**ПРИЛОЖЕНИЕ А**

**(обязательное)**

**Программный код проекта**

import cv2  
import face\_recognition  
from threading import Thread  
import datetime

def read\_imgs(nums: int)-> list:  
 images=[]  
 for i in range(1, nums+1):  
 images.append( cv2.imread('images/getty'+str(i)+'.jpg', cv2.IMREAD\_UNCHANGED) )  
 return images  
class ThreadedCamera(object):  
 def \_\_init\_\_(self, src=0):  
 self.names=["kirill"]  
 self.me = face\_recognition.load\_image\_file('images/me.jpeg')  
 self.me\_enc = face\_recognition.face\_encodings(self.me)[0]  
 self.known\_faces = [self.me\_enc]  
 #capture camera  
 self.capture = cv2.VideoCapture(src)  
 self.capture.set(cv2.CAP\_PROP\_BUFFERSIZE, 1)  
 self.frame = None  
 # Start frame retrieval thread  
 self.thread = Thread(target=self.update, args=())  
 self.thread.daemon = True  
 self.thread.start()  
 # FPS = 1/X  
 # X = desired FPS  
 self.FPS = 1 / 10  
 self.FPS\_MS = int(self.FPS \* 1000)  
 #find face  
 self.Founded = False  
 self.founde\_frame = None  
 self.threadFind = Thread(target=self.find\_face, args=())  
 self.threadFind.daemon = True  
 self.faces = True  
 self.face\_frame=None  
 #naming\_face  
 self.threadNaming = Thread(target=self.naming\_face, args=())  
 self.threadNaming.daemon = True  
 self.threadNaming.start()  
  
  
  
 def update(self):  
 while True:  
 if self.capture.isOpened():  
 (self.status, self.frame) = self.capture.read()  
 self.face\_frame = self.frame  
 if self.faces:  
 print("started thred")  
 self.threadFind.start()  
 self.faces = False  
 #time.sleep(self.FPS)  
 def show\_frame(self):  
 print("start", datetime.datetime.now())  
 cv2.imshow('frame', self.face\_frame)  
 cv2.waitKey(1)  
 print("fin", datetime.datetime.now())  
 def find\_face(self):  
 # gray = cv2.cvtColor(self.frame, cv2.COLOR\_BGR2GRAY)  
 while True:  
 small\_frame = cv2.resize(self.face\_frame, (0, 0), fx=0.25, fy=0.25)  
 rgb\_small\_frame = small\_frame[:, :, ::-1]  
 self.face\_loc = face\_recognition.face\_locations(rgb\_small\_frame)  
 face\_enc=face\_recognition.face\_encodings(rgb\_small\_frame, self.face\_loc)  
 for face\_en in face\_enc:  
 matches=face\_recognition.compare\_faces(self.known\_faces, face\_en)  
 if True in matches:  
 self.Founded = True  
 if self.Founded:  
 print("draw")  
 for (top, right, bottom, left), name in zip(self.face\_loc, self.names):  
 cv2.rectangle(self.face\_frame, (left, top), (right, bottom), (0, 0, 255), 2)  
 # Draw a label with a name below the face  
 cv2.rectangle(self.face\_frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)  
 font = cv2.FONT\_HERSHEY\_DUPLEX  
 cv2.putText(self.face\_frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)  
 # self.frame=self.face\_frame  
  
  
 def naming\_face(self):  
 print("namnig")  
if \_\_name\_\_ == '\_\_main\_\_':  
 cam='rtsp://192.168.8.101:8080/h264\_ulaw.sdp'  
 thr\_cam=ThreadedCamera(cam)  
 while True:  
 try:  
 thr\_cam.show\_frame()  
 except:  
 pass  
 cv2.destroyAllWindows()

app.py:

const h1 = document.getElementById('header\_\_h1');

const search = document.getElementById('search');

const table = document.getElementById('table');

const urlPersonal = 'personal';

const urlUsers = 'users';

const urlCameras = 'cameras';

const urlCabinets = 'cabinets';

const urlTime = 'time';

const resultsPersonal = [

{

id\_pers: 1,

name: {

username: 'kir',

last\_name: 'lname',

first\_name: 'fname',

},

dep\_id: {

name: 'dep1',

},

},

{

id\_pers: 2,

name: {

username: 'user1',

last\_name: 'lname',

first\_name: 'fname',

},

dep\_id: {

name: 'dep3',

},

},

{

id\_pers: 3,

name: {

username: 'kir',

last\_name: 'lname',

first\_name: 'fname',

},

dep\_id: {

name: 'dep3',

},

},

];

const resultsTime = [

{

id: 1,

timedate: '2022-12-28T11:50:08.688797Z',

direction: true,

per\_id: 1,

cab\_id: 1,

cam\_id: 1,

},

{

id: 2,

timedate: '2022-12-28T11:50:19.978811Z',

direction: false,

per\_id: 1,

cab\_id: 1,

cam\_id: 2,

},

];

const resultsCamera = [

{

id\_cam: 1,

cam\_model: 'model1',

cab\_id: {

name: 'cab1',

},

in\_pos: true,

},

{

id\_cam: 2,

cam\_model: 'cam2',

cab\_id: {

name: 'cab1',

},

in\_pos: false,

},

];

const resultsCabinets = [

{

id\_cab: 1,

name: 'cab1',

floor: 1,

dep\_id: {

name: 'dep1',

},

},

{

id\_cab: 2,

name: 'cab2',

floor: 2,

dep\_id: {

name: 'dep2',

},

},

{

id\_cab: 3,

name: 'cab3',

floor: 3,

dep\_id: {

name: 'dep3',

},

},

{

id\_cab: 4,

name: 'cab4',

floor: 9,

dep\_id: {

name: 'dep2',

},

},

];

class Router {

routes = [];

mode = null;

root = '/';

constructor(options) {

this.mode = window.history.pushState ? 'history' : 'hash';

if (options.mode) this.mode = options.mode;

if (options.root) this.root = options.root;

this.listen();

}

add = (path, cb) => {

this.routes.push({ path, cb });

return this;

};

remove = (path) => {

for (let i = 0; i < this.routes.length; i += 1) {

if (this.routes[i].path === path) {

this.routes.slice(i, 1);

return this;

}

}

return this;

};

flush = () => {

this.routes = [];

return this;

};

clearSlashes = (path) =>

path.toString().replace(/\/$/, '').replace(/^\//, '');

getFragment = () => {

let fragment = '';

if (this.mode === 'history') {

fragment = this.clearSlashes(

decodeURI(window.location.pathname + window.location.search)

);

fragment = fragment.replace(/\?(.\*)$/, '');

fragment = this.root !== '/' ? fragment.replace(this.root, '') : fragment;

} else {

const match = window.location.href.match(/#(.\*)$/);

fragment = match ? match[1] : '';

}

return this.clearSlashes(fragment);

};

navigate = (path = '') => {

if (this.mode === 'history') {

window.history.pushState(null, null, this.root + this.clearSlashes(path));

} else {

window.location.href = `${window.location.href.replace(

/#(.\*)$/,

''

)}#${path}`;

}

return this;

};

listen = () => {

clearInterval(this.interval);

this.interval = setInterval(this.interval, 50);

};

interval = () => {

if (this.current === this.getFragment()) return;

this.current = this.getFragment();

this.routes.some((route) => {

const match = this.current.match(route.path);

if (match) {

match.shift();

route.cb.apply({}, match);

return match;

}

return false;

});

};

}

const router = new Router({

mode: 'hash',

root: '/',

});

router

.add(/main\_page/, () => {

table.innerHTML = '';

h1.textContent = 'In-Out status';

table.style.marginRight = '1rem';

search.style.display = 'none';

// getData(urlTime);

getTableHead(resultsTime);

getTableBody(resultsTime);

})

.add(/cameras\_status/, () => {

table.innerHTML = '';

h1.textContent = 'Cameras status';

table.style.marginRight = '30rem';

search.style.display = '';

// getData(urlCameras);

getTableHead(resultsCamera);

getTableBody(resultsCamera);

})

.add(/cabinets\_status/, () => {

table.innerHTML = '';

h1.textContent = 'Cabinets status';

table.style.marginRight = '30rem';

search.style.display = '';

// getData(urlCabinets);

getTableHead(resultsCabinets);

getTableBody(resultsCabinets);

})

.add('', () => {

table.innerHTML = '';

h1.textContent = 'Personal info';

table.style.marginRight = '30rem';

search.style.display = '';

// getData(urlPersonal);

getTableHead(resultsPersonal);

getTableBody(resultsPersonal);

});

// async function getData(url) {

// await fetch(`http://192.168.0.101/api/${url}/`, {

// method: 'GET',

// mode: 'cors',

// headers: {

// 'Content-Type': 'application/json',

// },

// })

// .then((response) => {

// response.json().then((data) => {

// return data.results;

// });

// })

// .catch((e) => {

// console.log(e);

// });

// }

function createCircle(value, tr) {

const td = document.createElement('td');

const circle = document.createElement('div');

circle.style.width = '1rem';

circle.style.height = '1rem';

circle.style.margin = '0 auto';

circle.style.backgroundColor = value ? 'green' : 'red';

circle.style.borderRadius = '50%';

td.appendChild(circle);

tr.appendChild(td);

return table.appendChild(tr);

}

function createTh(value, tr) {

const th = document.createElement('th');

th.innerHTML = value.toString().replaceAll('\_', ' ');

tr.appendChild(th);

return table.appendChild(tr);

}

function createTd(value, tr) {

const td = document.createElement('td');

td.innerHTML = value.toString().replaceAll(',', ' ');

tr.appendChild(td);

return table.appendChild(tr);

}

function getTableHead(data) {

const keys = Object.keys(data[0]);

const tr = document.createElement('tr');

keys.forEach((key) => {

createTh(key, tr);

});

}

function getTableBody(data) {

data.forEach((item) => {

const values = Object.values(item);

const tr = document.createElement('tr');

values.forEach((value) => {

if (typeof value === 'object') {

const newValue = Object.values(value);

createTd(newValue, tr);

} else if (typeof value === 'boolean') {

createCircle(value, tr);

} else {

createTd(value, tr);

}

});

});

}