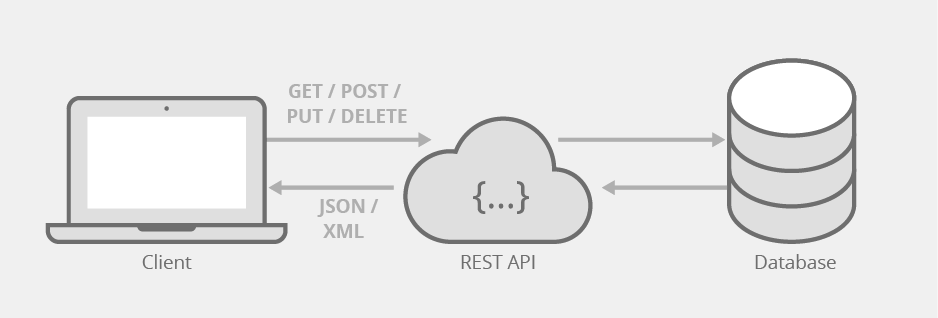
* About

So by the name it is sure that all are confused what is **DoCu\_It**  and how it will solve real world problem? Yes it is normal.

Well **DoCu\_It**  is a automation tool designed to search data and automate it for school projects. As a student I know what struggle we have to face when we have to prepare projects and atleast it takes 2-3 days to complete it. And the specialty of this tool is that it works on **Client-Server Model**. That means it will have the server running in the background to authenticate with all the happening in the GUI client. It has a **Realational Database** which is playing major role in the authentication.

**You Just Search , Arnab will automate**

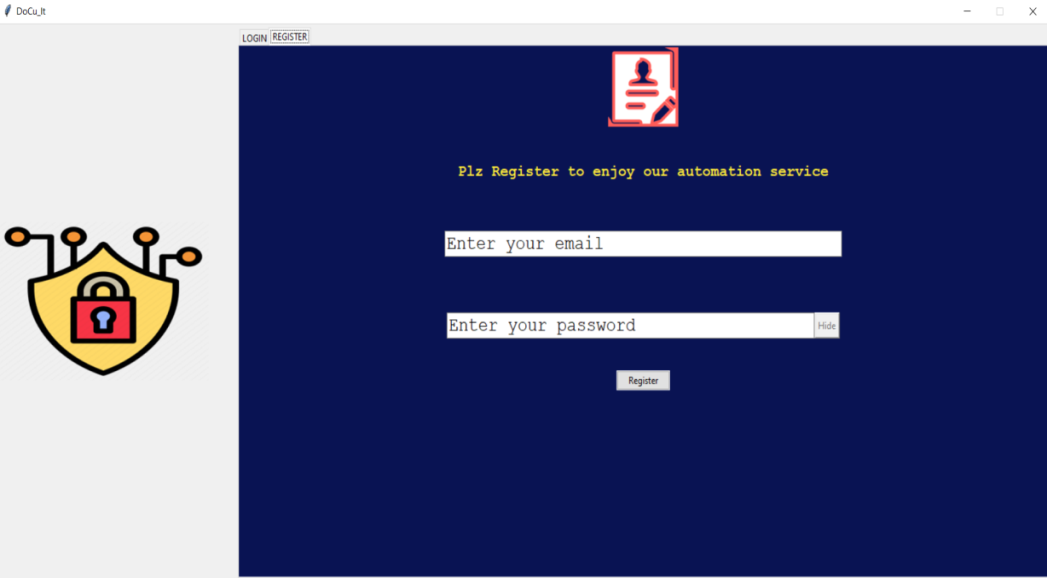
* Project Creation Process



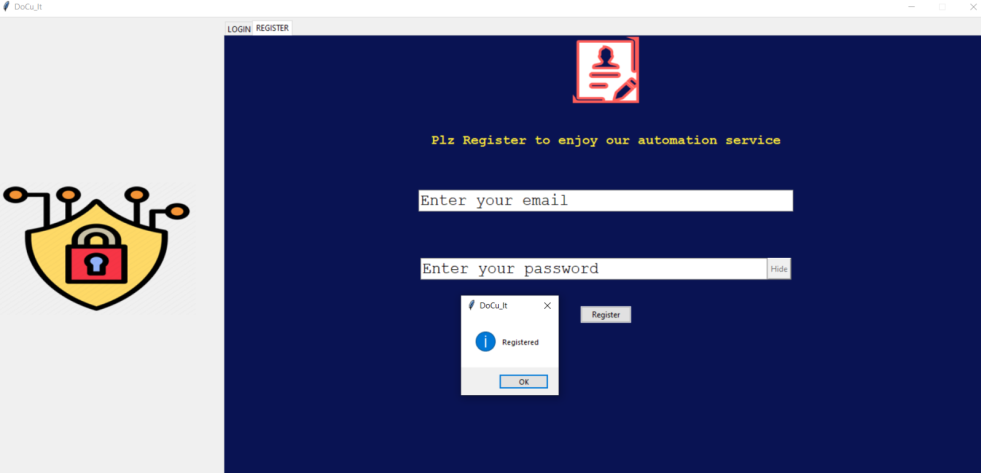
**DoCu\_It**  is running in this principle

1. First we have to start the server and GUI client.



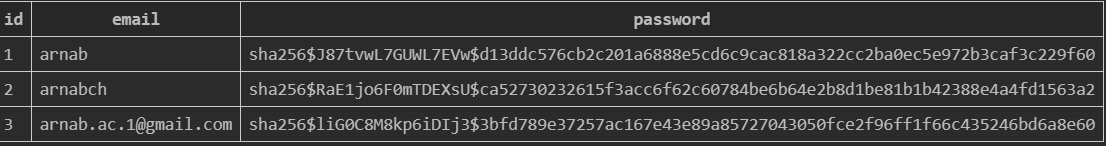


1. First of all user has to register through GUI client by an email and that must be unique.

****

I have registered using the GUI client.

Now if we see the database:



I have total 3 fields. I have shown you the database but I have no fear as the **password is encrypted with “sha256”encryption technique**.

1. Now the account has been created. So login using your credentials.

And here is the main window.

* You cannot open more than one main window at a time you may be logged .

1. Then search for the topic you want to make project. For example I want to make to project on databases. So search it.(\*Internet connection required otherwise it will warn you.)

It consists of multiple substeps:

* Search the topic. Here it is database.
* Then click overview to make changes to the data if required. Changes means to change any word or color combination.
* By default color combination is black. You can change color as well as the data. You scroll the data by the spin box. I have choosen multi color combination. Then click save button.

**99 paragraphs in just 5sec.**

* Now click automate and see the magic! File is saved in **Your Projects Folder**.

**And the project on database has been created.**

* It consists of some other great features like it is connection responsive.
* You show and hide password in entry section.
* Also it restricts the number of windows.
* Also it will not crash as it is running all the parallely.
* Data will be saved according to your login name.
* Cloud Saving Process
* Another thing which makes it unique is that you save all your project on the server database. So you can retrieve it anytime you want.
* Open the main window by logging in. Then move to the upload tab. Select the project you want to save to the server.
* Then you can download all the projects you want from your account.

**Upload**

**Download**

And the database is downloaded from server in the directory:-

DOCU\_IT\Your Projects\files from docuit server

So all the features are packed in this project.

* Features
* Works on the principle of Client-Server Model.
* Server is build on the **famous API architecture REST model**.
* Client works on multi thread which reduces the chances of application crash.
* Supports **manual editing along with the automations**.
* We can search any valid topic and it will give us the result.
* Password toggling(show and hide).
* **We can store our projects in the server which makes it a great tool for the students**.
* Responsive client means it will show you each and every event.
* It will not allow you to open more than one window which reduces the load on the CPU.
* It has a **virtual environment** which makes it separate from global environments.
* **Passwords are stored in encypted format**.
* Since server is build externally we can make multiple clients.

Here I have made a GUI client but we can also connect a mobile application or websites.

It can become a huge selling product.

* Folder Structure

All files are not included. Those are prior to this project are included.

Docu\_It:.

| client.py

| server.py

|

+---DATABASE

| DOCu\_It.db

|

+---Images

|

+---MY\_PACKAGE

| | main\_win.py

| | project\_parser.py

| | \_\_init\_\_.py

| |

| +---Images

|

+---Your Projects

| |

| \---files from docuit server

* Requirements

\*A requirements.txt is provided. In the shell just type **pip install –r requirements.txt**  to automatically download the modules.

Following modules are used:-

1. bs4
2. Flask
3. Flask-SQLAlchemy
4. Pillow
5. python-docx
6. requests
7. tkinter
8. os

* Techniques/Concepts used:-
* Full stack development :- It is a full stack project (Frontend,Database & Backend) totally based on my own idea.
* Web Scraping :- To collect data.
* HTML :- Since web scraping has been done, one have to know the HTML tree.
* Working of Internet :- HTTP methods are used to build a connection between the client and server.
* Python Packages,Modules & Paths :- I have made a MY\_PACKAGE including two modules main\_win.py & project\_parser.py . Both of them are executing the main and heavy tasks. They are used as packages to **encapsulate** them.
* Object Oriented Programming(OOP) :- Concepts of classes and objects are used in the packages made by me to give the modules a proper structure.
* Database Relationships :- It has been used to maintain the data in proper formatted way.
* Version Control & GitHub:- Since it was a huge project I had to use GIT to maintain the codebase. Since it solves a issue of student I hosted it on my online public repo **ArnabChatterjee20k** in Github.
* Automation Skills :- Last but not the least it is the technique used by me which is also the main attraction of the project.
* Code Base(Structure wise)
* **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}

#functions for buttons

def toggle\_pass(entry\_var,button):

"""to show/hide pass"""

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):

"""event handling when enter a field."""

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):

"""event handling when leave a field."""

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():

"""to restrict the number of automate window opened"""

global restrict

if messagebox.askokcancel("Quit", "Do you want to quit?"):

restrict=0

main.destroy()

data={

"email":email.get().strip(),

"password":password.get().strip()

}

# cross checking

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:

# sending data to server database

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()

}

# cross checking

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()

# print(status)

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):

'''SO that window doesnot crashes while registering or login'''

thread=threading.Thread(target=funcname)

thread.daemon=True

thread.start()

primary\_color="#091353"#dark\_blue

# configuring window

window.geometry("1500x700")

window.resizable(False,False)

# Making a layout

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

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="",textvariable=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)

# Register components

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=email\_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=None))

pass\_reg\_frame=ttk.Frame(register\_tab)

pass\_reg\_frame.pack()

password\_registry=ttk.Entry(pass\_reg\_frame,width=37,font=("Courier","18"),show="",textvariable=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\_reg))

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\_init))

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\_\_)

# DATABASE configurations

DB\_NAME="DATABASE\DOCu\_It.db"

app.config['SQLALCHEMY\_DATABASE\_URI'] = f'sqlite:///{DB\_NAME}'

app.config['SQLALCHEMY\_TRACK\_MODIFICATIONS'] = False

db=SQLAlchemy(app)

# It will work on one to many relationship

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"))

# endpoints and routes

@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="sha256"))

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():

"""to send a json object containing all filenames of a particulat email"""

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):

super().\_\_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.any\_project=True#needs to be false

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

# text var

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.primary\_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.search\_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\_project)

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\_project)

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\_var)

self.file\_directory.pack(side=LEFT)

self.browse\_file=ttk.Button(self.file\_upload\_frame,text="Browse",command=self.browse)

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=self.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

# print(self.project\_data\_encoded)

# print(project)

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 open\_modal(self):

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,width=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

#deleting unnecessary content

try:

for i in body.find\_all(class\_="reference"):

i.decompose()

except:

pass

# parsing html paragraphs to text

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")))