**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Программный код проекта**

import sys  
import cv2  
import face\_recognition as fa\_re  
import datetime  
from sqlalchemy import create\_engine  
from sqlalchemy.orm import sessionmaker  
from sqlalchemy import Boolean, Column, Integer, String, DateTime  
from sqlalchemy.dialects.postgresql import BYTEA  
from sqlalchemy.orm import declarative\_base  
import base64  
from PIL import Image  
import io  
import numpy  
import base64  
from http.server import BaseHTTPRequestHandler, HTTPServer  
import socketserver  
#from flask import Flask  
import time  
import numpy as np  
import setuptools  
  
  
# DB settings-------------------------------------------------  
db\_url = 'postgresql://userP:mypass@192.168.126.130:5432/facedb'  
engine = create\_engine(  
 db\_url, pool\_size=10, max\_overflow=20  
)  
Session = sessionmaker(  
 engine,  
 autocommit=False,  
 autoflush=False,  
)  
session = Session()  
  
Base = declarative\_base()  
  
# classes for DB-----------------------------------------------  
class facesDB(Base):  
 \_\_tablename\_\_ = "faces"  
  
 id = Column(Integer, primary\_key=True)  
 file = Column(BYTEA)  
 face\_data = Column(BYTEA)  
 personal\_id = Column(Integer)  
  
  
class face\_check(Base):  
 \_\_tablename\_\_ = "in\_out\_date"  
  
 time = Column(DateTime, default=datetime.datetime.now, primary\_key=True)  
 per\_id = Column(Integer)  
 cam\_id = Column(Integer)  
  
  
class undef\_face(Base):  
 \_\_tablename\_\_ = "undendified\_faces"  
 time = Column(DateTime, default=datetime.datetime.now, primary\_key=True)  
 cam\_id = Column(Integer)  
 file = Column(BYTEA)  
  
  
def get\_faces():  
 #ans = session.query(facesDB).filter(facesDB.id == 0).all()  
 ans = session.query(facesDB.personal\_id, facesDB.face\_data).all()  
  
 answer\_id = []  
 answer\_face = []  
 for i in ans:  
 bytes\_converted = np.frombuffer( i[1], np.float64)  
 answer\_id.append(i[0])  
 answer\_face.append(bytes\_converted)  
  
 return answer\_id, answer\_face  
  
  
def find\_faces(known\_faces, new\_faces):  
 for face\_new in face\_encodings:  
 t1 = time.time()  
 matches = fa\_re.compare\_faces(faces, face\_new)  
  
 if True in matches:  
 print(f'id={ matches.index(True) }')  
 else:  
 print('unknown')  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
 ids, faces = get\_faces()  
 print(ids)  
  
 video\_capture = cv2.VideoCapture(0)  
  
 #video\_capture = cv2.VideoCapture("rtsp://192.168.100.8:8554/webcam.h264")  
  
 process\_this\_frame = True  
  
 while True:  
  
 ret, frame = video\_capture.read()  
  
 if process\_this\_frame:  
 small\_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)  
  
 rgb\_small\_frame = np.ascontiguousarray(small\_frame[:, :, ::-1])  
  
 face\_locations = fa\_re.face\_locations(rgb\_small\_frame)  
  
 if len(face\_locations)==0:  
 continue  
  
 face\_encodings = fa\_re.face\_encodings(rgb\_small\_frame, face\_locations, model="small")  
  
 t1 = time.time()  
  
 for face\_new in face\_encodings:  
 #time.sleep(0.5)  
 matches = fa\_re.compare\_faces(faces, face\_new)  
  
 if True in matches:  
 p\_id=ids[ matches.index(True) ]  
 print(f'id={ p\_id }')  
 face\_add = face\_check(  
 per\_id=p\_id,  
 cam\_id =1  
 )  
  
 session.add(face\_add)  
 session.commit()  
  
 else:  
  
 face\_unden = undef\_face(  
 cam\_id =1,  
 file= cv2.imencode('.png', frame)[1].tobytes()  
 )  
 session.add(face\_unden)  
 session.commit()  
 print('unknown')  
  
 print(f"{time.time()-t1}")