Chat With Pdf

This application designed to allow users to upload documents (specifically PDF files) and query those documents using natural language questions. The system utilizes advanced AI technologies to generate responses based on the content of the uploaded documents. The main goal of this project is to enhance user interaction with digital documents through intuitive querying.

Technologies Used

- **Python**: Programming language used for development.
- **Flask**: Web framework for creating the application.
- **Lang chain**: Framework for building applications with LLMs.
- **Pinecone**: Vector database for managing embeddings.
- **Firebase**: Backend service for managing data and authentication.

Prerequisites

- Python 3.10
- Necessary libraries (installable via requirements.txt).

```
- flask
- PyPDF2
- pinecone
- firebase_admin
- python-dotenv
- google-generativeai
- langchain
- langchain-google-genai
- profanity-check
-
```

Link to GitHub Repository

[https://github.com/Atiqur78/Chat_With_Pdf]

```
To get started with the project, follow these steps:

1. Clone the repository: `git clone
https://github.com/Atiqur78/Chat_With_Pdf.git`
2. Install the required dependencies: `pip install -r requirements.txt`
3. Create .env file and write the required API_KEY
4. Run the application: `python main.py`
```

6. API Overview

Endpoints

1. **Upload Document**
   - **Endpoint**: /upload
   - **Method**: POST
   - **Request Format**: Multipart form data (file and chat_name).
   - **Response**: Redirects to index with success or error message.
2. **Query Document**
   - **Endpoint**: /query
   - **Method**: POST
   - **Request Format**: Form data (query_chat_name and question).
   - **Response**: JSON containing success status and answer.

Sample Requests

Upload document: curl -X POST -F "chat_name=your_chat_name" -F "file=@path_to_your_file.pdf" http://localhost:5000/upload

Query: curl -X POST -H "Content-Type: application/x-www-form-urlencoded" -d "query_chat_name=your_chat_name&question=your_question" http://localhost:5000/query

document_upload.py

```
from PyPDF2 import PdfReader

from src.logger import logging
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_google_genai import GoogleGenerativeAIEmbeddings
from firebase_admin import firestore
```

```python
from pinecone import Pinecone, ServerlessSpec
from src.exception import CustomException
from dotenv import load_dotenv
from firebase_admin import credentials, initialize_app, firestore
import sys
import os


load_dotenv()
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = "ragmodel-91184-firebase-
adminsdk-mn1rb-1729150ab9.json"
cred = credentials.Certificate(os.environ["FIRE_BASE"])
initialize_app(cred)

# for initializing Pinecone
pc = Pinecone(
    api_key=os.environ["PINCONE_API"]
)

index_name = 'langchainvector'

# for checking the index exits or not
if index_name not in pc.list_indexes().names():
    pc.create_index(
        name=index_name,
        dimension=768,
        metric='cosine',
        spec=ServerlessSpec(cloud='aws', region='us-east-1')
    )

def handle_document_upload(uploaded_file, chat_name):
    try:
        text = extract_pdf_text(uploaded_file)
        text_chunks = split_text_into_chunks(text)

        # for Ccreating  Pinecone Index
        index = pc.Index(index_name)
        embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-
001")
        vectors = embeddings.embed_documents(text_chunks)

        upsert_data = [
            {"id": f"{chat_name}_{i}", "values": vector, "metadata": {"text":
text_chunks[i]}}
            for i, vector in enumerate(vectors)
        ]

        # Upsert vectors to Pinecone
        index.upsert(vectors=upsert_data)
```

```
        # Store index metadata in Firestore
        db = firestore.client()
        db.collection("document_indices").document(chat_name).set({"index_name
": index_name})

        logging.info(f"Document {chat_name} indexed successfully")

    except Exception as e:
        logging.error(f"Error during document upload: {str(e)}")
        raise CustomException(e, sys)

#  to extract text from PDF
def extract_pdf_text(pdf_file):
    text = ""
    pdf_reader = PdfReader(pdf_file)
    for page in pdf_reader.pages:
        text += page.extract_text()
    return text

# to split text into chunks
def split_text_into_chunks(text):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=10000,
chunk_overlap=1000)
    return text_splitter.split_text(text)
```

document_query.py

```
from langchain_google_genai import GoogleGenerativeAIEmbeddings
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain.prompts import PromptTemplate
from langchain.chains.question_answering import load_qa_chain
from langchain.docstore.document import Document
from firebase_admin import firestore
from pinecone import Pinecone
from src.logger import logging
from src.exception import CustomException
import sys, os

# for initializing Pinecone
pc = Pinecone(
    api_key=os.environ["PINCONE_API"]
)

def handle_document_query(chat_name, question):
    try:
```

```
        # fectching index from firebase
        db = firestore.client()
        index_ref =
db.collection("document_indices").document(chat_name).get()

        if not index_ref.exists:
            raise ValueError(f"Document index {chat_name} not found")

        index_name = index_ref.to_dict().get("index_name")
        index = pc.Index(index_name)

        # embeding the questioin into vector
        embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-
001")
        vector_query = embeddings.embed_query(question)

        # Query Pinecone
        search_result = index.query(
            vector=vector_query, top_k=3, include_values=False,
include_metadata=True
        )

        docs = [
            Document(page_content=result['metadata']['text'])
            for result in search_result['matches']
            if 'metadata' in result
        ]

        chain = get_conversational_chain()
        response = chain({"input_documents": docs, "question": question},
return_only_outputs=True)

        return response['output_text']

    except Exception as e:
        logging.error(f"Error during document query: {str(e)}")
        raise CustomException(e, sys)

def get_conversational_chain():
    prompt_template = """
    Answer the question as detailed as possible from the provided context. If
the answer is not in the provided context,
    say 'answer is not available in the context'.\n\n
    Context:\n {context}?\n
    Question: \n{question}\n
    Answer:
    """
    model = ChatGoogleGenerativeAI(model="gemini-pro", temperature=0.3)
```

```
    prompt = PromptTemplate(template=prompt_template,
input_variables=["context", "question"])
    return load_qa_chain(model, chain_type="stuff", prompt=prompt)
```

validation.py

```
from profanity_check import predict

def validate_question(question):
    if len(question) < 5:
        return False, "Question too short"
    if predict([question])[0] == 1:
        return False, "Offensive content detected"
    return True, "Valid question"
```

app.py

```
from flask import Flask, request, render_template, redirect, url_for, jsonify
from src.document_upload import handle_document_upload
from src.document_query import handle_document_query
from src.logger import logging
from src.exception import CustomException
import sys


app = Flask(__name__)
app.secret_key = 'your_secret_key'




@app.route('/')
def index():
    return render_template('index.html')


@app.route('/upload', methods=['POST'])
def upload_document():
    try:
        if 'file' not in request.files or 'chat_name' not in request.form:
            logging.info('Missing file or chat name', 'error')
            return redirect(url_for('index'))

        uploaded_file = request.files['file']
```

```
        chat_name = request.form['chat_name']

        if uploaded_file.filename == '':
            logging.error('No file selected', 'error')
            return redirect(url_for('index'))

        handle_document_upload(uploaded_file, chat_name)
        logging.info('Document indexed successfully', 'success')

    except Exception as e:
        logging.error(f"Error during document upload: {str(e)}")
        raise CustomException(e, sys)

    return redirect(url_for('index'))


@app.route('/query', methods=['POST'])
def query_document():
    try:
        chat_name = request.form.get('query_chat_name')
        question = request.form.get('question')

        if not chat_name or not question:
            logging.info('Missing chat name or question', 'error')
            return redirect(url_for('index'))


        answer = handle_document_query(chat_name, question)
        logging.info(f"Answer: {answer}", 'success')
        return render_template('index.html', answer=answer)

    except Exception as e:
        logging.error(f"Error during document query: {str(e)}")
        raise CustomException(e, sys)


if __name__ == '__main__':
    app.run(debug=True)
```