• client.py

```
from MY_PACKAGE.main_win import LogIn
from tkinter import *
from tkinter import ttk,messagebox
from PIL import ImageTk,Image
from functools import partial
import requests,threading
window=Tk()
title="DoCu_It"
window.title(title)
server_link_register="http://127.0.0.1:5000/register"
server_link_login="http://127.0.0.1:5000/login"
server_link_filelist="http://127.0.0.1:5000/allfile"
email=StringVar()
password=StringVar()
email_reg=StringVar()
password_reg=StringVar()
email_placeholder="Enter your email"
password_placeholder="Enter your password"
place_holder={"email":email_placeholder,"password":password_placeholder}
def toggle_pass(entry_var,button):
    if entry_var.cget("show")=="*":
        entry_var.config(show="")
        button.config(text="Hide")
    elif entry_var.cget("show")=="":
        entry_var.config(show="*")
        button.config(text="Show")
```

```
def focus_out(place_hold_name=None,entry_var=None,button=None,textvar=None):
   if button!=None:
      button.config(state="disabled")
    if textvar.get().strip()=="":
         textvar.set(place_holder["password"])
   if entry_var!=None:
      if entry_var.get().strip()=="":
         entry_var.set(place_holder[f"{place_hold_name}"])
def focus_in(place_hold_name=None,entry_var=None,button=None,textvar=None):
   if button!=None:
      button.config(state="normal")
      if textvar.get().strip()==place_holder["password"]:
         textvar.set("")
   if entry_var!=None:
      if entry_var.get().strip()==place_holder[f"{place_hold_name}"]:
         entry_var.set("")
restrict=0#to restrict the number of windows
def login_init():
   global restrict
   def on_close():
      global restrict
      if messagebox.askokcancel("Quit", "Do you want to quit?"):
         restrict=0
         main.destroy()
   data={"email":email.get().strip(),"password":password.get().strip()}
   if (data["email"]=="" or data["email"]==email_placeholder) or (data["password"]==""
or data["password"]==password_placeholder):
      messagebox.showerror("DoCu_It","plz fill the details")
   else:
      try:
         response=requests.post(url=server_link_login,data=data)
         status=response.json()
         email.set(email_placeholder)
         password.set(password_placeholder)
         messagebox.showinfo("DoCu_It",status["message"])
         if status.get("user") and restrict==0:
             try:
                 credential=data["email"]
                 main=LogIn(email=credential)
                 restrict=1
                 window.wm_state('iconic')
                 main.protocol("WM_DELETE_WINDOW",on_close)
                 main.mainloop()
             except:
                 pass
         elif restrict==1:
             messagebox.showwarning(title,"At a time only window can be opened")
      except:
         messagebox.showerror("DOCu-It","Server connection not established")
def regester_init():
   data={ "email":email_reg.get().strip(),"password":password_reg.get().strip()  }
   if (data["email"]=="" or data["email"]==email_placeholder) or (data["password"]==""
or data["password"]==password_placeholder):
      messagebox.showerror("DoCu_It","plz fill the details")
   else:
      try:
         response=requests.post(url=server_link_register,data=data)
         status=response.json()
        email_reg.set(email_placeholder)
        password_reg.set(password_placeholder)
        messagebox.showinfo("DoCu_It",status["message"])
    except:
        messagebox.showerror("DOCu-It","Server connection not established")
def process(funcname):
  thread=threading.Thread(target=funcname)
  thread.daemon=True
  thread.start()
primary_color="#091353"#dark_blue
window.geometry("1500x700")
window.resizable(False,False)
img_frame=Frame(window,bg=primary_color)
auth_img=Image.open("Images/auth.png")
auth_img=auth_img.resize((300,200))
auth_img=ImageTk.PhotoImage(auth_img)
ttk.Label(img_frame,image=auth_img).pack(ipady=300,ipadx=20)
img_frame.pack(side=LEFT,fill="y")
tabs_frame=Frame(window,height=700,width=1400)
tabs_frame.pack(side=LEFT,fill=BOTH)
tabs=ttk.Notebook(tabs_frame,height=800,width=1400)
tabs.pack(pady=(5,0),fill="both")
login_tab=Frame(tabs,width=1400,height=700,bg=primary_color)
register_tab=Frame(tabs,width=1400,height=700,bg=primary_color)
login_tab.pack(fill="both")
register_tab.pack(fill="both")
tabs.add(login_tab,text="LOGIN")
tabs.add(register_tab,text="REGISTER")
```

```
log_image=Image.open("Images/login.png")

log_image=log_image.resize((100,100))

log_image=ImageTk.PhotoImage(log_image)

Label(login_tab,bg=primary_color,image=log_image).pack(pady=(0,40))

Label(login_tab,text="Login to get access to your saved automated projects.\nYour
safety our first
priority",bg=primary_color,font=("Courier","15","bold"),fg="#ffeb3b").pack(pady=(0,40
))

email_entry=ttk.Entry(login_tab,width=40,font=("Courier","18"),textvariable=email)

email_entry.pack(pady=(20,70))

email.set(place_holder["email"])

email_entry.bind("<FocusIn>",lambda
e:focus_in(place_hold_name="email",entry_var=email,button=None,textvar=None))

email_entry.bind("<FocusOut>",lambda
e:focus_out(place_hold_name="email",entry_var=email,button=None,textvar=None))

pass_frame=ttk.Frame(login_tab)

pass_frame.pack()

password_entry=ttk.Entry(pass_frame,width=37,font=("Courier","18"),show="",textvari
able=password)

show_pass=Button(pass_frame,text="Hide",state="disabled")

show_pass.pack(side=RIGHT,fill=BOTH)

show_pass.config(command=partial(toggle_pass,password_entry,show_pass))

password_entry.pack()

password.set(place_holder["password"])

password_entry.bind("<FocusIn>",lambda
e:focus_in(textvar=password,entry_var=None,button=show_passpassword_entry.bind(
"<FocusOut>",lambda
e:focus_out(textvar=password,entry_var=None,button=show_pass))

submit=ttk.Button(login_tab,text="LOG IN",command=partial(process,login_init))

submit.pack(pady=40)

registration_image=Image.open("Images/register.png")

registration_image=registration_image.resize((100,100))
```

```
registration_image=ImageTk.PhotoImage(registration_image)

Label(register_tab,bg=primary_color,image=registration_image).pack(pady=(0,40))

Label(register_tab,text="Plz Register to enjoy our automation
service",bg=primary_color,font=("Courier","15","bold"),fg="#ffeb3b").pack(pady=(0,40
))

email_registry=ttk.Entry(register_tab,width=40,font=("Courier","18"),textvariable=emai
l_reg)

email_registry.pack(pady=(20,70))

email_reg.set(place_holder["email"])

email_registry.bind("<FocusIn>",lambda
e:focus_in(place_hold_name="email",entry_var=email_reg,button=None,textvar=None)
)

email_registry.bind("<FocusOut>",lambda
e:focus_out(place_hold_name="email",entry_var=email_reg,button=None,textvar=Non
e))

pass_reg_frame=ttk.Frame(register_tab)

pass_reg_frame.pack()

password_registry=ttk.Entry(pass_reg_frame,width=37,font=("Courier","18"),show="",t
extvariable=password_reg)

show_pass_reg=Button(pass_reg_frame,text="Hide",state="disabled")

show_pass_reg.pack(side=RIGHT,fill=BOTH)

show_pass_reg.config(command=partial(toggle_pass,password_registry,show_pass_re
g))

password_registry.pack()

password_reg.set(place_holder["password"])

password_registry.bind("<FocusIn>",lambda
e:focus_in(textvar=password_reg,entry_var=None,button=show_pass_reg))

password_registry.bind("<FocusOut>",lambda
e:focus_out(textvar=password_reg,entry_var=None,button=show_pass_reg))

register=ttk.Button(register_tab,text="Register",command=partial(process,regester_ini
t))

register.pack(pady=40)

window.mainloop()
```

- **server.py**

```
from flask import Flask,request,jsonify,send_file
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash, check_password_hash
import os
from io import BytesIO
from json import JSONEncoder
app=Flask(__name__)
DB_NAME="DATABASE\DOCu_It.db"
app.config['SQLALCHEMY_DATABASE_URI'] = f'sqlite:///{DB_NAME}'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db=SQLAlchemy(app)
class User(db.Model):
   id = db.Column(db.Integer, primary_key=True)
   email = db.Column(db.String(100), unique=True)
   password = db.Column(db.String(100))
   doc=db.relationship("ProjectFile",backref="user")
class ProjectFile(db.Model):
   id = db.Column(db.Integer, primary_key=True)
   filename=db.Column(db.String(100))
   file=db.Column(db.LargeBinary)
   person_email = db.Column(db.String(100),db.ForeignKey("user.email"))
@app.route("/register",methods=["POST"])
def register():
   email=request.form.get("email")
   password=request.form.get("password")
   user=User.query.filter_by(email=email).first()
   if user:
        return {"message":"Email Already present"},409
    else:

new_user=User(email=email,password=generate_password_hash(password,method="s
ha256"))
        db.session.add(new_user)
        db.session.commit()
        return {"message":"Registered"},201
@app.route("/login",methods=["POST"])
def login():
    email=request.form.get("email")
    password=request.form.get("password")
    user=User.query.filter_by(email=email).first()
    if user:
        #comparing hash and given password
        if check_password_hash(user.password,password):
            return {"user":True,"message":"Found"}
        else:
            return {"message":"Password Not Matching"},409
    else:
        return {"message":"User not found"},404
@app.route("/upload",methods=["POST"])
def upload():
    file=request.files["upload"]
    name=request.form["email"]
    auth=ProjectFile.query.filter_by(filename=file.filename,person_email=name).first()
    if auth:
        return "Already Present",409
    else:
```

```
    new_file=ProjectFile(filename=file.filename,file=file.read(),person_email=name)
    db.session.add(new_file)
    db.session.commit()
    return "done",201
@app.route("/download",methods=["POST"])
def download():
  user=request.form["email"]
  request_file=request.form["file"]
  user_file=ProjectFile.query.filter_by(person_email=user,filename=request_file).first()
  if user_file:
    return send_file(BytesIO(user_file.file),attachment_filename=user_file.filename)
  else:
    return "not found",404
@app.route("/allfile",methods=["POST"])
def files():
  email=request.form["email"]
  data=ProjectFile.query.filter_by(person_email=email).all()
  filename_list={}
  for i in range(len(data)):
    filename_list[i]=data[i].filename
  return jsonify(filename_list)
```

- **main_win.py**

```
from tkinter import *
from tkinter import ttk,messagebox,colorchooser,filedialog
from PIL import Image,ImageTk
import threading as td
import requests
from MY_PACKAGE.project_parser import Parser#when calling this whole main_win as a
module
class LogIn(Toplevel):
  max_height=1500
  max_width=700
```

```
    primary_color="#091353"
    def __init__(self,email=None):
r().__init__()
        self.email=email#for verfication and connecting to server
        self.geometry(f"{self.max_height}x{self.max_width}")
        self.name="DoCu_It"
        self.title(self.name)
        self.resizable(0,0)
        self.any_project=False#needs to be false. Used for enabling options and disabling
options if nothing project is searched
        self.proj_title=None
        self.count_paras=0
        self.not_blank_position=0
        self.project_data_encoded=None
        self.docx_save=None
        self.color_choice=["000000"]*5
        self.search_var=StringVar()
        self.upload_var=StringVar()
        # Image frame
        self.img=Image.open("MY_PACKAGE\Images\icon.ico")
        self.img=self.img.resize((200,200))
        self.img=ImageTk.PhotoImage(self.img)
        self.img_frame=Frame(self)
        self.img_label=Label(self.img_frame,image=self.img,text="Project
Automation",compound=TOP,font=("Microsoft JhengHei UI Light","16"))
        self.img_label.pack()
        self.img_frame.pack(side=LEFT,ipadx=10)
        # tabs
        self.tab=ttk.Notebook(self,height=self.max_height)
        self.tab.pack(fill=BOTH,pady=10)

self.automate=Frame(self.tab,width=self.max_width,height=self.max_height,bg=self.pri
mary_color)
        self.upload=Frame(self.tab,width=self.max_width,height=self.max_height)
        self.automate.pack(fill=BOTH)
        self.upload.pack(fill=BOTH)
        self.tab.add(self.automate,text="Automate")
        self.tab.add(self.upload,text="Upload")
        self.api_img1=Image.open("MY_PACKAGE\Images\internet.png")
        self.api_img=ImageTk.PhotoImage(self.api_img1)
        Label(self.automate,image=self.api_img,bg=self.primary_color).pack()
        Label(self.automate,text="DoCu_IT",font=("Microsoft JhengHei UI
Light","24","bold"),bg=self.primary_color,fg="#F0A500").pack(pady=(10,0))
```

```
        Label(self.automate,text="You search,Arnab Chatterjee will
automate",font=("Microsoft JhengHei UI
Light","15","bold"),bg=self.primary_color,fg="#F0A500").pack(pady=(4,0))
        self.search=Frame(self.automate,width=37)
        self.search.pack(pady=(2,40))

self.search_bar=ttk.Entry(self.search,width=37,font=("Courier","18"),textvariable=self.s
earch_var)
        self.search_bar.pack(side=LEFT)

self.search_ico=ImageTk.PhotoImage(Image.open("MY_PACKAGE\Images\search.png")
)

self.search_btn=ttk.Button(self.search,image=self.search_ico,command=self.search_pro
ject)
        self.search_btn.pack(side=LEFT)
        self.btn_frame=Frame(self.automate,bg=self.primary_color)
        self.btn_frame.pack()


self.automate_btn=ttk.Button(self.btn_frame,text="Automate",command=self.save_pr
oject)
        self.automate_btn.pack(side=LEFT,padx=(0,7))


self.overview=ttk.Button(self.btn_frame,text="Overview",command=self.open_modal)
        self.overview.pack(side=LEFT)
        for child in self.btn_frame.winfo_children():
            if self.any_project==False:
                child["state"]="disabled"
        # upload/download section
        self.rocket= Image.open(r'MY_PACKAGE\Images\rocket.png').resize((300,300))
        self.rocket= ImageTk.PhotoImage(self.rocket)
        Label(self.upload,image=self.rocket).pack()
        #upload
        self.file_upload_frame=LabelFrame(self.upload,text="Upload File",padx=8,pady=4)
        self.file_upload_frame.pack()
        self.upload_icon=Image.open(r"MY_PACKAGE\Images\upload.png")
        self.upload_icon=ImageTk.PhotoImage(self.upload_icon.resize((50,50)))
        Label(self.file_upload_frame,image=self.upload_icon).pack(side=LEFT)

self.file_directory=ttk.Entry(self.file_upload_frame,width=50,textvariable=self.upload_v
ar)
        self.file_directory.pack(side=LEFT)
self.browse_file=ttk.Button(self.file_upload_frame,text="Browse",command=self.brow
se)
      self.browse_file.pack(side=LEFT,padx=5)

self.upload_file=ttk.Button(self.file_upload_frame,text="Upload",command=self.upload
_file)
      self.upload_file.pack(side=LEFT)
            #download
      self.file_download_frame=LabelFrame(self.upload,text="Download
File",padx=4,pady=4)
      self.file_download_frame.pack(pady=50)
      self.download_icon=
Image.open(r'MY_PACKAGE\Images\download.png').resize((50,50))
      self.download_icon= ImageTk.PhotoImage(self.download_icon)
      Label(self.file_download_frame,image=self.download_icon).pack(side=LEFT)
      self.file_view=ttk.Combobox(self.file_download_frame,width=50)
      self.file_view.pack(side=LEFT)

self.download_file=ttk.Button(self.file_download_frame,text="Download",command=s
elf.download_file)
      self.download_file.pack(side=LEFT,padx=5)

      self.download_file_options()

   def download_file_options(self):
      def process():
         uploded_file_link="http://127.0.0.1:5000/allfile"
         try:
            file_response= requests.post(uploded_file_link,data={"email":self.email})
            actual_data=file_response.json()
            data=[]
            for i in actual_data:
               data.append(actual_data[i])
            self.uploaded_file_server=data
            self.file_view["values"]=data
            self.file_view.update()
         except:
            data=None
            self.file_view["values"]=tuple(data)
            self.file_view.update()

      thread=td.Thread(target=process,daemon=True)
      thread.start()
   def browse(self):
      file_types=[ ("Word file",".docx") ]
      location = filedialog.askopenfilename(initialdir="Your Projects",title="Select
file",filetypes=file_types)
      self.upload_var.set(location)

   def upload_file(self):
      def process():
         if self.upload_var.get().strip()!="":
            try:
               file_content=open(self.upload_var.get(),"rb")
            except:
               messagebox.showerror(self.name,"Plz check the file location. Some error
occured")
            data={
               "email":self.email,
            }
            file={
               "upload":file_content
            }
            link="http://127.0.0.1:5000/upload"
            res=requests.post(link,data=data,files=file)
            messagebox.showinfo(self.name,res.text)
            file_content.close()
            self.download_file_options()
         else:
            messagebox.showwarning(self.name,"Plz select a file")
      thread=td.Thread(target=process)
      thread.daemon=True
      thread.start()
   def download_file(self):
      def process():
         req_file=self.file_view.get()
         if req_file.strip()!="":
            data={
               "email":self.email,
               "file":req_file
            }
            link="http://127.0.0.1:5000/download"
            res=requests.post(link,data=data)
            with open(fr"Your Projects\files from docuit server\{req_file}","wb") as f:
                  f.write(res.content)
            messagebox.showinfo(self.name,"Downloaded")
      thread=td.Thread(target=process)
      thread.daemon=True
      thread.start()
```

```
    def search_project_initialiser(self,var):
        project_to_be_automated=var.get().strip()
        try:
            if project_to_be_automated!="":
                self.proj_title=project_to_be_automated
                thread=td.Thread(target=lambda:messagebox.showinfo("DOCu-It","Getting
connected"),daemon=True)
                thread.start()
                project=Parser(project_to_be_automated)
                project.parse()
                self.project_data_encoded=project.collection_paragraphs
                self.docx_save=project.para_to_be_docxed
                for i in range(len(self.project_data_encoded)):
                    if self.project_data_encoded[i].strip()!="":
                        break
                self.not_blank_position=i
                self.proj_title=project_to_be_automated
                messagebox.showinfo("DOCu-It","Your project data is ready.\nClick automate
to save.\nClick overview to make changes")
                self.count_paras=project.project_paras
                self.any_project=True
                for child in self.btn_frame.winfo_children():
                    child["state"]="normal"
                self.btn_frame.update()
            else:
                messagebox.showerror("DOCu-It","Please enter the project name")
        except:
            messagebox.showerror("DOCu-It"," Network issuue")
    def search_project(self):
        """for search button. Thread has been used to conduct this process parallely and the
window does not get irresponsive"""
        thread=td.Thread(target=self.search_project_initialiser,args=(self.search_var,))
        thread.daemon=True
        thread.start()
    def save_project(self):
        try:

Parser.save_docx(self.proj_title,collection_paragraphs=self.docx_save,colors=self.color
_choice)
            messagebox.showinfo(self.name,f"Saved {self.proj_title}.docx")
        except Exception as e:
            messagebox.showerror(self.name,f"Fail to save {self.proj_title}.docx")
            print(e)
        def color_change(btn,button_index):
            selected_color = colorchooser.askcolor()[1]
            btn["bg"]=selected_color
            self.color_choice[button_index]=selected_color.strip("#")
        def view_para():
            para_number=int(para_count.get())-1
            project_display.delete("1.0",END)
            project_display.insert(INSERT,self.project_data_encoded[para_number])
            project_display.update()

        def save():
            current_change=project_display.get("1.0",END)
            current_index=int(para_count.get())-1
            self.project_data_encoded[current_index]=current_change
            self.docx_save[current_index]=current_change
            messagebox.showwarning("DOCu-It","current para changed")

        modal=Toplevel(self)
        modal.title(f"DOCu-It--Overview of ({self.proj_title})")
        modal.geometry("700x288")
        modal.resizable(0,0)
        Label(modal,fg="red",text="Some symbols are meant for encoding.They will be
alright in docx.").pack()

project_display=Text(modal,width=40,height=17,relief=SUNKEN,bd=2,wrap=WORD,font
=("10"),spacing2=5)
        project_display.pack(side=LEFT,pady=3,padx=4,anchor=N)
        project_display.insert(INSERT,self.project_data_encoded[self.not_blank_position])
        options_frame=Frame(modal,width=60,height=17,relief=SUNKEN,bd=2)
        options_frame.pack(anchor=CENTER,pady=20)
        para=LabelFrame(options_frame,text="See Para")
        para.grid(row=0,padx=6,pady=(10,15))

para_count=ttk.Spinbox(para,from_=(self.not_blank_position+1),to=self.count_paras,wi
dth=5)

        para_count.set(f"{self.not_blank_position+1}")
        para_count.pack()
        para_count.bind("<Button-1>",lambda e:view_para())
        color=LabelFrame(options_frame,text="Choose Colors")
        color.grid(row=1,ipadx=3,padx=10)
        color_1=Button(color,width=2,height=1,command=lambda:color_change(color_1,0))
        color_1.grid(row=0,column=0)
        color_2=Button(color,width=2,height=1,command=lambda:color_change(color_2,1))
        color_2.grid(row=0,column=1)
        color_3=Button(color,width=2,height=1,command=lambda:color_change(color_3,2))
```

```
    color_3.grid(row=0,column=2)

color_4=Button(color,width=2,height=1,command=lambda:color_change(color_4,3))
    color_4.grid(row=1,column=0)

color_5=Button(color,width=2,height=1,command=lambda:color_change(color_5,4))
    color_5.grid(row=1,column=1)
    for i in color.winfo_children():
      i["padx"]="2"
      i["pady"]="2"
      i["bg"]="black"

    save_btn=ttk.Button(options_frame,text="SAVE",command=save)
    save_btn.grid(row=2)
```

- **project_parser.py**

```
   import requests
   from bs4 import BeautifulSoup
   from docx import Document
   from docx.shared import Pt, RGBColor
   class Parser:
     source_link="https://en.wikipedia.org/wiki/"
     def __init__(self,project_topic):
       self.project_topic = project_topic
       self.project_paras=0
       self.completed=None
       self.collection_paragraphs=None#it will contain the whole data
       self.para_to_be_docxed=[]
     def parse(self):
       response=requests.get(Parser.source_link+self.project_topic)
       if response.status_code!=200:
         print("""Some problem occured... Plz make sure the content is heading is
   correct. If correct then some connection issue"""
   )
         return """Some problem occured... Plz make sure the content is heading is
   correct. If correct then some connection issue"""
       soup=BeautifulSoup(response.content,"html.parser")
       body=soup.body
       try:
         for i in body.find_all(class_="reference"):
           i.decompose()
       except:
         pass
       parsed_pragraphs="" #to store parsed paragraphs
```

```
    number_of_para=0 #to count number of para
para_list=[]#it will contain a list of all paragrahs stored in different tuples
    saving_list=[]
    html_para=body.find_all("p")
    for para in html_para:
      para_list_pointer=""
      number_of_para+=1
      parsed_pragraphs+=para.text
      para_list_pointer+=para.text
      saving_list.append(para_list_pointer)
      para_list_pointer=para_list_pointer.encode("utf-8","ignore")
      para_list.append((para_list_pointer))
      para_list_pointer=""
    self.project_paras=number_of_para
    self.completed=parsed_pragraphs
    self.collection_paragraphs=para_list
    self.para_to_be_docxed=saving_list
  @staticmethod
  def save_docx(file,collection_paragraphs,colors=[]):
    file=file.upper()
    document = Document()
    document.add_heading(file, 0)
    color_count=0
    if len(colors)==0:
      colors=["000000"]# #000000 should be 000000 and it means black
    for i in collection_paragraphs:
      if i.strip()=="":
        continue
      else:
        color_count+=1
        if color_count>(len(colors)-1):
          color_count=0
        para=document.add_paragraph().add_run(str(i))
        para.font.color.rgb = RGBColor.from_string(colors[color_count])
        para.font.size=Pt(12)
    document.save(f'Your Projects/{file}.docx')
  @staticmethod
  def save_txt(self,file):
    with open(f"{file}.txt","w") as f:
      f.write(self.completed)
  def __repr__(self) :
return(str(self.completed.encode("utf-8","ignore")))
```