UNIVERZITET CRNE GORE
ELEKTROTEHNIČKI FAKULTET

Nikola Živković

# Razvoj aplikacije i implementacija sistema za detekciju neovlašćenih lica unutar zone nadzora

MASTER RAD

Podgorica, 2023. godine

UNIVERZITET CRNE GORE
ELEKTROTEHNIČKI FAKULTET

Nikola Živković

# Razvoj aplikacije i implementacija sistema za detekciju neovlašćenih lica unutar zone nadzora

MASTER RAD

Podgorica, 2023. godina

## PODACI I INFORMACIJE O MAGISTRANDU

| | |
|---|---|
| Ime i prezime: | Nikola Živković |
| Datum i mjesto rođenja: | 09.08.1999. godine, Podgorica, Crna Gora |
| Naziv završenog osnovnog studijskog programa i godina završetka: | Studijski program Primijenjeno računarstvo, Elektrotehnički fakultet, Univerzitet Crne Gore, 180 ECTS kredita, 2021. godine. |

INFORMACIJE O MASTER RADU

| | |
|---|---|
| Naziv postdiplomskih studija: | Master studije primijenjenog računarstva |
| Naslov rada: | Razvoj aplikacije i implementacija sistema za detekciju neovlašćenih lica unutar zone nadzora |
| Fakultet na kom je rad odbranjen: | Elektrotehnički fakultet |

UDK, OCJENA I ODBRANA MASTER RADA

| | |
|---|---|
| Datum prijave master rada: | 12.06.2023. godine |
| Datum sjednice Vijeća na kojoj je prihvaćena tema: | 11.07.2023. godine |
| Komisija za ocjenu teme i podobnosti magistranda: | Prof. dr Milutin Radonjić<br>Prof. dr Nikola Žarić<br>Doc. dr Slavica Tomović |
| Mentor: | Prof. dr Nikola Žarić |
| Komisija za ocjenu rada: | Prof. dr Milutin Radonjić<br>Prof. dr Nikola Žarić<br>Doc. dr Slavica Tomović |
| Komisija za odbranu rada: | Prof. dr Milutin Radonjić<br>Prof. dr Nikola Žarić<br>Doc. dr Slavica Tomović |
| Lektor: | Prof. dr Nikola Žarić |
| Datum odbrane: | 25.12.2023. godine |

Ime i prezime autora: Nikola Živković, BAap. Primijenjenog računarstva

# E T I Č K A  I Z J A V A

U skladu sa članom 22 Zakona o akademskom integritetu i članom 18 Pravila studiranja na master studijama, pod krivičnom i materijalnom odgovornošću, izjavljujem da je magistarski rad pod naslovom

**"Razvoj aplikacije i implementacija sistema za detekciju neovlašćenih lica unutar zone nadzora"**

moje originalno djelo.

**Podnosilac izjave,**

**Nikola Živković, BAap**

U Podgorici, dana 23.10.2023. godine

# PREDGOVOR

Ovaj rad posvećujem mom ocu Jagošu, koji bi se svakog puta, nakon svakog mog položenog ispita, obradovao više od mene. Do samog kraja studija, koje moj otac nije doživio, a koje nikada ne bih ni upisao na prvom mjestu da nije bilo njega, služio mi je kao inspiracija i snaga. Meni ostaje da se nadam, da sam ispunio njegova očekivanja, opravdao njegovo povjerenje i neizmjernu ljubav koju mi je pružio, da je ponosan i da odmara na ljepšem i mirnijem mjestu.

Nikola Živković

# IZVOD RADA

Komercijalni sistemi nadzora se često susrijeću sa izazovima u vidu visokih troškova implementacije i održavanja što ih čini nedostupnim organizacijama ili pojedincima sa ograničenim budžetom ili resursima. Takođe, ovi sistemi uglavnom nemaju implementiranu mogućnost detekcije neovlašćenog pristupa, a rješenja koja omogućavaju tu vrstu integracije su izuzetno skupa.

Ovaj rad se bavi problemom i nudi rješenja u oblasti realizacije resursno i energetski efikasnog, automatizovanog sistema za potrebe adekvatnog prepoznavanja nepoznatih i/ili neželjenih osoba upotrebom tehnologije za prepoznavanje lica. Takođe, bavi se razvojem i implementacijom *web*-aplikativnog kontrolnog centra za monitoring i kontrolu prethodno navedenog sistema sa udaljene lokacije.

Primjenom tehnika za prepoznavanje lica uspješno je realizovan pristupačan, resursno i energetski efikasan sistem za detekciju neovlašćenog pristupa kao i notifikaciju korisnika sistema u skoro realnom vremenu. Takođe, upotrebnom baze podataka kao posrednika u komunikaciji između *raspberry-pi* uređaja i *web* servera, uspješno je realizovan resursno efikasan kontrolno nadzorni centar u vidu *web* aplikacije, koji korisnicima, sa udaljene tačke, omogućava nadgledanje stanja sistema kao i kontrolu nad istim.

# ABSTRACT

Commercial surveillance systems are often challenged by high implementation and maintenance costs that makes them inaccessible to organizations or individuals with limited budgets or resources. Also, these systems generally do not have implemented the possibility of detecting unauthorized access, and solutions that enable this type of integration are extremely expensive.

This paper deals with the problem and offers solutions in the area of developing a resource- and energy-efficient, automated system for the needs of adequate recognition of unknown and/or unwanted persons using facial recognition technology. It also deals with the development and implementation of a web-based control center for monitoring and controlling the aforementioned system from a remote location.

By applying facial recognition techniques, an affordable, resource- and energy-efficient system for detecting unauthorized access and notifying system users in almost real time was successfully implemented. Also, by using the database as an intermediary in the communication between the raspberry-pi device and the web server, a resource-efficient control and monitoring center was successfully implemented in the form of a web application, which enables users to monitor the state of the system as well as control it from a remote location.

# Sadržaj

# 1 UVOD

Sa konstantno rastućom i sve većom potrebom za privatnošću a ujedno i eksponencijalnom ekspanzijom i razvojem interneta stvari (eng. Internet of Things - IoT) raste i broj kako komercijalnih tako i softvera otvorenog koda (eng. Open source) koji, između ostalog nude i solucije u oblasti nadgledanja i detekcije nepoznatih osoba i neovlaštenog pristupa unutar zone nadzora.

U vremenu u kome su podaci i informacije neprocjenjive, a nikad lakše dostupne, potrebno je na svaki mogući način obezbijediti i sačuvati privatnost i integritet kako samog sistema i podataka koje sistem obrađuje, tako i krajnje korisnike istog, posebno kod sistema koji obrađuje multimedijalne podatke.

Baš iz tih razloga, sistem koji je razvijen za potrebe ovog rada i prezentovan u istom je dizajniran na način da niti u jednom svom dijelu nema posredničke tačke ni komunikacije sa eksternim aplikacijama koje bi ili preko kojih bi neko mogao doći do informacija koje se obrađuju unutar samog sistema.

Predmet istraživanja je razvoj sistema za detekciju prisustva neovlašćenih osoba unutar zone nadzora. Značaj ovog istraživanja ogleda se u resursnoj i energetskoj efikasnosti, jednostavnosti implementacije i korišćenja sistema od strane krajnjeg korisnika. U ovom istraživanju je predstavljen dizajn, opisana implementacija i razvoj zatvorenog i automatizovanog sistema za detekciju neovlaštenih osoba unutar određene zone nadzora, zasnovanog na detekciji lica, a softverski izgrađenog na *debian-based linux* server okruženju, *raspberry pi*-u (raspi operativnom sistemu), *python* programskom jeziku kao i *MySql* bazi podataka.

U cilju jednostavnijeg upravljanja kompletnim sistemom, kao i smanjenja mogućnosti greške i lakšeg otklanjanja istih, dizajnirali smo, i u vidu *web* aplikacije implementirali kontrolni/monitoring centar, preko kojeg će se pratiti trenutno stanje sistema, omogućiti pokretanje i zaustavljanje istog, kao i restartovanje čitavog fizičkog uređaja ukoliko smatramo da je došlo do greške.

Posebno je važno naglasiti da je jedan ovako realizovan sistem nije ograničen ni u kom smislu i uz male modifikacije koda ili dizajna u potpunosti je skalabilan na više načina i u više pravaca. Uz modifikaciju koda na način da sistem paralelno obrađuje podatke sa više kamera kao i da vodi evidenciju o zasebnim *child* procesima (podređenim odnosno procesima nastalim od glavnog, tj. procesa roditelja) kreiranim za svaku kameru zasebno, a uz to na serverskoj strani čitati podatke iz više tabela namijenjenih za više kamera (kamera1, …, kameraN, kameraN+1), moguće je napraviti složeniji *cluster* nadzorni sistem, naravno ukoliko obezbijedimo i resurse za ovako nešto. Takođe, mogućnost skalabilnosti sistema možemo diskutovati i u pravcu dodatne automatizacije u vidu omogućavanja korisniku/cima sistema da na daljinu vrše dotreniravanje modela na način što će nakon detekcije osobe kao nepoznate, istu dodati u listu poznatih ili slično. Još jedan pravac skalabilnosti je i bezbjednost istog, osigurati šifrovanu komunikaciju između svih djelova sistema uvođenjem bezbjednosnog kriptografskog protokola (eng. Transport layer security - TLS), integraciju sa sopstvenim SMTP (eng. Simple mail transfer protocol) serverom i slično.

U radu su predstavljeni i rezultati evaluacije performansi sistema u vidu kašnjenja notifikacija, procenta false-positive rezultata, mrežnog opterećenja, energetske potrošnje, iskorišćenosti hardverskih resursa, kao i mogućnosti širenja i nadogradnje čitavog sistema.

## 1.1 Motiv i cilj istraživanja

Sa sve bržim napretkom i razvojem video nadzornih tehnologija, IoT i Edge Computinga (Arhitektura ivičnog računarstva), eksponencijalno se povećava i potreba za efikasnim i pouzdanim sistemom za detekciju neovlašćenog pristupa.

Komercijalni nadzorni sistemi se često susrijeću sa izazovima u vidu visokih troškova implementacije i održavanja što ih čini nedostupnim organizacijama ili pojedincima sa ograničenim budžetom ili resursima. Takođe, ovi sistemi uglavnom nemaju implementiranu mogućnost detekcije neovlašćenog pristupa, a rješenja koja omogućavaju tu vrstu integracije su izuzetno skupa.

Cilj ovog istraživanja je obezbijediti isplativu alternativu korišćenjem pristupačnog hardware-a i software-a otvorenog koda, čineći sisteme za detekciju neovlašćenog pristupa a zasnovane na tehnologiji prepoznavanja lica, dostupnijim širem broju korisnika.

Sa sve većim naglaskom na održivost, energetski efikasni sistemi su ključni. Dizajniranjem energetski efikasnog sistema, ovo istraživanje doprinosi smanjenju potrošnje električne energije čineći ga pogodnim za dugoročnu primjenu.

Tehnologija prepoznavanja lica se pokazala kao efikasan alat u identifikaciji pojedinaca i otkrivanju potencijalnih prijetnji. Integracijom mogućnosti prepoznavanja lica u sistem za otkrivanje neovlašćenog pristupa nadzorne zone, možemo znatno smanjiti procenat lažnih uzbuna i poboljšati bezbjedonosne mjere preciznim identifikovanjem neovlašćenih pojedinaca unutar zone nadzora.

Ovo istraživanje ima za cilj da se pozabavi prethodno navedenim izazovima, razvojem troškovno i energetski efikasnog sistema za detekciju neovlašćenog pristupa nadzorne zone, upotrebnom tehnologije prepoznavanja lica kao i razvojem i implementacijom podsistema za orkestraciju i *web* aplikacije za kontrolu i monitoring čitavog, ali i pojedinačnih djelova sistema implementiranog na *raspberry pi* hardverskoj platformi. Primarni cilj je omogućiti potpunu skalabilnost i fleksibilnost sistema, tj. mogućnost istog da se prilagodi različitim pravnim, društvenim ili tehničkim zahtjevima organizacije ili pojedinca. Sistem uz male modifikacije koda omogućava jednostavnu integraciju sa dodatnim senzorima, kamerama omogućavajući proširenje i prilagođavanje sistema na osnovu specifičnih scenarija primjene.

Navedeni sistem je isplativa solucija, koja nudi odličan odnos cijene i kvaliteta/performansi, a koja i krajnjim korisnicima bez tehničkog predznanja, omogućava jednostavnu implementaciju sistema u njihovom okruženju. Takođe, još jedan od ciljeva je postizanje energetski efikasnog sistema koji minimizuje potrošnju električne energije i resursa samog hardware-a na kom je isti implementiran, čineći ga pogodnim za dugoročnu primjenu u okruženju sa ograničenim resursnim uslovima.

Istraživanje ima za cilj postizanje, visokog stepena tačnosti i pouzdanosti u vidu sposobnosti za prepoznavanja lica i detekcije neovlašćenog pristupa, kao i kontrolu greške i samoodrživosti radnog stanja sistema. Omogućavajući krajnjem korisniku, da sa udaljene lokacije, sam upravlja i kontroliše rad sistema, postižemo visok nivo pouzdanosti istog a samim tim i povećavamo stepen bezbjednosti nadzorne zone.

Takođe, istraživanje ima za cilj da doprinese akademskom i praktičnom znanju i primjeni u oblasti integracije tehnologije za prepoznavanje lica sa sistemom otkrivanja neovlašćenog

pristupa, kao i da pruži uvid u izazove dizajna, implementacije, optimizacije i primjene jednog ovakvog sistema u stvarnom svijetu.

## 1.2 Pregled dosadašnjih istraživanja

U poslednjih par godina, značajan je porast stope kriminaliteta, posebno u zemljama u razvoju. Brojne pojave kao što su krađe, provale i neželjene uzurpacije privatnog posjeda nastaju neočekivano [1].

Ovo zahtijeva instalaciju sistema sposobnog da spriječi ulazak u privatne objekte, mala preduzeća ili u najmanju ruku alarmira vlasnike istih. Mnoge konvencionalne metode bezbjednosnih sistema u vidu vrata sa lozinkom, čuvara i sl. su skupe i kao takve nepogodne za privatne prostore i mala preduzeća. Sa druge strane, obična nadzorna kamera, nema implementiran sistem prepoznavanja prijetnji i upozorenja, iz tog razloga je razvoj troškovno pristupačnog sistema za detekciju neovlašćenog pristupa značajan izazov [2].

Tehnologija prepoznavanja lica se koristi kao pomoć pri identifikaciji osoba. Istorija prepoznavanja lica doseže do 1960. godine, kada je matematičar i programer Woodrow Wilson Bledsoe osmislio tehniku mjerenja koja se može koristiti za kategorizaciju slika ljudskih lica [3].

Tehnološki zasnovan način prepoznavanja ljudskog lica poznat je kao tehnologija prepoznavanja lica eng. (Facial recognition technology). Tehnologija za prepoznavanje lica mapira facijalne karakteristike lica sa slike ili videa korišćenjem biometrije, nakon čega te metapodatke upoređuje sa bazom podataka sebi već poznatih lica kako bi došla do rezultata tj. pronašla poklapanje. Ljudi aktivno koriste tehnologiju prepoznavanja lica na svojim mobilnim uređajima [4-8]

Tamo gdje mi vidimo lice, tehnologija prepoznavanja vidi podatke, puno podataka. Ovi podaci se mogu čuvati (pisati) a kasnije i po potrebi čitati. Postoji više tehnika za prepoznavanje lica, ali sve se pridržavaju nekih osnovnih pravila. Neke od bitnih stavki su horizontalna razdaljina između očiju i dijagonalna razdaljina između čela i brade. U obzir se takođe uzimaju i crte lica, jedan sistem tehnologije za prepoznavanje lica uzme u obzir do 68 crta lica koje su bitne stavke u razlikovanju našeg od lica drugih osoba. Kao rezultat prikupljenih podataka, dobijamo unikatnu signaturu našeg lica. Kasnije, matematička formula signature našeg lica se

uporeduje sa bazom podataka ostalih signatura poznatih lica i na taj način dolazi do prepoznavanja [9].

Sudeći po izvještaju Centra za Strategijsku i Internacionalnu Studiju eng. (CSI), preciznost algoritama za prepoznavanje lica je visoka, oko 99.97% sa procentom mogućnosti greške oko 0.1 % u uslovima poređenja statičkih slika visokog kvaliteta [9]. Međutim, u praksi, tj. kada se signature lica upoređuju sa signaturama ljudi izračunatim, slikanim u javnosti, sa ne tako savršenim operativnim uslovima, procenat greške raste i do 9.3%. Procenat greške takođe raste i ukoliko osoba ne gleda direktno u objektiv kamere, ukoliko su u cjelosti ili djelimično pokriveni sjenkama ili objektima, ukoliko je nivo osvjetljenja loš, subjekti obrade prave izraze lica (osmijeh, zijevanje, tuga) a zanimljiv faktor greške takođe može biti i starenje [9-10].

U [10] je predloženo korišćenje OpenCV eng. (Open-Source Computer Vision) biblioteke koja se bazira na obradi slika i videa u realnom vremenu za čitanje, pisanje i obradu slika za prepoznavanje lica. Takođe, *dlib* biblioteka se koristi za mjerenje pomenutih 68 crta lica i predstavlja ih na Dekartovom pravouglom koordinatnom sistemu. Uz to još se koristi i *Numpy* python biblioteka kojom se simplificiraju razne matematičke operacije, vektorizacije niza objektnih podataka, nasumični proračuni, sve u cilju kreiranja ili upoređivanja digitalne signature konvertovane slike lica.

U [11] je sistem za prepoznavanje neželjenog pristupa nadzorne zone realizovan upotrebom tri programska jezika, i to python, javascripta i PHP-a. Svaki ima različita zaduženja. Python je zadužen za prepoznavanje lica korišćenjem kamere na *raspberry pi-u,* kao i slanje notifikacija ukoliko dođe do detekcije. Javascript je zadužen za slanje obradu i prijem podataka sa *raspberry pi*-a, dok se PHP koristi za prezentovanje rezultata skladištenih u bazi podataka.

U [12] vidimo da je za dataset od 5 lica, tj. 125 slika ukupno, 25 slika po jednom licu, raspberry pi-u potrebno 5.8 sekundi i 18.1 Mb memorije, što je dobro vrijeme treniranja modela uzimajući u obzir da se radi o niskobudžetnom *raspberry pi* 3B računaru koji na godišnjem nivou potroši oko 31 kWh električne energije.

U [13] je predloženo korišćenje eigenface vektorskog algoritma za računanje a kasnije i poređenje signatura lica, iz razloga što isti zahtijeva manje procesorske snage pa je samim tim pogodan za rad na raspberry-pi uređaju. Takođe, unutar sistema je implementiran modul koji pravi razliku između slike, fotografije i živog bića.

Bezbjednost sistema za prepoznavanje lica je takođe jedna posebna oblast bez koje ovakvi sistemi ne mogu u potpunosti da zažive u praksi, kao ni da dosegnu svoj puni potencijal. U [14] je predloženo cloud okruženje za obradu slika i vraćanje rezultata klijentu, međutim cloudu bi se slale samo neophodne signature lica u vidu nizova podataka, a takođe bi se samo pomenuti nizovi podataka čuvali na istom, pa bi se na taj način izbjegao rizik od kompromitacije ličnih podataka, u ovom slučaju slika. Kao dodatni faktor bezbjednosti, svakom nizu podataka koji se šalje na obradu, izračunat je SHA-128 hash, koji služi za provjeru integriteta i povjerljivosti primljenih podataka na serverskoj strani.

U [15] je fokus stavljen na razvoj hardverske strane sistema. Tako su korišćenjem raspberry-pi single-board računara i kontrolnog PCB-a smještenih iza mini LCD ekrana, razvili sistem za automatsko otključavanje vrata na osnovu rezulata sistema za prepoznavanja lica.

U [16] je predložen jednostavan sistem za detekciju neovlašćene osobe, koji kao rezultat te detekcije prikazuje poruku upozorenja na grafičkom interfejsu raspberry-pi uređaja. Na sličnom principu zasnovan se i sistem realizovan u [17] u kom se uzorak lica dotreniravao 6 mjeseci u toku kojih je koristeći se klasifikacionim algoritmom “k-najbližih susjeda” (eng. k-Nearest Neighbor – kNN) ujedno razvrstavao uzorke na poželjne i nepoželjne.

U [18] je sistem za detekciju neovlašćene osobe u potpunosti realizovan upotrebom python programskog jezika. Sistem takođe posjeduje integraciju sa Telegram API-em (Aplikativno programabilnim interfejsom) u cilju slanja notifikacija prilikom detekcije neovlašćene osobe unutar zone nadzora.

U [19] je predloženo korišćenje openCV biblioteke za program prepoznavanja lica, a za kreiranje web aplikacije čija namjena je prijem i prikazivanje lica neovlašćenih osoba, korišćen je Pyramid framework. U ovom istraživanju a u sklopu razvoja web aplikacije koriščeni su i MongoDB, AJAX, JQuery i XML.

Razvijen je veliki broj sistema za detekciju neovlašćenih lica ili radnji. Funkcionišu na principu senzora, kamera i/ili notifikacija. Neki imaju mogućnost prenošenja video zapisa u realnom vremenu (live streaminga) ali nemaju mogućnost notifikacija i slično. Dešava se i da sistemi imaju problem sa internet konekcijom ili napajanjem, pa iz tih i tome sličnih razloga dolazi do prekida rada samog sistema [20].

U [20] je predložena upotreba GSM (Globalnog sistema za mobilnu komunikaciju) modula za slanje notifikacije ali i konfigurisanje mrežne redudanse na način da u slučaju gubitka WiFi signala ili mrežne internet konekcije, raspberry-pi ostvari internet konekciju uz pomoć GSM modula.

U literaturi se, za slanje notifikacija predlaže upotreba servisa ili API-a aplikacija treće strane kao što su Telegram i razni chatbotovi. SMTP eng. (Simple Mail Transfer Protocol) je mail server koji može biti računar ili sistem računara koji prima sve odlazeće poruke svojih klijenata i prosleđuje ih dalje do njihovih adresiranih destinacija, tj. primaoca poruka [21]. U ovom radu smo koristitili SMTP kako bismo omogućili slanje notifikacija prilikom detektovanja neovlašćene osobe unutar zone nadzora.

Svi mail serveri  na sebi imaju implementiran SMTP protokol, a veliki broj njih radi na UNIX operativnim sistemima koristeći sendmail program [21-22].

## 1.3 Naučne metode

Koristeći se metodom razvoja rješenja predložili smo i realizovali upotrebu *mysql* baze podataka kao posrednika u komunikaciji *web-aplikacije* i *raspberry pi-a*, u cilju zaobilaženja potrebe za konstantnim održavanjem 1:1 konekcije te samim tim smanjenja potrebnih resursa. Rad na tezi obuhvata prikaz i implementaciju hardverske infrastrukture potrebne za pravilno funkcionisanje razvijenog sistema. Takođe, obuhvata i razvoj i implementaciju softverskog dijela rješenja, *python*, *bash* programe i skripte, *web-aplikaciju* realizovanu u *Django frameworku* kao i konfiguraciju *mysql* baze podatake.

Sistem je praktično realizovan i testiran. Evaluacija performansi sistema je realizovana sa dva *data* seta i utvrđeno je koji od njih daje bolje rezultate. Kroz metode testiranja i metode uzorkovanja i ispitivanja pouzdanosti sistema izvršeno je ispitivanje performansi sistema. Takođe, koristili smo metrike kao što su kašnjenje notifikacija, procenat *false-positive* rezultata, mrežno opterećenje, potrošnja energije, iskorišćenost hardverskih resursa.

Takođe, u radu od pojedinačnih zapažanja izvodimo uopštene i realne zaključke, uzevši u obzir da imaju oslonac u stvarnosti a koristeći se metodom generalizacije. Prednost ove metode ogleda se u mogućnosti da uz pomoć pojedinačnog zapažanja budu izvedeni zaključci o tome da

li je moguće realizovati pristupačan, resursno i energetski efikasan sistem za detekciju neovlašćenog pristupa nadzorne zone.

Nedostatak se ogleda u mogućnosti dolaska do greške usled izvođenja zaključaka prilikom pogrešnih zapažanja.

# 2 ANALIZA STRUKTURE, SOFTVERSKIH I HARDVERSKIH KOMPONENTI SISTEMA

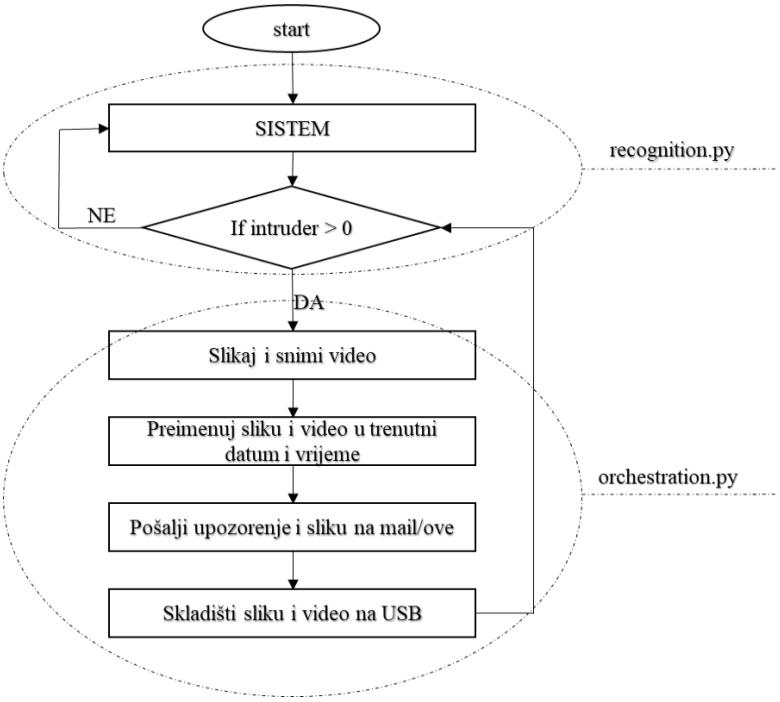
Unutar ovog poglavlja ćemo predstaviti softverske i hardverske resurse, potrebne za realizaciju sistema za detekciju neovlašćenog pristupa unutar nadzorne zone. Takođe, skiciraćemo i predstaviti arhitekturu kako samog sistema za detekciju i orkestraciju tako i dijela *MySql* baze podataka i same *web* aplikacije za kontrolu i monitoring.

## 2.1 Pojednostavljeno funkcionisanje sistema

Sistem kao cjelina se sastoji iz pet djelova i to:

- dijela za prepoznavanje neželjene osobe (***recognition.py***),
- dijela za orkestraciju (***orchestration.py***),
- dijela za monitoring (***monitor.sh***),
- *web* aplikacije (***0.0.0.0:8000***),
- *MySql* baze (***DB: Sistem, Tabela: kameramon***).

Algoritamski prikaz pojednostavljenog funkcionisanja sistema prikazan je na Slici 1.




Slika 1: Algoritamski prikaz pojednostavljenog sistema

Funkcija ovog sistema jeste da detektuje, tj. prepozna prisustvo neželjene ili neovlašćene osobe upotrebom tehnologije prepoznavanja lica, nakon čega je potrebno da isti alarmira korisnika ili korisnike sistema, na način što će im na njihove e-mail adrese proslijediti poruku upozorenja u kojoj se nalazi slika uljeza, tj. potencijalno neželjene osobe koju je sistem detektovao kao takvu. Sistem takođe zadržava lokalnu kopiju slike i videa te osobe, preimenuje ih u trenutni datum i vrijeme i sačuva ih na eksternoj memoriji (npr. *USB sticku*).

U gore prikazanom pojednostavljenom algoritmu primjećujemo da sistem funkcioniše korišćenjem modula ***recognition.py*** i ***orchestration.py*** i obavlja ono za šta je i namijenjen.

Međutim, šta se dešava ukoliko jedan od dva navedena modula iz nekog razloga prestanu sa radom? Da li, ukoliko je to slučaj, možemo reći da sistem još uvijek funkcioniše? Ne možemo, za optimalan rad sistema potrebno je da su oba navedena modula aktivna.

Kako bismo obezbjedili nesmetan i kontinualan rad sistema, moramo uzeti u obzir i mogućnost dolaska do greške i preduzeti mjere i korake potrebne za oporavak od istih. (u daljem tekstu ćemo koristiti eng. *error handling*).

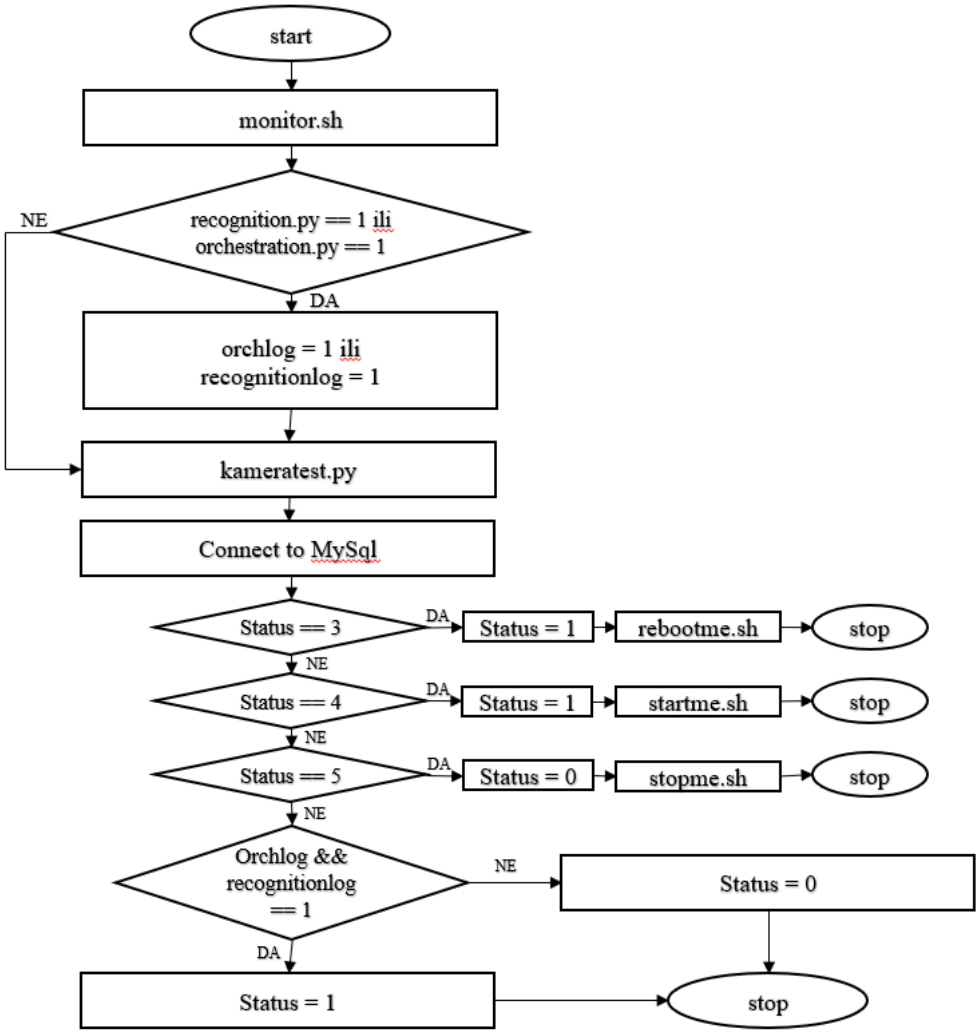
Kako bismo imali uvid u trenutno stanje sistema, tj. status ***recognition.py*** i ***orchestration.py*** procesa/modula, njihovo stanje zapisujemo u bazi podataka, u vidu 0 ili 1. (**0** = sistem/jedan dio sistema ne funkcioniše kako treba ; **1**= sistem funkcioniše u potpunosti). Modul koji između ostalog vodi računa o upisivanju stanja sistema u bazu je ***monitor.sh***

*Web* aplikacija koja se nalazi na "***x.x.x.x:8000***" služi kao monitoring ali i kontrolni centar, u kom možemo pratiti stanje sistema na način što *web* aplikacija u kontinuitetu provjerava trenutno stanje u bazi podataka, ali i naređivati mu šta da radi, upisivanjem novog statusa u bazu podataka, tj. možemo pokrenuti ili zaustaviti rad sistema kao i restartovati uređaj na kom se isti nalazi ukoliko smatramo da je to neophodno.

Moguća stanja za promjenu statusa:

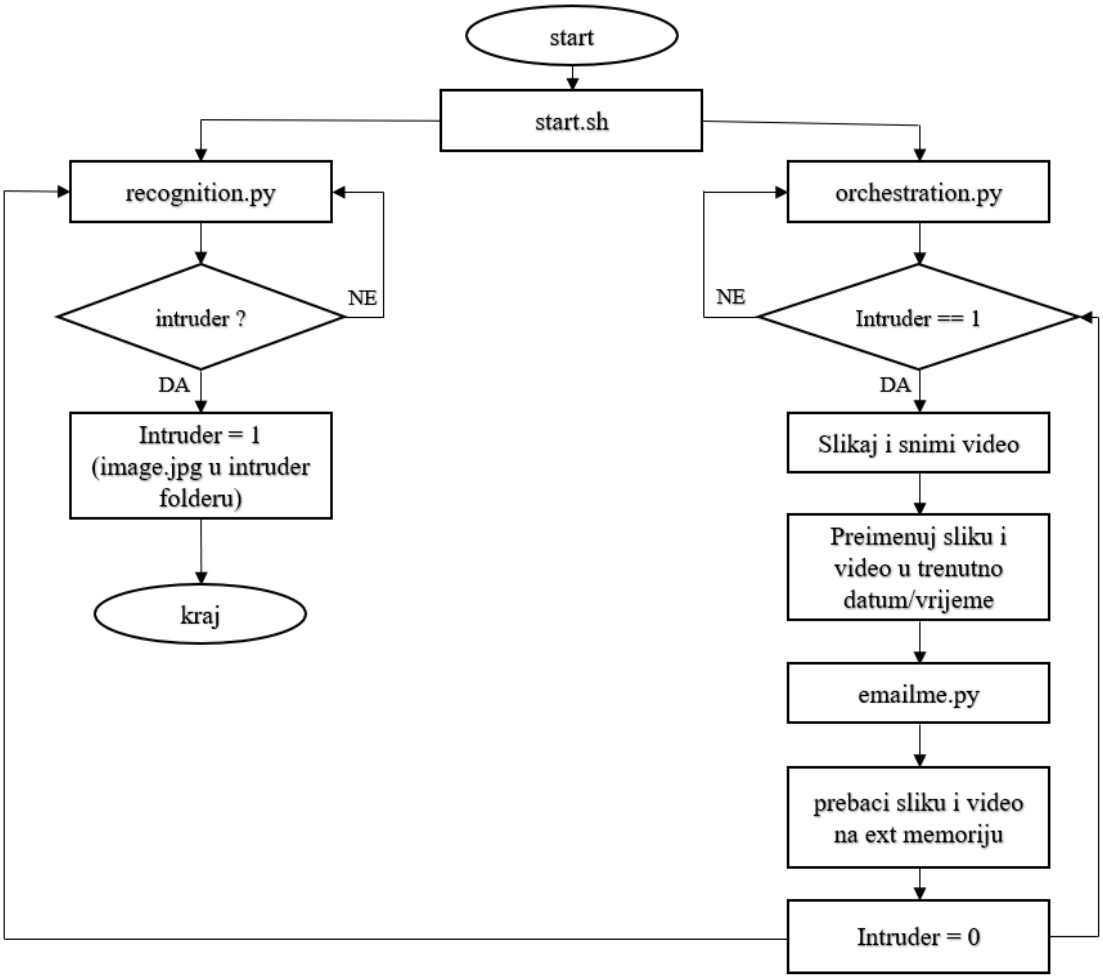
- 3 = Uradi *reboot* (restart *raspberry pi-a*),
- 4 = Pokreni sistem,
- 5 = Zaustavi sistem.

Rekli smo da modul **monitor.sh** između ostalog vodi računa o upisivanju statusa u bazu podataka, ali prije nego što uradi to, on takođe provjerava da li je korisnik sistema preko kontrolnog centra sistemu zadao neku od raspoloživih gore pomenutih komandi (3,4 ili 5) i ukoliko jeste, izvršiće je ali će i osvježiti trenutni status u zavisnosti od zadate komande. Metodologiju rada modula monitor.sh možemo vidjeti na Slici 2.




Slika 2: Algoritamski prikaz rada modula **monitor.sh**

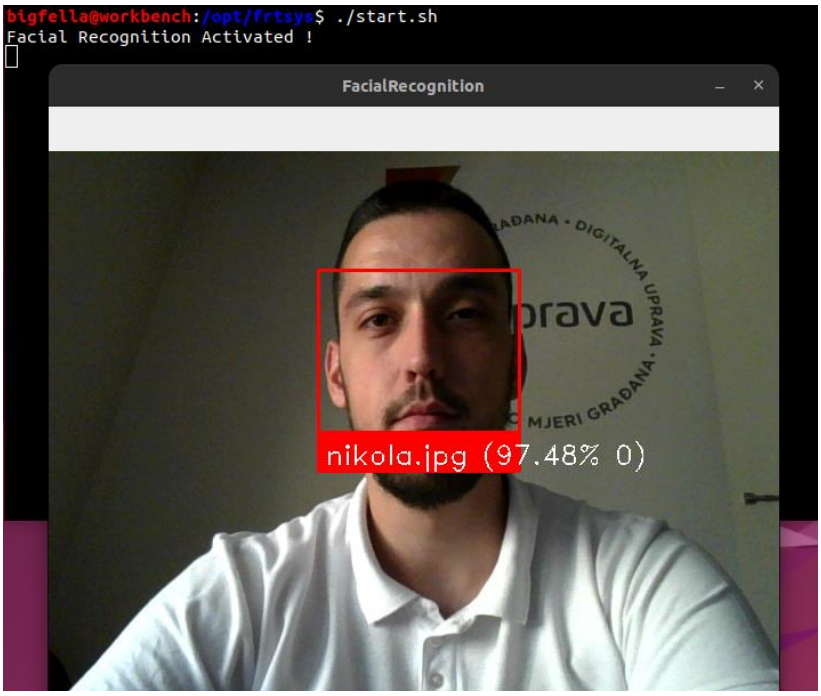
Kako bismo omogućili kontinualan rad recognition.py i orchestration.py modula, napravljena je *bash* skripta **start.sh** koja će se nakon svakog podizanja sistema automatski pokrenuti što će uzrokovati paralelno pokretanje dva gore navedena modula. Na Slici 3. možemo vidjeti malo detaljniji prikaz rada sistema.




Slika 3: Logika rada sistema

## 2.2 Vizuelni prikaz rada sistema

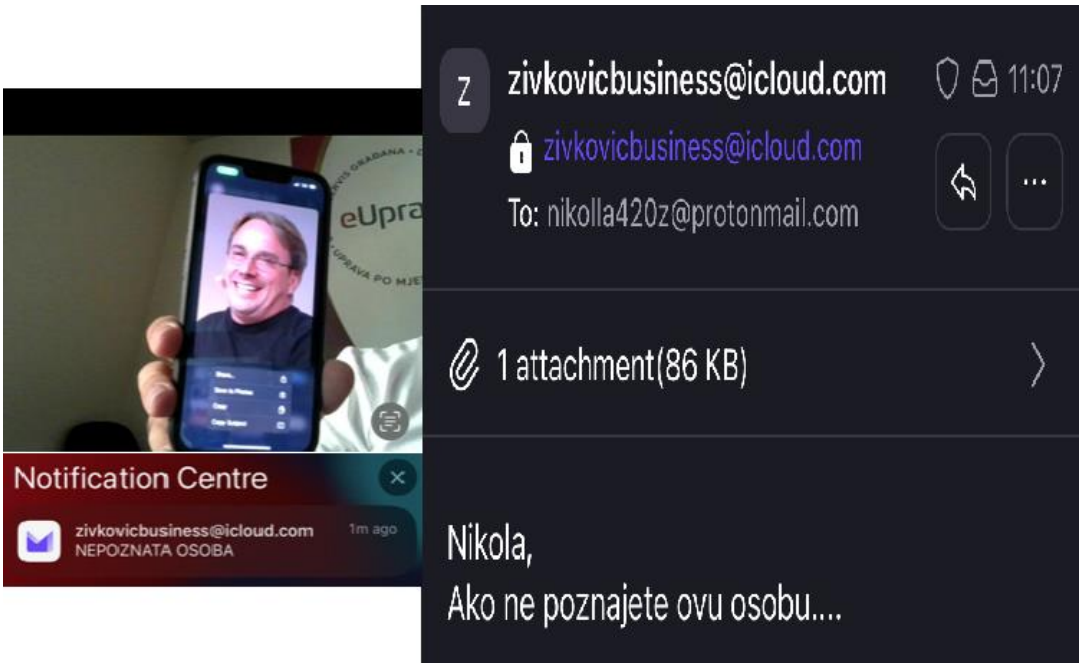
U ovom poglavlju ćemo vizuelno prikazati rad sistema. Na Slici 4. predstavljen je grafički interfejs modula recognition.py, dok se paralelno u pozadini izvršava i orchestration.py.




Slika 4: Vizuelni prikaz rada recognition.py modula

Kao što je prethodno objašnjeno, u trenutku kad recognition.py u trenutnom *frame-u* zabilježi neovlašćeno lice unutar zone nadzora, automatski stvara slikovni zapis *.jpg* formata, nakon čega završava sa radom, dok ostatak funkcija, tj. preimenovanje, skladištenje i slanje dokaza o nepoznatoj osobi obavlja *orchestration.py* modul, nakon čega ponovno pokreće *recognition.py* modul, što možemo vidjeti na Slikama 5, 6 i 7 respektivno.


Slika 5: Preimenovani i uskladišteni dokazi o nepoznatoj osobi

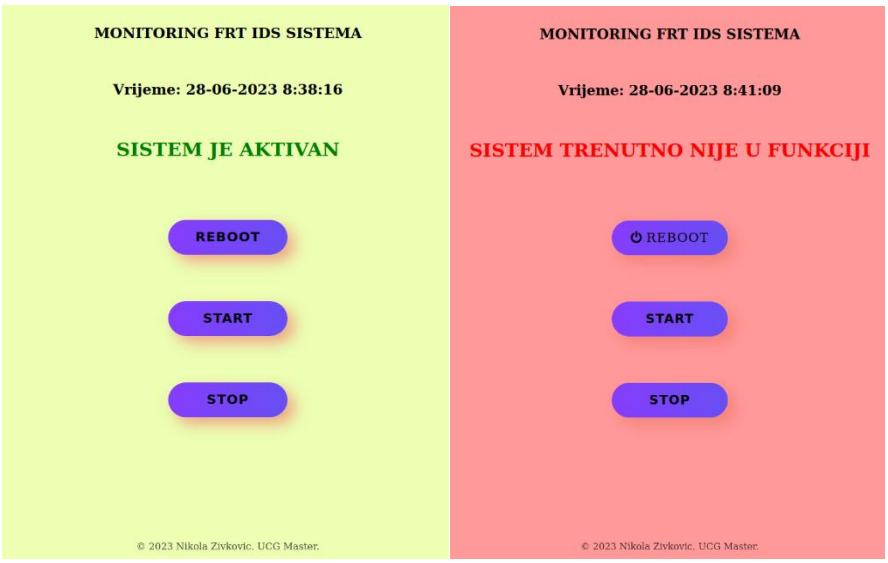



Slika 6: Primljeni email sa dokazima o nepoznatoj osobi

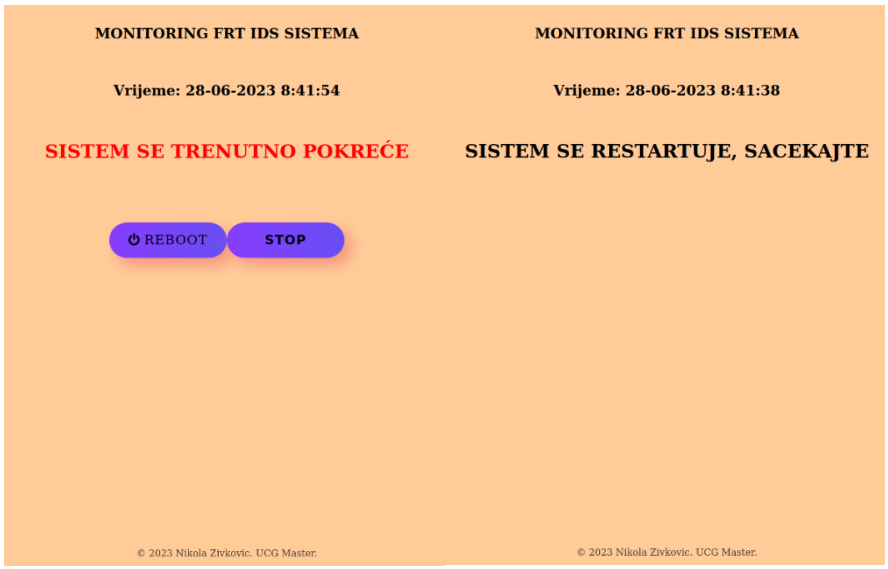
```
Facial Recognition Activated !

 Alert Sent to >>

 Nikola,
Ako ne poznajete ovu osobu....

Facial Recognition Activated !
```

Slika 7: orchestration.py ponovo pokreće recognition.py

*Web* aplikacija ima za cilj da u skoro realnom vremenu korisniku sistema pruži uvid u trenutno stanje istog, ovo se između ostalog postiže kroz ***hello.html*** koji će sadržaj prikazivati dinamički u zavisnosti od trenutnih stanja sistema zabilježenih u *MySql* bazi podataka. Na Slici 8 možemo vidjeti izgled *web* aplikacije kad je sistem aktivan, isključen, restartovan ili pokrenut.




(**a**) (**b**)

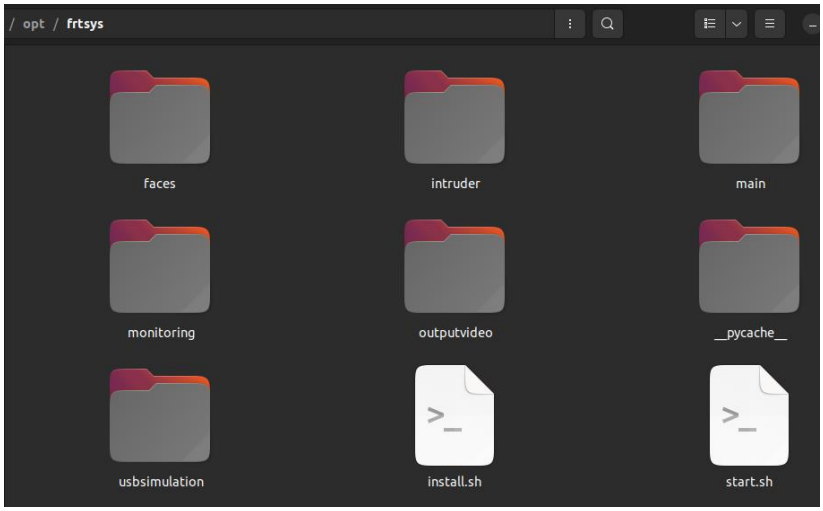



(**c**) (**d**)

Slika 8: Prikaz *web* aplikacije kada je sistem; (**a**) aktivan; (**b**) nije u funkciji; (**c**) u fazi pokretanja; (**d**) u fazi restartovanja.

## 2.3 Struktura projektnih datoteka

U prethodnom poglavlju smo se upoznali sa logikom i principom rada samog sistema i nadzorno-kontrolnog aspekta istog, u ovom poglavlju ćemo ukratko opisati strukture samih datoteka projekta. Na Slici 9. vidimo direktorijum u kom se sistem nalazi.




Slika 9: **/opt/frtsys** – projektni direktorijum

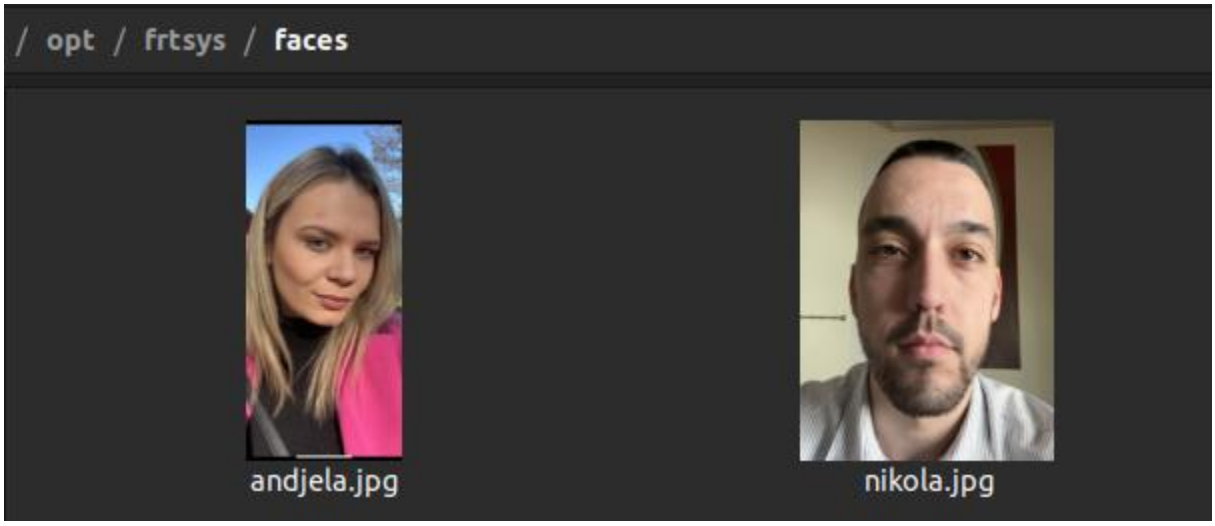
Lista direktorijuma i datoteka sistema :

- /opt/frtsys,
    - faces,
        - *dataset1.jpg*,
        - *dataset2.jpg*,
    - intruder,
    - main,
        - mailassets,
            - *getcontacts.py*,
            - *readtemplate.py*,
            - *message.md*,
            - *mycontacts.md*,
        - *recognition.py*,
        - *orchestration.py*,
        - *recordvideo.py*,
        - *emailme.py*,
    - monitoring,
        - *kameratest.py*,
        - *monitor.sh*,
        - *rebootme.sh*,
        - *startme.sh*,
        - *stopme.sh*,
        - *orchlog.md*,
        - *recognitionlog.md*,
    - outputvideo,

- usbsimulation,
- *install.sh*,
- *start.sh.*

U direktorijumu ***faces*** smiještamo *datasetove* tj, slike osoba koje želimo da sistem prepoznaje kao prijateljske. Na primjer, ukoliko bi se jedan ovako realizovan sistem implementirao na ulazu u stan ili u kuću, onda bismo željeni da sistem kao prijateljska lica prepoznaje ukućane tog domaćinstva dok bi ostale smatrao kao nepoželjne. U trenutku kad sistem prepozna takvu nepoželjnu osobu, aktivira niz drugih događaja koji omogućavaju istom da pošalje e-mail upozorenja i da do znanja korisniku sistema da je nepoželjna osoba uočena u nadzornoj zoni.

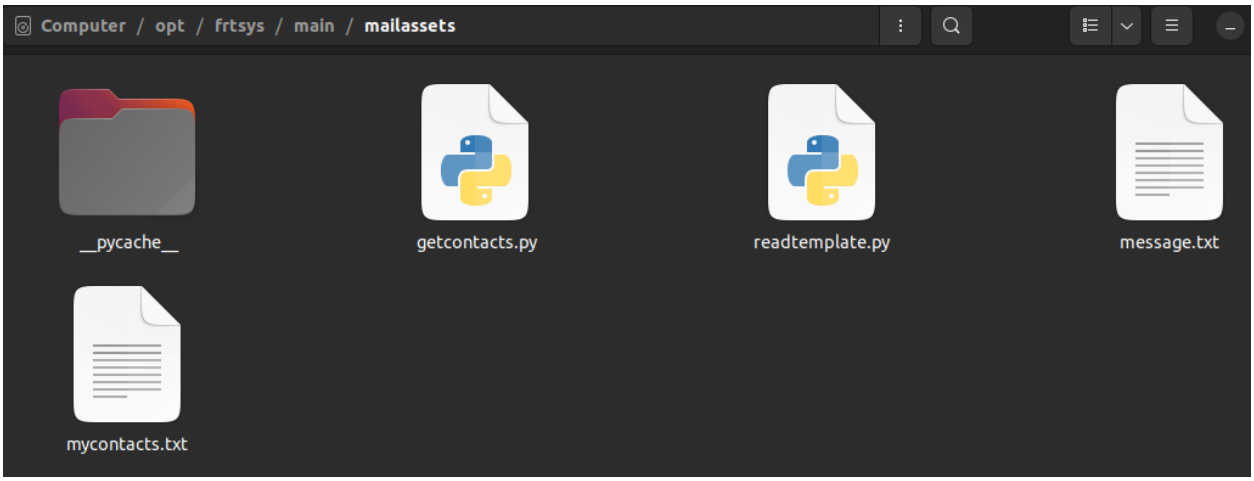
Primjer inicijalnog dataseta sa početka projekta vidimo na Slici 10.



Slika 10: Inicijalni dataset

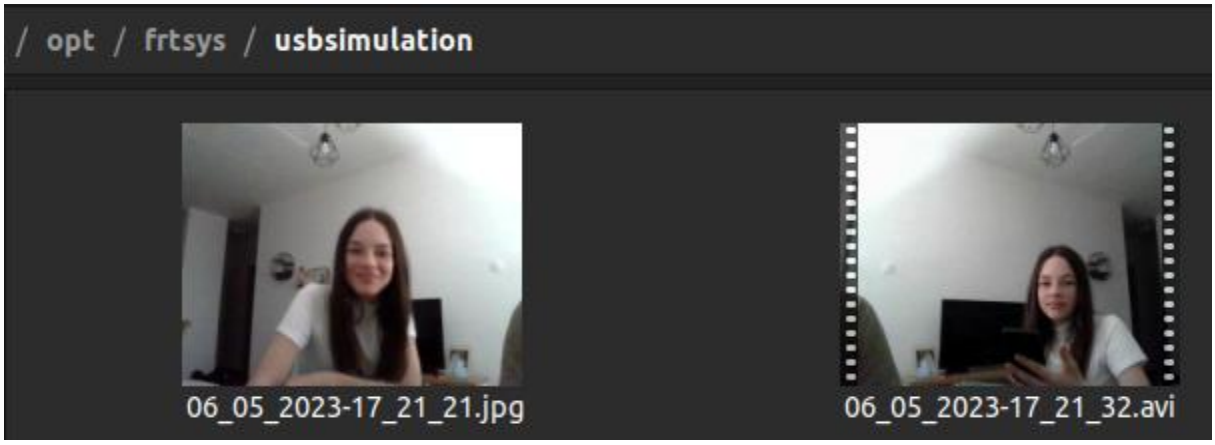
Direktorijum **intruder** je mjesto u kom se kreira ***image.jpg*** tj. slika nepoželjne osobe, ukoliko dođe do otkrivanja iste od strane recognition.py modula.

Glavni direktorijum projekta je **main**, a osim glavnih programa sistema u njemu se nalazi i **mailassets** (Slika 11, direktorijum u kom se može vršiti dodatno prilagođavanje sistema upozorenja, tačnije, možemo precizirati kakvu poruku i kome sve želimo da je pošaljemo ukoliko dođe do detekcije nepoželjne osobe unutar nadzorne zone.




Slika 11: **mailassets** za dodatnu konfiguraciju upozorenja

U **outputvideo** direktorijumu se inicijalno čuva video zapis snimljen prilikom detekcije nepoznate osobe, nakon čega zajedno sa slikom biva preimenovan u trenutni datum i vrijeme (*Eng. timestamp)* nakon čega se zajedno premiještaju i skladište na eksternoj memoriji, *USB sticku*, ili u našem slučaju simuliranom *USB sticku*, što možemo vidjeti na Slici 12.




Slika 12: Primjer uspješne detekcije i skladištenja dokaza u ***usbsimulation*** direktorijumu

Kao što smo vidjeli u prethodnom poglavlju, skripta **start.sh** osigurava pokretanje glavnih programskih modula recognition.py i orchestration.py, dok je **install.sh** modul zadužen za automatizaciju instalacije svih preduslova i biblioteka potrebnih za uspješno funkcionisanje sistema za detekciju neovlašćenog pristupa nadzornoj zoni.

### 2.3.1 Struktura datoteka *web monitoring* aplikacije

*Web* aplikacija za kontrolu i monitoring sistema za detekciju neovlašćenog pristupa je realizovana pomoću *Django framework-a.*

Lista direktorijuma i datoteka *Django web* aplikacije:

- /master/webmonitorapp,
    - monitorcenter,
        - settings.py,
        - urls.py,
    - playground,
        - templates,
            - hello.html,
        - static,
            - css,
                - style.css,
        - urls.py,
        - views.py,
    - manage.py,
    - models.py.

U **monitorcenter** direktorijumu se nalazi globalna konfiguracija i putanje potrebne za funkcionisanje *django web* aplikacije.

*Web* aplikaciji pristupamo na način što u *address bar-u* našeg pretraživača unesemo, zavisno od konfiguracije, domen ili ip adresu kao i port sa kojim smo povezali našu *web* aplikaciju, npr: kamera2stan.me ili *X.X.X.X*:8000.

## 2.4 Pregled potrebnog Software-a

U ovom poglavlju ćemo se osvrnuti na cjelokupan softwer potreban za realizaciju sistema za detekciju neovlašćenog pristupa kao i kontrolno-nadzornog centra realizovanog u vidu *web* aplikacije.

### 2.4.1 Python

Python je univerzalan i moćan programski jezik koji je posljednjih godina postao izuzetno popularan. Prepoznatljiv je po svojoj jednostavnosti i čitljivosti, a nudi širok spektar mogućnosti i funkcionalnosti koje ga čine jezikom koji se često koristi.

Jedna od glavnih prednosti *pythona* jeste ustvari njegova jedostavna sintaksa. Za razliku od nekih drugih programskih jezika, python stavlja akcenat na čitljivost koda, čineći ga gotovo kao pisanje na običnom jeziku. Ova karakteristika ne samo da pojednostavljuje proces

programiranja i pisanja koda, već i olakšava održavanje istog. Sintaksa pythona omogućava programerima da izraze svoje ideje i algoritme na koncizan i elegantan način.

Python podržava više paradigmi programiranja, uključujući proceduralno, objektno orjentisano i funkcionalno programiranje. Ova fleksibilnost omogućava programerima da odaberu najprikladniji pristup u zavisnosti od specifičnih potreba projekta. Bilo da je potrebno kreirati male skripte, napraviti *web* aplikacije, izvršiti analizu podataka ili razviti složene softverske sisteme, python može da prihvati različite stilove programiranja i da obezbjedi neophodne alate i biblioteke.

Jedna od prednosti pythona leži u njegovoj opsežnoj standardnoj biblioteci, koja nudi širok spektar modula za različite svrhe. Standardna biblioteka uključuje module za rukovanje datotekama, mrežne operacije, regularne izraze (*RegEx*), serijalizaciju podataka i još mnogo toga.

Osim toga, pythonov menadžer paketa, **pip**, omogućava lak pristup ogromnom ekosistemu biblioteka i okvira trećih strana, omogućavajući programerima da iskoriste postojeća rješenja i ubrzaju svoj razvojni proces.

Ove biblioteke pokrivaju različite domene kao što su *web* razvoj, naučno računanje, analiza podataka, mašinsko učenje, vještačka inteligencija i mnoge druge. Na primjer, biblioteke poput *NumPy* i *Pandas* nude moćne alate za numeričke proračune i manipulaciju podacima, dok *TensorFlow* i *PyTorch* pružaju okvire za izgradnju i obuku dubokih neuronskih mreža.

Univerzalnost pythona se prostire izvan biblioteka i modula. Može se koristiti za različite vrste aplikacija, počevši od *web* razvoja do pisanja skripti, naučnih istraživanja i automatizacije. Sa okvirima poput Djanga i Flaska, python je postao često korišćen jezik u razvoju *web* aplikacija.

Ovi okviri pružaju programerima neophodne alate i komponente za izgradnju robusnih ali i skalabilnih veb aplikacija. Dodatno, sposobnost skriptovanja kroz python čini ga odličnim izborom za automatizaciju repetitivnih zadataka i upravljanje sistemskim operacijama.

Pythonova međuplatformska kompatibilnost je još jedna prednost. Može da radi na različitim operativnim sistemima, uključujući Windows, macOS i Linux, što ga čini veoma dostupnim programerima. Pythonov interpreter i mogućnost pisanja koda nezavisnog od

platforme omogućavaju programerima da kreiraju aplikacije koje se mogu implementirati u različitim okruženjima bez većih modifikacija.

Python takođe ima ogromnu zajednicu koja konstantno raste i pruža podršku. Python zajednica je poznata po svojoj inkluzivnosti, spremnosti da pomogne kao i aktivnom doprinosu u razvoju jezika. Posvećenost zajednice dovela je do stvaranja brojnih resursa, tutorijala i dokumentacije koji olakšavaju učenje i pružaju pomoć kako početnicima tako i iskusnim programerima.

Dolazimo do zaključka da je python programski jezik koji je postao ključan u svijetu tehnologije. Njegova jednostavnost, čitljivost i fleksibilnost čine ga idealnim izborom za sve vrste softverskih projekata [23-24].

### 2.4.2 OpenCV

OpenCV (*Eng. Open Source Computer Vision*) je biblioteka otvorenog koda koja sadrži sveobuhvatan skup alata i funkcija za zadatke pri obradi slika i video zapisa, a u našem slučaju, zajedno sa bibliotekom *face_recognition* je od velike važnosti za funkcionisanje sistema detekcije neovlašćenog pristupa.

Jedna od ključnih prednosti OpenCV pythona je njegova opsežna kolekcija funkcija i algoritama. Nudi širok spektar mogućnosti, uključujući obradu slike i videa, otkrivanje i praćenje karakteristika, prepoznavanje objekata i mašinsko učenje. Ove funkcionalnosti su ključne za različite aplikacije kao što su robotika, nadzor i slično.

OpenCV pruža jednostavan i intuitivan interfejs za rad sa slikama i video zapisima. Omogućava lako čitanje, pisanje i manipulisanje slikama i video materijalima. Biblioteka podržava različite formate slikovnih datoteka, što olakšava import i export slika iz različitih izvora.

OpenCV se dobro integriše sa drugim popularnim bibliotekama i okvirima, kao što su *NumPy* i *TensorFlow*. Na primjer, korišćenjem *NumPy* nizova, možemo efikasno obrađivati slike i izvoditi složene matematičke operacije nad vrijednostima piksela, dok integracija sa *TensorFlow-om* omogućava izradu naprednih sistema za prepoznavanje slika [25-26].

### 2.4.3 Face_recognition

*Face_recognition* je *python* biblioteka koja pruža različite metode za prepoznavanje lica i detekciju karakteristika lica, a izgrađena je na principima *dlib*, *NumPy* i *Cython* biblioteka i omogućava raznovrse akcije nad licima.

Glavna karakteristika *face_recognition* biblioteke je njena sposobnost prepoznavanja i identifikacije lica na slikama i video zapisima. Koristeći modele dubokog učenja izračunava 128-dimenzionalnu mapu, tj. signaturu lica, za svako lice na slici. Ove signature predstavljaju jedinstvene karakteristike svakog lica te iz tog razloga omogućavaju efikasno poređenje i identifikaciju svih lica.

Proces prepoznavanja lica u *face_recognition* modulu sastoji se od nekoliko koraka. Prvo, biblioteka detektuje lica na slici koristeći *dlib* model detekcije lica, koji se zasniva na karakteristikama histograma orijentisanih gradijenta (eng. *Histogram of Oriented Gradients - HOG*) [27] Jednom kada su lica otkrivena, biblioteka koristi prethodno istrenirani model duboke konvolucione neuronske mreže (eng. *Convolutional Neural Network - CNN*) za izračunavanje 128-dimenzionalnih signatura za svaki region lica. Ove signature se zatim koriste za upoređivanje lica.

Biblioteka *face_recognition* takođe sadrži metode za otkrivanje karakteristika lica, kao što su lociranje položaja orijentira lica, uključujući oči, nos, usta i bradu. Ova funkcija omogućava obavljanje različitih zadataka kao što su analiza izraza lica, prepoznavanje pogleda i slično.

Još jedna prednost ove biblioteke jeste njena brzina i efikasnost. Biblioteka koristi optimizovani *C++* kod, čineći je znatno bržom od nekih drugih biblioteka za prepoznavanje lica dostupnih u *pythonu*. Ova brzina je posebno ključna za aplikacije koje rade u realnom vremenu kao što je slučaj sa našim sistemom.

*Face_recognition* podržava različite formate slika, što ga čini kompatibilnim sa uobičajnim tipovima datoteka kao što su *JPEG* i *PNG*. Osim toga, biblioteka može obraditi slike s više lica, pa čak i prepoznati lica u pretrpanim ili prepunim scenama, zahvaljujući svojim kompleksnim algoritmima za detekciju i prepoznavanje lica.

Biblioteka je dobro dokumentovana, sa dostupnom opsežnom dokumentacijom i primjerima koda [28].

### 2.4.4 Dlib

Dlib je sveobuhvatna biblioteka koja nudi širok spektar alata i algoritama za mašinsko učenje, kompjutersku grafiku i obradu slika. Takođe, pruža širok spektar alata za mašinsko učenje, uključujući (*eng. Support Vector Machine – SVM*), k-najbližih susjeda (eng. *k-nearest neighbors - k-NN*) ali i druge algoritme za klasifikaciju.

Mogućnosti mašinskog učenja biblioteke omogućavaju kreiranje prilaogđenih klasifikatora i modela za specifične zadatke i potrebe, pa čak i biblioteka kao što je slučaj sa *face_recognition* [29].

### 2.4.5 MySqlConnector-Python

Mysql-connector-python je biblioteka koja pruža interfejs za povezivanje i interakciju sa *MySQL* bazama podataka. Biblioteku je razvio i održava *MySQL* tim u *Oracleu,* a ona omogućava lako povezivanje sa *MySQL* bazama podataka, izvršavanje *SQL* upita i efikasno upravljanje operacijama nad bazom podataka. Aktivno se održava i ažurira, osiguravajući da ostane kompatibilna s najnovijim verzijama *pythona* i *MySQL*-a.

Jedna od glavnih karakteristika *mysql-connector-pythona* je njegova jednostavnost upotrebe. Biblioteka pruža jednostavan *API* koji pojednostavljuje proces povezivanja na *MySQL* bazu podataka, specificirajući adresu baze podataka, korisničko ime, lozinku i druge potrebne parametre.

*Mysql-connector-python* nudi podršku za više metoda povezivanja, uključujući standardne *TCP/IP* konekcije, kao i modul skladištenja konekcija (*eng. Connection Pooling*), koji omogućava efikasno rukovanje višestrukim vezama, smanjujući troškove stalnog uspostavljanja i zatvaranja veze za bazom podataka, što može biti iskorišćeno prilikom ekspanzije sistema i uvođenja većih i kompleksnijih baza podataka kao i eventualno kompleksnijih upita što bi u mnogome resursno rasteretilo sistem i ubrzalo vrijeme potrebno za ivršavanje upita.

Biblioteka pruža efikasno rukovanje greškama i izvještavanje o izuzetcima, olakšavajući identifikaciju i rješavanje problema sa upitima ili samom bazom podataka.

*Mysql-connector-python* je pouzdana biblioteka, bilo da je u pitanju projekat manjeg obima ili veliki produkcioni sistem, *mysql-connector-python* pruža neophodne funkcionalnosti za efikasno i sigurno upravljanje operacijama nad bazom podataka [30].

### 2.4.6 Smtplib

*Smtplib* je python biblioteka koja pruža jednostavan i efikasan način za slanje elektronske pošte koristeći jednostavan mail transfer protokol (*eng. Simple Mail Transfer Protocol – SMTP*). Uz *smtplib* lako se može integrisati funkcionalnost slanja elektronske pošte unutar *python* aplikacija, bilo da se radi o slanju automatizovanih obavještenja, upozorenja ili potvrde kreiranja naloga i slično.

Bibliteoka dolazi instalirana uz *python*, tako da nema potrebe naknadno je instalirati. Da bi koristili biblioteku, prvo je moramo importovati nakon čega deklarišemo adresu *SMTP* servera, broj porta na kom isti sluša za nadolazeće konekcije kao i potrebne faktore autentifikacije. U našem slučaju faktori autentifikacije su korisničko ime i *third-party-authentication-token*, koji generišemo na samom *mail* serveru.

Glavni metod koji se koristi za slanje elektronske pošte je *sendmail*(). Ovoj metodi kao argumente prosleđujemo adresu elektronske pošte pošiljaoca, adresu elektronske pošte primaoca kao i sadržaj e-pošte.

*Smtplib* omogućava i dodavanje priloga, tj. datoteka unutar tijela e-pošte. Koristeći višenamjensku internet poštansku ekstenziju (*eng. Multipurpose Internet Mail Extensions – MIME*) omogućili smo prilaganje slike neovlašćenog lica unutar elektronske poruke.

Takođe, moguće je i korišćenje sigurnih veza putem SSL/TLS protokola, što ga čini sigurnom opcijom za slanje osjetljivih informacija. Sigurna veza se konfiguriše unutar *SMTP_SSL* klase, a pokreće putem *starttls()* metode [31].

### 2.4.7 Bash

Bash, skraćenica od “Bourne Again Shell” je komandi interfejs (eng. *command-line interface – CLI)* i jezik za pisanje skripti u *Linux* operativnom sistemu. On je podrazumjevani komandni interfejs za mnoge *Linux* distribucije i servere.

Jedna od ključnih prednosti Basha je njegova svestranost. Omogućava korisnicima da izvršavaju različite sistemske komande, manipulišu datotekama i direktorijumima, upravljaju

procesima i obavljaju širok spektar administrativnih zadataka direktno iz komandne linije. Bash pruža širok spektar ugrađenih komandi i uslužnih programa, kao i mogućnost kreiranja prilagođenih skripti, što ga čini moćnim alatom za interaktivnu upotrebu i automatizaciju.

Bash skripte su u suštini nizovi komandi napisanih u običnom tekstualnom fajlu sa ekstenzijom "*.sh*". Ove skripte se mogu izvršiti direktno, pružajući način za automatizaciju složenih zadataka i izvođenje operacija koje se ponavljaju. Bash skripte se široko koriste za administraciju sistema, implementaciju softvera ili automatizaciju sistema.

Bash jezik podržava varijable, omogućavajući korisnicima da skladište i manipulišu podacima unutar skripti. Varijable su fleksibilne i mogu da sadrže nizove, cijele brojeve i druge tipove podataka. Uz to, Bash nudi različite kontrolne strukture, uključujući uslovne naredbe (*if-else*), petlje (*for, while*), što omogućava kreiranje složenih i logički vođenih skripti.

Još jedna vrijedna karakteristika Basha je njegova sposobnost rukovanja argumentima unutar komandne linije. Prilikom pokretanja Bash skripte, korisnici mogu prenijeti argumente skripti, što doprinosi fleksibilnosti same skripte, čineći ih prilagodljivim različitim scenarijima.

Bash podržava preusmjeravanje ulazno izlaznih podataka i parametara (eng. *input-output – I/O).*

Za složenije zadatke, Bash omogućava upotrebnu spoljnih alata, kao što su *sed, awk, grep* i drugi. Ovo omogućava Bashu da iskoristi moć ovih alata za efikasno obavljanje obrade teksta, upoređivanje uzoraka ili ekstrakciju i manipulaciju podacima i datotekama što upravo i radimo unutar samog sistema prilikom provjere statusa istog.

Iako je Bash moćan i fleksibilan jezik, vrijedno je napomenuti da su složeni i resursno intenzivni zadaci često prikladniji za druge programske jezike kao što su *Python*, *Perl* ili *Ruby*. Za brze jednolinijske zadatke ili jednostavnu automatizaciju, Bash ostaje neprocjenjiv alat za sistemske administratore, programere i *Linux* entuzijaste [32].

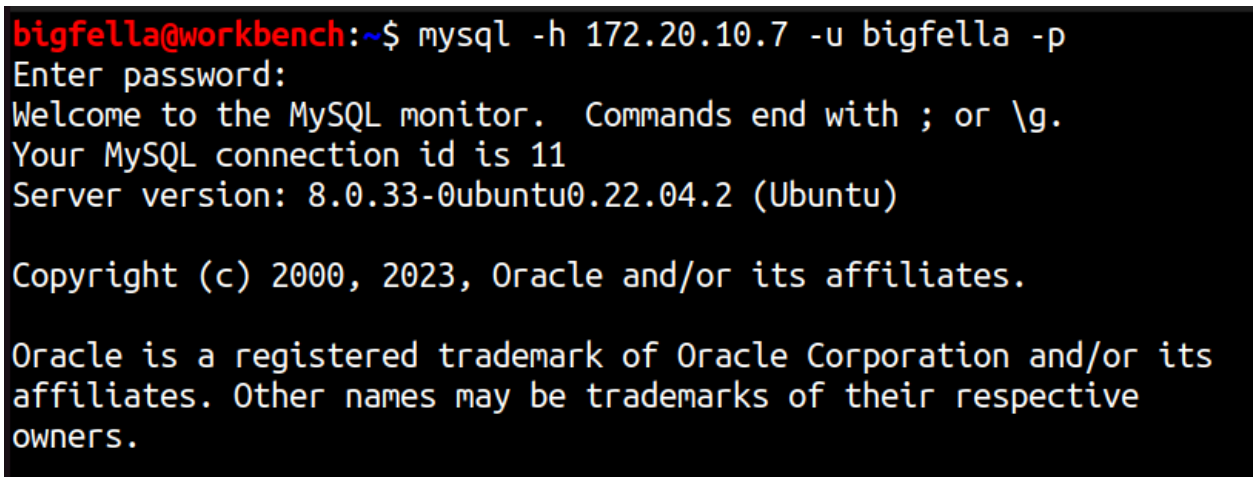
### 2.4.8 MySql

*MySQL* je popularan i široko rasprostranjen sistem za upravljanje relacionim bazama podataka otvorenog koda *(eng. Relational Data Base Management System – RDBMS)* koji je razvio Oracle Corporation. MySQL je jedna od najčešće korišćenih baza podataka u *web* aplikacijama i poznata je po svojoj jednostavnosti, skalabilnosti i performansama.

Nudi mogućnost upita kroz upotrebu jezika struktuiranih upita (*eng. Structured Query Language – SQL* ) i omogućava korisnicima da izvode operacije kao što su SELECT, INSERT, UPDATE i DELETE, itd.

MySQL je veoma skalabilan, što ga čini pogodnim i za male i za velike projekte i aplikacije. Može da obrađuje značajnu količinu podataka i podržava veliki broj istovremenih korisnika, što ga čini poželjnim izborom za *web* aplikacije [33].

Za potrebe našeg sistema smo koristili komandni interfejs MySQL baze podataka što možemo vidjeti na Slici 13. iako postoji više razvijenih grafičkih interfejsa koji omogućavaju vizuelizaciju baze podataka kao i grafičko i intuitivno upravljanje istima.




Slika 13: MySQL CLI okruženje

MySQL se lako integriše sa popularnim programskim jezicima, kao što smo prethodno naveli, u našem projektu, integraciju sistema i baze podataka uspješno smo realizovali upotrebnom *MySql-connector-python* biblioteke.

### 2.4.9 Ubuntu

Ubuntu je popularna *Linux* distribucija otvorenog koda, bazirana na *debianu*. To je jedan od najčešće korišćenih i subjektivno najprijatnijih *Linux* operativnih sistema, poznat po svojoj jednostavnosti, redovnim ažuriranjima i velikoj podršci zajednice.

Jedna od ključnih karakteristika *ubuntua* je njegov fokus na upotrebljivost i pristupačnost. Distribucija dolazi sa desktop okruženjem prilagođenim korisniku, obično GNOME (eng. *GNU Network Object Model Environment*), ili *x11*, koje pruža intuitivno i poznato okruženje za nove

ali i iskusne korisnike. Ubuntuov dizajn naglašava jednostavnost i efikasnost, što korisnicima olakšava navigaciju i obavljanje zadataka s lakoćom.

Ubuntuov softverski ekosistem je ogroman i raznolik. Uključuje širok spektar unaprijed instaliranog softvera kao što je LibreOffice (kancelarijski paket), Firefox (*web* pretraživač), Thunderbird (klijent e-pošte) i razne multimedijalne aplikacije. Ubuntu softverski centar pruža korisničko grafičko okruženje za instaliranje, ažuriranje i upravljanje softverskim paketima, što olakšava proširivanje funkcionalnosti sistema.

Kao Linux distribucija, *ubuntu* ima korist od ogromnih Linux softverskih bazi podataka, tj. apt-repozitorijuma dajući korisnicima pristup hiljadama besplatnih softverskih paketa otvorenog koda.

Ubuntu prati redovan ciklus izdavanja, s novim verzijama koje se obično objavljuju svakih šest mjeseci. Svako novo izdanje se ažurira narednih devet mjeseci. Uz to, izdanja za dugoročnu podršku (eng. *Long Term Support - LTS*) objavljuju se svake dvije godine, uz produženje podrške na pet godina. *LTS* izdanja su posebno popularna za implementaciju servera i sistema kritične infrastrukture, jer nude proširenu stabilnost i održavanje.

Za potrebe našeg sistema, tj. *web* aplikacije na virtuelnoj mašini u lokalnoj mreži, iskoristili smo i instalirali Ubuntu 22.04 LTS verziju, sa desktop okruženjem kako bismo lakše vršili izmjene u slučaju potrebe za proširivanjem sistema i kako bismo imali bolju preglednost izvornog koda i aplikativnih direktorijuma. Za produkcione potrebe, sasvim je u redu i poželjno koristi CLI verziju operativnog sistema po želji, kako bismo dobili što bolje performanse i izbjegli nepotrebni utrošak hardverskih resursa.

Još jedan značajan aspekt Ubuntua je njegova posvećenost sigurnosti. Redovna sigurnosna ažuriranja se objavljuju kako bi se riješile trenutne ranjivosti, a sistem održao bezbjednim [34].

Osim Ubuntua, *Debian* i *CentOs* su takođe pogodni operativni sistemi za hostovanje *web* aplikacija.

### 2.4.10 Django Framework

Django je *web* okvir otvorenog koda, napisan u *pythonu* koji se zasniva na *Model-View-Template* arhitekturi.

Dizajniran je tako da pojednostavi i ubrza proces izgradnje *web* aplikacije tako što nudi skup alata i gotovih funkcija.

Neke od glavnih karakteristika django okvira [35]:

1. *Model-View-Template - Model* upravlja operacijama nad bazom podataka, *view* upravlja prezentacijom podataka i korisničkim interfejsom, dok je *template* odgovoran za dinamičko prikazivanje *HTML-a.*
2. Objektno relaciono mapiranje (eng. *Object-Relational Mapping – ORM)* – omogućava komunikaciju sa bazom podataka koristeći *python* klase i metode. Ova funkcionalnost pojednostavljuje upite baze podataka i smanjuje rizik od *SQL injection-a*.
3. Administratorski interfejs – Djangov administratorski interfejs je automatski i prilagodljiv admin panel koji omogućava lako upravljanje sa podacima aplikacije bez potrebe za dodatnim pisanjem koda.
4. *URL* rutiranje – Django sadrži i fleksibilan način za mapiranje *URL-ova,* kako bi omogućili korisnicima intuitivnu navigaciju.
5. Template Engine – Omogućava kreiranje dinamičkih *web* stranica ugrađivanjem *python* koda unutar *HTML* šablona. Ova funkcija omogućava odvajanje logike od prezentacije, što rezultuje čistim i održivim kodom.

Django je pogodan za izgradnju kako malih, jednostavnih projekata tako i velikih i složenih *web* aplikacija. Njegova skalabilnost čini ga prilagodljivim rješenjem za različite slučajeve upotrebe.

Sve u svemu *django* je moćan i fleksibilan *web* okvir koji omogućava efikasno kreiranje skalabilnih *web* aplikacija. Njegov naglasak je na čistom i jednostavnom kodu za višestruku upotrebu, ugrađenim alatima i pridržavanju najboljih praksi, što ga čini odličnim izborom za projekte *web* razvoja svih veličina [36].

### 2.4.11 Raspberry Pi OS

*Raspberry Pi OS* je zvanični operativni sistem za *Raspberry Pi* računare sa jednom pločom. To je besplatni operativni sistem otvorenog koda baziran na *Linuxu, debian* porijekla i optimizovan za *Raspberry Pi* hardver. Dizajniran je da bude resursno lagan i efikasan.

Operativni sistem je posebno dizajniran za *Raspberry Pi ARM* arhitekturu, osiguravajući optimalne performanse i hardversku kompatibilnost.

Grafički korisnički interfejs (*eng. Graphical User Interface - GUI*) - *Raspberry Pi OS* dolazi sa desktop okruženjem koje uključuje grafički interfejs, što korisnicima olakšava navigaciju kroz sistem, pokretanje aplikacija i upravljanje datotekama.

Operativni sistem dolazi s raznim unaprijed instaliranim softverom, uključujući *LibreOffice* paket i nekoliko programskih alata, što ga čini pogodnim za obrazovne i produktivne zadatke.

*Raspberry Pi OS* uključuje *Raspberry Pi* alat za konfiguraciju, dostupan sa desktopa ili terminala, koji pruža jednostavan način za konfigurisanje sistemskih podešavanja, upravljanje perifernim uređajima i omogućavanje hardverskih funkcija.

Kao i svaki operativni sistem zasnovan na *Linuxu*, *Raspberry Pi OS* pruža potpuni pristup terminalu, omogućavajući korisnicima da izvršavaju komande, instaliraju pakete i izvršavaju zadatke na nivou sistema.

*Raspberry Pi OS* pruža biblioteke i alate za pristup opštenamjenskim ulazno izlaznim pinovima (*eng. General-Purpose Input/Output - GPIO*) *Raspberry Pi* ploče. Ova funkcija omogućava korisnicima interakciju s vanjskim uređajima i senzorima, što je čini popularnim izborom za IoT projekte.

Operativni sistem nudi podršku za zvanični *Raspberry Pi* modul kamere i zvanični *Raspberry Pi Touch Display*, omogućavajući korisnicima da snimaju slike i video zapise i kreiraju interaktivne projekte [37].

### 2.4.12 Hardware

Lista neophodnog hardware-a za realizaciju sistema za detekciju neovlašćenog pristupa unutar zone nadzora:

1. Raspberry Pi 4b 2GB+ RAM
2. Raspberry Pi 5,1V Power Adapter
3. Micro Sd kartica, 16GB+
4. Ethernet RJ45 kablo ili WiFi adapter
5. Usb ili Raspberry kamera.

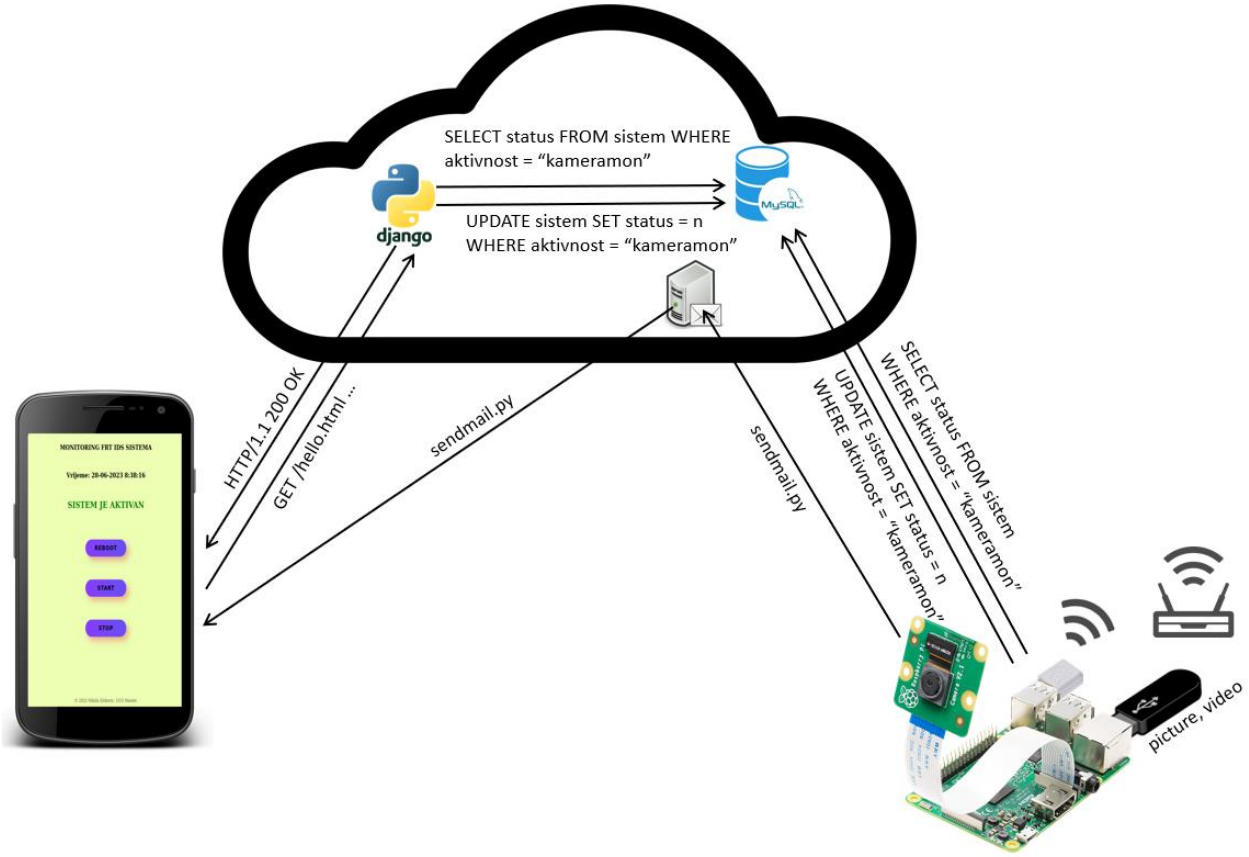
Na prethodnu listu potrebno je dodati monitor, hdmi kablo, miš i tastaturu kako bismo izvršili evaluaciju performansi sistema i otklonili potencijalne greške, nastale prilikom inicijalne konfiguracije sistema ukoliko do istih dodje.

Na slici 14. vidimo potrebnu hardversku opremu za realizaciju sistema.

Slika 14: Potreban hardver

# 3 DIZAJN I IMPLEMENTACIJA SISTEMA

Na slici 15. je prikazan dizajn sistema za detekciju neovlašćenog pristupa kao i upiti, zahtjevi i odgovori koji se realizuju preko nivoa mreže.




Slika 15: Dizajn sistema

U cilju dodatnog doprinosa i fokusa u dijelu optimizacije energetske i resursne iskorišćenosti, sistem je osmišljen na način da komunikaciju na nivou mreže ostvaruje samo na kratko i to u trenutku kada je nepoznata osoba detektovana u nadzornoj zoni kako bi kontaktirao *SMTP* server i proslijedio poruku upozorenja.

Sistem će takođe ostvariti mrežnu komunikaciju i onda kada se izvrši monitor.sh modul koji šalje jednostavan i brz upit ka *MySql* bazi podataka kako bi provjerio da li je korisnik zatražio pokretanje, zaustavljanje ili restartovanje sistema i u isto vrijeme ažurirao trenutno stanje sistema.

*Web* aplikativni server i *MySql* baza podataka su za potrebe ovog istraživanja implementirani na simuliranom serveru u oblaku (u daljem tekstu *eng. Cloud*) tj. računaru u lokalnoj mreži.

Način na koji je sistem kao cjelina dizajniran, omogućava podsistemu zaduženom za detekciju nepoznatih osoba u nadzornoj zoni koji je implementiran na *Raspberry Pi-u*, da nema dodirnih tačaka sa *web* serverom što u mnogome resursno rasterećuje isti.

U narednim poglavljima ćemo analizirati izvorni kod značajnih modula sistema.

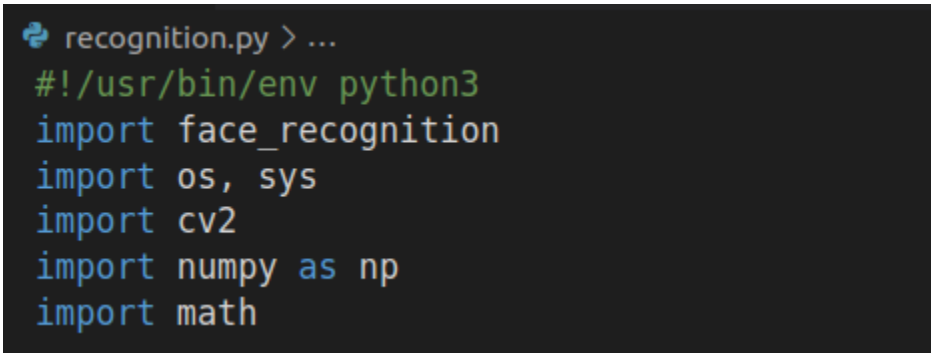
## 3.1 Analiza izvornog koda *frtsys* direktorijuma

U poglavlju **2.3** (Struktura projektnih datoteka) smo ostvarili uvid u direktorijume **frtsys**, tj. glavnog direktorijuma u kom se nalaze projektne datoteke i pod direktorijumi. U sledećim poglavljima ćemo analizirati i dodatno objasniti rad i logiku svih modula sistema.

### 3.1.1 Recognition.py

Logika iza funkcionisanja *recognition.py* modula preuzeta je iz [38] a dalje razvijena u skladu sa potrebama i zamisli sistema za detekciju neovlašenog lica unutar zone nadzora.

Na Slici 16. vidimo da recognition.py importuje biblioteke *face_recognition, os, sys, cv2 (OpenCV), NumPy, math.* Ove biblioteke pružaju funkcionalnosti potrebne za uspješno prepoznavanje lica, manipulaciju sistemskih datoteka i direktorijuma kao i raznih numeričkih operacija.




Slika 16: Biblioteke koje koristi recognition.py

Takođe, prva linija koda koja počinje sa "#!" naziva se "*Shebang*" linija i ona u *unix-like* sistemima označava putanju do interpretera sa kojim bi sve ispod nje trebalo da se pokrene, u

našem slučaju to je *python3*, tako da će prilikom poziva ovog programa, zbog uticaja *Shebang* linije, ona uvijek biti pokrenuta sa *python3 interpreterom*.

Na Slici 17. je deklarisana metoda *face_confidence* koja računa nivo pouzdanosti tj. tačnosti poklapanja signatura lica u trenutnom *frame-u* sa onim učitanim i enkodiranim signaturama iz *dataseta*, što će biti opisano u nastavku. Metoda će na kraju vratiti rezultat pouzdanosti u procentima tipa string.

```
# Helper
def face_confidence(face_distance, face_match_threshold=0.6):
    range = (1.0 - face_match_threshold)
    linear_val = (1.0 - face_distance) / (range * 2.0)

    if face_distance > face_match_threshold:
        return str(round(linear_val * 100, 2)) + '%'
    else:
        value = (linear_val + ((1.0 - linear_val) * math.pow((linear_val - 0.5) * 2, 0.2))) * 100
        return str(round(value, 2)) + '%'
```

Slika 17: face_confidence metoda koja računa procenat pouzdanosti poklapanja signatura lica

Nakon ovoga kreirana je klasa *FaceRecognition* koju vidimo na Slici 18.

*FaceRecognition* klasa sadrži *encode_faces* metodu za enkodiranje signatura lica iz *dataseta* koji se nalazi u direktorijumu *"faces"*, a koja se poziva od strane "*__init__*" konstruktora klase *FaceRecognition*.

```
class FaceRecognition:
    face_locations = []
    face_encodings = []
    face_names = []
    known_face_encodings = []
    known_face_names = []
    process_current_frame = True
    intruder = ""

    def __init__(self):
        self.encode_faces()

    def encode_faces(self):
        for image in os.listdir('/opt/frtsys/faces'):
            face_image = face_recognition.load_image_file(f"/opt/frtsys/faces/{image}")
            face_encoding = face_recognition.face_encodings(face_image)[0]

            self.known_face_encodings.append(face_encoding)
            self.known_face_names.append(image)
        print("Facial Recognition Activated !")
```

Slika 18: Prvi dio klase FaceRecognition

Na Slici 19. vidimo run_recognition metodu koja pokreće proces prepoznavanja lica koristeći kameru kao izvor. Ukoliko nije pronađen izvor video materijala, u ovom slučaju kamera, proces će se prekinuti.

```
def run_recognition(self):
    video_capture = cv2.VideoCapture(0)

    if not video_capture.isOpened():
        sys.exit('Video source not found...')

    while True:
        ret, frame = video_capture.read()

        # Only process every other frame of video to save time
        if self.process_current_frame:
            # Resize frame of video to 1/4 size for faster face recognition processing
            small_frame = cv2.resize(frame, (0, 0), fx=0.25, fy=0.25)

            # Convert the image from BGR color (which OpenCV uses) to RGB color (which face_recognition uses)
            rgb_small_frame = small_frame[:, :, ::-1]

            # Find all the faces and face encodings in the current frame of video
            self.face_locations = face_recognition.face_locations(rgb_small_frame)
            self.face_encodings = face_recognition.face_encodings(rgb_small_frame, self.face_locations)

            self.face_names = []
            for face_encoding in self.face_encodings:
                # See if the face is a match for the known face(s)
                matches = face_recognition.compare_faces(self.known_face_encodings, face_encoding)
                intruder = 1
                name = "Nepoznato"
                confidence = '?'
```

Slika 19: Drugi dio klase Face_Recognition

Unutar beskonačne while petlje, u kojoj se dešava konverzija formata, poređenje signatura u cilju utvrđivanja prisustva neovlašćene osobe a koja se može prekinuti pritiskom "Q" tastera na tastaturi, ili u našem slučaju i praktičnoj primjeni pouzdanijim načinom, a to je uništavanje samog procesa, što se postiže kroz nadzorno-kontrolni centar na *web* aplikaciji.

Unutar petlje se vrši konverzija BGR formata koji koristi *OpenCV* u RGB format koji *face_recognition* biblioteka koristi kako bi odredila da li postoji podudaranje. Takođe, obrađuje se svaki drugi *frame* kako bi se ubrzalo vrijeme potrebno za samu obradu.

Signature lica iz frame-a se upoređuju sa signaturama *dataseta* učitanim u face_encodings listi, nakon čega se ukoliko nemamo poklapanje, između ostalog promjenjiva intruder setuje na 1 što djeluje kao okidač za čuvanje slike trenutnog *frame-a* koja se skladišti u intruder folderu, nakon čega proces recognition.py uništava sam sebe, što vidimo na Slici 20.

```
                # Calculate the shortest distance to face
                face_distances = face_recognition.face_distance(self.known_face_encodings, face_encoding)
                best_match_index = np.argmin(face_distances)
                if matches[best_match_index]:
                    intruder = 0
                    name = self.known_face_names[best_match_index]
                    confidence = face_confidence(face_distances[best_match_index])
                # check if intruder was detected
                if intruder == 1:
                    img_name = "/opt/frtsys/intruder/image.jpg"
                    cv2.imwrite(img_name, frame)
                    exit()


                self.face_names.append(f'{name} ({confidence} {intruder})')
```

Slika 20: Treći dio klase Face_Recognition

Na Slici 21. vidimo dio koda pomoću kojeg se prikazuje rezultat poređenja signatura enkodiranih lica, kao i dio u kom se vrši pozivanje unutar *main* dijela programa.

```
            # Display the results
            for (top, right, bottom, left), name in zip(self.face_locations, self.face_names):
                # Scale back up face locations since the frame we detected in was scaled to 1/4 size
                top *= 4
                right *= 4
                bottom *= 4
                left *= 4

                # Create the frame with the name
                cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)
                cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)
                cv2.putText(frame, name, (left + 6, bottom - 6), cv2.FONT_HERSHEY_DUPLEX, 0.8, (255, 255, 255), 1)

            # Display the resulting image
            cv2.imshow('FacialRecognition', frame)

            # Hit 'q' on the keyboard to quit!
            if cv2.waitKey(1) == ord('q'):
                break

        # Release handle to the webcam
        video_capture.release()
        cv2.destroyAllWindows()


if __name__ == '__main__':
    fr = FaceRecognition()
    fr.run_recognition()
```

Slika 21: Kraj recognition.py modula

### 3.1.2 Orchestration.py

Kako bismo osigurali nesmetan rad sistema, osmišljen je kontrolni proces koji vodi računa o kontinuitetu rada sistema, nazvan orchestration.py. Modul orchestration.py je jedini modul na sistemu unutar *raspberry pi-a* koji nikad ne završava sa radom, tj. mora uvijek biti aktivan kako bi između ostalog pratio rad recognition.py modula i ponovno pokretao isti, a

takođe i vršio orkestraciju ostalih djelova sistema, tačnije pokretao ostale djelove sistema u trenucima kada je to potrebno i na taj način obezbjedio pravilno i sinhronizovano funkcionisanje sistema.

Na Slici 22. vidimo biblioteke *shutil, time, os i runpy* koje modulu orchestration.py omogućavaju rad i interakciju sa sistemskim procesima, preimenovanje datoteka, čitanje trenutnog datuma i vremena zarad stavljanja vremenskog pečata na novokreirane dokaze o neovlašćenom pristupu ukoliko se isti dogodio.

Takođe, vidimo i učitanu funkciju *sendmail* iz modula *emailme.py* koja će, kasnije ćemo vidjeti, za argument uzeti sliku neovlašćene osobe i istu proslijediti ostatku *emailme.py* modula koji će sliku iz argumenta dostaviti na željeni email, željenim osobama u vidu *attachment-a.*

Primjećujemo i promjenjive *imgdir, Imgsrc, Vidsrc* u kojima skladištimo apsolutne putanje do direktorijuma intruder kao i datoteka *image.jpg* (slika uljeza koja nastane u trenutku kada modul *recognition.py* zapazi nepoželjnu osobu), kao i output.avi (video snimljen od strane *recordvideo.py* modula koji se poziva unutar *orchestration.py*, takođe u trenutku kad *recognition.py* zapazi nepoželjnu osobu u kadru)

orchestration.py > ...

```
#!/usr/bin/env python3
import shutil
import time
import os
import runpy
from emailme import sendmail


# Paths,
imgdir = "/opt/frtsys/intruder"
Imgsrc = '/opt/frtsys/intruder/image.jpg'
Vidsrc = '/opt/frtsys/outputvideo/output.avi'
```

Slika 22: Biblioteke i deklaracija promjenjivih unutar orchestration.py modula

Na Slici 23. vidimo da se unutar beskonačne *while* petlje, u intervalu izvršavanja od svake dvije sekunde, provjerava da li je prethodno deklarisana promjenjiva *imgdir,* tj. apsolutna putanja do direktorijuma *intruder* prazna.

Ukoliko to nije slučaj, tačnije ukoliko postoji datoteka unutar *intruder* direktorijuma, a može se nalaziti samo *image.jpg* datoteka koja se kreira od strane *recognition.py* modula u trenutku kad nepoželjna osoba uđe unutar zone nadzora.

Dakle, ukoliko *intruder* direktorijum nije prazan, tačnije ukoliko jeste došlo do kreiranja *image.jpg* datoteke, modul *orchestration.py* preduzima niz akcija u cilju kreiranja i čuvanja dokaza, slanja mail obavještenja kao i ponovnog pokretanja *recognition.py* modula.

```
while True:

#Check if intruder was captured
    checkDir = os.listdir(imgdir)
    #If there is something (image.jpg) in intruder dir, trigger following
    if len(checkDir) != 0:
        time.sleep(1)
        #Start recordvideo.py
        runpy.run_path(path_name='/opt/frtsys/main/recordvideo.py')
        #Define Pics Label / timestamp of event occurance
        Imgdst = f'/opt/frtsys/usbsimulation/{time.strftime("%d_%m_%Y-%H_%M_%S")}.jpg'
        shutil.move(Imgsrc, Imgdst)
        time.sleep(1)
        #Send mail and parse intruder pic with timestamp as arg (attachment))
        sendmail(Imgdst)
        time.sleep(5)
        shutil.move(Vidsrc, f'/opt/frtsys/usbsimulation/{time.strftime("%d_%m_%Y-%H_%M_%S")}.avi' )
        os.system('python3 /opt/frtsys/main/recognition.py')
        # could check if image.jpg spawned again in meanwhile and delete for optimal

    time.sleep(2)
```

Slika 23 Funkcionalni dio orchestration.py modula unutar beskonačne while petlje sa dvosekundnim delayom.

Prva stvar koja se izvršava ukoliko jeste došlo do prepoznavanja nepoželjne osobe unutar nadzorne zone jeste snimanje video zapisa iste pokretanjem *recordvideo.py* modula korišćenjem metode *run_path*, biblioteke *runpy*.

Dok se snimanje video zapisa još uvijek odvija, u pozadini se izvršava skladištenje i preimenovanje *image.jpg-a* u trenutni datum i vrijeme radi kasnije lakše evidencije, nakon čega se tako preimenovana slika proslijeđuje *sendmail* metodi kao argument za email *attachment*.

Na kraju će modul preimenovati i video zapis nakon čega će i njega uskladištiti na eksternoj memoriji.

### 3.1.3 Emailme.py

Modul emailme.py je zadužen za slanje email notifikacije prilikom identifikacije nepoželjne osobe unutar zone nadzora korišćenjem *smtplib*, *email* i *os* biblioteka.

Modul se prilikom pozivanja autentifikuje, u našem slučaju *Apple-e iCloud SMTP* serveru, sa dodijeljenom email adresom i prethodno generisanim tokenom za aplikativno programabilni interfejs (u daljem tekstu ćemo koristiti *Eng. Application programming interface*) – API.

Na Slici 24. vidimo učitavanje (Eng. Multipurpose Internet Mail Extensions) – MIME, modula email biblioteke koji omogućava dodavanje multimedijalnog sadržaja. Takođe, vidimo i smiještanje autentifikacionih kredencijala unutar promjenjivih koje ćemo kasnije koristiti za autentifikaciju *SMTP* serveru.

```
emailme.py > ...
#!/usr/bin/env python3
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.image import MIMEImage
from mailassets.readtemplate import read_template
from mailassets.getcontacts import get_contacts
import os

MY_ADDRESS = 'zivkovicbusiness@icloud.com'
PASSWORD = '[redacted]'
```

Slika 24: Učitavanje biblioteka i autentifikacionih kredencijala unutar *emailme.py* modula

Takođe, učitane su i metode *read_template* i *get_contacts* koje vode računa o tome kome je potrebno dostaviti notifikaciju kao i o tome koji je to sadržaj koji je potrebno proslijediti.

Na Slici 25. vidimo *sendmail* metodu unutar *emailme.py* modula koja kao argument uzima apsolutnu putanju do slike detektovane nepoželjne osobe.

```
def sendmail(imagepath):
    names, emails = get_contacts('mycontacts.txt') # read contacts
    message_template = read_template('message.txt') # read message template

    # set up the SMTP server
    s = smtplib.SMTP(host='smtp.mail.me.com', port=587)
    s.starttls()
    s.login(MY_ADDRESS, PASSWORD)

    # For each contact from mycontacts send the email:
    for name, email in zip(names, emails):
        msg = MIMEMultipart()       # create a message

        # Swap $person_name with first row from mycontacts
        message = message_template.substitute(PERSON_NAME=name.title())

        # Prints message in term for test purpose
        print('\n Alert Sent to >>\n\n',message,'\n')

        # Read img
        with open(imagepath, 'rb') as f:
            img_data = f.read()

        # setup the parameters of the message
        msg['From']=MY_ADDRESS
        msg['To']=email
        msg['Subject']="NEPOZNATA OSOBA"

        # add in the message body
        msg.attach(MIMEText(message, 'plain'))

        # add attachment
        image = MIMEImage(img_data, name=os.path.basename(imagepath))
        msg.attach(image)

        # send the message via the server set up earlier.
        s.send_message(msg)
        del msg

    # Terminate the SMTP session and close the connection
    s.quit()

if __name__ == '__main__':
    sendmail()
```

Slika 25: *sendmail* metoda unutar *emailme.py* modula

Unutar sendmail metode vidimo da se pozivaju prethodno učitane metode *get_contacts* i *read_ template*, kojima se kao argument proslijeđuju relativna putanja do *mycontacts.txt* i *message.txt* datoteke, respektivno.

Kako bismo bolje razumjeli get_contacts i read_template metode korišćene u nastavku koda, osvrćemo se na izvorni kod modula unutar kojih iste i postoje.

Na Slici 26. vidimo princip rada *get_contacts* modula. Isti iščitava sadržaj *mycontacts.txt* datoteke liniju po liniju i u svakoj iteraciji uzima sadržaj prve kolone, tj. sadržaj prije prvog razmaka i dodaje ga na kraj liste *names*, a po istom principu dodaje sadržaj i u listu *emails*.

mailassets > getcontacts.py > ...

```python
def get_contacts(filename):

    names = []
    emails = []

    # append names to names and emails to emails, from each row of mycontacts.txt using .split method
    # pos0 being name and pos1 being email -- return both lists
    with open('/opt/frtsys/main/mailassets/mycontacts.txt', mode='r', encoding='utf-8') as contacts_file:
        for a_contact in contacts_file:
            names.append(a_contact.split()[0])
            emails.append(a_contact.split()[1])
    return names, emails
```

Slika 26: Izvorni kod *get_contacts* modula

Na Slici 27. prikazan je primjer sadržaja unutar *mycontacts.txt* datoteke.

mailassets > mycontacts.txt

```
Nikola nikolla420z@protonmail.com
xxx yyy@zzz.me
```

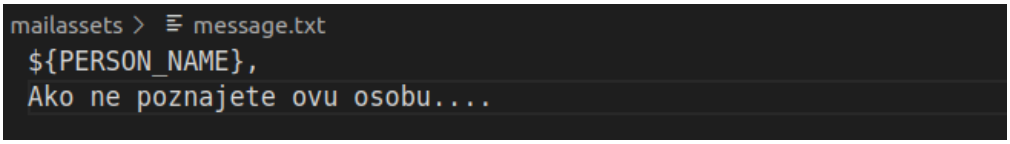
Slika 27: Primjer sadržaja *mycontacts.txt* datoteke

Na Slici 28. vidimo izvorni kod *read_template* metode unutar *readtemplate.py* modula, koja učitava sadržaj datoteke *message.txt* i isti vraće kao *template* objekat.

```python
from string import Template

def read_template(filename):

    #return template obj
    with open('/opt/frtsys/main/mailassets/message.txt', 'r', encoding='utf-8') as template_file:
        template_file_content = template_file.read()
    return Template(template_file_content)
```

Slika 28: Izvorni kod *read_template* modula

Na Slici 29. prikazan je primjer sadržaja unutar *message.txt* datoteke.




Slika 29: Primjer sadržaja *message.txt* datoteke

U nastavku sendmail metode vidimo da je potrebno konfigurisati željeni SMTP server (host, port i pristupne parametre). Nakon toga unutar *for* petlje je postignuto da svaka osoba kojoj se šalje email upozorenja dobije različitu poruku, tj. promjenjiva *${PERSON_NAME}* će se dinamički mijenjati u zavisnosti od sadržaja *mycontacts.txt* datoteke. U nastavku podešavamo parametre same poruke, naslov poruke, ko je šalje a ko prima, ali i dodajemo sliku neovlašćenog lica detektovanog unutar zone nadzora.

Nakon što pošaljemo email, prekidamo *SMTP* sesiju.

### 3.1.4 Recordvideo.py

U prethodnom dijelu ovog rada smo naveli da u trenutku kada je detektovano neovlašćeno lice unutar zone nadzora, pokreće se niz operacija a između ostalog pokrene se i izvršavanje *recordvideo.py* modula čiji je zadatak da napravi video zapis potencijalne uzurpacije nadzorne zone u *.avi* formatu, koji će nakon toga, orkestracioni modul sistema preimenovati u trenutni datum i vrijeme radi lakše evidencije kao i isti sačuvati na eksternoj memoriji.

Na Slici 30. vidimo izvorni kod *recordvideo.py* modula u kom deklarišemo vrijeme trajanja video zapisa (u našem slučaju tri sekunde što je dovoljno za analizu snimka u testnom okruženju, međutim u produkcionim uslovima moguće je i poželjno povećati vrijeme trajanja snimka), izvor video kamere kao i apsolutnu putanju mjesta na kom će se upisati video zapis nakon uspješnog završetka procesa snimanja.

Takođe prilikom snimanja video zapisa korišćenjem *cv2* biblioteke, potrebno je iskoristiti *flip* metodu koja vrši horizontalnu rotaciju, te na taj način dobijamo ispravno pozicioniran video zapis.

```python
recordvideo.py > ...
#!/usr/bin/env python3
import cv2
import time


# The duration in seconds of the video captured
capture_duration = 3
cap = cv2.VideoCapture(0)

fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('/opt/frtsys/outputvideo/output.avi',fourcc, 20.0, (640,480))

start_time = time.time()
while( int(time.time() - start_time) < capture_duration ):
    ret, frame = cap.read()
    if ret==True:
        frame = cv2.flip(frame, 1)
        out.write(frame)
    else:
        break

cap.release()
out.release()
cv2.destroyAllWindows()
```

Slika 30: Izvorni kod *recordvideo.py* modula

### 3.1.5 Kameratest.py

Nadzorno kontrolni dio predloženog sistema čini modul *kameratest.py* čiji je zadatak kako prijem i izvršavanje komandi zadatih od strane korisnika putem *web* aplikativnog grafičkog interfejsa, tako i aktivno praćenje i ažuriranje trenutnog stanja sistema.

U opisu orkestracionog dijela smo napomenuli da se sistem za detekciju neovlašćenih lica unutar zone nadzora automatski pokreće nakon uspješnog pokretanja operativnog sistema na kom je isti implementiran, dok se korišćenjem istog servisa (*cronjob-a*), svakog minuta obezbjeđuje pokretanje *monitor.sh* skripte, što vidimo na slikama 31 i 32 respektivno.

```
* * * * * /opt/frtsys/monitoring/./monitor.sh
@reboot sleep 15 && /opt/frtsys/./start.sh
```

Slika 31: *Cronjob* kojim se obezbjeđuje kontinuitet i monitoring sistema

```
toring > $ monitor.sh
#!/bin/bash

ps -aux | grep -v grep | grep recognition.py | tee /opt/frtsys/monitoring/recognitionlog;
ps -aux | grep -v grep | grep orchestration.py | tee /opt/frtsys/monitoring/orchlog;
python3 /opt/frtsys/monitoring/kameratest.py
```

Slika 32: Sadržaj *monitor.sh* skripte

Na Slici 32. vidimo način na koji se provjerava da li su procesi aktivni, a to je korišćenjem *ps* komande sa argumentima *aux* koja kao rezultat prikazuje sve trenutno aktivne korisničke procese, nakon čega rezultat filtriramo pomoću pajp karaktera " | " (*eng. Pipe*) i *grep* komande tako da prikaz rezultata svedemo samo na *recognition.py* i *orchestration.py*.

Ukoliko su procesi aktivni, njihov PID (*Eng. Process Identifier – PID*) ali i ostale informacije o njima će se upisati u *recognitionlog* ili *orchlog* datoteke. Ukoliko to nije slučaj, tj. ukoliko procesi nijesu aktivni, onda će sadržaj *log* datoteka ostati prazan, a ta informacija će nam kasnije biti od značaja.

Nakon što odradimo provjeru stanja procesa, pokrećemo *kameratest.py* modul koji vidimo na Slici 33.

```
oring > kameratest.py > ...
#!/usr/bin/env python3
import os
import time
import mysql.connector
import subprocess

time.sleep(1)

#check if file has any content inside, use for checking if process is alive.
def isitempty(path):
    return os.stat(path).st_size == 0

# connect to mysql db
mydb = mysql.connector.connect(
  host="[redacted]",
  user="bigfella",
  password="[redacted]",
  database="monitoring"
)
```

Slika 33: Princip rada kameratest.py modula

Unutar kameratest.py modula po prvi put koristimo *mysql.connector* biblioteku koja nam omogućava uspostavljanje konekcije sa MySql bazom podataka u cilju čitanja i pisanja podataka u/iz baze. Ovog puta ćemo za pokretanje drugih procesa koristiti biblioteku subprocess.

U prvim koracima kroz funkciju *isitempty* koja za argument uzima apsolutnu putanju do direktorijuma, vraćemo *boolean* vrijednost, tj. 0 ili 1 korišćenjem metode *.st_size* koja će provjeriti da li je direktorijum prazan i ukoliko jeste vratiti 1, a u suprotnom vratiće 0.

Takođe, metodi *.connect* biblioteke *mysql.connector* proslijedimo autentifikacione parametre koje prethodno kreiramo unutar same baze podataka u *MySql* okruženju.

Sledeće što ovaj modul radi, a što možemo vidjeti na Slici 34. jeste da izvršava *MySql* upit "*SELECT status FROM sistem WHERE aktivnost='kameramon'*" a rezultat upita skladišti u promjenjivoj *myresult*.

Nakon što rezultat bude uskladišten unutar *myresult* promjenjive, modul vrši provjeru trenutnog stanja sistema, tj. ispituje da li je korisnik u međuvremenu zatražio neku akciju od samog sistema.

```
#check if reboot,start,stop is requested
mycursor = mydb.cursor()
mycursor.execute("SELECT status FROM sistem WHERE aktivnost='kameramon'")
myresult = mycursor.fetchall()

#is reboot requested
if ((3,) in myresult):
    print('rebooting system...')
    mycursor = mydb.cursor()
    time.sleep(4)
    mycursor.execute("UPDATE sistem SET status = 1 WHERE aktivnost = 'kameramon'")
    mydb.commit() #todo- could add startup script that sets status to 1 when process starts
    subprocess.call('/opt/frtsys/monitoring/./rebootme.sh')

#is start requsted
elif ((4,) in myresult):
    print('Starting system...')
    mycursor = mydb.cursor()
    mycursor.execute("UPDATE sistem SET status = 1 WHERE aktivnost = 'kameramon'")
    mydb.commit()
    subprocess.call('/opt/frtsys/monitoring/./startme.sh')

#is stop requested
elif ((5,) in myresult):
    print('Stopping system...')
    mycursor = mydb.cursor()
    mycursor.execute("UPDATE sistem SET status = 0 WHERE aktivnost = 'kameramon'")
    mydb.commit()
    subprocess.call('/opt/frtsys/monitoring/./stopme.sh')
```

Slika 34: kameratest.py modul provjerava da li je korisnik zatražio akciju nad sistemu

Kao što smo već pomenuli pri opisivanju dizajna sistema, korisnik preko *web* aplikativnog grafičkog interfejsa od sistema traži akciju pritiskanjem na jedno od ponuđenih dugmića (Reboot, Start ili Stop). U trenutku kad korisnik pritisne jedno od ponuđenih dugmića stanje unutar baze podataka se mijenja na 3, 4 ili 5 respektivno u zavisnosti od izabrane opcije.

Ukoliko je prilikom provjere *kameratest.py* modula unutar *if* uslova vrijednost status kolone 3, to znači da je korisnik zatražio *reboot* sistema, što unutar prvog *if* uslova modul i radi, prvo će vratiti vrijednost status kolone na 1, nakon čega će pokrenuti *rebootme.sh* skriptu koja će *rebootovati* operativni sistem.

Po sličnom principu unutar *elif* uslova u nastavku koda vidimo da se provjerava da li je zatražen start ili stop sistema, ukoliko je to slučaj, zatražena akcija će biti izvršena, a stanje sistema ažurirano u zavisnosti od tražene akcije.

Na Slici 35. vidimo da ukoliko korisnik nije zatražio ništa od navedenih i mogućih akcija, tačnije ako se u rezultatu izvršenog upita ne nalaze vrijednosti statusa 3, 4 ili 5 onda program ide u *else* petlju kako bi utvrdio koje je zaista trenutno stanje sistema i kako bi kao rezultat toga, mogao da ažurira stanje sistema unutar baze podataka.

Prvo provjeravamo da li su recognitionlog i orchlog popunjeni sadržajem, a pošto je jedini očekivani sadržaj unutar njih putanja do procesa kao i *PID*, to znači da je u tom trenutku proces živ, a ukoliko je to slučaj sa obje log datoteke, to znači da sistem funkcioniše u potpunosti i u tom trenutku izvršavamo *UPDATE* upit ka bazi podataka koji će vrijednost status kolone promijeniti u 1.

Ukoliko je jedna od log datoteka prazna, to znači da taj proces nije aktivan i da sistem ne funkcioniše u potpunosti te da je potrebno preduzeti akciju, a stanje unutar status kolone mijenjamo u 0.

```
# if there are no requests, then just update current sys status
else:
    #if recognitionlog AND orchlog are active update to 1
    if isitempty("/opt/frtsys/monitoring/recognitionlog") == 0 and isitempty("/opt/frtsys/monitoring/orchlog") == 0 :
        mycursor = mydb.cursor()
        mycursor.execute("UPDATE sistem SET status = 1 WHERE aktivnost = 'kameramon'")
        mydb.commit()
        print("Sistem Funkcionise u potpunosti !")
    else:
        mycursor = mydb.cursor()
        mycursor.execute("UPDATE sistem SET status = 0 WHERE aktivnost = 'kameramon'")
        mydb.commit()
        print ("Sistem/Jedan dio sistema ne funkcionise kako treba, potreban restart !")
```

Slika 35: Provjera trenutnog stanja sistema unutar kameratest.py modula

Dakle, da zaključimo, *kameratest.py* je kontrolni modul koji se pokreće svakog minuta upotrebom *cronjob-a,* a ima za cilj da korisniku obezbijedi uvid u trenutno stanje sistema, na način što ažurira vrijednost kolone status u *MySql* bazi podataka.

Takođe, modul će, ukoliko je to korisnik zatražio, izvršiti restartovanje, zaustavljanje ili pokretanje sistema na način što će prije ažuriranja stanja sistema u bazi podataka, provjeriti trenutnu vrijednost kolone status, i ukoliko je ta vrijednost jednaka 3, 4 ili 5 modul će izvršiti zatraženu akciju.

Na kraju analize koda dijela sistema koji je implementiran na *raspberry pi-u*, važno je naglasiti i istaći prednosti upotrebe *python* i *bash* jezika.

Upoznati smo sa činjenicom da *raspberry-pi* koristi *raspi*-os *unix-like* operativni sistem koji je zasnovan na *debian* distribuciji.

Većina *linux* operativnih sistema dolazi instalirana sa *python* i *bash* okruženjem, pa je to slučaj i sa *raspi-os,* koji je kao takav najbolja opcija za ovaj i slične projekte koji zahtijevaju optimalnu resursnu iskorišćenost.

## 3.2 Konfiguracija i analiza Django frameworka

U prethodnim poglavljima smo imali uvid u izvorni kod dijela sistema koji se nalazi na *raspberry pi-u*, takođe vidjeli smo dizajn cjelokupnog sistema, što znači da do sada već imamo znanje o principu rada istog.

Za potrebe ovog istraživanja, Django okvir je implementiran unutar python virtuelnog okruženja u *webmonitorapp* direktorijumu koji se nalazi na laptop računaru u lokalnoj mreži sa instaliranim *Ubuntu 22.04 LTS* operativnim sistemom.

Django okvir ima za cilj olakšavanje procesa kreiranja kompleksnih *web* aplikacija koje zahtijevaju čitanje i pisanje u/iz baze podataka.

Možemo reći da naš *web* aplikativni kontrolno-nadzorni centar i nije pretjerano kompleksna ili velika aplikacija u pogledu resursa ili funkcionalnosti, te iz tog razloga ne bismo pogriješili ukoliko bi kao alternativu *Django* okviru izabrali *Flask* koji je dosta lakši za implementaciju, međutim kako je ovo istraživanje zamišljeno sa ciljem da kao konačan proizvod stvori između ostalog i skalabilan sistem za detekciju neovlašćenih lica unutar zone nadzora, i uz male izmjene omogući kreiranje *cluster-a* sa više podsistema ili kamera, izabran je Django okvir.

Django okvir radi na principu model-template-views arhitekture, pa je iz tog razloga pogodno rješenje za buduća proširenja sistema, pisanje novih funkcija, kao i dodatnih modelovanja objekata baza podataka.

U ovom poglavlju ćemo prikazati način na koji su razni moduli Django okvira konfigurisani, kao i diskutovati o tome zbog čega su određene konfiguracije primijenjene.

### 3.2.1 Opšta Django konfiguracija

Na samom početku potrebno je kreirati direktorijum u kom želimo da smjestimo našu *web* aplikaciju.

Nakon kreiranja direktorijuma, pozicioniramo se u isti i komandom "*pipenv install django*" kreiramo novo, django virtuelno okruženje.

Na Slici 36. vidimo na koji način možemo provjeriti putanju do našeg novokreiranog virtuelnog okruženja ukoliko istu zaboravimo, kao i na koji način aktiviramo virtuelno okruženje.

```
bigfella@workbench:~/master/webmonitorapp$ pipenv --venv
/home/bigfella/.local/share/virtualenvs/webmonitorapp-E-059A33
bigfella@workbench:~/master/webmonitorapp$ source /home/bigfella/.local/share/virtualenvs/
webmonitorapp-E-059A33/bin/activate
(webmonitorapp) bigfella@workbench:~/master/webmonitorapp$ ls
db.sqlite3  manage.py  models.py  monitorcenter  Pipfile  Pipfile.lock  playground
(webmonitorapp) bigfella@workbench:~/master/webmonitorapp$
```

Slika 36: Aktivacija virtuelnog okruženja

Nakon što uspješno aktiviramo virtuelno okruženje, potrebno je kreirati Django projekat komandom "*django-admin startproject webmonitorapp* ." Razlog stavljanja "." jeste taj da želimo da se projekat kreira unutar direktorijuma u kom se trenutno nalazimo, a ne da se isti kreira u zasebnom direktorijumu.

Nakon uspješnog kreiranja Django projekta, možemo započeti opštu konfiguraciju naše *web aplikacije.*

Na Slici 37. vidimo konfiguraciju koja se nalazi unutar *settings.py* modula u *monitorcenter* direktorijumu, a koja služi kao globalna konfiguracija projekta. Unutar iste smo definisali aplikacije koje će Django koristiti, između ostalog smo dodali direktorijum *playground* u kom se nalaze datoteke vezane za samu aplikaciju, kao i *built-in staticfiles* koji ćemo kasnije koristiti kako bismo učitavali *css.*

```
INSTALLED_APPS = [
    'clearcache',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'playground',
    'django.contrib.sessions',
]
```

Slika 37: Installed_apps konfiguracija

Takođe, potrebno je dodati *URL (eng.* Uniform Resource Locator*)* putanju unutar konfiguracije *urls.py* modula koja se nalazi u *monitorcenter* direktorijumu, tačnije unutar globalne konfiguracije projekta.

Putanju dodajemo na način da uključujemo tj. uvozimo sve lokalne putanje iz projekta *playground*, što možemo vidjeti na Slici 38.

```
urlpatterns = [
    path('admin/clearcache/', include('clearcache.urls')),
    path('admin/', admin.site.urls),
    path('', include('playground.urls'))
]
```

Slika 38: Globalna konfiguracija putanja

### 3.2.2 Analiza konfiguracije web aplikacije

Nakon što smo uspješno implementirali virtuelno okruženje, predefinisali globalnu konfiguraciju potrebnu za funkcionisanje *web* servera, potrebno je konfigurisati samu *web* aplikaciju na način da ispunjava naše potrebe i zahtjeve. Dakle, potrebno je da u zavisnosti od trenutnog stanja sistema, na *web* aplikaciji dinamički serviramo neki sadržaj, koji će nam dati uvid u trenutno stanje sistema. Takođe, potrebno je omogućiti korisniku da pošalje zahtjev samom sistemu, tj. pokrene ili zaustavi isti, kao i restartuje čitav uređaj.

Na početku smo pomenuli da django okvir funkcioniše na principu model-template-views arhitekture, pa ćemo istu i objasniti.

Unutar django okvira, model je jedinstveni izvor informacije i nekim našim podacima, u suštini, svaki model se veže za jednu tabelu unutar baze podataka.

Unutar modela imamo i atribute koji predstavljaju polje u datoj tabeli baze podataka, što ćemo detaljnije objasniti i vidjeti u daljem radu.

Modeli nam omogućavaju da čitamo i pišemo podatke u/iz baze podataka i na taj način imamo informaciju o trenutnom stanju sistema kao i da omogućimo korisniku da zatraži neku akciju od istog.

Sad je potrebno ovu informaciju dinamički prezentovati, što činimo korišćenjem template-a koji nam omogućavaju dinamičko renderovanje html-a i pisanje koda u sintaksi sličnoj python programskom jeziku.

Na kraju, ali ništa manje bitno, *views* modul nam omogućava primanje korisničkih zahtjeva, obradu i odgovaranje na iste.

Na Slici 39. vidimo konfiguraciju *URL* putanja same aplikacije, tačnije, ukoliko korisnik pošalje zahtjev sa praznom putanjom, tj. pristupi *web* aplikaciji, pokrenuće se *say_hello* metoda modula *views*, koja će korisniku vratiti *hello.html* dinamički sadržaj, koji smo imali prilike da vidimo na Slici 8.

```
from django.urls import path, include
from . import views


urlpatterns = [
    path('', views.say_hello),
    path('reqReboot',views.reqReboot),
    path('reqStart',views.reqStart),
    path('reqStop',views.reqStop)
]
```

Slika 39: Mapiranje putanja aplikacije

Na Slici 39. također vidimo i metode reqReboot, reqStart i reqStop koje se pozivaju ukoliko korisnik pritisne neku od opcija kako bi zatražio akciju na sistemskoj strani.

Na Slici 40. vidimo izvorni kod *views* modula.

```
#sistemDBmysql
def say_hello(request):
    sistems = Sistem.objects.all()

    context = {
        'sistems':sistems
    }
    return render(request, 'hello.html', context)


def reqReboot(request): #Reboot
    newValue=3
    Sistem.objects.filter(aktivnost="kameramon").update(status=newValue)
    response = redirect('/')
    return response

def reqStart(request):  #START
    newValue=4
    Sistem.objects.filter(aktivnost="kameramon").update(status=newValue)
    response = redirect('/')
    return response

def reqStop(request):  #STOP
    newValue=5
    Sistem.objects.filter(aktivnost="kameramon").update(status=newValue)
    response = redirect('/')
    return response
```

Slika 40 : Izvorni kod *views.py* modula

Unutar say_hello metode vidimo da se inicijalizuje model baze podataka kako bi se iz iste čitalo trenutno stanje sistema, kao i da se na kraju korisniku vraće, tj. servira *hello.html* template.

Ukoliko je korisnik zatražio akciju nad sistemom, pozivaju se neke od tri sledeće metode: reqReboot, reqStart, reqStop. One će pristupiti modelu baze podataka, ažurirati stanje *status* atributa, a korisnika redirektovati na početku stranu koja će opet prikazati dinamički *hello.html* sadržaj.

Na Slici 41. vidimo izvorni kod *hello.html*-a.

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  {% load static %}
  <link rel="stylesheet" href="{% static 'css/style.css' %}">
  <link rel="stylesheet" href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/4.7.0/css/font-awesome.min.css">
  <title>KontrolaFRTIDS</title>
</head>
<body>

{% for sistem in sistems %}

{% if sistem.status == 1 %}
<body style="background-color: rgb(236, 255, 179);">
<h2 class="content">MONITORING FRT IDS SISTEMA</h2>
<h2 class="content">Vrijeme: {% now "d-m-Y G:i:s" %}</h2>
<h1 class="content" style="color:green;">SISTEM JE AKTIVAN</h1>
<form action="/reqReboot" class="content">
  <button id="stat-btn" class="button">REBOOT</button>
</form>
<form action="/reqStart" class="content">
  <button id="stat-btn" class="button">START</button>
</form>
<form action="/reqStop" class="content">
  <button id="stat-btn" class="button">STOP</button>
</form>
```

Slika 41: Izvorni kod *hello.html*

Vidimo da se logički dio koda piše između *"{%  %}"* karaktera.

*"{% load static %}"* nam omogućava stilizovanje *html-a* uz pomoć *css-a* (eng. Cascading Style Sheets).

Dalje provjeravamo da li je vrijednost atributa *status*, modela *sistem*, jednak 1, ukoliko jeste, pozadinsku boju mijenjamo u zelenu što označava aktivnost sistema, serviramo poruku da je sistem aktivan, kao i nudimo opcije, tj. forme u vidu dugmadi reboot, start i stop, kako bi korisnici mogli da izvrše akcije nad sistemom.

Ukoliko vrijednost atributa status nije 1, onda provjeravamo dalje, što možemo vidjeti na Slici 42.

```
{% elif sistem.status == 3 %}
<body style="background-color: rgb(255, 204, 153);">
<h2 class="content">MONITORING FRT IDS SISTEMA</h2>
<h2 class="content">Vrijeme: {% now "d-m-Y G:i:s" %}</h2>
<h1 class="content">SISTEM SE RESTARTUJE, SACEKAJTE</h1>

{% elif sistem.status == 4 %}
<body style="background-color: rgb(255, 204, 153);">
  <h2 class="content">MONITORING FRT IDS SISTEMA</h2>
  <h2 class="content">Vrijeme: {% now "d-m-Y G:i:s" %}</h2>
  <h1 class=content style="color:red;"> SISTEM SE TRENUTNO POKREĆE</h1>
  <form action="/reqReboot" class="content">
    <button id="stat-btn" class="button"><i class="fa fa-power-off" aria-hidden="true"> REBOOT</i>
    </button>
  <form action="/reqStop" class="content">
    <button id="stat-btn" class="button">STOP</button>
  </form>

{% else %}
<body style="background-color: rgb(255, 153, 153);">
<h2 class="content">MONITORING FRT IDS SISTEMA</h2>
<h2 class="content">Vrijeme: {% now "d-m-Y G:i:s" %}</h2>
<h1 class=content style="color:red;"> SISTEM TRENUTNO NIJE U FUNKCIJI</h1>
<form action="/reqReboot" class="content">
  <button id="stat-btn" class="button"><i class="fa fa-power-off" aria-hidden="true"> REBOOT</i>
  </button>
</form>
<form action="/reqStart" class="content">
  <button id="stat-btn" class="button">START</button>
</form>
<form action="/reqStop" class="content">
  <button id="stat-btn" class="button">STOP</button>
</form>
{% endif %}
{% endfor %}
```

Slika 42: Logika dinamičkog serviranja html-a

Prikaz sadržaja na *web* aplikaciji, mijenjaće se u zavisnosti od vrijednosti atributa status. Ukoliko je vrijednost 3, dobićemo poruku da se sistem trenutno restartuje, ukoliko je 4, da se pokreće, a u slućaju 0 ili 5, da sistem nije u funkciji kao i crvenu pozadinu.

Takođe vidimo da je na svakom prikazu prisutno i trenutno vrijeme. Kako bismo izbjegli potrebu za konstantnim osvježavanjem aplikacije, implementirali smo *javascript* funkciju unutar *hello.html* koja će svake tri sekunde automatski osvježiti i samim tim ažurirati sadržaj aplikacije, što možemo vidjeti na Slici 43.

```
<!-- auto refresh -->
<script>
window.setTimeout(function () {
  location.href = "http://192.168.1.234:8000";
}, 3000);
</script>
```

Slika 43: *auto-refresh* funkcija

Unutar *settings.py* modula potrebno je komentarisati *defaultnu sqlite* bazu podataka i konfigurisati pristupne parametre za pristup našoj *MySql* bazi podataka, što vidimo na Slici 44.

```
# DATABASES = {
#     'default': {
#         'ENGINE': 'django.db.backends.sqlite3',
#         'NAME': BASE_DIR / 'db.sqlite3',
#     }
# }

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'monitoring',
        'USER': 'bigfella',
        'PASSWORD': '[redacted]',
        'HOST': '[redacted]',
        'PORT': '3306',
    }
}
```

Slika 44: Konfiguracije pristupnih parametara baze podataka

Nakon toga, potrebno je još i kreirati prethodno pomenute modele i atribute kako bismo lakše upravljali podacima iz baze podataka, što vidimo na Slici 45.

```python
from django.db import models


class Sistem(models.Model):
    ids = models.IntegerField(db_column='IDs', primary_key=True)  # Field name made lowercase.
    aktivnost = models.CharField(max_length=25)
    status = models.IntegerField(blank=True, null=True)

    class Meta:
        managed = False
        db_table = 'sistem'
```

Slika 45: Kreiranje modela baze podataka

Kreiramo klasu koju nazivamo *sistem,* zatim kreiramo atribute i povezujemo model sa bazom podataka *sistem*.

*Web* aplikaciju pokrećemo koristeći se metodom *runserver* modula *manage.py,* kao na Slici 46.

```
(webmonitorapp) bigfella@workbench:~/master/webmonitorapp$ python manage.py runserver 0.0.0.0:8000
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).
July 04, 2023 - 07:42:47
Django version 4.2, using settings 'monitorcenter.settings'
Starting development server at http://0.0.0.0:8000/
Quit the server with CONTROL-C.

[04/Jul/2023 07:42:54] "GET / HTTP/1.1" 200 1208
[04/Jul/2023 07:42:54] "GET /static/css/style.css HTTP/1.1" 200 1874
[04/Jul/2023 07:42:56] "GET / HTTP/1.1" 200 1208
[04/Jul/2023 07:42:58] "GET / HTTP/1.1" 200 1208
```

Slika 46: Pokretanje *web* aplikacije

*Web* aplikaciju pokrećemo na portu 8000, a adresi 0.0.0.0, što znači javnoj adresi u datom trenutku, tako da aplikacija bude javno dostupna ukoliko je ostatak mreže pravilno konfigurisan.

### 3.3 Dizajn MySql baze podataka

Kao što smo vidjeli u prethodnim poglavljima, čitava "komunikacija" između klijenata i servera se odvija posredovanjem baze podataka.

Za potrebe ovog istraživanja, napravili smo jednostavnu bazu podataka u kojoj smo skladištili vrijednost trenutnog stanja sistema u vidu 0 ili 1.

Na *linux* serveru u simuliranom oblaku, komandom "*sudo apt install mysql-server*" instalirali smo MySql bazu podataka, nakon čega smo napravili nalog "*bigfella*" i istom dodijelili

potrebne privilegije, koristeći komandu "*GRANT ALL PRIVILEGES ON *.* TO 'bigfella'@'%' WITH GRANT OPTION;*" .

MySql server je po *default-u* podešen tako da sluša za nadolazeće konekcije, samo na *localhostu,* tj. *127.0.0.1,* pa je to potrebno promijeniti kako bi *web-aplikacija* i ostatak sistema mogli upsješno da se autentifikuju i vrše izmjene u bazi podataka. Navedena konfiguracija se nalazi u "*/etc/mysql/mysql.conf.d/mysqld.cnf"* datoteci, pa je potrebno dodati *"#*" karakter ispred *bind-address* argumenta, kako bi server slušao na svojoj javnoj i privatnoj adresi, što vidimo na Slici 47.

```
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
#bind-address           = 127.0.0.1
#mysqlx-bind-address    = 127.0.0.1
```

Slika 47: Komentarisanje bind-address linije kako bi omogućili serveru da sluša za eksterne konekcije

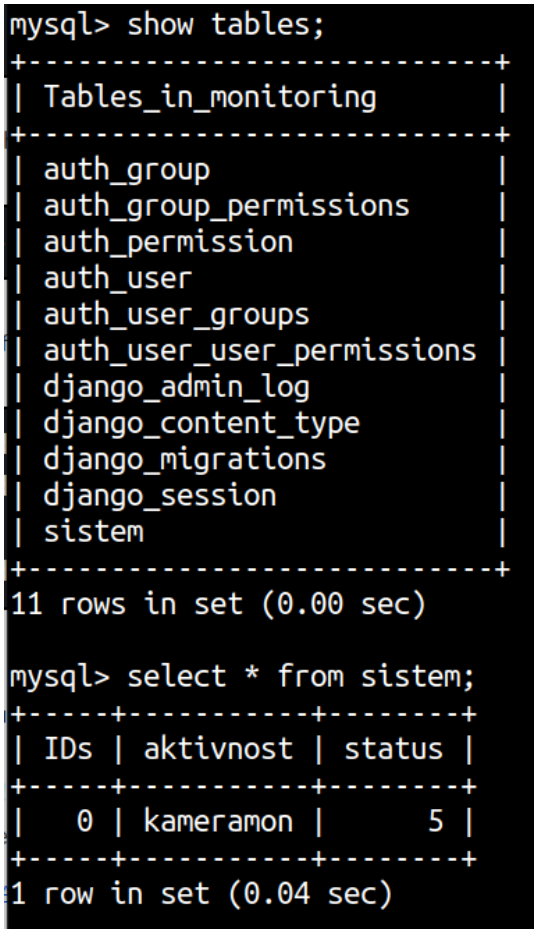
Potrebno je i restartovati MySql bazu podataka, koristeći komandu "*sudo systemctl restart mysql.service*" kako bi prethodna izmjena u konfiguraciji bila primijenjena.

Nakon što smo uspješno podigli i konfigurisali bazu podataka, potrebno je povezati se na istu komandom "mysql -h x.x.x.x -u bigfella -p" kao i kreirati bazu podataka i tabelu, što vidimo na Slici 48.

```
mysql> CREATE TABLE siistem (
    -> IDS int NOT NULL,
    -> aktivnost varchar(25) NOT NULL,
    -> status int,
    -> PRIMARY KEY (IDs) );
Query OK, 0 rows affected (0.08 sec)
```

Slika 48: Kreiranje tabele

Nakon što tabelu popunimo nekom generičkom vrijednošću, korišćenjem *INSERT* upita, sadržaj iste možemo vidjeti kao na Slici 49.




Slika 49: Sadržaj tabele sistem

Prilikom povezivanja Django okvira sa bazom podataka, takođe su se kreirale dodatne tabele, koje vidimo na slici 49.

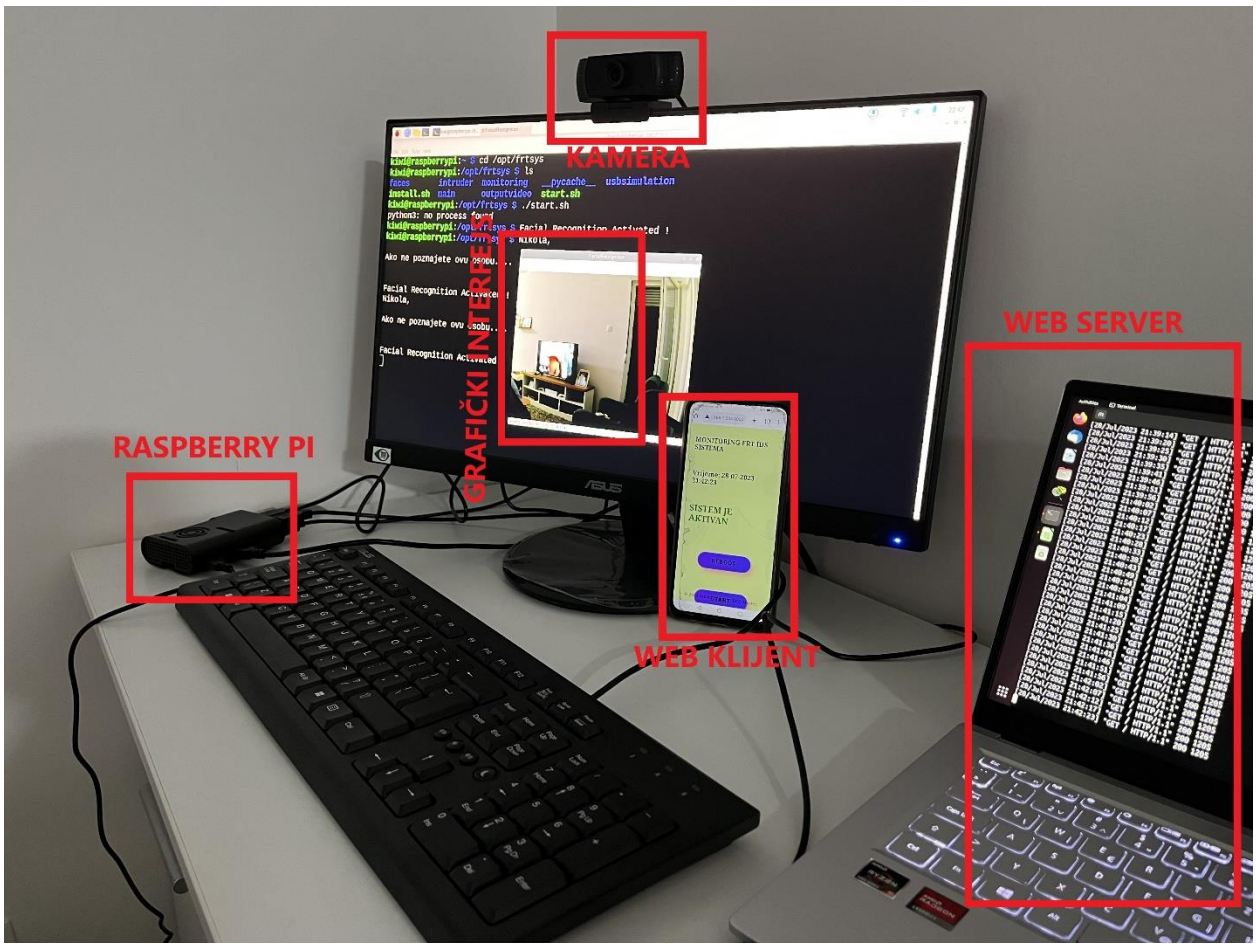
Ako se ukratko osvrnemo i na bezbjednost same baze podataka, možemo reći da je ista ranjiva na *Sql injection* napad zbog svoje otvorenosti, tačnije nedostatka autentifikacije prilikom pristupa samoj *web* aplikaciji, kao i nedovoljne validacije inputa od strane samog sistema.

Međutim, kako bi izbjegli dodatne komplikacije sa kreiranjem korisničkih polisa, naloga i privilegija, za potrebe ovog istraživanja, i dokazivanje koncepta projekta, dovoljno je mrežno ograničiti pristup *web* aplikaciji na *MAC* (eng. *Media Access Control*) adresu/adrese mobilnih ili desktop uređaja sa kojih želimo ili planiramo da pristupamo grafičkom interfejsu *web* aplikacije, i na taj način možemo osigurati da niko, osim onih uređaja, a ujedno i korisnika, ne može pristupiti sadržaju *web* aplikacije, pa samim tim ni narušiti integritet iste.

# 4 EVALUACIJA PERFORMANSI

Evaluacijom performansi smo stekli uvid u ostvarene rezultate unaprijed postavljenih i jasno definisanih ciljeva. Mjerenjem efikasnosti realizovanog sistema, došli smo do realnih podataka o stanju sistema, tačnije stekli smo uvid u sve njegove vrline i mane, pa samim tim smo uvidjeli u kojim to djelovima postoji prostor za poboljšanje istog.

Na Slici 50. vidimo kako izgleda testno okruženje.




Slika 50: Testno okruženje

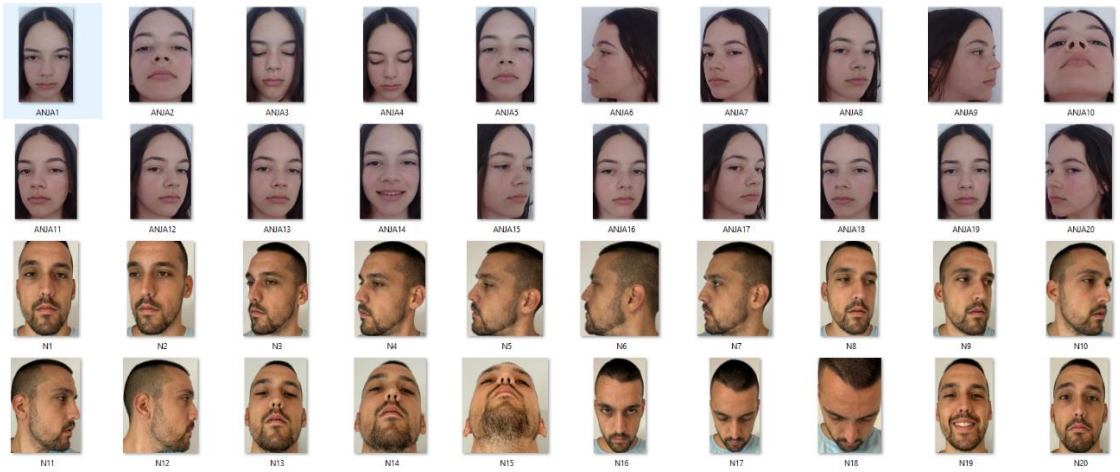
Na monitoru vidimo grafički prikaz dijela sistema koji je implementiran na *raspberry-pi* uređaju, a upotrebnom mobilnog uređaja pristupamo *web* aplikaciji u cilju monitoringa i kontrole samog sistema.

Evaluaciju perfomansi smo sproveli u pet različitih faza, koristeći tri različita *dataseta*, tj. uzorka, a svaka od faza je testirana kroz pet iteracija.

Faza jedan podrazumijeva testiranje uspješnosti prepoznavanja neovlašćene osobe, procenat false positive resultata, koliko je sekundi potrebno da notifikacija stigne korisniku nakon uspješnog prepoznavanja neovlašćene osobe, kao i koliko sekundi je potrebno sistemu da se pokrene, čime smo utvrdili da li veličina uzorka diktira performanse sistema. U prvoj fazi smo koristili manji uzorak, nazvan "uzorak A" koji možemo vidjeti na Slici 51.


Slika 51: Uzorak A

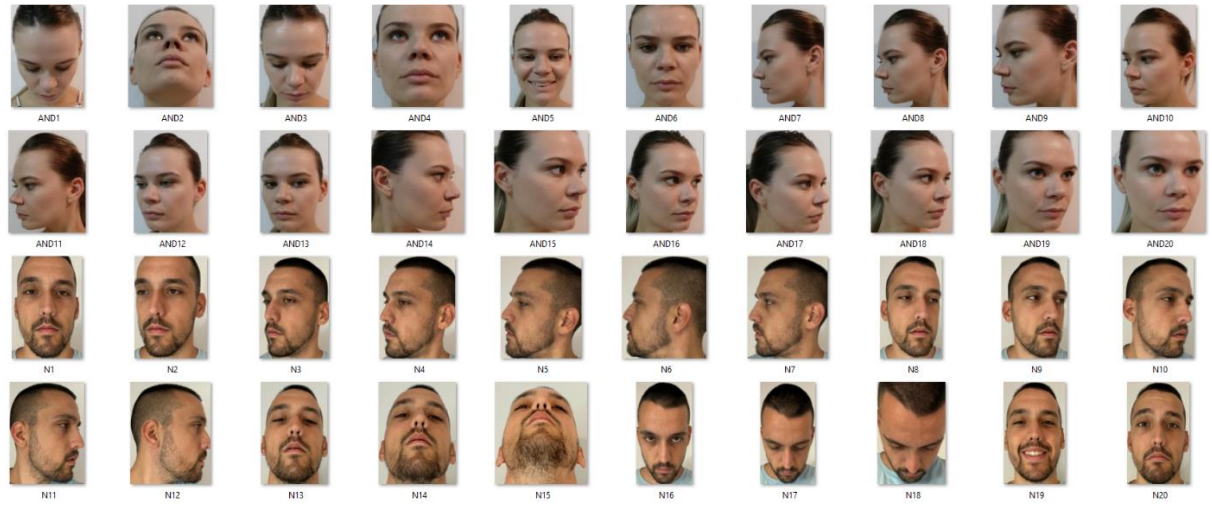
Faza dva podrazumjeva identično testiranje kao i u fazi jedan, stim da smo u fazi dva koristiti prilično veći uzorak koji broji 20 slika lica po osobi, pod različitim uglovima, kako bismo došli do zaključka da li će sa povećanjem uzorka, doći do povećanja preciznosti samog sistema. Uzorak koji smo koristili prilikom testiranja faze dva, jeste uzorak B, koji vidimo na Slici 52.




Slika 52: Uzorak B

U trećoj fazi smo se ja (subjekat N), i moja supruga (subjekat AND) fokusirali isključivo na mogućnost greške samog sistema, tačnije potrudili smo se da kroz 5 iteracija "isprovociramo" sistem, praveći neuobičajene izraze lica, stavljajući peškire ili kape na glavu, tako da ga "natjeramo" da "pomisli" kako je identifikovao neovlašćenu osobu.

Cilj treće faze jeste da detaljnije uvidimo slabosti sistema, kako bismo ih evidentirali, naučili iz istih kao i potpomogli buduća istraživanja na ovu ili ovoj sličnu temu. Uzorak koji smo koristili  u trećoj fazi, a koji možemo vidjeti na Slici 53. jeste uzorak C.




Slika 53: Uzorak C

U četvrtoj fazi, a na uzorku B, smo testirali uspješnost prepoznavanja sistema u uslovima smanjenog osvjetljenja.

U petoj i završnoj fazi ove evaluacije performansi smo testirali funkcionalnost *web*-aplikacije, na način što smo provjeravali vrijeme koje je potrebno sistemu za izvršavanje akcija zadatih od strane klijenta, koje je u mnogome zavisilo od trenutka zadavanja komande iz razloga što se modul *monitor.sh* izvršavao na svakih 60 sekundi.

## 4.1 Prva faza

Na početku prve faze, nakon pokretanja uzorka A, kome je trebalo 10,32 sekundi da se enkodira i da se njegove signature upišu u radnu memoriju sistema, došli smo do očekivanih rezultata koje vidimo u Tabeli 1.

**Tabela 1:** Rezultati prve faze evaluacije perfomansi sistema.

| FAZA 1 | Uspješno prepoznavanje neovlašćene osobe | False positive | Odziv notifikacije (s) | Učitavanje uzorka (s) |
|---|---|---|---|---|
| ITERACIJA 1 | DA | NE | 35 | 10.32 |
| ITERACIJA 2 | NE | NE | 26.05 | |
| ITERACIJA 3 | NE | NE | 23.75 | |
| ITERACIJA 4 | DA | NE | 30 | |
| ITERACIJA 5 | DA | NE | 40 | |

Iteracije 1-5 predstavljaju testiranje sa vertikalno pozicioniranom, podignutom, spuštenom i glavom okrenutom u desno-lijevo, respektivno. Iz priloženih rezultata vidimo da sistem nije uspješno prepoznao neovlašćenu osobu u slučaju kad je glava iste bila podignuta ka gore ili spuštena na dolje.

## 4.2 Druga faza

Prilikom pokretanja druge faze, tj. učitavanja uzorka B, uviđamo da sistem ne može da enkodira slike ANJ6, ANJ10, N4-7, N11-12, N15 i N18, a kako se radi o slikama iz profila ili izuzetno podignute ili izuzetno spuštene glave, analizom *error* koda dolazimo do zaključka da *face_recognition* biblioteka, navedene slike ne prepoznaje kao lice, te ih iz tog razloga ne može enkodirati.

Unutar izvornog koda *recognition.py* modula smo dodali *try*, *except* blok za potrebe ovog *error handlinga* i nastavili sa daljim testovima, čije rezultate vidimo u Tabeli 2.

**Tabela 2:** Rezultati druge faze evaluacije performansi.

| FAZA 2 | Uspješno prepoznavanje neovlašćene osobe | False positive | Odziv notifikacije (s) | Učitavanje uzorka (s) |
|---|---|---|---|---|
| ITERACIJA 1 | DA | NE | 25 | 43.31 |
| ITERACIJA 2 | DA | NE | 20 | |
| ITERACIJA 3 | DA | NE | 24 | |
| ITERACIJA 4 | DA | DA | 27.7 | |
| ITERACIJA 5 | DA | NE | 21.32 | |

Vidimo da se sa većim i detaljnijim uzorkom, povećalo vrijeme koje je potrebno sistemu za obradi isti, ali se isto tako povećala i vjerovatnoća uspješnosti samog sistema. Interesantan je podatak da je u jednom trenutku četvrte iteracije, za vrijeme testiranja desnog profila, pod uglom

od 60 stepeni, kad je za razliku od ugla od 90 stepeni već bio u mogućnosti da izračuna signaturu lica, dao „lažni“ podatak, tj. izračunao je da testni subjekat (AND sa uzorka C), ima 97% poklapanja u crtama lica sa subjektom ANJ9, koja na toj slici takođe ima izračunate signature desnog profila pod sličnim uglom kao testni subjekt. Nakon što je testni subjekt nastavio da smanjuje ugao zakrivljenosti, sistem je istog identifikovao kao nepoželjnog i poslao notifikaciju kojoj je trebalo 27.7 sekundi da stigne na klijentsku stranu.

## 4.3 Treća faza

U trećoj fazi smo izvršili testiranje identično onom u fazi dva, samo u uslovima smanjenog pozadinskog osvjetljenja, a rezultati istog su prikazani u Tabeli 3

**Tabela 3:** Rezultati treće faze evaluacije performansi.

| FAZA 3 | Uspješno prepoznavanje neovlašćene osobe | False positive |
|---|---|---|
| ITERACIJA 1 | DA | NE |
| ITERACIJA 2 | DA | NE |
| ITERACIJA 3 | DA | NE |
| ITERACIJA 4 | NE | DA |
| ITERACIJA 5 | DA | NE |

U Tabeli 3. vidimo rezultate treće faze, koji su identični onima iz druge faze. Takođe, ponovo smo imali 97% poklapanja između AND i ANJ subjekata.

## 4.4 Četvrta faza

U četvrtoj fazi smo koristili uzorak C i kao što smo naveli na početku ovog poglavlja, cilj ove faze jeste navesti sistem da “pomisli” kako su subjekti iz uzorka C nepoželjne osobe.

Na ovaj način dobijamo bližu sliku o propustima unutar uzorka ili samog Sistema. U Tabeli 4. vidimo rezultate četvrte faze evaluacije performansi sistema.

**Tabela 4:** Rezultati četvrte faze.

| FAZA 4 | False positive |
|---|---|
| ITERACIJA 1 | NE |
| ITERACIJA 2 | NE |
| ITERACIJA 3 | NE |
| ITERACIJA 4 | DA |
| ITERACIJA 5 | DA |

U ovoj fazi iteracije 1-5 se ne odnose na poziciju lica subjekta, već na neuobičajene izraze lica kao i na testiranje prilikom nošenja peškira ili kape na glavi.

### 4.5 Peta faza

U petoj i završnoj fazi smo testirali vrijeme koje je potrebno sistemu da izvrši neku akciju koju je klijent prethodno zatražio putem *web* kontrolno nadzornog centra. Rezultate vidimo u Tabeli 5.

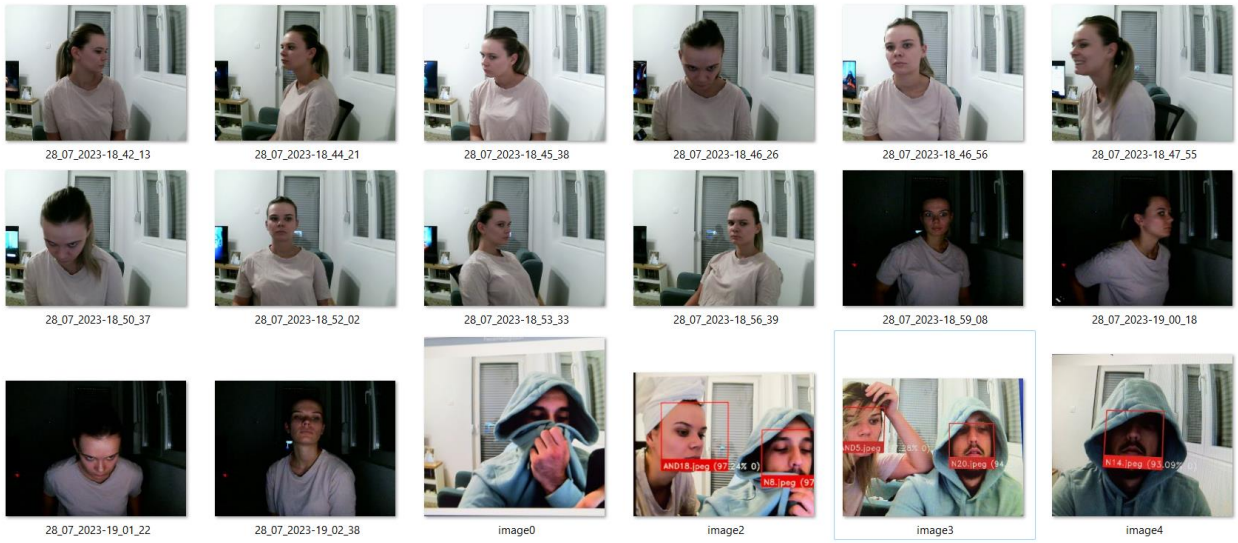
**Tabela 5:** Rezultati izvršavanja klijentskih zahtjeva.

| FAZA 5 | Odziv za reboot sistema (s) | Odziv za pokretanje sistema (s) | Odziv za zaustavljanje sistema (s) |
|---|---|---|---|
| ITERACIJA 1 | 18 | 8.78 | 32.34 |
| ITERACIJA 2 | 58.61 | 57.61 | 35.64 |
| ITERACIJA 3 | 43.46 | 52.56 | 45.44 |
| ITERACIJA 4 | 36.8 | 25.08 | 17.81 |
| ITERACIJA 5 | 43.09 | 12.65 | 29.47 |

Kako je kroz *crontab/cronjob* definisano da se modul *monitor.sh* izvršava svakih 60 sekundi, a kako je isti između ostalog zadužen za izvršavanje akcija zatraženih od strane klijenta, onda vrijeme koje je potrebno sistemu da izvrši neku od tih akcija, nikad neće biti duže od 60 sekundi, a biće jednako trenutku u kom se akcija zatražila plus onoliko sekundi koliko je ostalo do izvršavanja *cronjob-a* na sistemskoj strani.

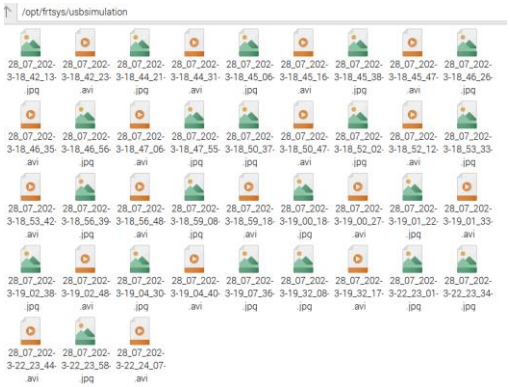
Tako u rezultatima testiranja pete faze vidimo da vrijeme odziva za zaustavljanje, pokretanje ili restartovanje sistema varira. U četvrtoj iteraciji vidimo da je sistemu bilo potrebno 17.81 sekund da zaustavi sistem, što znači da je korisnik zatražio/zadao akciju 17.81 sekund prije nego što se *cronjob* na sistemskoj strani trebao izvršiti.

Na Slici 54. vidimo dokaze o uspješnosti prepoznavanja neovlašćenog lica, tj. subjekta AND, kroz faze 1,2 i 3 u kojima su bili učitani uzorci A i B kojima navedeni subjekt ne pripada, kao i rezultate testiranja kroz fazu 4.



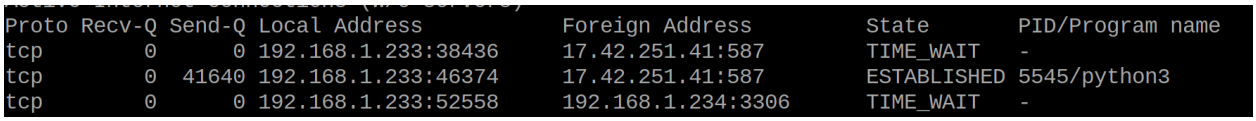

Slika 54: Rezultati evaluacije performansi kroz prve četiri faze.

Na Slici 55. vidimo sadržaj unutar simulirane eksterne memorije, nakon završenih 5 faza testiranja sistema.




Slika 55: Sadržaj simulirane eksterne memorije.

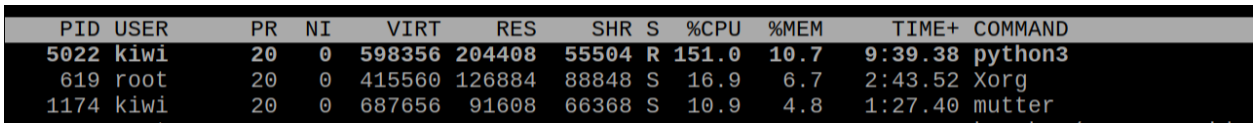
U petoj fazi smo također testirali mrežno i resursno opterećenje sistema. Kao što smo već rekli, sistem ostvaruje mrežnu komunikaciju samo u trenutku kad mijenja vrijednost unutar baze podataka ili šalje notifikaciju o detekciji neovlašćene osobe, što možemo vidjeti na Slici 56.




Slika 56: Mrežna komunikacija sistema u trenutku slanja notifikacije.

Vidimo da sistem komunicira samo ka *Apple* mail serveru na *SMTP* portu 587, kao i *MySQL* serverom u lokalnoj mreži na standardnom 3306 *MySQL* portu.

Takođe, sistem u toku najvećeg radnog opterećenja radi na 151% ukupne procesorske iskorišćenosti, a što znači da koristi više od jednog čitavog procesorskog jezgra, a to možemo vidjeti na Slici 57.




Slika 57: Procesorska iskorišćenost sistema.

## 4.6 Energetska efikasnost i troškovi

Ako uzmemo u obzir da *raspberry pi* 4 model B, u prosjeku troši 5W, ako bi sistem radio 24 časa, to znači da bismo za jedan dan utrošili 0.12 kWh, za jedan mjesec 3.6 kWh, a za jednu godinu 43.8 kWh električne energije. To znači da bismo po trenutnoj cijeni od 9.7 eurocent/kWh u Crnoj Gori, utrošili nešto više od 4 eura na godišnjem nivou, što je zanemarljivo a pritom imali priliku da koristimo sistem u režimu 24/7.

U Tabeli 6. vidimo neophodni hardver za realizaciju ovakvog sistema kao i njegovu novčanu vrijednost.

**Tabela 6:** Novčana vrijednost hardvera neophodnog za realizaciju sistema

| TROŠKOVI | CIJENA |
|---|---|
| Raspberry Pi 4 Model B 2GB | 60e |
| Napajanje | 15e |
| VPS | 80e |
| Kamera | 10e |
| Wifi adapter | 15e |

Ukoliko odlučimo da hostujemo *web*-aplikaciju na virtuelnom serveru (eng. *Virtual Private Server – VPS*), cijena na gore prikazanoj slici je izračunata na godišnjem nivou. Međutim postoji dosta novčano efikasnija varijanta, a to je da se i *MySQL* baza podataka i *Web* aplikacija hostuju na drugom *raspberry pi* uređaju.

Dolazimo do zaključka da cijena potrebna za realizaciju ovako dizajniranog sistema za detekciju neovlašćenog lica unutar zone nadzora varira između 100 i 180 eura.

# 5 ZAKLJUČAK

Predloženi model sistema za detekciju neovlašćenog lica unutar zone nadzora je predviđen za korišćenje u uslovima privatnog posjeda, iako ne zadržava, tj. ne skladišti i ne pruža uvid u biometrijske podatke, fotografije subjekata obrade, osim u slučaju kada je neovlašćena osoba identifikovana (npr. provalnik ukoliko dovoljno dobro istreniramo model).

Primjenom tehnika za prepoznavanje lica uspješno smo realizovali pristupačan, resursno i energetski efikasan sistem za detekciju neovlašćenog pristupa kao i notifikaciju korisnika sistema u skoro realnom vremenu. Na ovaj način smo eliminisali potrebu za konstatnim nadzorom. Takođe smo utvrdili činjenicu da će različiti uzorci podataka, davati različite rezultate, tačnije, došli smo do zaključka da će obimniji uzorak doprinijeti većoj preciznosti sistema prilikom prepoznavanja neovlašćenog lica.

Upotrebnom baze podataka kao posrednika u komunikaciji između *raspberry-pi* uređaja i *web* servera, maksimalno smo rasteretili *raspberry-pi* a uspješno smo postigli realizaciju *web* aplikativnog konrolno nadzornog centra u skoro realnom vremenu. Ovo ne samo što omogućava sistemu da funkcioniše i na *raspberry-pi* uređajima starijih generacija ili slabijih performansi, već i doprinosi energetskoj efikasnosti cijelog sistema.

Sistem takođe nije zahtjevan u vidu internet protoka, kao ni pretjerano stabilne konekcije iz razloga što jedini put kad sistem koristi Internet jeste kad prosleđuje informacije SMTP serveru i upisuje ili čita stanje iz baze podataka.

Sistem je teoretski moguće primjenjivati i u uslovima javnog posjeda ukoliko bi procenat *false-positive* rezultata bio dovoljno nizak, međutim kako to nije u slučaj u praksi, ovo bi predstavljalo osnovno ograničenje korišćenja predloženog modela u uslovima javnog prostora. Još jedno ograničenje sistema bi bila nemogućnost identifikacije lica udaljenog objekta kao i nemogućnost sistema da razliku stvarnu osobu od fotografije (eng. Liveness detection). Trenutni model prepoznaje lica na udaljenosti do 1.50 metara što ga čini pogodnim za nadgledanje hodnika zgrade, ili ulaznih vrata stana, kuće, poslovnih prostora i slično.

Mogući pravac daljeg istraživanja je prilagođavanje ili optimizacija algoritama za prepoznavanje lica na način da isti prate regulative *GDPR* eng. *General Data Protection Regulation,* tj. da se dozvoli procesuiranje biometrijskih podataka samo u slučaju kad se za to

steknu uslovi, npr. ukoliko osoba počini krivično djelo, prekršaj ili slično.

Još jedan mogući pravac daljeg istraživanja bi bio pronalaženje modela za dodatno i *ad hoc* dotreniravanje sistema preko *web* aplikacije, mogućnost integracije sa već postojećim sistemima *CCTV* nadzora, kao i enkripcije kompletnog mrežnog saobraćaja u cilju dodatne zaštite i osiguravanja bezbjednosti protoka podataka.

Izvorni kod sistema za detekciju neovlašćenog lica i kontrolno nadzorne *web* aplikacije koji je nastao kao rezultat ovog istraživanja možete naći na: https://github.com/goodfella-afk/sistem-za-detekciju-neovlascenog-lica-unutar-zone-nadzora, a isti možete koristiti, redistribuirati i dodatno razvijati u skladu sa licencom otvorenog koda.

# 6 LITERATURA


[1] Van Dijk J, Nieuwbeerta P, Joudo Larsen J.Global crime patterns: An analysis of survey data from 166 countries around theworld, 2006–2019. J. Quant. Criminol;2021.

[2] Piza EL, Welsh BC, Farrington DP, Thomas AL. CCTV surveillance for crime prevention. Criminol. Public Policy. 2019; 18(1):135–159. DOI:https://doi.org/10.1111/1745-9133.12419.

[3] W. W. Bledsoe and I. Browning. 1959. Pattern recognition and reading by machine. In Papers presented at the December 1-3, 1959, eastern joint IRE-AIEE-ACM computer conference (IRE-AIEE-ACM '59 (Eastern)). Association for Computing Machinery, New York, NY, USA, 225–232. https://doi.org/10.1145/1460299.1460326

[4] Koley, S., Roy, H., Dhar, S., & Bhattacharjee, D. (2022). Illumination invariant face recognition using Fused Cross Lattice Pattern of Phase Congruency (FCLPPC). Information Sciences, 584, 633-648.

[5] Lu, Y., Khan, M., & Ansari, M. D. (2022). Face recognition algorithm based on stack denoising and self-encoding LBP. Journal of Intelligent Systems, 31(1), 501-510.

[6] Pawar, A. B., Gawali, P., Gite, M., Jawale, M. A., & William, P. (2022, April). Challenges for Hate Speech Recognition System: Approach based on Solution. In 2022 International Conference on Sustainable Computing and Data Communication Systems (ICSCDS) (pp. 699-704). IEEE

[7] Raharja, N. M., Fathansyah, M. A., & Chamim, A. N. N. (2022). Vehicle parking security system with face recognition detection based on eigenface algorithm. Journal of Robotics and Control (JRC), 3(1), 78-85.

[8] Smith, M., & Miller, S. (2022). The ethical application of biometric facial recognition technology. Ai & Society, 37(1), 167-175.

[9] Salama, Ramiz & Nour, Mohamed. (2023). Security Technologies Using Facial Recognition. Global Journal of Computer Sciences: Theory and Research. 13. 01-27. 10.18844/gjcs.v13i1.8294.

[10] Zainal, M. M., & Hamdan, A. (2022). Artificial intelligence in healthcare and medical imaging: Role in fighting the spread of COVID-19. In Advances in Data Science and Intelligent Data Communication Technologies for COVID-19 (pp. 173-193). Springer, Cham.

[11] Chatisa, Ivan & Syahbana, Yoanda & Wibowo, Agus. (2023). Object Detection and Monitor System for Building Security Based on Internet of Things (IoT) Using Illumination Invariant Face Recognition. Kinetik: Game Technology, Information System, Computer Network, Computing, Electronics, and Control. 10.22219/kinetik.v8i1.1622.

[12] Ijaradar, Jyotirmaya & Xu, Jinjing. (2022). A Cost-efficient Real-time Security Surveillance System Based on Facial Recognition Using Raspberry Pi and OpenCV. Current Journal of Applied Science and Technology. 1-12. 10.9734/cjast/2022/v41i531665.

[13] Srihari. K, Ramesh. R, Udayakumar. E, Gaurav Dhiman. An Innovative Approach for Face Recognition Using Raspberry Pi. Artificial Intelligence Evolution [Internet]. 2020 Aug. 8 [cited 2023 Oct. 5];1(2):102-7. Available from: https://ojs.wiserpub.com/index.php/AIE/article/view/62

[14] Muhammad Sajjad, Mansoor Nasir, Khan Muhammad, Siraj Khan, Zahoor Jan, Arun Kumar Sangaiah, Mohamed Elhoseny, Sung Wook Baik. 2017 May 9

[15] Nikisins O, Fuksis R, Kadikis A, Greitans M. Face recognition system on Raspberry Pi. Institute of Electronics and Computer Science. 2015 Apr 17;14.

[16] Boxey, Aasawari & Jadhav, Anushri & Gade, Pradnya & Ghanti, Priyanka & Mulani, Altaf. (2022). Face Recognition using Raspberry Pi. Journal of Image Processing and Intelligent Remote Sensing. 15-23. 10.55529/jipirs.24.15.23.

[17] Amato G, Carrara F, Falchi F, Gennaro C, Vairo C. Facial-based intrusion detection system with deep learning in embedded devices. InProceedings of the 2018 International Conference on Sensors, Signal and Image Processing 2018 Oct 12 (pp. 64-68).

[18] Zuma M, Owolawi PA, Malele V, Odeyemi K, Aiyetoro G, Ojo JS. Intrusion Detection System using Raspberry Pi and Telegram Integration. InProceedings of the International Conference on Artificial Intelligence and its Applications 2021 Dec 9 (pp. 1-7).

[19] Rodelas NC, Ballera MA. Intruder detection and recognition using different image processing techniques for a proactive surveillance. Indonesian Journal of Electrical Engineering and Computer Science. 2021 Nov;24(2):843-52.

[20] Bhardwaj R, Bera K, Jadhav O, Gaikwad P, Gupta T. Intrusion Detection through Image Processing and getting notified via SMS and Image.

[21] Klenshin, (2001) ‘Simple Mail Transfer Protocol on Internet’ IETF RFC 2821

[22] Patni, Jagdish. (2016). A simple mail server for Universities and Colleges. International Journal of Control Theory and Applications.

[23] https://www.python.org/success-stories/building-a-dependency-graph-of-our-python-codebase/

[24] Python.org [online] available at: https://www.python.org/about/

[25] Culjak I, Abram D, Pribanic T, Dzapo H, Cifrek M. A brief introduction to OpenCV. In2012 proceedings of the 35th international convention MIPRO 2012 May 21

(pp. 1725-1730). IEEE.

[26] OpenCv-Python [online] available at: https://pypi.org/project/opencv-python/

[27] Voronov V, Strelnikov V, Voronova L, Trunov A, Vovik A. Faces 2D-recognition and identification using the HOG descriptors method. InConference of Open Innovations Association, FRUCT 2019 (No. 24, pp. 783-789). FRUCT Oy.

[28] Face_recognition [online] available at: https://pypi.org/project/face-recognition/

[29] Dlib.net [online] available at: https://dlib.net/

[30] Krogh JW, Krogh G, Gennick. MySQL Connector/Python Revealed. New York: Apress; 2018.

[31] Smtplib [online] available at: https://docs.python.org/3/library/smtplib.html

[32] Newham C. Learning the bash shell: Unix shell programming. " O'Reilly Media, Inc."; 2005 Mar 29.

[33] MySQL AB. MySQL; 2001.

[34] Raggi E, Thomas K, Van Vugt S. Beginning Ubuntu Linux. Apress; 2011 Jan 10.

[35] Djangoproject.com [online] available at: https://www.djangoproject.com

[36] Ghimire D. Comparative study on Python web frameworks: Flask and Django.

[37] Raspberry Pi OS [online] available at: https://www.raspberrypi.com/documentation/computers/os.html

[38] Webcam facial recognition [online] available at: https://github.com/federicoazzu/webcam_face_recognition