Team ID	PNT2022TMID02158
Project Name	A Novel Method for Handwritten Digit Recognition System

Build python Part-2:

MAIN.(PYTHON):

```
from flask import Flask, render_template, request,redirect,session, url_for
from flask_mail import Mail, Message
from itsdangerous import URLSafeTimedSerializer, SignatureExpired
import mysql.connector
import os
from flask_mysqldb import MySQL
from recognize import recognize
import requests
from io import BytesIO
from werkzeug.utils import secure_filename
app = Flask(_name_)
app.secret_key=os.urandom(24)
app.config['MYSQL_HOST'] = 'localhost'
app.config['MYSQL_USER'] = 'root'
app.config['MYSQL_PASSWORD'] = ''
app.config['MYSQL_DB'] = 'digit_recognition'
mysql = MySQL(app)
@app.route('/')
def index():
    return render_template('index.html')
@app.route('/login')
def login():
    return render_template('login.html')


@app.route('/register/')
```

```
def about():
    return render_template('form.html')
@app.route('/home')
def home():
    if 'email' in session:
        return render_template('home.html')
    else:
        return redirect('/')
@app.route('/login_validation',methods=['POST'])
def login_validation():
    if request.method == "POST":
        email=request.form.get('email')
        password=request.form.get('password')
        error = None
        if mysql:
            print("Connection Successful!")
            cursor = mysql.connection.cursor()
            cursor.execute("""SELECT * FROM `users` where `Email` LIKE '{}'
"""".format(email))
            users = cursor.fetchall()
            cursor.close()
            cursor1 = mysql.connection.cursor()
            cursor1.execute("""SELECT * FROM `users` where `Email` LIKE '{}' and
`Password` LIKE '{}'""".format(email, password))
            users1 = cursor1.fetchall()
            cursor1.close()
        else:
            print("Connection Failed!")
        if len(users)>0:
            if len(users1)>0:
                session['email'] = users[0][1]
                return redirect('/home')
            else:
                error = "Wrong password"
        else:
```

```
            error = "Email not available"
    return render_template('login.html',error=error)
@app.route('/add_user',methods=['POST'])
def add_user():
    username=request.form.get('username')
    email = request.form.get('email')
    password = request.form.get('password')
    phone = request.form.get('phone')
    gender = request.form.get('gender')
    if mysql:
        print("Connection Successful!")
        cursor = mysql.connection.cursor()
        cursor.execute(
            """INSERT INTO `users` (`FullName`,`Email`,`Password`,`PhoneNo`,`Gender`)
VALUES ('{}','{}','{}','{}','{}')""".format(username,email, password,phone,gender))
        mysql.connection.commit()
        cursor.close()
    else:
        print("Connection Failed!")
    return redirect('/login')
@app.route('/logout')
def logout():
    return redirect('/')
@app.route('/predictpage',methods=['POST'])
def predictpage():
    return render_template('prediction.html')
@app.route('/submit',methods=['POST'])
def submit():
    if request.method == 'POST':
        # Upload file flask
        uploaded_img = request.files['image']
        # Upload file to database (defined uploaded folder in static path)
        uploaded_img.save('./static/data/1.jpg')
        # Storing uploaded file path in flask session
        session['uploaded_img_file_path'] = "./static/data/1.jpg"
        return render_template('prediction.html')
@app.route('/prediction',methods=('POST', "GET"))
```

```
def predict():
        # Retrieving uploaded file path from session
        img_file_path = session.get('uploaded_img_file_path', None)
        best, img1 = recognize(img_file_path)
        return render_template("prediction.html", best=best, img_name=img1)
if__name_=="_main_":
    app.run(debug=True)
```

RECOGNIZER(PYTHON):

```
import os
import random
import string
from pathlib import Path
import numpy as np
from tensorflow.keras.models import load_model
from PIL import Image, ImageOps
import cv2
def recognize(image: bytes) -> int:
    """
    Predicts the digit in the image.


    Args:
        image (bytes): The image data.
    Returns:
        tuple: The best prediction, other predictions and file name
    """
    model=load_model(Path("./model/digit.h5"))
    image = cv2.imread(image)
    grey = cv2.cvtColor(image.copy(), cv2.COLOR_BGR2GRAY)
    ret, thresh = cv2.threshold(grey.copy(), 75, 255, cv2.THRESH_BINARY_INV)
    contours, _ = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    preprocessed_digits = []
    for c in contours:
        x,y,w,h = cv2.boundingRect(c)
        cv2.rectangle(image, (x,y), (x+w, y+h), color=(0, 255, 0), thickness=2)
```

```
    digit = thresh[y:y+h, x:x+w]
    resized_digit = cv2.resize(digit, (18,18))
    padded_digit = np.pad(resized_digit, ((5,5),(5,5)), "constant", constant_values=0)
    preprocessed_digits.append(padded_digit)
for digit in preprocessed_digits:
    prediction = model.predict(digit.reshape(1, 28, 28, 1))
    best= np.argmax(prediction)
return best, "1.jpg"
```