RepoGPT-main/requirements.txt

```
gradio
requests
PyPDF2
reportlab
PILLOW
langchain
flask
unstructured
unstructured[local-inference]
openai
faiss-cpu
```

RepoGPT-main/app.py

```
import gradio as gr
import os
import shutil
import requests
import zipfile
from PyPDF2 import PdfFileReader, PdfFileWriter
import PyPDF2
from io import BytesIO
from reportlab.lib.pagesizes import letter
from reportlab.platypus import SimpleDocTemplate,Preformatted
from reportlab.platypus import Image  as RLImage
from reportlab.platypus import Paragraph, Spacer
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib.utils import ImageReader
from PIL import Image
import os
from langchain.indexes.vectorstore import VectorstoreIndexCreator
from langchain.chains import VectorDBQA,VectorDBQAWithSourcesChain
from langchain import OpenAI
from langchain.document_loaders import UnstructuredPDFLoader
from langchain.vectorstores.faiss import FAISS
from langchain.embeddings.openai import OpenAIEmbeddings
from flask import send_file
# from IPython.display import Markdown, display


class REPOGPT:
    def __init__(self) -> None:

        self.repo_link = None
        self.api_key = None

    def init_agent(self, api_key, repo_link = None,  load_vectorstore = None):
        try:
            os.remove('merged.pdf')
        except:
            pass
        self.repo_link = repo_link
        self.api_key = api_key
        self.load_vectorstore = load_vectorstore
        #assert if api key is valid
        assert self.api_key != None, "You need to provide an API key"
        self.REPOGPT_Initialized()
        return gr.update(visible = True),'Initialize Finished'



    def REPOGPT_Initialized(self,image_included = False):


        os.environ["OPENAI_API_KEY"] = self.api_key
        if self.load_vectorstore == None:
            loader = UnstructuredPDFLoader( self.create_repo_pdf(self.repo_link,image_included = image_i
            # pages = loader.load_and_split()
            self.index = VectorstoreIndexCreator(vectorstore_cls = FAISS).from_loaders([loader])
            self.vectorstore = self.index.vectorstore
            print(' vectorstore created')
        else:
            embeddings = OpenAIEmbeddings()
            self.vectorstore = FAISS.load_local(self.load_vectorstore,embeddings =embeddings)
            print(' vectorstore loaded')

        self.qa = VectorDBQA.from_chain_type(llm =OpenAI(temperature=0, model_name="gpt-3.5-turbo"), cha
    def download_repo_zip(self, link, output_folder = "main.zip"):
        username =  link.split('/')[3]
        repo = link.split('/')[4]
        zip_url = f"https://github.com/{username}/{repo}/archive/refs/heads/master.zip"
        self.zip_url = zip_url
```

```
        response = requests.get(zip_url)
        response.raise_for_status()
        #down load the zip file
        with open('main.zip', 'wb') as f:
            f.write(response.content)
        # return the name of the extracted folder
        # return self.extract_zip("main.zip", output_folder)
        # return BytesIO(response.content)

    def extract_zip(self, zip_file, destination_folder):
        with zipfile.ZipFile(zip_file) as zf:
            zf.extractall(destination_folder)
        #get the name of the extracted folder
        folder_name = zf.namelist()[0]
        return folder_name

    def convert_to_pdf(self, input_path, output_path):
        if input_path.endswith(".pdf"):
            # Create a new PDF with the file path heading
            buffer = BytesIO()
            doc = SimpleDocTemplate(buffer, pagesize=letter)
            styles = getSampleStyleSheet()
            elements = []
            heading = Paragraph(f"File path: {input_path}", styles["Heading2"])
            elements.append(heading)
            elements.append(Spacer(1, 12))
            doc.build(elements)

            # Read the newly created PDF with heading
            buffer.seek(0)
            new_pdf = PdfFileReader(buffer)

            # Read the input PDF
            with open(input_path, "rb") as f:
                input_pdf = PdfFileReader(f)

            # Merge the new PDF with heading and the input PDF
            pdf_writer = PdfFileWriter()
            for page_num in range(new_pdf.getNumPages()):
                pdf_writer.addPage(new_pdf.getPage(page_num))

            for page_num in range(input_pdf.getNumPages()):
                pdf_writer.addPage(input_pdf.getPage(page_num))

            # Save the merged PDF to the output file
            with open(output_path, "wb") as f:
                pdf_writer.write(f)

        elif input_path.lower().endswith((".jpg", ".jpeg", ".png", ".gif", ".bmp", ".tiff")):
            img = Image.open(input_path)
            img_reader = ImageReader(img)
            img_width, img_height = img.size
            aspect_ratio = img_height / img_width


            max_pdf_width = letter[0] - 2 * 72  # 1 inch margin on each side
            max_pdf_height = letter[1] - 2 * 72  # 1 inch margin on top and bottom

            if img_width > max_pdf_width:
                    img_width = max_pdf_width
                    img_height = img_width * aspect_ratio
            if img_height > max_pdf_height:
                img_height = max_pdf_height
                img_width = img_height / aspect_ratio
            img_width = int(img_width)
            img_height = int(img_height)
            # Resize the image
            img = img.resize((int(img_width), int(img_height)))

            img = img.resize((int(img_width), int(img_height)))

            img.save(output_path, "PNG")
```

```
            # Create a new PDF with the image
            doc = SimpleDocTemplate(output_path, pagesize=letter)
            styles = getSampleStyleSheet()

            elements = []
            heading = Paragraph(f" {input_path}", styles["Heading2"])
            elements.append(heading)
            elements.append(Spacer(1, 12))

            img_rl = RLImage(input_path, width=img_width, height=img_height, kind='proportional')
            elements.append(img_rl)

            doc.build(elements)

        else:
            with open(input_path, "r") as f:
                content = f.read()

            doc = SimpleDocTemplate(output_path, pagesize=letter)
            styles = getSampleStyleSheet()
            elements = []

            # Add the file path heading
            heading = Paragraph(f"{input_path}", styles["Heading2"])
            elements.append(heading)
            elements.append(Spacer(1, 12))

            # Add the content as Preformatted text
            text = Preformatted(content, style=styles["Code"])
            elements.append(text)

            doc.build(elements)

    def merge_pdfs(self, pdf_files, output_path):
        pdf_writer = PyPDF2.PdfWriter()
        for pdf_file in pdf_files:
            with open(pdf_file, "rb") as f:
                try:
                    pdf_reader = PyPDF2.PdfReader(f)
                    if pdf_reader.is_encrypted:
                        print(f"{pdf_file} is encrypted. Skipping.")
                        continue
                except:
                    print(f"{pdf_file} is not a valid PDF. Skipping.")
                    continue


                for page_num in range(len(pdf_reader.pages)):
                    pdf_writer.add_page(pdf_reader.pages[page_num])

        with open(output_path, "wb") as f:
            pdf_writer.write(f)

    def get_pdf(self):
        return self.merged_pdf_path

    def save_indexDB(self,save_path = 'indexDB.json'):
        self.vectorstore.save_local(save_path)
        print("indexDB saved at: ", save_path)



    def create_repo_pdf(self, repo_link, image_included = False,  merged_pdf = "temp_merged.pdf"):
        self.merged_pdf_path = merged_pdf
        self.download_repo_zip(repo_link)
        folder_name = self.extract_zip('./main.zip', './')
        ingnore_list = ['__pycache__',]
        if not image_included:
            ingnore_list.append('.jpg')
            ingnore_list.append('.png')
            ingnore_list.append('.jpeg')
            ingnore_list.append('.gif')
```

```
            ingnore_list.append('.bmp')
            ingnore_list.append('.tiff')

        print('folder_name: ', folder_name)
        pdf_files = []
        for root, dirs, files in os.walk(folder_name):
            for file in files:

                input_file = os.path.join(root, file)
                #if the file contains any of the strings in the ignore list, skip it
                if any(x in input_file for x in ingnore_list):
                    continue
                #create a temp folder to store the pdf files
                os.makedirs("temp", exist_ok=True)
                output_file = os.path.join("temp", os.path.splitext(file)[0] + ".pdf")

                try:
                    self.convert_to_pdf(input_file, output_file)
                except:
                    print("Error converting file: ", input_file)
                    continue
                pdf_files.append(output_file)



        self.merge_pdfs(pdf_files, self.merged_pdf_path)
        #clean up the temp folder and downloaded zip file
        os.remove("main.zip")
        shutil.rmtree(folder_name)
        shutil.rmtree("temp")

        return self.merged_pdf_path


    def Answer_quetsion(self, question):
        return self.qa.run(question)

    def Answer_quetsion_with_source(self, question):
        return self.qa({"question": question}, return_only_outputs = True)



def call_output(string = 'REPOGPT Initializing'):
    return string

def download_file(filename = 'merged.pdf'):
    # filename = repogpt.get_pdf()
    return send_file(filename, as_attachment=True)


if __name__ == "__main__":
    repogpt = REPOGPT()


    with gr.Blocks() as demo:
        with gr.Row():
            gr.Markdown("<h3><center>REPOGPT</center></h3>")
            gr.Markdown(
                """This is a demo to the work [REPOGPT](https://github.com/wuchangsheng951/RepoGPT).<br>
                This space connects ChatGPT and RepoGPT is a Python library that allows you to search an
                """
            )
        with gr.Row():
            apikey = gr.Textbox(
                placeholder="Paste your OpenAI API key here to start Visual ChatGPT(sk-...) and press En
                show_label=True,
                label = 'OpenAI API key',
                lines=1,
                type="password",
            )
        with gr.Row():
            repo_link = gr.Textbox(
```

```
            placeholder="Paste your repo_link and press Enter ↵■",
            label = 'repo_link',

            show_label=True,
            lines=1,
        )

    with gr.Column(scale=0.7):
            Initialize = gr.Button("Initialize RepoGPT")

    output = gr.Textbox(label="Output Box")

    with gr.Row(visible=False) as input_raws:
        with gr.Column(scale=0.7):
            txt = gr.Textbox(show_label=False, placeholder="Enter your question").style(container=Fa

        with gr.Column(scale=0.4):
            AQ = gr.Button("Ask a Question").style(container=False)

        # with gr.Row():
        #     Download = gr.Button("Download PDF")


    gr.Examples(
        examples=["Whats the name of this repo?",
                "Whats this repo for?",
                "How can I use this. Example code ? Step by step",
                "how can I use this Experiment trackers ? Step by step",
                "how can I Performing gradient accumulation with Accelerate? Step by step?",
                "Make it like water-color painting",
                "What is the background color",
                "Describe this image",
                "please detect the depth of this image",
                "Can you use this depth image to generate a cute dog",
                ],
        inputs=txt
    )

    apikey.submit(repogpt.init_agent, [apikey,repo_link], [input_raws, output])
    Initialize.click(repogpt.init_agent, [apikey,repo_link], [input_raws, output])
    apikey.submit(call_output, [],[output])
    txt.submit(repogpt.Answer_quetsion, [txt], [output])
    AQ.click(repogpt.Answer_quetsion, [txt], [output])
    # Download.click(download_file, [], [Download])


demo.launch()
```

RepoGPT-main/README.md

````
# RepoGPT
RepoGPT is a Python library that allows you to search and answer questions about a GitHub repository's c



Usage
Install the dependencies:
```pip install -r requirements.txt```

Initialize the REPOGPT object:
```python
repogpt = REPOGPT()
repogpt.init_agent(api_key="your_api_key", repo_link="https://github.com/user/repo")
answer = repogpt.Answer_question("What is the purpose of this repository?")
print(answer)

```
## Example
```python
qa.run('How can I use this. Example code ? Step by step?')
```


This project is licensed under the MIT License.
````
````