Setting up

```
!pip uninstall langchain
!pip install langchain
!pip install pymupdf
!pip install pinecone-client
#!pip install openai
!pip install tiktoken
#!pip install chromadb
!pip install --quiet langchain
!pip install pymupdf
!pip install -U langchain-community
!pip install fastapi
#!pip install fastapi nest-asyncio pyngrok uvicorn
!pip install langchain_google_genai
#!pip install uvicorn
```

Show hidden output

```
#!wget https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.zip
#!unzip -o ngrok-stable-linux-amd64.zip
```

Show hidden output

Setting up NGROK

From this code we can setup the ngrok tunnel service to run the application on collab iteslf without exposing the ip google collab

```
#import os
#from google.colab import userdata
#from pyngrok import conf
#os.environ["NGROK"] = userdata.get("NGROK")
#conf.get_default().auth_token = os.environ["NGROK"]
```

```
#from fastapi import FastAPI, UploadFile, File
#from fastapi.responses import JSONResponse
#from fastapi.middleware.cors import CORSMiddleware
```

```
#app = FastAPI()
#app.add_middleware(
 #   CORSMiddleware,
  #  allow_origins=['*'],
   # allow_credentials=True,
    #allow_methods=['*'],
   # allow_headers=['*'],
#)
#@app.get("/")
#async def root():
 #   return {"message": "Hello World"}
```

```
# Run this cell and paste the API key in the prompt
import os
import getpass

os.environ['GOOGLE_API_KEY'] = getpass.getpass('Gemini API Key:')
```

```
Gemini API Key:··········
```

Set up Pinecone API keys

```
os.environ['PINECONE_API_KEY'] = getpass.getpass('Pinecone API Key:')
```

```
Pinecone API Key:··········
```

Index

Split the text from pdf into smaller chunks

There are many ways to split the text. We are using the text splitter that is recommended for generic texts. For more ways to slit the text check the documentation

Create embeddings

Creating a vectorstore

A vectorstore stores Documents and associated embeddings, and provides fast ways to look up relevant Documents by embeddings.

There are many ways to create a vectorstore. We are going to use Pinecone.

```
from langchain_google_genai import GoogleGenerativeAIEmbeddings
from langchain.document_loaders import PyMuPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.vectorstores import Pinecone
       # Read the uploaded file content
loader = PyMuPDFLoader("/content/Fullstack Internship Assignment.pdf")
pages = loader.load_and_split()
text_splitter = RecursiveCharacterTextSplitter(
    chunk_size = 1000,
    chunk_overlap  = 200,
    length_function = len,
)

docs = text_splitter.split_documents(pages)
# If there is no environment variable set for the API key, you can pass the API
# key to the parameter `google_api_key` of the `GoogleGenerativeAIEmbeddings`
# function: `google_api_key = "key"`.
gemini_embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")
index_name = "langchain-demo"
docsearch = Pinecone.from_documents(docs, gemini_embeddings, index_name=index_name)
# if you already have an index, you can load it like this
# docsearch = Pinecone.from_existing_index(index_name, embeddings)



query = "What are the deliverables?"
docs = docsearch.similarity_search(query)
# print(len(docs))
# print(docs[0])
print(docs[0].page_content)
```

```
Asking Questions:
•
Users can ask questions related to the content of an uploaded PDF.
•
The system processes the question and the content of the PDF to provide an answer.
Displaying Answers:
•
The application displays the answer to the user’s question.
•
Include the functionality to ask follow-up or new questions on the same document.
Non-Functional Requirements:
Usability: Ensure the user interface is intuitive and easy to navigate.
Performance: Optimize the processing of PDF documents and the response time for answering
questions.
Backend Specification:
FastAPI Endpoints:
```

The following code block uses Croma for creating a vectorstore. Uncomment it if you don't have access to pinecone and use it instead.

```
# from langchain.vectorstores import Chroma
# docsearch = Chroma.from_documents(docs, embeddings)
```

Vectorstore is ready. Let's try to query our docsearch with similarity search

Making a question answering chain

```
from langchain.chains import RetrievalQA
from langchain import OpenAI
from langchain_google_genai import ChatGoogleGenerativeAI
#defining LLM
llm = ChatGoogleGenerativeAI(model="gemini-1.5-pro-latest" , temperature = .2)
result = llm.invoke("Write about LangChain")
qa = RetrievalQA.from_chain_type(llm=llm, chain_type="stuff", retriever=docsearch.as_retriever(search_kwargs={"k": 2}))
```

Query the chain to test that it's working

```
query = "describe the assignment"
qa.run(query)
```

```
/usr/local/lib/python3.10/dist-packages/langchain_core/_api/deprecation.py:139
  warn_deprecated(
'The assignment is to create a full-stack application, the design of which is
provided in a Figma file. You need to deliver:\n\n1. **Source Code:** Well-st
ructured and commented frontend and backend code.\n2. **Documentation:** A RE
ADME file containing setup instructions, API documentation, and an overview o
f the application's architecture.\n3. **Demo:** A live demo or screencast sho
wcasing the application's functionality.\n\nThe assignment will be evaluated
based on:\n\n* **Functionality:** Whether the application meets all the funct
ional and non-functional requirements.\n* **Code Quality:** Cleanliness, orga
```

Preparing Zapier tool

First, you need to get a Zapier API key here https://nla.zapier.com/get-started/ and add the the actions that you are going to use in Zapier

```
os.environ["ZAPIER_NLA_API_KEY"] = os.environ.get("ZAPIER_NLA_API_KEY", "your zapier api key")
```

Setting up a zapier toolkit. For more information visit documentation

```
from langchain.agents.agent_toolkits import ZapierToolkit
from langchain.utilities.zapier import ZapierNLAWrapper

zapier = ZapierNLAWrapper()
toolkit = ZapierToolkit.from_zapier_nla_wrapper(zapier)
```

```
!pip install --upgrade langchain-community
!pip install --upgrade --quiet langchain-google-genai
```

```
Requirement already satisfied: langchain-community in /usr/local/lib/python3.10/dist-packages (0.2.5)
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.10/dist-packages (from langchain-community) (6.0.1)
Requirement already satisfied: SQLAlchemy<3,>=1.4 in /usr/local/lib/python3.10/dist-packages (from langchain-community)
Requirement already satisfied: aiohttp<4.0.0,>=3.8.3 in /usr/local/lib/python3.10/dist-packages (from langchain-communit
Requirement already satisfied: dataclasses-json<0.7,>=0.5.7 in /usr/local/lib/python3.10/dist-packages (from langchain-c
Requirement already satisfied: langchain<0.3.0,>=0.2.5 in /usr/local/lib/python3.10/dist-packages (from langchain-commun
Requirement already satisfied: langchain-core<0.3.0,>=0.2.7 in /usr/local/lib/python3.10/dist-packages (from langchain-c
Requirement already satisfied: langsmith<0.2.0,>=0.1.0 in /usr/local/lib/python3.10/dist-packages (from langchain-commun
Requirement already satisfied: numpy<2,>=1 in /usr/local/lib/python3.10/dist-packages (from langchain-community) (1.25.2
Requirement already satisfied: requests<3,>=2 in /usr/local/lib/python3.10/dist-packages (from langchain-community) (2.3
Requirement already satisfied: tenacity<9.0.0,>=8.1.0 in /usr/local/lib/python3.10/dist-packages (from langchain-communi
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->lan
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3-
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=3.8.3->la
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp<4.0.0,>=
Requirement already satisfied: marshmallow<4.0.0,>=3.18.0 in /usr/local/lib/python3.10/dist-packages (from dataclasses-j
Requirement already satisfied: typing-inspect<1,>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from dataclasses-jso
Requirement already satisfied: langchain-text-splitters<0.3.0,>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from la
Requirement already satisfied: pydantic<3,>=1 in /usr/local/lib/python3.10/dist-packages (from langchain<0.3.0,>=0.2.5->
Requirement already satisfied: jsonpatch<2.0,>=1.33 in /usr/local/lib/python3.10/dist-packages (from langchain-core<0.3.
Requirement already satisfied: packaging<25,>=23.2 in /usr/local/lib/python3.10/dist-packages (from langchain-core<0.3.0,
Requirement already satisfied: orjson<4.0.0,>=3.9.14 in /usr/local/lib/python3.10/dist-packages (from langsmith<0.2.0,>=
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2-
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langchain-c
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langc
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests<3,>=2->langc
Requirement already satisfied: typing-extensions>=4.6.0 in /usr/local/lib/python3.10/dist-packages (from SQLAlchemy<3,>=
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.10/dist-packages (from SQLAlchemy<3,>=1.4->lan
Requirement already satisfied: jsonpointer>=1.9 in /usr/local/lib/python3.10/dist-packages (from jsonpatch<2.0,>=1.33->la
Requirement already satisfied: mypy-extensions>=0.3.0 in /usr/local/lib/python3.10/dist-packages (from typing-inspect<1,
```

```
from langchain.agents import AgentType, initialize_agent
from langchain_community.agent_toolkits import ZapierToolkit
from langchain_community.utilities.zapier import ZapierNLAWrapper
from langchain_google_genai import GoogleGenerativeAI

llm = GoogleGenerativeAI(model="models/text-bison-001", google_api_key="Your_api_key", temperature=0.1)
zapier = ZapierNLAWrapper()
toolkit = ZapierToolkit.from_zapier_nla_wrapper(zapier)
agent = initialize_agent(
    toolkit.get_tools(), llm, agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose=True)
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
<ipython-input-11-07763639ffe3> in <cell line: 9>()
      8 toolkit = ZapierToolkit.from_zapier_nla_wrapper(zapier)
      9 agent = initialize_agent(
---> 10     toolkit.get_tools(), llm, agent=AgentType.ZERO_SHOT_REACT_DESCRIPTION, verbose=True)

1 frames
/usr/local/lib/python3.10/dist-packages/langchain_core/_api/deprecation.py in warn_deprecated(since, message, name,
alternative, alternative_import, pending, obj_type, addendum, removal, package)
    361         if not removal:
    362             removal = f"in {removal}" if removal else "within ?? minor releases"
--> 363             raise NotImplementedError(
    364                 f"Need to determine which default deprecation schedule to use. "
    365                 f"{removal}"

NotImplementedError: Need to determine which default deprecation schedule to use. within ?? minor releases
```

Building agent

Assembling it all together into an agent.

```
from langchain.agents import AgentType
from langchain.agents import initialize_agent, Tool
from langchain.memory import ConversationBufferMemory
from langchain_google_genai import ChatGoogleGenerativeAI


#defining the tools for the agent
tools = [
    Tool(
        name = "Demo",
        func=qa.run,
        description="use this as the primary source of context information when you are asked the question. Always search fo
    ),
] + toolkit.get_tools()


#setting a memory for conversations
memory = ConversationBufferMemory(memory_key="chat_history")

#Setting up the agent
agent_chain = initialize_agent(tools, llm, agent=AgentType.CONVERSATIONAL_REACT_DESCRIPTION, verbose=True, memory=memory)
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
<ipython-input-18-8b495f62a955> in <cell line: 9>()
     12         description="use this as the primary source of context information when you are asked the question.
Always search for the answers using this tool first, don't make up answers yourself"
     13     ),
---> 14 ] + toolkit.get_tools()
     15
     16

1 frames
/usr/local/lib/python3.10/dist-packages/langchain_core/_api/deprecation.py in warn_deprecated(since, message, name,
alternative, alternative_import, pending, obj_type, addendum, removal, package)
    361         if not removal:
    362             removal = f"in {removal}" if removal else "within ?? minor releases"
--> 363             raise NotImplementedError(
    364                 f"Need to determine which default deprecation schedule to use. "
    365                 f"{removal}"

NotImplementedError: Need to determine which default deprecation schedule to use. within ?? minor releases
```

Quering the agent

To get an agent do what you want, the prompt should be constructed properly. For user facing apps, we need to look at prompt templates and also figure out if chat is the best interface

```
agent_chain.run(input="What Techstack does the project requiee?")
```

```
agent_chain.run(input="Email the answer to email@gmail.com and mention that this email was sent by AI")
```

Runnign the agent using private tunnel

```
#import nest_asyncio
#from pyngrok import ngrok
#import uvicorn
#conf.get_default().auth_token = "your_auth_token"  # Set your authtoken here
#ngrok_tunnel = ngrok.connect(8000)
#print('Public URL:', ngrok_tunnel.public_url)
#nest_asyncio.apply()
#print(f"Received request with method: {request.method}")
#uvicorn.run(app, port=8000)
```

```
Public URL: https://87cc-34-148-89-108.ngrok-free.app
INFO:     Started server process [175]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     49.43.112.248:0 - "GET / HTTP/1.1" 200 OK
INFO:     49.43.112.248:0 - "GET /favicon.ico HTTP/1.1" 404 Not Found
INFO:     49.43.112.248:0 - "GET /upload HTTP/1.1" 405 Method Not Allowed
INFO:     49.43.112.248:0 - "GET / HTTP/1.1" 200 OK
```