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}")
```