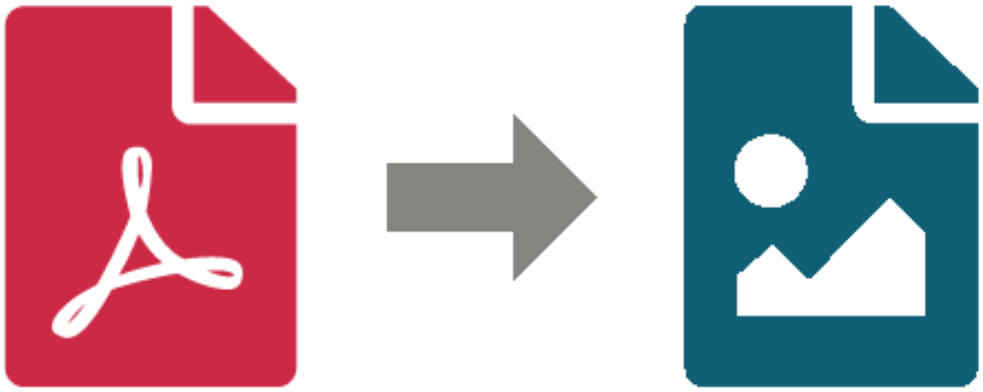


Convert pdf -> png

פרויקט גמר

למילוי חלקי של הדרישות לקבלת תואר הנדסאי

הנדסת תוכנה

בהתמחות : מחשבים, סמינר בנות אלישבע

נושא הפרויקט: חישוב מקבילי ומבוזר, המרת קובצי pdf לקובצי png

שם הסטודנטית: תמר גיגי

העבודה בוצעה בהנחיית: גב' תמר כהן

תוכן

הקדמה

מבוא ורקע לפרויקט

על מנת להבין ולהתעמק בנושא המהותי של חישוב מקבילי ומבוזר ניסיתי לחשוב על נושא שישלב את שתי הבחינות - חישוב מקבילי ומבוזר. ולכן שילבתי בפרויקט גם את הבחינה של מקביליות, ואפשרתי על ידי threads לכמה משתמשים בו זמנית להשתמש בו, ובנוסף שימוש במערכות מבוזרות, כל ידי שימוש בחיבור של socket.

לשם כך חשבתי על מערכת שממירה קובצי pdf לקובצי png בצורה מהירה ומקוונת.

מפני שהמשתמש יכול להעלות מספר לא מוגבל של קבצים להמרה, וכן קובצי pdf גדולים-(עם הרבה דפים) ההמרה לוקחת זמן רב... כי הדרך לעשות זאת זה ע"י לולאה שעוברת על כל הקבצים שהתקבלו מהמשתמש ולולאה שעוברת על מספר הדפים של הקובץ ה pdf הנוכחי וכך היא ממירה דף דף לתמונה.

וכן הקבצים שהמשתמש בחר צריכים לעבור לשרת בשביל שהם ימרו לתמונות.

לכן השתמשתי עם threads לפי כמות הקבצים שהמשתמש העלה כך שכל thread יהיה אחראי על המרה של קובץ אחד, וכן עבור כל thread שאחראי על קובץ יצרתי גם כן TREADS לפי כמות הדפים באותו קובץ כך שכל thread יטפל הדף אחר.

כדי שהמשתמש יוכל להעלות את הקבצים להמרה השתמשתי עם socket שמתקשר בין ה client (הממשק שבו המשתמש מעלה את הקבצים) לבין ה server (הקוד של ההמרה).

תהליך המחקר

כדי לכתוב את הפרויקט הייתי צריכה להתעמק ולהבין את המקביליות ואת פרוטוקולי התקשורת בפיתון. השתמשתי threads וב socket בפיתון, ובפרוטוקול TCP. התעמקתי בנושא של מקביליות ובמערכת מבוזרת.

בנוסף התעמקתי בלימוד שפת פיתון על מנת לכתוב את הקוד בצורה המדויקת ביותר.

אתגרים

האתגרים בפרויקט היו בהבנת הקשר שבין השרת והלקוח והתמודדות עם הסנכרון ביניהם. ניסיתי למצוא רעיון שיממש בצורה הטובה ביותר את הקשר שבין הלקוח והשרת ויממש את ה threads.

מדדי הצלחה למערכת

במידה והמשתמשים מצליחים להעלות את הקבצי pdf ולקבל אותם חזרה כקובצי png אז המערכת תיחשב כהצלחה. אם החיבור לא מצליח או שההמרה לא מתבצעת כראוי, אזי המערכת נכשלה.

רקע תיאורטי

על מנת לפתור את הבעיה של מקביליות בהמרה שכמה יוכלו להעלות ביחד, וכל אחד יוכל להעלות כמה קבצים שהוא רוצה, יש לאפשר פרוטוקולי רשת ומקביליות. על מנת לאפשר את זה השתמשתי ב sockets הפועלים בפרוטוקול TCP ובנוסף כדי לאפשר מקבול בהמרות השתמשתי ב thread.

השרת מורכב מ socket שמחכה לבקשות מהלקוח. ברגע שמתקבלת בקשת התחברות חדשה השרת יוצר עבורה תהליכון חדש שיטפל בחיבור הנתון.

השרת מקבל מהלקוח דרך ה socket את הקבצים כקלט ושולח אותם לפונקציה שממירה אותם לתמונות ומחזירה אותם חזרה ללוקח בתקיה להורדה.

פתרון האלגוריתם

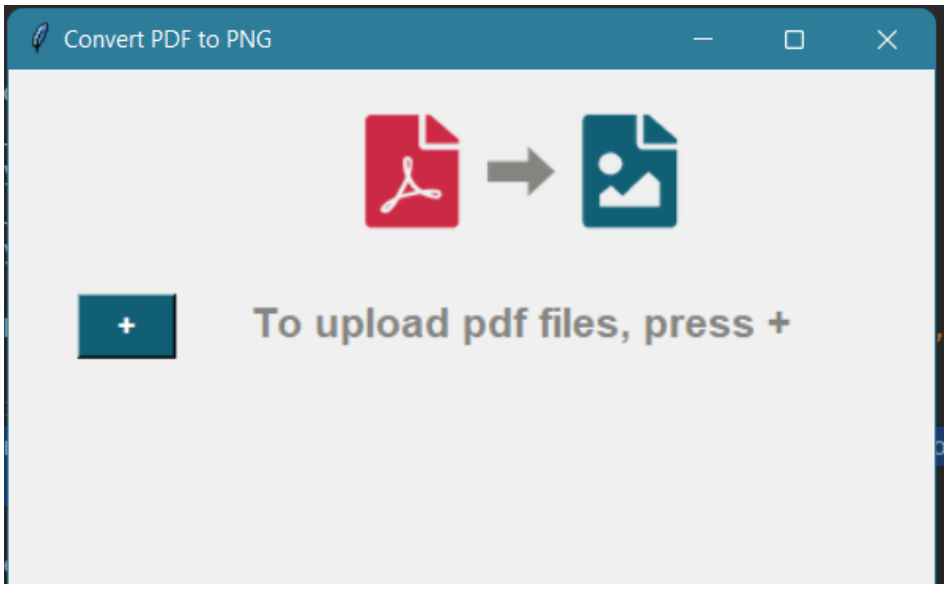
במערכת ההמרה אלקטרונית זו הלקוח יוצר חיבור עם השרת, כלומר נעשה שימוש בפרוטוקול TCP. על השרת להקצות תהליכון חדש עבור כל כניסת משתמש חדשה, כדי להשיג תכונה זו מימשתי זאת ע"י תהליכונים במקביל- כל תהליכון אינו מפריע לתהליכון אחר, לכן, סנכרנתי את התהליכונים.

תכנון ויישום

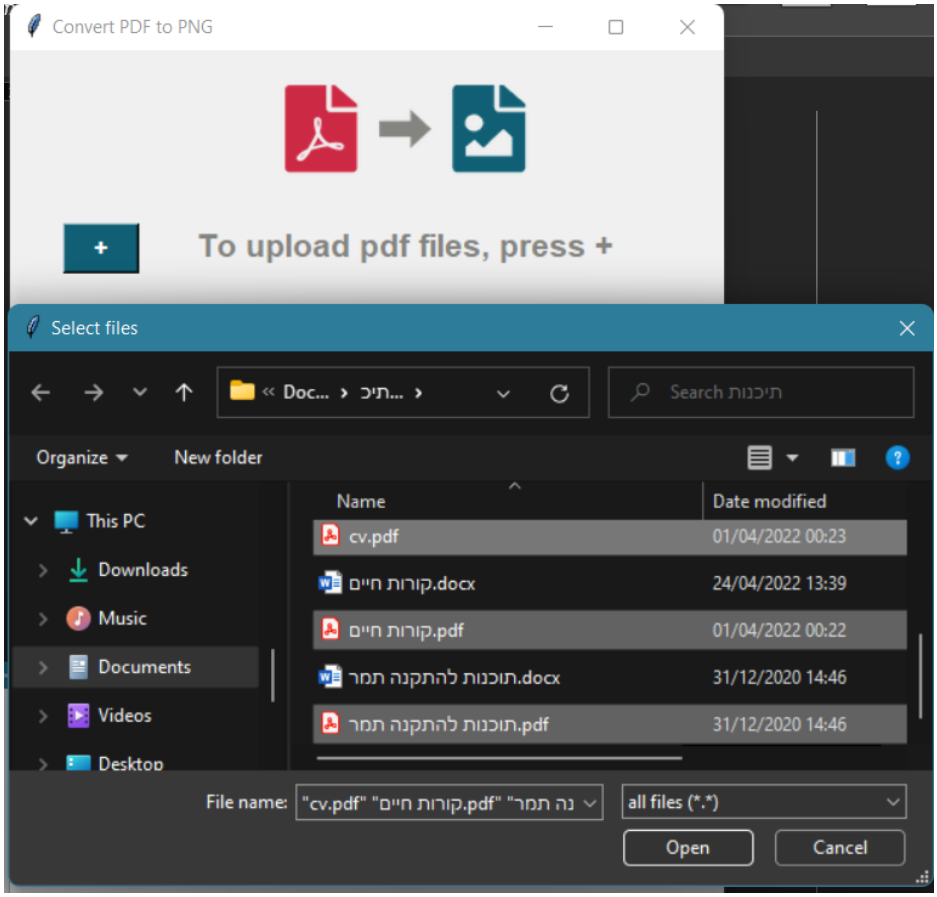
- שרת המקבל בקשות מלקוחות שונים
- יצירת התחברות עבור כל בקשת התחברות נוספת
- ניהול ההמרה בזמן אמת
- ניהול התנתקות מהמערכת וסיום תהליכון

מסכים

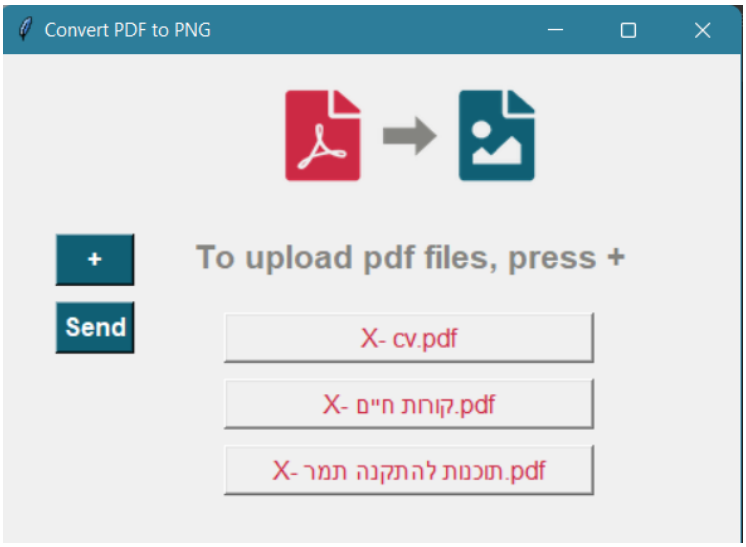
מסך פתיחה



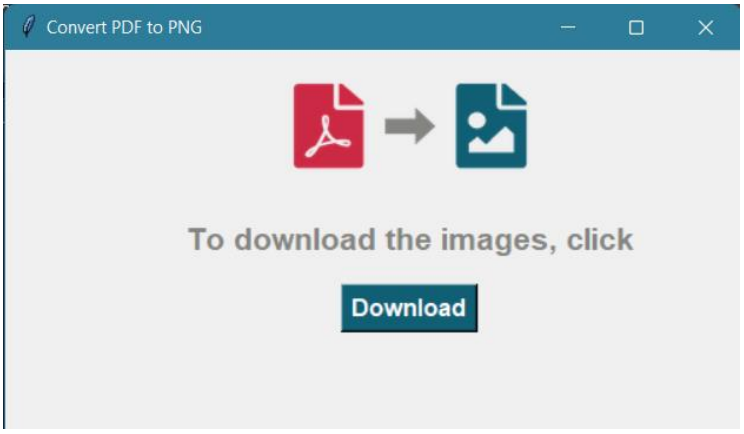
בחירת קובץ מהמחשב



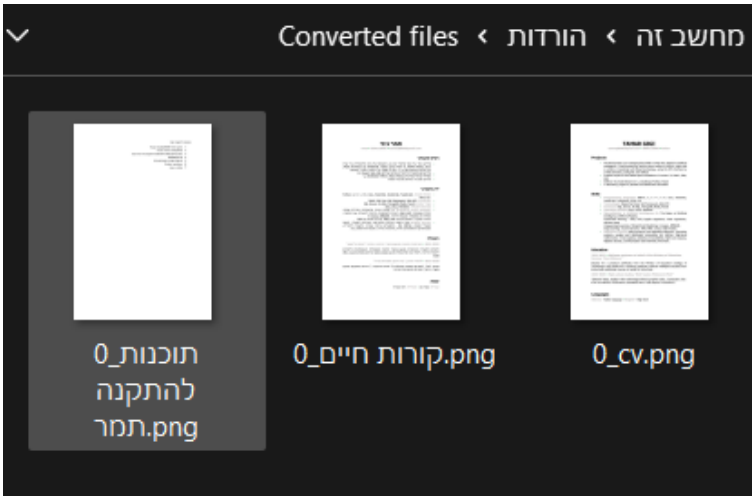
הצגת הקבצים שנבחרו



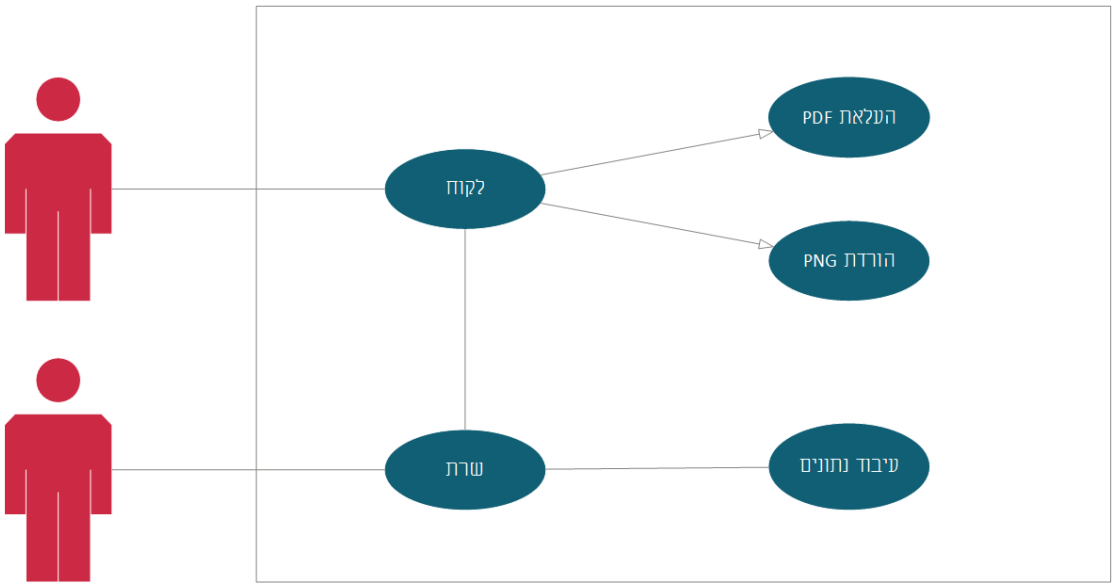
הצגת כפתור הורדת התמונות המומרות



הקבצים שהורדו להורדות של אותו משתמש



תרשים UML



קטעי קוד

Server

הקוד הראשי

```
import shutil
import socket
import pickle
from savePDF.savePDF import savePDF
from convert.convert import convert
from threading import Thread, Lock


HOST = socket.gethostname()
PORT = 640
mutex = Lock()
numberThread = 0


def createThreadsClient(connectClient):
    global numberThread
    with connectClient:
        print(f"Connected by {address}")
        data = b""
        while True:
            binPdf = connectClient.recv(1024)
            if len(binPdf) < 1024:
                data += binPdf
                break
            data += binPdf
        files_obj = pickle.loads(data)
        mutex.acquire()
        routePdf = savePDF(files_obj, numberThread)
        images = convert(routePdf, numberThread)
        print(numberThread)

shutil.rmtree(fr'savePDF/pdf_file/{numberThread}')

shutil.rmtree(fr'convert/png_file/{numberThread}')
        numberThread += 1
        image_obj = pickle.dumps(images)
        mutex.release()
connectClient.sendall(pickle.dumps(len(image_obj)))
        connectClient.sendall(image_obj)
```

```
if __name__ == "__main__":
    with socket.socket(socket.AF_INET,
socket.SOCK_STREAM) as serverSocket:

        try:
            serverSocket.bind((HOST, PORT))
        except socket.error as error:
            print(str(error))
        serverSocket.listen()
        while True:
            (clientSocket, address) =
serverSocket.accept()
            t = Thread(target=createThreadsClient,
args=(clientSocket,))
            t.start()
```

שמירת הקבצים

```
import os

from reportlab.pdfgen import canvas
import multiprocessing


def poolPDF(file, name, res):
    path = fr"{name}/{file[0]}"
    my_canvas = canvas.Canvas(path)
    my_canvas.drawString(0, 0, '')
    my_canvas.save()
    with open(path, 'wb') as pdf:
        pdf.write(file[1])
    res.append(path)


def savePDF(files, numberThread):
    path = f'savePDF/pdf_file/{numberThread}'
    os.mkdir(f'{path}')
    res = []
    with multiprocessing.Manager() as manager:
        newImage = manager.list()
        p_arr = []
        for file in files:
            p = multiprocessing.Process(target= poolPDF,
```

```
args=(file, path, newImage))
            p_arr.append(p)
        for p in p_arr:
            p.start()
        for p in p_arr:
            p.join()
        res = list(newImage)
    return res
```

המרת הקבצים

```
from pdf2image import convert_from_path
import multiprocessing
import os


def poolPage(file, path, index, page, newPDF):
    path1 =
f'{path}/{index}_{os.path.splitext(os.path.basename(file)
)[0]}.png'
    page.save(f'{path1}')
    with open(f'{path1}', 'rb') as image:
        binImage = image.read()
    newPDF.append([os.path.basename(path1), binImage])


def poolPdf(file, path, poppler_path, newPDF):
    pages = convert_from_path(pdf_path=f'{file}',
dpi=500, poppler_path=poppler_path)

    p_arr = []
    for index, page in enumerate(pages):
        p = multiprocessing.Process(target=poolPage,
args=(file, path, index, page, newPDF))
        p_arr.append(p)
    for p in p_arr:
        p.start()
    for p in p_arr:
        p.join()


def convert(files, numberThread):
    path = f'convert/png_file/{numberThread}'
    os.mkdir(f'{path}')
```

```
    poppler_path = r'convert/poppler-0.68.0_x86/poppler-
0.68.0/bin'
    ret_list = []
    with multiprocessing.Manager() as manager:
        newPDF = manager.list()
        p_arr = []
        for file in files:
            p = multiprocessing.Process(target=poolPdf,
args=(file, path, poppler_path, newPDF))
            p_arr.append(p)
        for p in p_arr:
            p.start()
        for p in p_arr:
            p.join()
        ret_list = list(newPDF)
    return ret_list
```

client

```
from tkinter.filedialog import askopenfilename,
askdirectory
import tkinter as tk
import pickle
import socket
import os
from PIL import ImageTk, Image

HOST = socket.gethostname()
PORT = 640

bColor = '#105F74'
fColor = '#FFFFFF'
tColor = '#CC2944'
hColor = '#848480'


def add_file():
    # global pathDownload
    type_files = 'pdf'
    file_path = askopenfilename(filetypes=[('all files',
'.*')], title="Select files", multiple=True)
    if not file_path:
        return
```

```
    for file in file_path:
        extension = file.split('.')
        if type_files == extension[1]:
            with open(file, 'rb') as binaryPdf:
                binPdf = binaryPdf.read()
            list_files.append(file)
            binary_files.append([os.path.basename(file),
binPdf])
    render_list_item()


def delete_file(index):
    list_files.pop(index)
    binary_files.pop(index)
    render_list_item()


def render_list_item():
    global lab, btn_add
    for widgets in frm_names_item.winfo_children():
        widgets.destroy()
    for widgets in frm_buttons.winfo_children():
        widgets.destroy()
    if len(list_files) > 0:
        btn_save = tk.Button(frm_buttons, text="Send",
width=4, command=save_file, bg=bColor, fg=fColor,
                             font=('Helvetica 12 bold'))
        btn_save.grid(row=1, column=0, sticky=tk.EW,
padx=5)
    btn_add = tk.Button(frm_buttons, text="+",
command=add_file, width=4, bg=bColor, fg=fColor,
                        font=('Helvetica 12 bold'))
    btn_add.grid(row=0, column=0, sticky=tk.EW, padx=5,
pady=10)
    for index, item in enumerate(list_files):
        extension = item.split('/')
        btn_del = tk.Button(frm_names_item, width=25,
text=f"X- {extension[-1]}", fg=tColor, font=('Helvetica
12'),
                            command=lambda key=index:
delete_file(key))
        btn_del.grid(row=index, column=10, padx=5,
pady=5)
```

```
def save_file():
    with socket.socket(socket.AF_INET,
socket.SOCK_STREAM) as serverSocket:
        serverSocket.connect((HOST, PORT))
        binPdf = pickle.dumps(binary_files)
        serverSocket.sendall(binPdf)
        size = serverSocket.recv(1024)
        recSize = pickle.loads(size)
        data = b""
        while True:
            binImage = serverSocket.recv(recSize)
            if len(binImage) <= recSize:
                data += binImage
                break
            data += binImage
    print(len(data))
    files_obj = pickle.loads(data)
    viewResponse(files_obj)


def viewResponse(imagePNG):
    for widgets in frm_names_item.winfo_children():
        widgets.destroy()
    for widgets in frm_buttons.winfo_children():
        widgets.destroy()
    lab.config(text='To download the images, click')
    btn_download = tk.Button(frm_names_item,
text=f"Download", bg=bColor, fg=fColor,
                             command=lambda obj=imagePNG:
save_image(obj), font=('Helvetica 12 bold'))
    btn_download.pack()


def save_image(images):
    pathDownload = askdirectory()
    for widgets in frm_names_item.winfo_children():
        widgets.destroy()
    labRes = tk.Label(frm_names_item, text='The download
is in progress, please wait ....', fg=bColor,
font=('Helvetica 12 bold'))
    labRes.grid()
    for imagePNG in images:
        path = fr'{pathDownload}/{imagePNG[0]}'
```

```
        with open(path, 'wb') as save:
            save.write(imagePNG[1])
    for widgets in frm_names_item.winfo_children():
        widgets.destroy()
    labRes = tk.Label(frm_names_item, text='Finished
....', fg=bColor, font=('Helvetica 12 bold'))
    labRes.grid()


if __name__ == '__main__':
    list_files = []
    binary_files = []

    # create window main
    window = tk.Tk()
    window.title("Convert PDF to PNG")
    window.rowconfigure(2, minsize=400)
    window.columnconfigure(2, minsize=70)

    # create frame body
    frm = tk.Frame(window)
    # title
    frm_title = tk.Frame(frm, height=20)
    lab = tk.Label(frm_title, text="To upload pdf files,
press +", fg=hColor, font=('Helvetica 16 bold'))
    lab.pack()
    frm_title.grid(row=0, column=1, sticky=tk.N, pady=10)
    # buttons- files selected
    frm_names_item = tk.Frame(frm, height=50)
    frm_names_item.grid(row=1, column=1, sticky=tk.NS,
pady=5)
    frm.grid(row=1, column=1)

    # create frame side
    frm_buttons = tk.Frame(window)
    # add file
    btn_add = tk.Button(frm_buttons, text="+",
command=add_file, width=4, bg=bColor, fg=fColor,
                        font=('Helvetica 12 bold'))
    btn_add.grid(row=0, column=0, sticky=tk.EW, padx=5,
pady=10)
    # button send show only when choose files
    frm_buttons.grid(row=1, column=0, sticky=tk.NS,
padx=30)
```

```
    # logo
    frm_image = tk.Frame(window)
    image = Image.open(f'logo.png')
    # f'client/logo.png'
    reSizeImage = image.resize((180, 100),
Image.ANTIALIAS)
    img = ImageTk.PhotoImage(reSizeImage)
    label = tk.Label(frm_image, image=img)
    label.pack()
    frm_image.grid(row=0, column=1)

    window.mainloop()
```

הסבר קטעי קוד

Server

הפונקציה הראשית

יצירת socket

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
serverSocket:
```

הפונקציה bind מקבלת tuple, ומקשרת בין הפורט (מפתח) הרצוי לבין ה IP כדי לקבל חיבורים ולהעביר מידע במפתח זה, אם היא לא מצליחה ליצור חיבור מודפסת השגיאה

```
try:
    serverSocket.bind((HOST, PORT))
except socket.error as error:
    print(str(error))
```

מאזין- בשביל מספר החיבורים כאשר לא רושמים מספר(מגביל את מספר החיבורים) אין הגבלה...

```
serverSocket.listen()
```

מזהה מתי יש client שמנסה להתחבר ויוצר לו thread

```
while True:
    (clientSocket, address) = serverSocket.accept()
    t = Thread(target=createThreadsClient,
args=(clientSocket,))
    t.start()
```

פונקציית יצירת ה thread

אובייקט שנותן שם לכל thread חדש- אובייקט זה הוא גלובאלי לכל ה threads

```
global numberThread
```

כל עוד החיבור ל client פתוח

```
with connectClient:
```

קריאת הנתונים שנמצאים בצינור- socket אופן הקיאה הוא 1024 ביטים כל פעם

```
data = b""
while True:
    binPdf = connectClient.recv(1024)
    if len(binPdf) < 1024:
        data += binPdf
        break
    data += binPdf
```

המרת הנתונים שהתקבלו מה client חזרה לצורה המקורית שלהם

```
files_obj = pickle.loads(data)
```

קטע קריטי- לצורך נתינת השם לכל thread

```
mutex.acquire()
routePdf = savePDF(files_obj, numberThread)
images = convert(routePdf, numberThread)
shutil.rmtree(fr'savePDF/pdf_file/{numberThread}')
shutil.rmtree(fr'convert/png_file/{numberThread}')
numberThread += 1
image_obj = pickle.dumps(images)
mutex.release()
```

החזרת הנתונים ללקוח- הקבצים המומרים לתמונות

```
connectClient.sendall(pickle.dumps(len(image_obj)))
connectClient.sendall(image_obj)
```

פונקציה ששומרת את הקובצי pdf ומחזירה את הניתובים שלהם- הסבר מפורט על הפונקציה בהמשך....

```
routePdf = savePDF(files_obj, numberThread)
```

פונקציה שמקבל את הניתובים לקובצי pdf וממירה כל קובץ ל png ומחזירה מערך שמכיל איברים של שם תמונה ותמונה- הסבר על הפונקציה מפורט בהמשך

```
images = convert(routePdf, numberThread)
```

מחיקת הקבצים מהשרת(כל הנתונים שצריכים שמורים במשתנים...)

```
shutil.rmtree(fr'savePDF/pdf_file/{numberThread}')
shutil.rmtree(fr'convert/png_file/{numberThread}')
```

קידום המשתנה שמכיל את מספר ה threads כדי שיוכל לתת שם ל thread הבא

```
numberThread += 1
```

החזרת התשובות לשרת

```
image_obj = pickle.dumps(images)
connectClient.sendall(pickle.dumps(len(image_obj)))
connectClient.sendall(image_obj)
```

הסבר מפורט על פונקציית שמירת הקבצים

בשביל לשמור את הקבצים בצורה מהירה השתמשתי ב threads כך שכל thread יהיה אחראי על שמירה של קובץ pdf אחר

יצירת תיקייה עם השם של הthread שלשם הקובצי pdf ישמרו

```
path = f'savePDF/pdf_file/{numberThread}'
os.mkdir(f'{path}')
```

מערך שיכיל את התשובה- ניתובי הקבצים

```
res = []
```

יצירת מערך מסוג של multiprocessing

```
with multiprocessing.Manager() as manager:
    newImage = manager.list()
```

מערך שיכיל את כל ה threads שנוצרים עבור כל קובץ pdf

```
p_arr = []
```

מעבר על רשימת הקבצים שהתקבלו ויצירת thread לכל קובץ- הסבר על הפונקציה poolPDF בהמשך...

```
for file in files:
    p = multiprocessing.Process(target= poolPDF,
args=(file, path, newImage))
    p_arr.append(p)
```

הפעלת ה threads והמתנה עד שכל ה threads יסיימו

```
for p in p_arr:
    p.start()
for p in p_arr:
    p.join()
```

הכנסת הניתובים שנמצאים מבערך מסוג multiprocessing למערך רגיל

```
res = list(newImage)
```

הסבר על הפונקציה poolPDF

פונקציה זו כותבת לתוך קובץ pdf את התוכן של הקובץ pdf שהמשתמש העלאה

יצירת קנווס בנתיב בשביל לכתוב לתוכו את התוכן של הקובץ pdf שהמשתמש העלאה ושמירה כ pdf

```
path = fr"{name}/{file[0]}"
my_canvas = canvas.Canvas(path)
my_canvas.drawString(0, 0, '')
my_canvas.save()
```

כתיבת נתוני הקובץ pdf לתוך הקנווס

```
with open(path, 'wb') as pdf:
    pdf.write(file[1])
```

שרשור הנתיב של הקובץ למערך

```
res.append(path)
```

הסבר מפורט על פונקציית המרת קובצי pdf ל png

פונקציה זו מקבלת את כל הקובצי pdf שהמשתמש העלה וממירה אותם לתמונות

יצירת תיקייה בשם של ה thread שלתוכה התמונות ישמרו

```
path = f'convert/png_file/{numberThread}'
os.mkdir(f'{path}')
```

משתנה שמכיל את הכלי של ההמרה בפועל

```
poppler_path = r'convert/poppler-0.68.0_x86/poppler-
0.68.0/bin'
```

מערך שיכיל את התשובה- ניתובי התמונות

```
ret_list = []
```

יצירת מערך מסוג של multiprocessing

```
with multiprocessing.Manager() as manager:
    newPDF = manager.list()
```

מערך שיכיל את כל ה threads שנוצרים עבור כל קובץ pdf

```
p_arr = []
```

מעבר על רשימת ניתובי הקבצים שנשמרו ויצירת thread לכל קובץ- הסבר על הפונקציה poolPDF בהמשך...

```
for file in files:
    p = multiprocessing.Process(target=poolPdf,
args=(file, path, poppler_path, newPDF))
    p_arr.append(p)
```

הפעלת ה threads והמתנה עד שכל ה threads יסיימו

```
for p in p_arr:
    p.start()
for p in p_arr:
    p.join()
```

הכנסת הניתובים שנמצאים מבערך מסוג multiprocessing למערך רגיל

```
ret_list = list(newPDF)
```

הסבר על הפונקציה poolPDF

פונקציה זו מקבלת קובץ pdf מחלקת אותו לדפים ושולחת לפונקציה ששומרת כל דף כתמונת png

ממיר קובץ pdf לדפים כך שאחר כך נשמור כל דף כתמונת png- מחזיר מערך של דפים

```
pages = convert_from_path(pdf_path=f'{file}', dpi=500,
poppler_path=poppler_path)
```

בשביל לשמור את הדפים עבור כל קובץ עשיתי גם כן כ threads כדי שמתי שיגיע קובץ עם הרבה דפים התהליך יהיה יותר מהר

יצירת thread עבור כל דף- הסבר מפורט על הפונקציה poolImage בהמשך...

```
p_arr = []
for index, page in enumerate(pages):
    p = multiprocessing.Process(target=poolPage,
args=(file, path, index, page, newPDF))
    p_arr.append(p)
```

הפעלת ה threads והמתנה עד שכל ה threads יסיימו

```
for p in p_arr:
    p.start()
for p in p_arr:
    p.join()
```

הסבר מפורט על הפונקציה poolInage

פונקציה זו מקבלת דף מהקובץ pdf ושומרת אותו כתמונה

```
path1 =
f'{path}/{index}_{os.path.splitext(os.path.basename(file)
)[0]}.png'
page.save(f'{path1}')
```

כדי להחזיר למשתמש את התמונה שהומרה יש לקרוא בצורה של ביטים את התמונה ולהחזיר אותה כמחרוזת ביטים

```
with open(f'{path1}', 'rb') as image:
    binImage = image.read()
```

הכנסת שם התמונה והתמונה למערך שיחזור למשתמש כדי שהוא יוכל להוריד- לשמור אצלו במחשב את התמונות

```
newPDF.append([os.path.basename(path1), binImage])
```

client

יצירת מסך- חלון ראשי

```
window = tk.Tk()
window.title("Convert PDF to PNG")
window.rowconfigure(2, minsize=400)
window.columnconfigure(2, minsize=70)
```

יצירת frame עבור הלוגו

```
frm_image = tk.Frame(window)
...
frm_image.grid(row=0, column=1)
```

פתיחת תמונת הלוגו ושינוי הגודל שלה לגודל הרצוי

```
image = Image.open(f'client/logo.png')
reSizeImage = image.resize((180, 100), Image.ANTIALIAS)
```

הגדרת התמונה מסוג imageTkinter והצגתה בתוך label

```
img = ImageTk.PhotoImage(reSizeImage)
label = tk.Label(frm_image, image=img)
label.pack()
```

יצירת frame עבור הכפתורים של הוספת קבצים וכפתור שליחה

```
frm_buttons = tk.Frame(window)
...
frm_buttons.grid(row=1, column=0, sticky=tk.NS, padx=30)
```

כפתור הוספת קבצים- מפעילה את הפונקציה add_file הסבר מפורט עליה בהמשך...

```
btn_add = tk.Button(frm_buttons, text="+",
command=add_file, width=4, bg=bColor, fg=fColor,
                    font=('Helvetica 12 bold'))
btn_add.grid(row=0, column=0, sticky=tk.EW, padx=5,
pady=10)
```

כפתור שליחה יוצג רק כאשר נבחרו קבצים

יצירת frame עבור הכותרות המתאימות והצגת הקבצים שנבחרו

```
frm = tk.Frame(window)
...
frm.grid(row=1, column=1)
```

יצירת frame ו label כבור כותרת ראשונית

```
frm_title = tk.Frame(frm, height=20)
lab = tk.Label(frm_title, text="To upload pdf files,
press +", fg=hColor, font=('Helvetica 16 bold'))
lab.pack()
frm_title.grid(row=0, column=1, sticky=tk.N, pady=10)
```

יצירת frame עבור הקבצים שנבחרו

```
frm_names_item = tk.Frame(frm, height=50)
frm_names_item.grid(row=1, column=1, sticky=tk.NS,
pady=5)
```

הסבר מפורט על פונקציית add_file

כאשר המשתמש ילחץ על הכפתור ה + הוא יגיע לפונקציה הזו שתפתח לו חלון לבחירת קבצים מהמחשב

```
type_files = 'pdf'
file_path = askopenfilename(filetypes=[('all files',
'.*')], title="Select files", multiple=True)
```

שמירת תוכן הקובצי pdf שנבחרו כביטים במערך מתאים עם השם של הקובץ

```
for file in file_path:
    extension = file.split('.')
    if type_files == extension[1]:
```

```
        with open(file, 'rb') as binaryPdf:
            binPdf = binaryPdf.read()
        list_files.append(file)
        binary_files.append([os.path.basename(file),
binPdf])
```

הפעלת פונקציית רינדור התצוגה

```
render_list_item()
```

הסבר מפורט על פונקציית רינדור התצוגה

מחיקת הנתונים שנמצאים ב frame של הכפתורים של ההוספה והשליחה ותצוגת הקבצים שהועלו לצורך הצגתם מחדש לפי הנתונים המעודכנים

```
for widgets in frm_names_item.winfo_children():
    widgets.destroy()
for widgets in frm_buttons.winfo_children():
    widgets.destroy()
```

במידה ונבחרו קבצים הצגת כפתור השליחה- כפתור זה מפעיל את פונקציית save_file שיפורט עליה בהמשך...

```
if len(list_files) > 0:
    btn_save = tk.Button(frm_buttons, text="Send",
width=4, command=save_file, bg=bColor, fg=fColor,
                         font=('Helvetica 12 bold'))
    btn_save.grid(row=1, column=0, sticky=tk.EW, padx=5)
```

הצגה מחדש של כפתור הוספת קבצים

```
btn_add = tk.Button(frm_buttons, text="+",
command=add_file, width=4, bg=bColor, fg=fColor,
                    font=('Helvetica 12 bold'))
btn_add.grid(row=0, column=0, sticky=tk.EW, padx=5,
pady=10)
```

הצגת שמות כל הקובצי pdf שנבחרו ككפתורים כך שבלחיצה עליהם אפשר להסירם

```
for index, item in enumerate(list_files):
    extension = item.split('/')
    btn_del = tk.Button(frm_names_item, width=25,
text=f"X- {extension[-1]}", fg=tColor, font=('Helvetica
12'),
                        command=lambda key=index:
delete_file(key))
    btn_del.grid(row=index, column=10, padx=5, pady=5)
```

הסבר מפורט על פונקציית delete_file

פונקציה זו מקבלת אינדקס מסוים ומשירה אותו מרשימת הקבצים שנבחרו ושולחת שוב לפונקציית רינדור התצוגה כדי לעדכן את השינויים

```
list_files.pop(index)
binary_files.pop(index)
render_list_item()
```

הסבר מפורט על פונקציית save_file

פונקציה זו אחראית על שליחת הקבצים ל server לצורך המרתם לתמונות וקבלת התשובה- התמונות המומרות

יצירת socket -חיבור ל server

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as
serverSocket:
    serverSocket.connect((HOST, PORT))
```

המרת מערך הקבצים שנבחרו לביטים מפני שהעברת הנתונים דרך ה socket יכולה להיות רק באמצעות ביטים

```
binPdf = pickle.dumps(binary_files)
serverSocket.sendall(binPdf)
```

קבלת התשובה מה server התמונות המומרות

```
size = serverSocket.recv(1024)
recSize = pickle.loads(size)
data = b""
while True:
    binImage = serverSocket.recv(recSize)
    if len(binImage) <= recSize:
        data += binImage
        break
    data += binImage
```

המרת מחרוזת הביטים שהתקבלה מה server לצורת הנתונים במקורית

```
files_obj = pickle.loads(data)
```

שליחה לפונקציית viewResponse

```
viewResponse(files_obj)
```

הסבר מפורת על פונקציית viewResponse

פונקציה זו אחראית על הפיכת תצוגת המסך למסך הורדה שבה המשתמש יוכל ללחוץ על הכפתור להורדת התמונות

מחיקת כל הנתונים מה frame של כפתורי הוספה ושליחה וכפתורי תצוגת הקבצים שנבחרו

```
for widgets in frm_names_item.winfo_children():
    widgets.destroy()
```

```
for widgets in frm_buttons.winfo_children():
    widgets.destroy()
```

שינוי שם הכותרת

```
lab.config(text='To download the images, click')
```

יצירת כפתור ההורדה- בלחיצה עליו הוא מפעיל את פונקצית save_image

```
btn_download = tk.Button(frm_names_item,
text=f"Download", bg=bColor, fg=fColor,
                          command=lambda obj=imagePNG:
save_image(obj), font=('Helvetica 12 bold'))
btn_download.pack()
```

הסבר מפורט על פונקצית save_image

פונקציה זו אחראית על קבלת מערך התמונות שהתקבל מהשרת ולשמור אותם בתיקייה שהמשתמש יבחר

בחירת התיקייה

```
pathDownload = askdirectory()
```

לולאה שמוחקת את הכפתור download ומציגה הודעה מתאימה על כך שהקבצים בהורדה

```
for widgets in frm_names_item.winfo_children():
    widgets.destroy()
labRes = tk.Label(frm_names_item, text='The download is
in progress, please wait ....', fg=bColor,
font=('Helvetica 12 bold'))
labRes.grid()
```

לולאה שעוברת על מערך התמונות שהתקבל מהשרת ויוצרת עבור כל תמונה קובץ png וכותבת לתוכו בביטים לפי הנתונים של התמונה שהגיעה מה server

```
for imagePNG in images:
    path = fr'{pathDownload}/Converted
files/{imagePNG[0]}'
    with open(path, 'wb') as save:
        save.write(imagePNG[1])
```

שינוי ההודעה כאשר ההורדה הסתימה

```
labRes.config(text='Finished ....')
```