Backend/server.py

```
import shutil
from flask import Flask, request, jsonify
from flask_cors import CORS
import firebase_admin
from firebase_admin import credentials, firestore, auth, storage
import asyncio
import random
import re
import os
import bcrypt
from werkzeug.utils import secure_filename
import cv2
import json
import numpy as np
import tensorflow as tf
from deepface import DeepFace
from pprint import pprint
import dlib
from imutils import face_utils
from scipy.spatial import distance as dist
import azure.cognitiveservices.speech as speechsdk
from moviepy.editor import VideoFileClip
from scipy.spatial.distance import euclidean
from fastdtw import fastdtw
import librosa


app = Flask(__name__)
cors = CORS(app, resources={r"/*": {"origins": "http://emac75.ddns.net:5555/"}})

# Inicialização da base de dados Firebase
cred = credentials.Certificate('ual-tech-firebase-adminsdk-p5fbv-f1103dd8a0.json')
firebase_admin.initialize_app(cred)
db = firestore.client()

# Inicializa o detector de pontos faciais do Dlib
detector = dlib.get_frontal_face_detector()
predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")

# Inicializa a configuração do Azure Speech
speech_key, service_region = "493fa347013244cba16555bb69a9a157", "westeurope"
language = "pt-PT"
speech_config = speechsdk.SpeechConfig(subscription=speech_key,
region=service_region)
speech_config.speech_recognition_language = language

# Função para validar e-mails
def email_valido(email):
    pattern = r'^\S+@\S+\.\S+VSCODE_PRINT_CONTENT#x27;
    return re.match(pattern, email)

@app.route("/")
```

```
285 def helloworld():
286     return "SERVER UP!"
287 
288 # Endpoint para registar um novo utilizador
289 @app.route('/register', methods=['POST'])
290 def register():
291     data = request.json
292     if not all(data[field] for field in ['username', 'email', 'password']):
293         return jsonify({'success': False, 'message': 'Campos obrigatórios
    ausentes'}), 400
294     if not email_valido(data['email']):
295         return jsonify({'success': False, 'message': 'Formato de e-mail inválido'}),
    400
296     print('-> REGISTER: Registo de utilizador: ' + data['username'])
297     try:
298         existing_user = db.collection('Users').document(data['email']).get()
299         if existing_user.exists:
300             return jsonify({'success': False, 'message': 'E-mail já registado'}),
    400
301         hashed_password = bcrypt.hashpw(data['password'].encode('utf-8'),
    bcrypt.gensalt())
302         hashed_password = hashed_password.decode('utf-8')
303         user_info = {
304             'username': data['username'],
305             'email': data['email'],
306             'password': hashed_password,
307         }
308         db.collection('Users').document(user_info['email']).set(user_info)
309         return jsonify({'success': True, 'message': 'Utilizador registado com
    sucesso!'}), 201
310     except Exception as e:
311         return jsonify({'success': False, 'message': str(e)}), 400
312 
313 # Endpoint para login de utilizadores
314 @app.route('/login', methods=['POST'])
315 def login():
316     data = request.json
317     if not all(data[field] for field in ['email', 'password']):
318         print("-> LOGIN: Campos obrigatórios ausentes")
319         return jsonify({'success': False, 'message': 'Campos obrigatórios
    ausentes'}), 400
320     try:
321         doc_ref = db.collection("Users").document(data['email'])
322         doc = doc_ref.get()
323         if doc.exists:
324             user_data = doc.to_dict()
325             hashed_password = user_data.get('password')
326             if bcrypt.checkpw(data['password'].encode('utf-8'),
    hashed_password.encode('utf-8')):
327                 print("-> LOGIN: Login bem-sucedido!")
328                 return jsonify({'success': True, 'message': 'Login bem-sucedido!'}),
    200
329             else:
330                 print("-> LOGIN: Senha incorreta!")
```

```
331                 return jsonify({'success': False, 'message': 'Senha incorreta!'}),
401
332         else:
333             print("-> LOGIN: Utilizador não encontrado!")
334             return jsonify({'success': False, 'message': 'Utilizador não
encontrado!'}), 404
335     except Exception as e:
336         print("-> LOGIN: Erro ao fazer login:", e)
337         return jsonify({'success': False, 'message': str(e)}), 400
338
339 # Endpoint para obter informações de um utilizador
340 @app.route('/userinfo', methods=['POST'])
341 def userinfo():
342     data = request.json
343     try:
344         doc_ref = db.collection("Users").document(data['email'])
345         doc = doc_ref.get()
346         if doc.exists:
347             doc_data = doc.to_dict()
348             return jsonify({'success': True, 'message': 'Obtidos dados de
Utilizador!', 'data': doc_data}), 200
349         else:
350             return jsonify({'success': False, 'message': 'Não exixtem dados de
Utilizador!'}), 404
351     except Exception as e:
352         return jsonify({'success': False, 'message': str(e)}), 400
353
354 # Endpoint para guardar obtidos pela câmera
355 @app.route('/camera_save', methods=['POST'])
356 def camera_save():
357     if 'email' not in request.form:
358         print("-> CAMERA_SAVE: Erro Email")
359         return jsonify({'error': 'Email não fornecido'}), 400
360     email = request.form['email']
361     if 'frase' not in request.form:
362         print("-> CAMERA_SAVE: Erro Frase")
363         return jsonify({'error': 'Frase não fornecida'}), 400
364     frase = request.form['frase']
365     if 'video' not in request.files:
366         print("-> CAMERA_SAVE: Erro Video")
367         return jsonify({'error': 'Video não fornecido'}), 400
368     file = request.files['video']
369     if file.filename == '':
370         return jsonify({'error': 'No selected file'}), 400
371     filename = secure_filename(file.filename)
372     os.makedirs(os.path.join(app.instance_path, email), exist_ok=True)
373     file.save(os.path.join(app.instance_path, email, filename))
374     return jsonify({'success': True}), 200
375
376 # Endpoint para registar um utilizador
377 @app.route('/register_user', methods=['POST'])
378 async def register_user():
379     data = request.get_json()
380     if not data or 'email' not in data:
```

```
381         print("-> REGISTER_USER: Erro Email")
382         return jsonify({'error': 'Email não fornecido'}), 400
383     try:
384         response_face = True
385         response_phrase = True
386         response_voice = True
387         response_liveness = True
388         response_face, response_phrase, response_voice, response_liveness = await asyncio.gather(face_register(data), recognize_speech(data), voice_register(data), liveness(data))
389         if ((response_face) and (response_phrase) and (response_voice) and (response_liveness)):
390             print('-> REGISTER_USER: Face:'+str(response_face)+' Frase:'+str(response_phrase)+' Liveness:'+str(response_liveness))
391             message = 'Utilizador Registado!'
392             success = True
393         else:
394             print('-> REGISTER_USER: Face:'+str(response_face)+' Frase:'+str(response_phrase)+' Liveness:'+str(response_liveness))
395             message = 'Utilizador Não Registado!'
396             success = False
397             eliminar_cameradata(data['email'])
398         delete_path = os.path.join(app.instance_path, f"{data['email']}/")
399         delete_video(delete_path)
400         return jsonify({'success': success, 'message': message, 'face': response_face, 'phrase': response_phrase, 'voice': response_voice, 'liveness': response_liveness}), 200
401     except Exception as e:
402         print(str(e))
403         return jsonify({'error': str(e)}), 500
404 
405 # Endpoint para validar um utilizador
406 @app.route('/validate_user', methods=['POST'])
407 async def validate_user():
408     data = request.get_json()
409     if not data or 'email' not in data:
410         print("-> VALIDATE_USER: Erro Email")
411         return jsonify({'error': 'Email não fornecido'}), 400
412     try:
413         response_face = True
414         response_phrase = True
415         response_voice = True
416         response_liveness = True
417         response_face, response_phrase, response_voice, response_liveness = await asyncio.gather(face_detect(data), recognize_speech(data), recognize_voice(data), liveness(data))
418         if ((response_face) and (response_phrase) and (response_voice) and (response_liveness)):
419             print('-> VALIDATE_USER: Face:'+str(response_face)+' Frase:'+str(response_phrase)+' Voz:'+str(response_voice)+' Liveness:'+str(response_liveness))
420             message = 'Utilizador Validado!'
421             success = True
422         else:
```

```
423             print('-> VALIDATE_USER: Face:'+str(response_face)+'
    Frase:'+str(response_phrase)+' Voz:'+str(response_voice)+'
    Liveness:'+str(response_liveness))
424             message = 'Utilizador Não Validado!'
425             success = False
426         delete_path = os.path.join(app.instance_path, f"{data['email']}/")
427         delete_video(delete_path)
428         return jsonify({'success': success, 'message': message, 'face':
    response_face, 'phrase': response_phrase, 'voice': response_voice, 'liveness':
    response_liveness}), 200
429     except Exception as e:
430         print(str(e))
431         return jsonify({'error': str(e)}), 500
432 
433 # Endpoint para obter uma frase aletória da base de dados
434 @app.route('/phrase', methods=['POST'])
435 def phrase():
436     try:
437         phrase_id = random.randint(1, 100)
438         doc_ref = db.collection("Frases").document('pt')
439         doc = doc_ref.get()
440         if doc.exists:
441             phrase_data = doc.to_dict()
442             if phrase_data[str(phrase_id)]:
443                 print('-> PHRASE: '+phrase_data[str(phrase_id)])
444                 return jsonify({'success': True, 'message': 'Obtidos dados de
    frases!', 'frase': phrase_data[str(phrase_id)]}), 200
445             else:
446                 print(f"-> PHRASE: Frase com ID {phrase_id} não encontrada no
    documento.")
447                 return jsonify({'success': False, 'message': ''}), 404
448         else:
449             print('-> PHRASE: Erro a obter Frase: Documento não existe.')
450     except Exception as e:
451         print("-> PHRASE: Erro a obter Frase!:", e)
452         return jsonify({'success': False, 'message': str(e)}), 400
453 
454 # Eliminar videos guardados no servidor
455 def delete_video(video_folder):
456     for filename in os.listdir(video_folder):
457         file_path = os.path.join(video_folder, filename)
458         try:
459             if os.path.isfile(file_path) or os.path.islink(file_path):
460                 os.unlink(file_path)
461             elif os.path.isdir(file_path):
462                 shutil.rmtree(file_path)
463         except Exception as e:
464             print(f'-> DELETE_VIDEO: Failed to delete {file_path}. Reason: {e}')
465 
466 # Registar o rosto do utilizador
467 async def face_register(data):
468     email = data['email']
469     videofile = data['videofile']
470     known_encodings = []
```

```
    face_confidence_counter = 0
    face_confidence_total = 0
    video_path = os.path.join(app.instance_path, f"{email}/{videofile}")
    if not os.path.exists(video_path):
        print("-> FACE_REGISTER: Erro Video")
        return(False)
    video_capture = cv2.VideoCapture(video_path)
    if not video_capture.isOpened():
        print("-> FACE_REGISTER: Erro Video")
        return(False)
    frame_count = int(video_capture.get(cv2.CAP_PROP_FRAME_COUNT))
    fps = int(video_capture.get(cv2.CAP_PROP_FPS))
    duration = frame_count / fps
    interval = duration / 10
    for i in range(10):
        video_capture.set(cv2.CAP_PROP_POS_MSEC, i * interval * 1000)
        success, frame = video_capture.read()
        faces = DeepFace.extract_faces(img_path=frame, detector_backend='opencv', enforce_detection=False)
        detected_face = faces[0]["face"]  # Extract the face image
        face_confidence = faces[0]['confidence']
        output_file = f"{app.instance_path}/{email}/frame_reg_{i}.jpg"
        detected_face_write = detected_face * 255
        cv2.imwrite(output_file, detected_face_write[:, :, ::-1])
        if (face_confidence >= 0.90):
            face_confidence_counter = face_confidence_counter + 1
            face_confidence_total = face_confidence_total + face_confidence
            face_embedding = DeepFace.represent(img_path=detected_face_write, model_name='Facenet512', enforce_detection=False)[0]["embedding"]
            known_encodings.append(face_embedding)
            if len(known_encodings) >= 5:
                json_file_path = f"registered_user_encodings_{email}.json"
                with open(json_file_path, "w") as f:
                    json.dump(known_encodings, f)
                with open(json_file_path, "r") as j:
                    lista = json.load(j)
                    doc_ref = db.collection('Users').document(email)
                    doc_ref.update({'cameradata': json.dumps(lista)})
                video_capture.release()
                print(f'-> FACE_REGISTER: Reconhecimento: {face_confidence*100:.0f}%')
                return(True)
    return(False)

# Detetar o rosto do utilizador
async def face_detect(data):
    try:
        # Definir variaveis
        count_auth = 0
        count_similarity = 0
        frame_attempts = 0
        face_result = 0
        face_count = 0
        registered_encodings = []
```

```
522         email = data['email']
523         videofile = data['videofile']
524         video_path = os.path.join(app.instance_path, f"{email}/{videofile}")
525         if not os.path.exists(video_path):
526             print("-> FACE_DETECT: Erro Video")
527             return(False)
528         video_capture = cv2.VideoCapture(video_path)
529         if not video_capture.isOpened():
530             print("-> FACE_DETECT: Erro Video")
531             return(False)
532         try:
533             doc_ref = db.collection('Users').document(email)
534             doc = doc_ref.get()
535             if doc.exists:
536                 registered_encodings.append(doc.to_dict().get("cameradata", None))
537                 registered_encodings = json.loads(registered_encodings[0])
538                 registered_encodings = [np.array(enc) for enc in
registered_encodings]
539             else:
540                 print("-> FACE_DETECT: Registo não encontrado na Base de Dados.")
541                 return(False)
542         except FileNotFoundError:
543             print("-> FACE_DETECT: Nenhum Utilizador registado encontrado. Por
favor, registe um Utilizador primeiro.")
544             return(False)
545         video_capture = cv2.VideoCapture(video_path)
546         if not video_capture.isOpened():
547             print("-> FACE_DETECT: Erro a abrir Video.")
548             return(False)
549         frame_count = int(video_capture.get(cv2.CAP_PROP_FRAME_COUNT))
550         fps = int(video_capture.get(cv2.CAP_PROP_FPS))
551         duration = frame_count / fps
552         interval = duration / 10
553         for i in range(10):
554             video_capture.set(cv2.CAP_PROP_POS_MSEC, i * interval * 1000)
555             success, frame = video_capture.read()
556             faces = DeepFace.extract_faces(img_path=frame,
detector_backend='opencv', enforce_detection=False)
557             face_confidence = faces[0]['confidence']
558             detected_face = faces[0]['face']  # Extract the face image
559             output_file = f"{app.instance_path}/{email}/frame_val_{i}.jpg"
560             detected_face_write = detected_face * 255
561             cv2.imwrite(output_file, detected_face_write[:, :, ::-1])
562             if (face_confidence >= 0.90):
563                 frame_attempts += 1
564                 face_embedding = DeepFace.represent(img_path=detected_face_write,
model_name='Facenet512', enforce_detection=False)[0]["embedding"]
565                 matches = [np.linalg.norm(face_embedding - reg_enc) for reg_enc in
registered_encodings]
566                 if(matches):
567                     count_auth += 1
568                 for recorded_face in registered_encodings:
569                     for j in range(5):
```

```
                        dot_product = np.dot(face_embedding, registered_encodings[j])
                        norm1 = np.linalg.norm(face_embedding)
                        norm2 = np.linalg.norm(recorded_face)
                        similarity = dot_product / (norm1 * norm2)
                        face_result = face_result + similarity
                        face_count = face_count + 1
                        if (similarity) > 0.5:
                            count_similarity += 1
        video_capture.release()
        if count_auth == 0:
            similarity_percent = 0
        else:
            similarity_percent =  round((count_similarity / (count_auth * 25)) * 100, 0)
        print(f'-> FACE_DETECT: # Autenticações:{str(count_auth)} / # Similaridades: {count_similarity}')
        if ((count_auth >= 5) and ((count_similarity/count_auth) > 15)):
            message = f'-> FACE_DETECT: Utilizador autorizado! Fotos: {str(count_auth)} Similaridades:{count_similarity} {similarity_percent:.0f}%'
            print(message)
            return(True)
        else:
            message = f'-> FACE_DETECT: Utilizador não autorizado! Fotos: {str(count_auth)} Similaridades:{count_similarity} {similarity_percent:.0f}%'
            print(message)
            return(False)
    except Exception as e:
        return(False)

# Fazer a prova de vida do utilizador
async def liveness(data):
    # Data
    email = data['email']
    videofile = data['videofile']
    video_path = os.path.join(app.instance_path, f"{email}/{videofile}")
    video = cv2.VideoCapture(video_path)
    # Limites para olhos e boca
    MAR_THRESH = 0.30
    MAR_CONSEC_FRAMES = 1
    # Movimento da cabeça
    prev_frame = None
    motion_threshold = 0.02
    motion_return = False
    # Contadores de frames
    mouth_counter = 0
    mouth_return = False
    # Índices dos marcos dos olhos e boca
    (mStart, mEnd) = face_utils.FACIAL_LANDMARKS_IDXS["mouth"]
    # Processar
    while video.isOpened():
        ret, frame = video.read()
        if not ret:
            break
```

```
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        rects = detector(gray, 0)
        for rect in rects:
            shape = predictor(gray, rect)
            shape = face_utils.shape_to_np(shape)
            # Boca
            mouth = shape[mStart:mEnd]
            mar = mouth_aspect_ratio(mouth)
            if mar > MAR_THRESH:
                mouth_counter += 1
                mouth_return = True
            else:
                if mouth_counter >= MAR_CONSEC_FRAMES:
                    mouth_counter += 1
                    mouth_return = True
        # Analise de movimento da cabeca
        if prev_frame is not None:
            frame_delta = cv2.absdiff(prev_frame, gray)
            thresh = cv2.threshold(frame_delta, 25, 255, cv2.THRESH_BINARY)[1]
            motion = np.sum(thresh) / (frame.shape[0] * frame.shape[1])
            # Considera movimento se a mudança for maior que o limite
            if motion > motion_threshold:
                motion_return = True
        prev_frame = gray.copy()
    print('-> LIVENESS: Boca movimentos: '+str(mouth_counter))
    print('-> LIVENESS: Motion valor: '+str(motion))
    # Liberta video
    video.release()
    if ((motion_return) and (mouth_return)):
        return(True)
    else:
        return(False)

# Reconhecer a fala do utilizador
async def recognize_speech(data):
    try:
        email = data['email']
        frase = data['frase']
        frase_rec = ''
        videofile = data['videofile']
        video_path = os.path.join(app.instance_path, f"{email}/{videofile}")
        try:
            video_clip = VideoFileClip(video_path)
            audio_path = os.path.join(app.instance_path, f"{email}/audio.wav")
            video_clip.audio.write_audiofile(audio_path, codec='pcm_s16le')  # 
Salvar como WAV
        except Exception as e:
            print(f"-> RECOGNIZE_SPEECH: Erro ao extrair áudio: {e}")
            return jsonify({'error': 'Erro ao extrair áudio do vídeo'}), 500
        audio_config = speechsdk.audio.AudioConfig(filename=audio_path)
        speech_recognizer = speechsdk.SpeechRecognizer(speech_config=speech_config, 
audio_config=audio_config)
        result = speech_recognizer.recognize_once()
        if result.reason == speechsdk.ResultReason.RecognizedSpeech:
```

```
671             frase_rec = format(result.text)
672             print("-> RECOGNIZE_SPEECH: Frase Original: "+frase.lower()[:-1])
673             print("-> RECOGNIZE_SPEECH: Frase Reconhecida: "+frase_rec.lower()[:-1])
674             if (frase.lower()[:-1] == frase_rec.lower()[:-1]):
675                 return(True)
676             else:
677                 return(False)
678         elif result.reason == speechsdk.ResultReason.NoMatch:
679             return(False)
680         elif result.reason == speechsdk.ResultReason.Canceled:
681             return(False)
682     except Exception as e:
683         print(str(e))
684         return(False)
685 
686 # Função para calcular o aspeto do olho
687 def eye_aspect_ratio(eye):
688     A = np.linalg.norm(eye[1] - eye[5])
689     B = np.linalg.norm(eye[2] - eye[4])
690     C = np.linalg.norm(eye[0] - eye[3])
691     ear = (A + B) / (2.0 * C)
692     return ear
693 
694 # Função para calcular o aspecto da boca
695 def mouth_aspect_ratio(mouth):
696     A = dist.euclidean(mouth[3], mouth[9])
697     B = dist.euclidean(mouth[0], mouth[6])
698     mar = A / B
699     return mar
700 
701 # Eliminar dados biométricos de um utilizador
702 def eliminar_cameradata(email):
703     doc_ref = db.collection('Users').document(email)
704     doc_ref.update({'cameradata': firestore.DELETE_FIELD})
705     doc_ref.update({'voicefeatures': firestore.DELETE_FIELD})
706 
707 # Carregar áudio de um ficheiro
708 def carregar_audio(file_path, sample_rate=22050):
709     audio, sr = librosa.load(file_path, sr=sample_rate)
710     audio = librosa.util.normalize(audio)
711     return audio, sr
712 
713 # Extrair características do áudio
714 def extrair_features(audio, sr, n_mfcc=13):
715     mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=n_mfcc)
716     chroma = librosa.feature.chroma_stft(y=audio, sr=sr)
717     mel = librosa.feature.melspectrogram(y=audio, sr=sr)
718     contrast = librosa.feature.spectral_contrast(y=audio, sr=sr)
719 
720     features = np.concatenate((mfccs, chroma, mel, contrast), axis=0)
721     return features
722 
723 # Remover silêncios do áudio
724 def remove_silence(y):
```

```
    non_silent_intervals = librosa.effects.split(y, top_db=20)
    non_silent_audio = np.concatenate([y[start:end] for start, end in non_silent_intervals])
    return non_silent_audio

# Calcular a distância DTW entre duas sequências de características
def calcular_distancia_dtw(features1, features2):
    distancia, _ = fastdtw(features1.T, features2.T, dist=euclidean)
    return distancia

# Extrair características devoz de um áudio
def feature_voz(audio_path):
    audio, sr1 = carregar_audio(audio_path)

    audio = remove_silence(audio)

    features = extrair_features(audio, sr1)

    return features

# Reconhecer a voz de um utilizador
async def recognize_voice(data):
    try:
        email = data['email']
        videofile = data['videofile']
        video_path = os.path.join(app.instance_path, f"{email}/{videofile}")
        try:
            video_clip = VideoFileClip(video_path)
            audio_path = os.path.join(app.instance_path, f"{email}/detect_audio.wav")
            video_clip.audio.write_audiofile(audio_path, codec='pcm_s16le')  # Salvar como WAV
            user_ref = db.collection("Users").document(email)
            user_data = user_ref.get()
            if user_data.exists:
                voice_features_json = user_data.to_dict().get('voicefeatures', '{}')
                voice_features_list = json.loads(voice_features_json)
                voice_features_array = np.array(voice_features_list)
                distancia = calcular_distancia_dtw(feature_voz(audio_path), voice_features_array)
                if os.path.exists(audio_path):
                    os.remove(audio_path)
                if distancia < 41688:
                    print('-> RECOGNIZE_VOICE: Autenticado com sucesso! Distância:' + str(distancia))
                    return(True)
                else:
                    print('-> RECOGNIZE_VOICE: Falha na autenticação: Voz não reconhecida! Distância:' + str(distancia))
                    return(False)
            else:
                print("-> RECOGNIZE_VOICE: Utilizador não encontrado na Base de Dados.")
                return(False)
```

```
772         except Exception as e:
773             print(f"-> RECOGNIZE_VOICE: Erro ao extrair áudio: {e}")
774             return(False)
775     except Exception as e:
776         print(str(e))
777         return(False)
778 
779 # Registar a voz de um utilizador
780 async def voice_register(data):
781     try:
782         email = data['email']
783         videofile = data['videofile']
784         video_path = os.path.join(app.instance_path, f"{email}/{videofile}")
785         try:
786             video_clip = VideoFileClip(video_path)
787             audio_path = os.path.join(app.instance_path, f"
    {email}/register_audio.wav")
788             video_clip.audio.write_audiofile(audio_path, codec='pcm_s16le')  #
    Salvar como WAV
789             features_json = feature_voz(audio_path).tolist()
790             doc_ref = db.collection('Users').document(email)
791             doc_ref.update({'voicefeatures': json.dumps(features_json)})
792             print("-> VOICE_REGISTER: Utilizador registado na Base de Dados.")
793             return(True)
794         except Exception as e:
795             print(f"-> VOICE_REGISTER: Erro ao extrair áudio: {e}")
796             return(False)
797     except Exception as e:
798         print(str(e))
799         return(False)
800 
801 #Start Server
802 if __name__ == '__main__':
803     app.run(
804         debug=True,
805         host="0.0.0.0",
806         port=5555
807         )
```