
        directory_path = f'./{persist_directory}'
        shutil.rmtree(directory_path)

    vectordb = Chroma.from_documents(
        documents=chunks,
        embedding=embedding,
        persist_directory=persist_directory
    )
```

(variable) persist_directory: str

136 ⚠ 0 0 Live Share Spaces: 4 UTF-8 CRLF Python 3.11.5 64-bit

17:48
05/11/2023

Retrieval vector db

With the vector database stored locally, we can easily retrieve it to use as a basis for answering questions. Each time the customer asks a question, the database is loaded and retrieved for use.

```python
# Load the Chroma object from the file
embedding = OpenAIEmbeddings()
vector_db = Chroma(persist_directory= persist_directory, embedding_function= embedding)

#Retrieve db
retriever = vector_db.as_retriever(search_type="similarity", search_kwargs={"k": k})
```

0 0 Live Share Ln 44, Col 24 Spaces: 4 UTF-8 CRLF Python 3.11.5 64-bit

17:51
05/11/2023

Answering Question

With the vectorstore db retrieved, we use it to create a conversational retrieval chain, building upon the retrieval QA chain in LLM.

Question answering is able to deal with historical questions and context through memory buffer.

Answering Question

```python
# Create a ConversationBufferMemory
memory = ConversationBufferMemory(
memory_key="chat_history",
return_messages=True  # Return chat history as a list of messages
)

qa = ConversationalRetrievalChain.from_llm(
    llm=ChatOpenAI(model_name="gpt-3.5-turbo", temperature=0),
    chain_type=chain_type,
    retriever=retriever,
    return_source_documents=False,
    return_generated_question=False,
    memory=memory,
    output_key='answer'  # Specify the desired output key
)
```

36 ⚠ 0 0 Live Share Ln 113, Col 43 Spaces: 4 UTF-8 CRLF Python 3.11.5 64-bit

17:56
05/11/2023

Answering Question

With ConversationalRetrievalChain we can answer questions for customers based on indexed files separated into blocks.

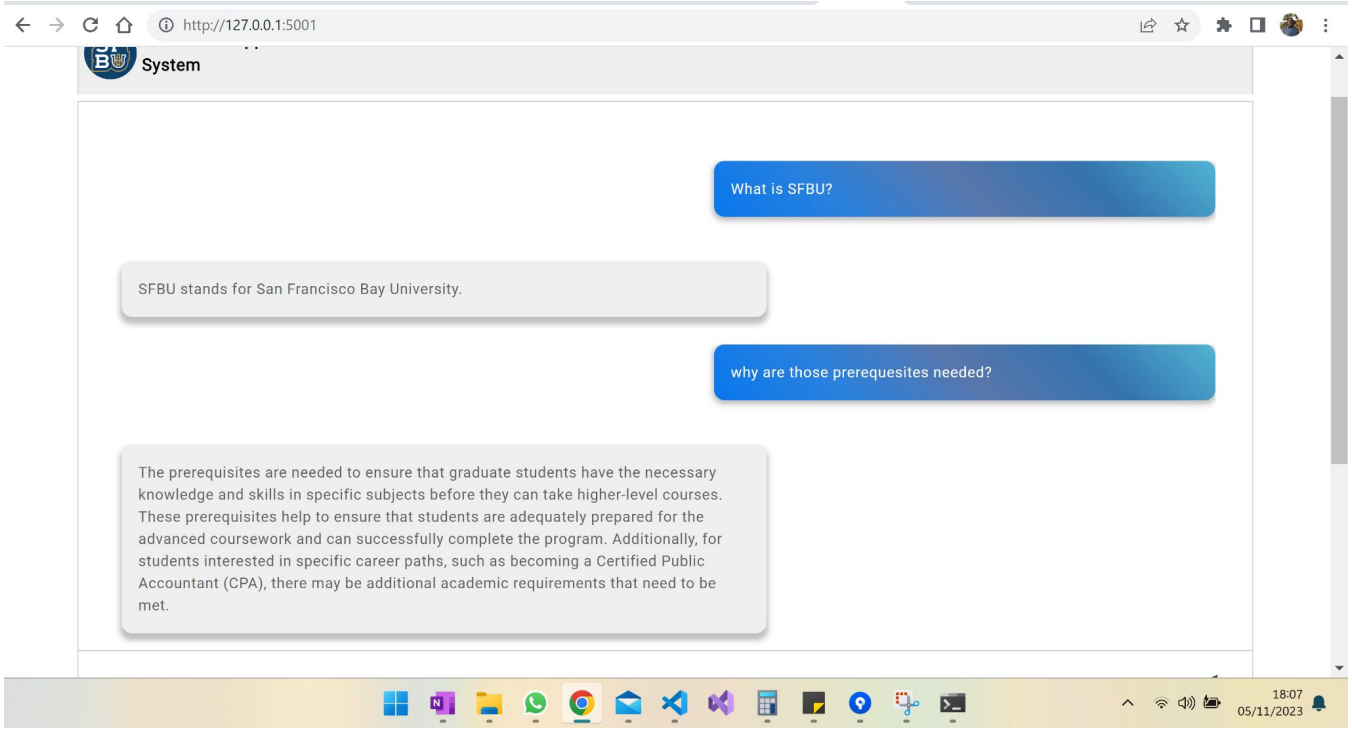
```
chatbot = self.__load_chatbot()
response = chatbot({'question': question,'chat_history': []})

return response
```

0 Live Share Ln 121, Col 30 Spaces: 4 UTF-8 CRLF Python 3.11.5 64-bit

17:58
05/11/2023

Answering Question

Example of answering customers' questions:



Conclusion

This project demonstrated implement a web application of a customer support system for SFBU that answers customer’s questions based in a loading PDF.

I applied techniques and showed solid examples of how to load, split, embed indexes, store vectorstore db and retrive it to a LLM in order to answer questions in a ConversationalRetrievalChain.