**Tehničko veleučilište u Zagrebu**

**Stručni diplomski studij Informacijske sigurnosti i digitalne forenzike**

# Razvoj aplikacije za prepoznavanje osoba i oružja na temelju vizualnih značajki

**Diplomski rad**

**Jan Lamza**

**Zagreb, srpanj, 2025.**

**Tehničko veleučilište u Zagrebu**

**Stručni diplomski studij Informacijske sigurnosti i digitalne forenzike**

# Razvoj aplikacije za prepoznavanje osoba i oružja na temelju vizualnih značajki

**Jan Lamza, 0036502455**

**Mentor: doc. dr. sc., Aleksander Radovan**

**Zagreb, srpanj, 2025.**

# Sadržaj

# 1. Uvod

Ovaj rad izrađen je kao diplomski rad studija Informacijske sigurnosti i digitalne forenzike na Tehničkom veleučilištu u Zagrebu. Rad se bavi razvojem i analizom *web* aplikacije koja omogućuje detekciju lica i prepoznavanje oružja u stvarnom vremenu, pri čemu se primjenjuju suvremene metode umjetne inteligencije i računalnog vida. U teorijskom dijelu rada obrađeni su ključni pojmovi i tehnike vezane za umjetnu inteligenciju i računalni vid, s naglaskom na njezinu primjenu u domeni vizualnog nadzora i sigurnosti. Osim toga, detaljno su opisani softverski i hardverski zahtjevi nužni za implementaciju aplikacije, uključujući konkretne tehnologije korištene na klijentskoj i poslužiteljskoj strani. Rad također prikazuje funkcionalnosti same aplikacije, objašnjava način njihove implementacije te daje uvid u programski kod koji ostvaruje zadane ciljeve. Poseban naglasak stavljen je na prikaz rezultata i evaluaciju učinkovitosti razvijenog sustava. U završnom dijelu rada izneseni su prijedlozi za budući razvoj aplikacije, moguća poboljšanja u pogledu performansi, preciznosti i skalabilnosti sustava, kao i zaključci proizašli iz provedenog istraživanja i praktične realizacije.

# 2. Predmet istraživanja, ciljevi i metodologija rada

Ovo poglavlje bavi se opisivanjem problema i predmeta istraživanja kao i svrhom i ciljevima samoga rada. Poglavlje također uključuje metodologiju pristupa rješavanju problema kao i konkretne zadatke uključene za postizanje postavljenih ciljeva. Na kraju poglavlja objašnjava se kratka struktura diplomskog rada.

## 2.1. Problem i predmet istraživanja

Predmet istraživanja ovog rada je razvoj *web* aplikacije koja omogućuje prijenos, analizu i obradu video sadržaja u stvarnom vremenu s ciljem detekcije i prepoznavanja osoba u kontekstu sigurnosti. Sam rad istražuje funkcionalnosti prepoznavanja i identifikacije pojedinaca te opasnih predmeta i oružja. Osim tehničke realizacije same aplikacije, u radu su objašnjene i metode detekcije, kao i potreban hardver za izradu aplikacija koje koriste umjetnu inteligenciju. Teorijski su predstavljene i moguće nadogradnje aplikacije prepoznavanjem događaja nasilnog ponašanja.

## 2.2. Svrha i ciljevi istraživanja

Svrha ovog rada je izrada *web* aplikacije za unapređenje sigurnosti kroz automatizirani nadzor. Glavni cilj rada je razviti *web* aplikaciju sposobnu za detekciju i prepoznavanje osoba u stvarnom vremenu, s mogućnošću identifikacije osoba s kriminalnom prošlošću. Aplikacija ima i mogućnost detekcije oružja. Dodatni ciljevi uključuju istraživanje učinkovitosti modela za prepoznavanje nasilnog ponašanja u video sadržaju, čime bi se dodatno povećala sigurnosna funkcionalnost aplikacije u budućnosti. Kroz ovu aplikaciju želi se demonstrirati kako se na učinkovit način, suvremene tehnologije poput umjetne inteligencije mogu primijeniti u svrhu javne sigurnosti.

## 2.3. Metodologija pristupa rješavanju problema

Za razvoj softvera izabran je *web* razvoj kao moderan i relevantan pristup. *Web* aplikacija omogućava brz prijenos video podataka (*frame*-ova) s klijenta na poslužitelj korištenjem *WebSocket* veze [1], čime se dobiva funkcionalnost stvarnog vremena. Razvojni okvir korišten na poslužitelju je *FastAPI* [2] u programskom

jeziku *Python* [3]. *FastAPI* služi za jednostavnu izradu *API* (*Application Programing Interface*) zahtjeva [4], čime se ostvaruje *SOC* (*Separation Of Concerns*) arhitektura [5]. Na klijentskoj strani korišteni su programski jezici *HTML* [6], *CSS* [7] i *JavaScript* [8]. Navedeni programski jezici na klijentskoj strani izabrani su kao dovoljni za potrebe funkcionalnosti aplikacije. Za korak detekcije i identifikacije lica koristi se *face-recognition python* knjižnica [9]. Za korak detekcije oružja koristi se *Ultralytics* knjižnica te modul *YOLOv11* koji odgovara istoimenom modelu [10].

## 2.4. Zadaci uključeni za postizanje ciljeva

Rad predstavlja aplikaciju koja ima mogućnosti obrađivanja video sadržaja u stvarnom vremenu te na istome detekciju i prepoznavanje više osoba. Prepoznavanje osoba odvija se u stvarnom vremenu iscrtavajući pravokutnike oko lica. Detektirana lica provjeravaju se u bazi podataka te u slučaju prepoznavanja, relevantni podaci osobe ispisuju se iznad iscrtanog pravokutnika u stvarnom vremenu. Kada prepoznata osoba ima kriminalni dosje u bazi, odgovarajući pravokutnik i podaci iscrtavaju se crvenom bojom. Aplikacija omogućuje automatsko detektiranje nepoznatih osoba, spremanje istih u bazu i uređivanje za daljnje nadziranje. Aplikacija omogućuje i prepoznavanje oružja koja se iscrtavaju pravokutnikom crvene boje i ažuriraju u stvarnom vremenu. U radu je opisan i model za prepoznavanje nasilja kao nadogradnja aplikacije u budućnosti.

## 2.5. Struktura diplomskog rada

Diplomski rad razrađen je u sedam logičkih cjelina: Uvod, umjetna inteligencija i računalni vid, korištene tehnologije i hardver, funkcionalnosti aplikacije, nadogradnja i optimizacija aplikacije, zaključak i literatura. U uvodu se daje kratki prikaz pitanja i radnih hipoteza na koje je rad usmjeren te zašto su pitanja koja su obrađena važna. U poglavlju umjetne inteligencije i računalnog vida objašnjavaju se teoretska znanja potrebna za razumijevanje ostatka rada. Poglavlje korištene tehnologije i hardver navodi tehnologije i hardver koji su korišteni za ostvarivanje funkcionalnosti aplikacije. U funkcionalnostima aplikacije prikazuju se mogućnosti aplikacije i objašnjava programski kod istih. Poglavlje nadogradnje i optimizacija opisuje moguća proširenja aplikacije i daljnji razvoj. U zaključku su izloženi rezultati

diplomskog rada. Poglavlje literatura sadrži popis korištenih materijala za izradu samoga rada i aplikacije.

# 3. Umjetna inteligencija i računalni vid

Umjetna inteligencija je područje računalnih znanosti koje je usmjereno na razvoj sustava sposobnih za obavljanje zadataka koji zahtijevaju ljudsku inteligenciju. Umjetna inteligencije uključuje: strojno učenje, „duboko učenje“, obradu prirodnog jezika, računalni vid i ostale podskupove. Računalni vid, kao podskup umjetne inteligencije omogućuje računalima razumijevanje i interpretaciju vizualnih podataka i učenje na istima [11].

Računalni vid pripada polju umjetne inteligencije koja koristi strojno učenje i neuronske mreže za učenje računala detekciji i prepoznavanju informacija sa slika, video zapisa i ostalih vizualnih ulaza. Računalni vid započeo je 1960-ih godina s pokušajima identifikacije objekata u slikama. Kasnije s razvojem hardvera i algoritama, a osobito dolaskom konvolucijskih neuronskih mreža (*CNN*), mogućnosti računalnog vida rapidno se proširuju [12], [13].

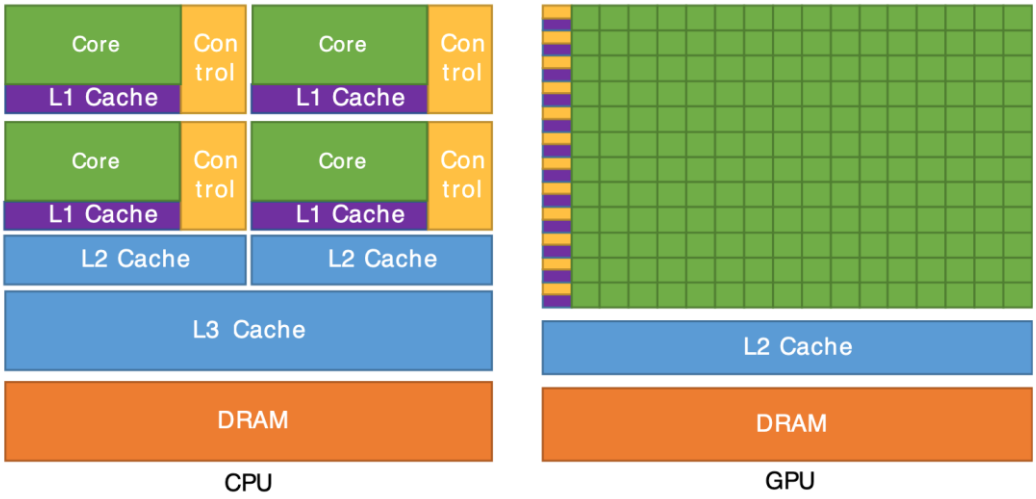
Računalni vid danas pomaže čovjeku u brojnim zadacima, od prepoznavanja proizvoda loše kvalitete na produkcijskoj traci, do detekcije neprijateljskog kretanja u ratu. Tradicionalni sustavi nadzora koji ovise isključivo o ljudskoj procjeni često su skloni pogreškama i operativnim ograničenjima. Takvi nedostaci mogu se učinkovito umanjiti automatizacijom te uvođenjem tehnologija temeljenih na umjetnoj inteligenciji [14], [15], [16].

U kontekstu ovoga rada, računalni vid koristi se za identifikaciju i evidenciju osoba na temelju značajki lica u stvarnom vremenu, detekciju oružja te teorijski za mogućnosti detekcije nasilnog ponašanja.

## 3.1. Hardverski zahtjevi

Za izradu aplikacija koje koriste računalni vid, od velike važnosti je posjedovanje adekvatnog hardvera. Jačina hardvera omogućuje brže izvršavanje i obradu, a u konačnici to dovodi do boljih rezultata. Programi računalnog vida mogu koristiti dodijeljene resurse procesora (*CPU*) ili grafičke procesne jedinice (*GPU*). Glavna razlika je u strukturi samih komponenti; centralna procesorska jedinica ima manji broj moćnih jezgri koje su optimizirane za serijsku obradu, dok grafička

procesorska jedinica ima stotine ili tisuće manjih jezgri dizajnirane za masivnu paralelnu obradu. Razlika u arhitekturi centralne procesorske jedinice i grafičke procesorske jedinice uprizorena je na slici broj 1.

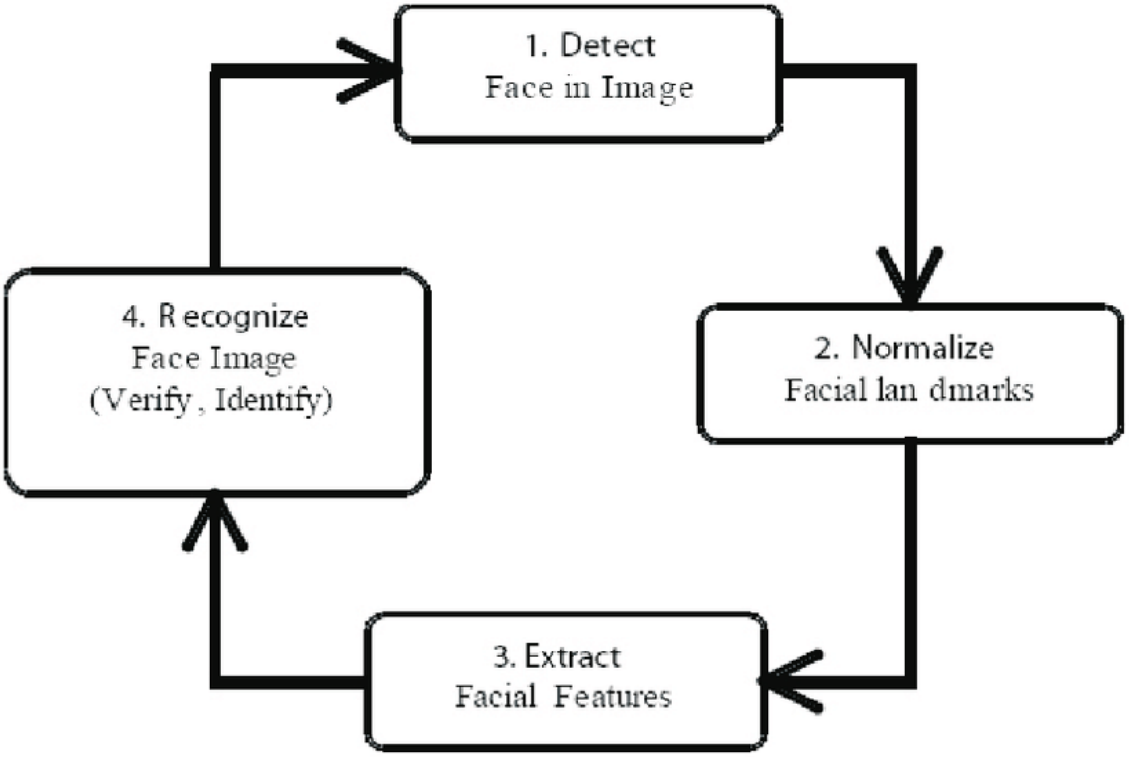



*Slika 1. Arhitektura centralne procesorske jedinice i grafičke procesne jedinice [17]*

S obzirom da u računalnom vidu mnoge funkcije koriste operacije nad većim brojem matrica i vektora (množenje, konvolucije), grafička procesorska jedinica daje znatno bolje rezultate [18], [19], [20].

## 3.2. Koraci u prepoznavanju lica

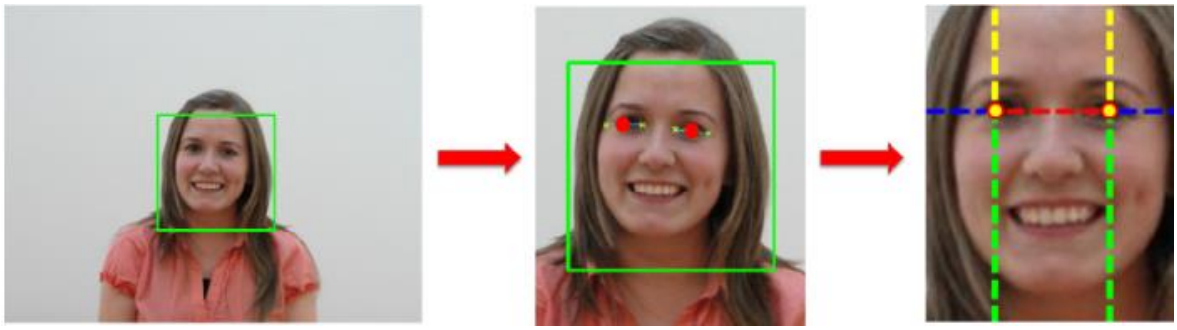
Prepoznavanje lica kroz računalni vid generalno se provodi kroz četiri standardna koraka: detekcija lica, normalizacija lica, obrada i prepoznavanje. Ciklus prepoznavanja lica prikazan je na slici broj 2. Proces prepoznavanja lica vrlo je sličan neovisno o tipu aplikacije ili programskom okviru [21], [22].




*Slika 2. Koraci u prepoznavanju lica [23]*

Prvi korak u procesu prepoznavanja lica je detekcija lica. Korak detekcije, bavi se automatskim identificiranjem ljudskog lica u vizualnom mediju (digitalna slika ili video). Cilj detekcije je izdvajanje koordinatnih lokacija lica, koje se obično prikazuju u obliku pravokutnog okvira (eng. *Bounding Box*). Ovisno o korištenom modelu za detekciju, jedinica koordinate može se razlikovati, no standard je pikselska dimenzija odnosno: *top*, *right*, *bottom*, *left*. Ovdje *top* označava pikselsku udaljenost od gornjeg ruba slike do gornjeg ruba okvira, a *right* od lijevog ruba slike do desnog ruba okvira slike itd [24], [25].

Korak normalizacije služi da detektirana lica koja ulaze u model za ekstrakciju budu obrađena pod istim uvjetima u pogledu: orijentacije, veličine i kontrasta. Ovakvom predobradom postiže se veća uniformnost ulaznih podataka što rezultira većom preciznošću samih rezultata. Jednostavan primjer normalizacije uključuje operacije poput: translacije, skaliranja i rotacije, kao što je prikazano na slici 3 [26].



*Slika 3.Translacija, skaliranje, rotacija [27]*

U koraku ekstrakcije, iz normalizirane slike lica izdvaja se numerički vektor koji računalno predstavlja to lice. Vektor se naziva *embedding* a služi kao reprezentacija lica koja se kasnije uspoređuje s drugim vektorima u koraku prepoznavanja. Jedan od poznatijih primjera *embedding*-a je 128-dimenzionalni vektor koji sadrži 128 realnih brojeva (*float* vrijednosti). Ovi vektori se generiraju korištenjem dubokih konvolucijskih neuronskih mreža, gdje svaka komponenta vektora opisuje određenu značajku lica. Modeli se treniraju tako da projiciraju lica u prostor u kojem su slična lica blizu, a različita lica udaljena. Tako se minimizira udaljenost između vektora istog identiteta, a maksimizira između različitih.

Četvrti korak je prepoznavanje lica usporedbom *embedding*-a lica koji se uspoređuju s referentnim licima pohranjenima u bazi podataka. Usporedba se provodi mjerenjem sličnosti između vektora, najčešće korištenjem euklidske udaljenosti. Ako je udaljenost između vektora ispod prethodno definirane granične vrijednosti (*Threshold*), ulazno lice se smatra prepoznatim, ako je iznad smatra se neprepoznatim. Euklidska distanca između dva vektora računa se kao korijen iz zbroja kvadrata razlika njihovih odgovarajućih komponenti [28] [29].

Opće formule za izračun euklidske distance prikazane su pod 1, 2, 3 i 4:

$$A = (a_1, a_2, \dots, a_n) \tag{1}$$

$$B = (b_1, b_2, \dots, b_n) \tag{2}$$

$$d(A,B) = \sqrt{\{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2\}} \quad (3)$$

odnosno za 128 dimenzionalne vektore A i B formula se proširuje na svih 128 komponenti:

$$d(A,B) = \sqrt{\left[\sum_{i=1}^{128}(a_i - b_i)^2\right]} \quad (4)$$

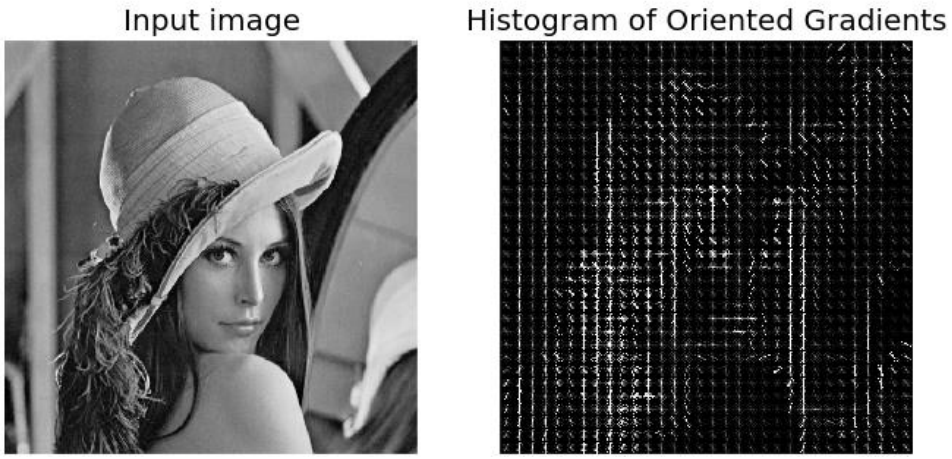
Dakle, ako distanca padne ispod proizvoljnog praga (npr. 0.6), dva lica se smatraju istom osobom, odnosno prepoznavanje je pozitivno.

## 3.3. Metode detekcije

U radu je već opisano kako je detekcija lica prvi korak u sustavima za prepoznavanje lica. Na tržištu su razvijene različite metode za detekciju lica koje se razlikuju u pristupu, složenosti i točnosti. U ovom poglavlju uspoređuju se tri poznata pristupa: *HOG* (*Histogram of Oriented Gradients*) [30] [31], *YOLO* (*You Only Look Once*) [32] i *CNN* (*Convolutional Neural Network*) [33], od kojih se u radu koriste dva od navedenih: *HOG* i *YOLO*.

### 3.3.1. *HOG (Histogram Of Oriented Gradients)*

*HOG* je metoda detekcije lica koja se temelji na analizi promjena intenziteta piksela na slici. Temeljna ideja *HOG*-a je opisivanje strukture slike nizom lokalnih gradijenata, odnosno smjera i jačine promjene svjetline na određenim regijama slike, kao što je prikazano na slici 4.




*Slika 4. HOG prikaz referentne slike [34]*

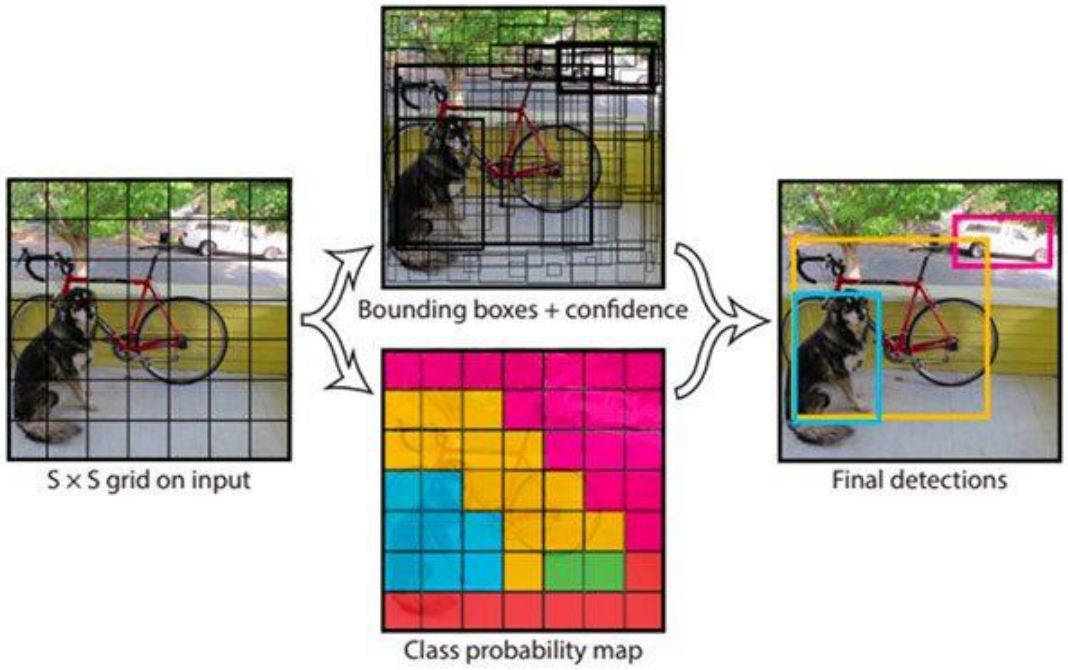
Slika se dijeli u manje ćelije, unutar kojih se za svaki piksel računa smjer gradijenta, a zatim se ti smjerovi agregiraju u histogram. Dobiveni histogrami sadrže informacije o rubovima, zakrivljenostima i teksturama na slici koje su ključne za prepoznavanje oblika lica.

*HOG* detektor koji se koristi unutar biblioteke *dlib* [35], vrlo je učinkovit za aplikacije kojima je funkcionalnost realno vrijeme jer ne koristi duboke neuronske mreže, već isključivo „orijentacijske“ značajke. Iz istog razloga *HOG* model je brz i prikladan za rad na centralnoj procesorskoj jedinici (*CPU*) bez potrebe za grafičkom procesorskom jedinicom (*GPU*). Najveće ograničenje *HOG* modela je osjetljivost na orijentaciju glave i rezoluciju, odnosno lica manja od *80x80* piksela te lica u poluprofilu ili profilu. Sukladno tome detekcija je optimalna za slike gdje je lice frontalno okrenuto prema kameri.

Unatoč ograničenjima, *HOG* model je upotrebljavan zbog svoje brzine i jednostavnosti. Model je također vrlo robustan na umjerene varijacije lica, gdje varijacije, odnosno okluzija označavaju modifikacije na samom licu poput: sunčanih naočala, maske za lice, brade, kape itd.

### 3.3.2. *YOLO (You Only Look Once)*

*YOLO (You Only Look Once)* model koristi se za detekciju u kojoj se cijela slika obrađuje u jednom prolazu kroz neuronsku mrežu. Za razliku od ostalih (tradicionalnih) metoda koje generiraju prijedloge regija (eng. *Region Proposals*), u *YOLO* modelu se slika podijeli na mrežu (eng. *Grid*), pri čemu se svakoj ćeliji dodjeljuje odgovornost za predikciju objekata čije se središte nalazi unutar te ćelije. Jednostavan prikaz prolaska kroz neuronsku mrežu YOLO modela nalazi se na slici broj 5.




*Slika 5. YOLO model, prolazak kroz neuronsku mrežu [36]*

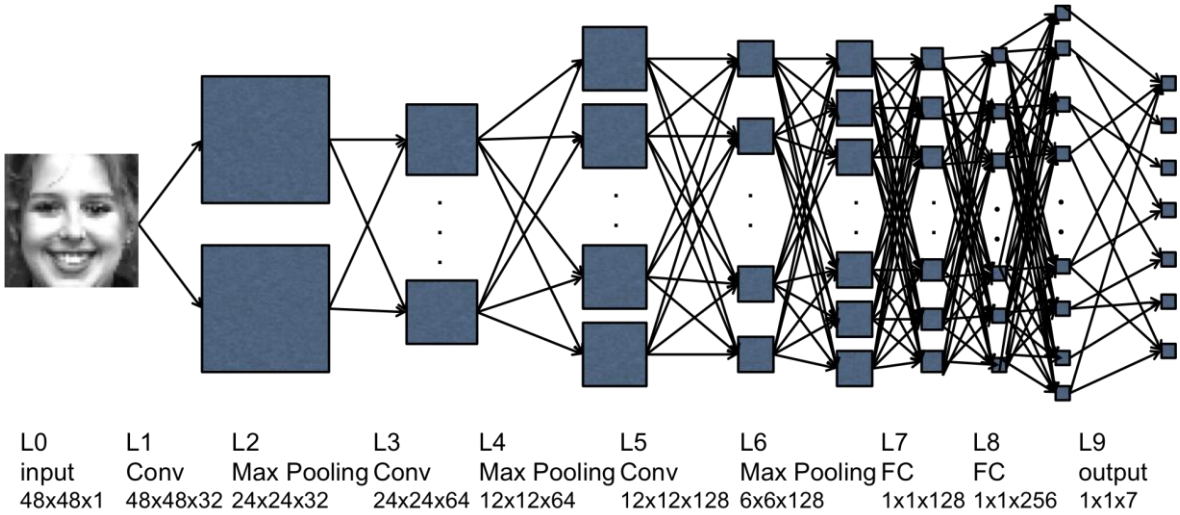
Na slici se također vidi da se za svaku ćeliju mreže predviđa unaprijed definirani broj *bounding boxova* zajedno s pripadajućim informacijama: koordinate okvira (*x, y, širina, visina*), vjerojatnost da okvir sadrži objekt (eng. *Objectness Score*) i pripadnost klasama (eng. *Class Probabilities*). Predikcija se vrši u jednoj konvolucijskoj mreži, što omogućuje brzo izvođenje u stvarnom vremenu uz odgovarajući hardver.

Trening modela provodi se s jednog kraja na drugi (*End-To-End*), pri čemu se koristi zajednička funkcija gubitka koja kombinira pogreške u klasifikaciji, lokalizaciji i pouzdanosti.

*YOLO* arhitekture od verzije 1 do najnovije 11, razlikuju se po dubini mreže, vrstama slojeva, strategijama skaliranja i načinima dekodiranja predikcija, no temeljna ideja je ista: detekcija više objekata u jednoj mrežnoj evaluaciji, bez potrebe za odvojenim koracima. Na ovaj način postiže se ravnoteža između brzine i točnosti, ipak zbog velikog broja paralelnih izračuna prolaskom kroz konvolucijsku mrežu *YOLO* model se najčešće koristi na grafičkoj procesorskoj jedinici.

### 3.3.3. *CNN (Convolutional Neural Network)*

Konvolucijske neuronske mreže (*CNN*) predstavljaju suvremeni pristup detekciji lica u kojem se umjesto ručno definiranih obilježja (npr. rubovi, gradijenti) koriste visoko-dimenzionalne značajke naučene iz podataka. Ulazna slika prolazi kroz kaskadu konvolucijskih slojeva koji generiraju višekanalne frekventne karte značajki; svaka karta naglašava drugačiji aspekt strukture lica (npr. kontura oka, luk obrve, nosni greben, itd.). Ilustrativni primjer tipične konvolucijske neuronske mreže za klasifikaciju lica prikazan je na slici 6. Na slici svaki plavi kvadratić predstavlja trodimenzionalan skup značajki (širina, visina, dubina), dok se strelicama označava prijenos podataka između slojeva, dakle, kako se mreža produbljuje, rezolucija se smanjuje, a broj kanala raste [37].




*Slika 6. Konvolucijska neuronska mreža za klasifikaciju lica [38]*

*CNN* model znatno bolje podnosi varijacije orijentacije (profil, blagi nagib, pogled usmjeren gore/dolje) i okluzije poput naočala, kape itd. Veća preciznost zahtjeva jači hardver na kojem se detekcija obavlja; stotine tisuća konvolucija po

slici otežavaju rad na procesoru pa je korištenje grafičke procesorske jedinice za ubrzanje najčešće obavezno. Zaključuje se da je brzina izvođenja kompenzirana preciznošću [39].

# 4. Korištene tehnologije i hardver

Za izradu aplikacije koja u stvarnom vremenu prepoznaje lica i oružje, bitna stavka je pravilan izbor softverskih alata i računalnih resursa. Pravilan izbor znatno utječe na performanse same aplikacije, što korisniku aplikacije na kraju pruža bolje iskustvo.

Veliki broj knjižnica za strojno učenje i umjetnu inteligenciju napisan je za *Python* programski jezik. *Python* programski jezik ima mogućnosti izrade desktop aplikacija, korištenjem *GUI* (*Graphical User Interface*) knjižnica poput: *Tkinter* [40], *PyQt* [41] ili *Kivy* [42], kao i mogućnost izrade *web* aplikacija pomoću raznih *Python* radnih okvira. Za tip aplikacije izabran je *web* razvoj s obzirom da takav daje puno bolje iskustvo u prepoznavanju lica i oružja u stvarnom vremena.

*Desktop* aplikacije u stvarnom vremenu puno teže istodobno renderiraju video i računaju konvolucije u odvojenim nitima, osim ako se ručno ne uvede multi-procesuiranje ili druge ekstenzije. To rezultira zamrzavanjem slike svaki put kad se obrađuje pojedini *frame*, pa korisnik ima dojam zastajkivanja cijelog video prikaza. *Web* aplikacija daje mogućnost samostalnog prikazivanja video sadržaja, a slanje *frame*-ova odvija se periodički kroz *WebSocket* na poslužitelju. Detekcija se tako izvodi asinkrono na poslužitelju a rezultat je prikaz videa u stvarnom vremenu neovisno o procesuiranju na poslužitelju, uz učestalo osvježavanje pravokutnika prepoznatog lica ili oružja.

## 4.1. Korišten hardver

Prilikom inicijalnog razvoja aplikacija se izvršavala na *laptop* računalu *Lenovo ThinkPad* s procesorom *13th Generation Intel Core i5-1335U*, frekvencije *1.30GHz*. Računalo je davalo zadovoljavajuće rezultate prilikom prepoznavanja lica, ovisno o broju lica koja se usporedno prepoznaju. Za usporedno prepoznavanje više lica, pravokutnici za prepoznavanje vidljivo bi zaostajali za video prikazom. S obzirom da računalo ne podržava *CUDA* (*Compute Unified Device Architecture*) arhitekturu [43] za ubrzavanje izvršavanja na grafičkoj procesorskoj jedinici, prelazak na računalo boljih performansi bilo je neizbježno.

Računalo *Lenovo Legion* pokazalo se zadovoljavajućim za potrebe aplikacije jer koristi jači procesor frekvencije *3.7GHz*. Konkretan procesor korišten za izvršavanje aplikacije je *7th Generation Intel Core i7* s brojem procesora: *14650HX*. Procesor ima 16 jezgri i 24 dretve. Novo računalo davalo je znatno bolje rezultate korištenjem samo procesorskih mogućnosti.

## 4.2. Tehnologije

Za izradu aplikacije korištene su popularne tehnologije i knjižnice u području računalnog vida i *web* aplikacija. Poslužiteljski dio aplikacije napisan je u programskom jeziku *Python* te bitne knjižnice koje koristi su: *FastApi*, *OpenCV* [44], *face_recognition*, *dlib*, *ultralytics*, *asyncio* [45]. Tehnologije koje se koriste na klijentu su: *JavaScript*, *HTML* i *CSS*. Za bazu podataka koristi se *PostgreSQL* [46] zbog jednostavnije mogućnosti dokerizacije [47] u budućnosti.

### 4.2.1. *Python* radni okvir *FastAPI*

*FastAPI* je *Python* radni okvir koji služi za izradu asinkronih *web* aplikacija temeljenih na *REST* (*Representation State Transfer*) [48] ili *WebSocket* komunikaciji. U aplikaciji je korištena verzija 0.111.0. *FastAPI* radni okvir u aplikaciji se koristi za definiciju *REST API* zahtjeva, omogućavanje komunikacije u stvarnom vremenu putem *WebSocket*-a te kao poslužitelj statičkih datoteka poput *HTML*-a, *CSS*-a i *JavaScripta*. *FastAPI* također omogućava asinkrono izvršavanje programskog koda što omogućuje brzu i skalabilnu komunikaciju između korisničkog sučelja i sustava za detekciju.

### 4.2.2. *OpenCV* knjižnica

*OpenCV* je knjižnica za računalni vid i strojno učenje. Verzija korištenog *OpenCV*-a je 4.9.0.80. U ovoj aplikaciji *OpenCV* koristi se kroz glavni modul *cv2* koji omogućava čitanje, obradu i prikazivanje slika. *OpenCV* omogućava i rad s video *stream*-ovima te razne transformacije nad slikama. Modul *cv2* u aplikaciji se konkretno koristi za: dekodiranje slika koje se šalju putem *WebSocket*-a te konverziju boja iz *BGR (*eng. *Blue, Green, Red)* u *RGB (*eng. *Red, Green, Blue)* itd.

### 4.2.3. *Face_recognition* i *dlib* knjižnice

*Face_recognition* je popularna *Python* knjižnica koja se koristi za automatsko detektiranje i prepoznavanje lica u slikama. Temelji se na *dlib* knjižnici koja sadrži implementaciju dubokih konvolucijskih mreža za generiranje 128-dimenzionalnih vektora. Korištena verzija *face_recognition* knjižnice je ujedno i trenutno zadnja: 1.3.0. Verzija *dlib* knjižnice je 19.24.0 kao kompatibilna s korištenom verzijom knjižnice *face_recognition.*

*Face_recognition* knjižnica se u aplikaciji koristi za funkcije učitavanja slika iz baze, generiranje 128-dimenzionalnih vektora, detekciju u realnom vremenu, te uspoređivanje i prepoznavanje korištenjem euklidske udaljenosti. Face_recognition knjižnica podržava dva modela detekcije lica: *HOG* i *CNN*. *HOG* model dolazi kao standardni izbor i koristi se za detekciju lica u ovoj aplikaciji.

### 4.2.4. *Ultralytics* knjižnica i *YOLOv11* model

Knjižnica *Ultralytics* koristi se za rad s *YOLO* modelima. Knjižnica omogućuje jednostavno treniranje, testiranje, evaluaciju i implementaciju *YOLO* modela u programskom jeziku *Python*. U ovoj aplikaciji koristi se *YOLOv11* model prethodno treniran na uzorku oružja (hladno i vatreno oružje).

### 4.2.5. *HTML* programski jezik

*HTML* (*HyperText Markup Language*) koristi se za definiranje strukture i sadržaja *web*-stranice. U ovoj aplikaciji, *HTML* služi za prikaz elemenata korisničkog sučelja, *GUI*-a. U korisničkom sučelju nalazi se prikaz videa u stvarnom vremenu te prikaz detektiranih osoba i oružja kroz dinamički generirane elemente. Korisniku je također omogućena interakcija s aplikacijom korištenjem raznih *HTML* skočnih prozora (*Pop-up Form*-i) povezanih s *JavaScript* funkcionalnostima. Kroz *HTML* je omogućen i prikaz referentnih slika iz baze podataka određenih osoba u zasebnom prozoru.

### 4.2.6. *JavaScript* programski jezik

*JavaScript* je skriptni programski jezik koji omogućuje dinamičku interakciju korisnika s *web* aplikacijom. Izvršava se u *web* pregledniku samog korisnika, a

omogućava dinamičke promjene sadržaja, reagiranje na korisničke akcije te komunikaciju s poslužiteljem bez ponovnog učitavanja stranice. U kontekstu ove aplikacije koristi se za: uspostavu komunikacije s poslužiteljem u stvarnom vremenu putem *WebSocket*-a, za prikazivanje rezultata prepoznavanja na video *stream*-u te za upravljanje korisničkim sučeljem (unos novih osoba, uređivanje postojećih, prikaz galerije). *JavaScript* obrađuje svu kompleksniju logiku na klijentskoj strani a direktno se odražava kroz sam *GUI (HTML i CSS).*

### 4.2.7. *CSS (Cascading Style Sheets)* programski jezik

*CSS* je jezik koji se koristi za stiliziranje izgleda *HTML* elemenata na *web* stranicama. Omogućuje odvajanje *HTML* sadržaja od prezentacije čime se postiže dosljednost, lakše održavanje i bolji vizualni doživljaj. U aplikaciji se *CSS* koristi za: dizajn pozadinske slike, stiliziranja video prikaza i sloja za iscrtavanje detekcije lica, rasporeda i stila galerije, oblikovanja skočnih prozora (*Pop-up Form*-i) za unos / uređivanje osoba, te vizualno označavanje prijetnji. Rezultat *CSS*-a u aplikaciji je pregledno, responzivno i estetski ugodno korisničko sučelje.

### 4.2.8. *PostgreSQL* relacijska baza podataka

*PostgreSQL* je sustav za upravljanje relacijskim bazama podataka otvorenog koda. *PostgreSQL* Omogućuje spremanje, dohvaćanje i upravljanje strukturiranim podacima uz podršku za napredne *SQL* upite, transakcije, indeksiranje i sigurnosne mehanizme. U ovoj aplikaciji *PostgreSQL* se koristi za pohranu osnovnih informacija o osobi, spremanje putanje do referentne slike osobe, te generalno za ažuriranje postojećih podataka i spremanje novih unosa nepoznatih osoba putem *web* forme. *PostgreSQL* koristi *SQL* upitni jezik za manipulaciju relacijskim bazama.

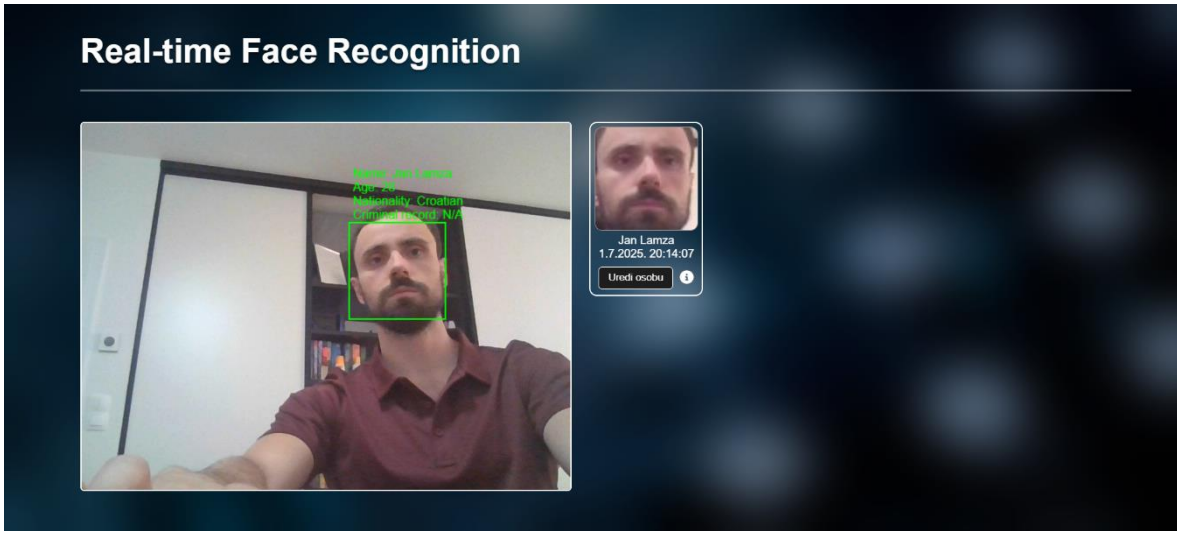
# 5. Funkcionalnosti

## 5.1. Funkcionalnosti aplikacije

U ovom poglavlju prikazane su funkcionalnosti grafičkog korisničkog sučelja (*GUI*) aplikacije razvijene za prepoznavanje lica i oružja u stvarnom vremenu. Funkcionalnosti su sistematizirane u više cjelina koje uključuju prepoznavanje jedne ili više osoba istovremeno, prepoznavanje oružja te ponašanje sustava u uvjetima djelomične okluzije. Svaka funkcionalnost demonstrirana je putem praktičnih primjera na slikama.

### 5.1.1. Prepoznavanje osoba

Detekcija lica i oružja počinje automatski prilikom paljenja aplikacije, tj. Spajanjem klijenta na poslužitelj i otvaranjem *WebSocket* veze. Slika 7 prikazuje *GUI* aplikaciju i jedno prepoznato lice u stvarnom vremenu. Na lijevoj strani slike vidi se prozor u kojem se iscrtava video u stvarnom vremenu. Na slici je također vidljivo da je osoba detektirana isrctavanjem zelenog pravokutnika oko lica. Također je vidljivo da je osoba prepoznata ispisivanjem osobnih informacija iznad zelenog prozora.

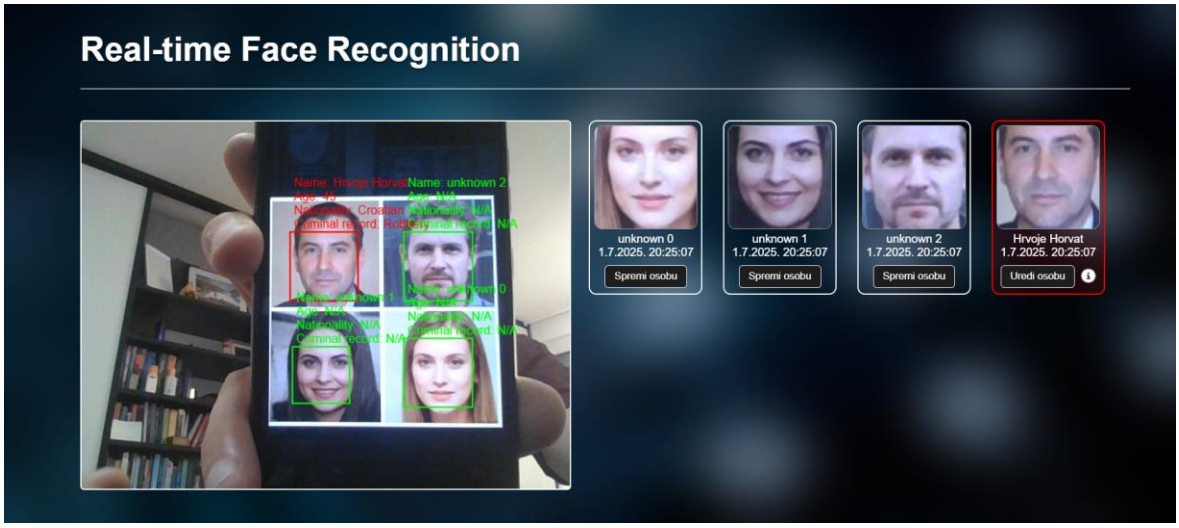



*Slika 7. Prepoznata osoba spremljena u bazi podataka*

Kada se osoba pomiče u lijevom prozoru, zeleni pravokutnik za detekciju prati osobu brzinom osvježavanja detekcije i prepoznavanja na poslužitelju. Na istoj

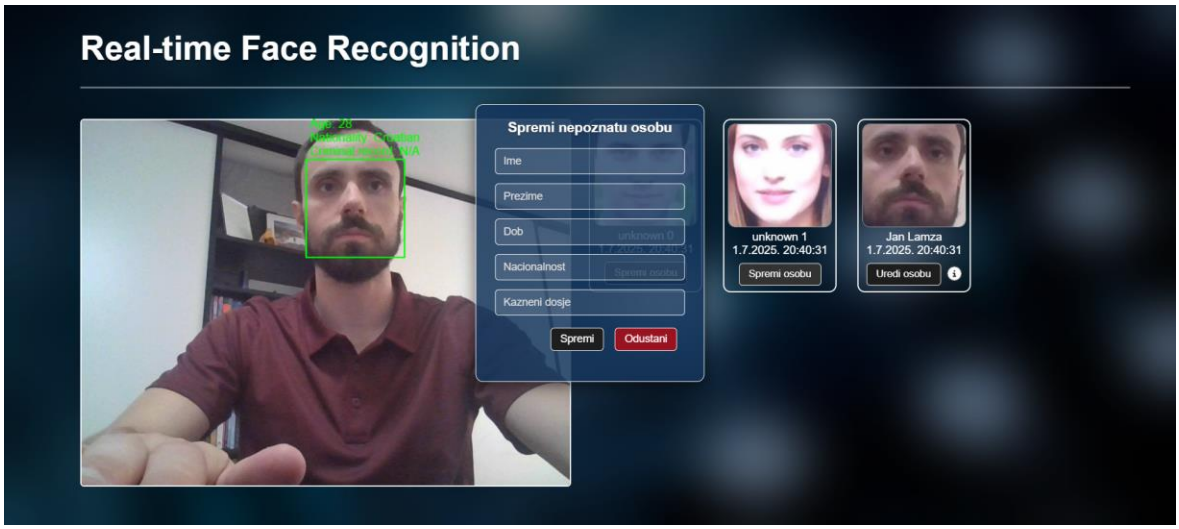
slici vidljivo je kako je osoba zabilježena na desnoj strani aplikacije u zasebnoj kartici. Kartica prethodno pohranjene osobe u bazi sadržava ime i prezime osobe, vrijeme identifikacije, gumb za uređivanje te ikonu „i“ za prikaz referentne slike u bazi po kojoj se prepoznavanje izvršilo.

Na slici broj 8 prikazan je scenarij četiri prepoznate osobe u isto vrijeme. Osobe koje nisu spremljene u bazu podataka detektiraju se i prepoznaju kao nepoznate (eng. *Unknown*). Ako se sve četiri osobe pomiču u lijevom prozoru, sva četiri pravokutnika prate osobe u realnom vremenu. Osobe koje u bazi podataka imaju polje *criminal_record* različito od *null* detektiraju se crvenim okvirom kako bi se istaknule. Sukladno prethodnome osobama koje imaju *criminal_record* različit od *null* na desnoj strani obrub kartice poprima crvenu boju.



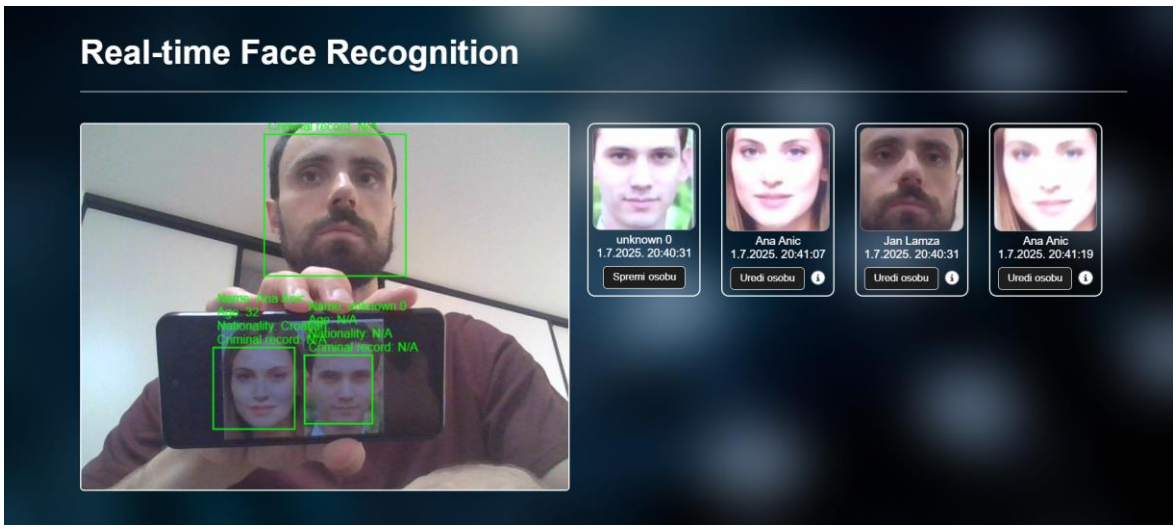

*Slika 8. Četiri detektirane i jedna prepoznata osoba*

Primjećuje se da osobe koje aplikacija nije prepoznala imaju gumb „Spremi osobu“ čijim klikom se otvara skočni prozor, kao što je prikazano na slici 9.



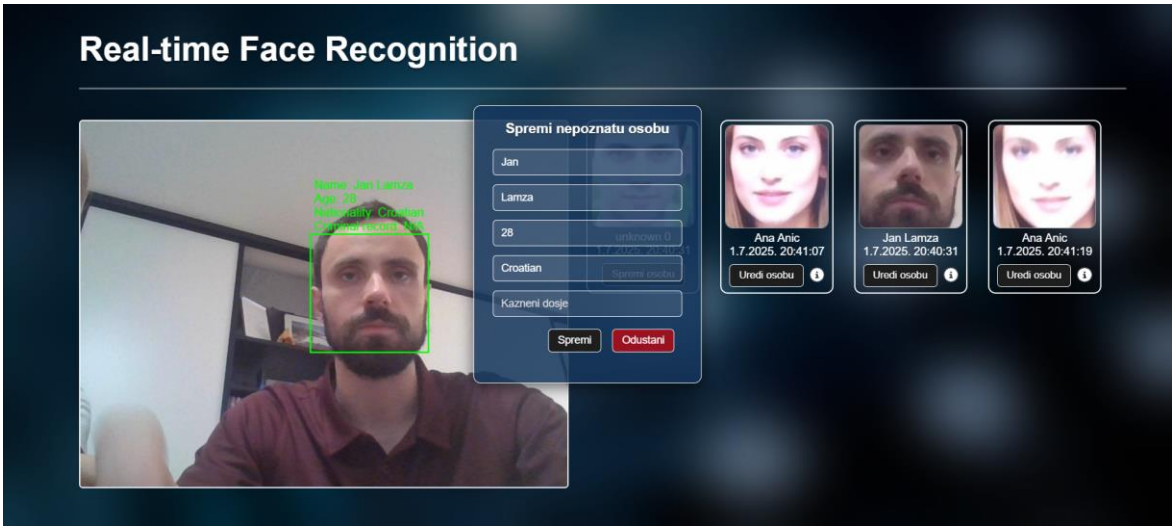

*Slika 9. Skočni prozor za pohranu nepoznate osobe*

Popunjavanjem forme i klikom na gumb „Spremi“, podaci nepoznate osobe spremaju se u bazu podataka. Kartica nepoznate osobe dinamički mijenja prikaz imena „*unknown*“ u ime i prezime iz obrasca. Dinamički je također promijenjen i gumb „Spremi osobu“ u gumb „Uredi osobu“ i dodijeljena ikona „i“ (s kojom se otvara referentna slika). Spremanjem osobe u bazu podataka osoba se odmah može detektirati i prepoznavati, čime dobiva novu karticu sada novog vremena prepoznavanja. Takav slučaj demonstriran je na slici 10. Prethodne akcije su u potpunosti dinamičke i ne ovise o gašenju ili ponovnom pokretanju aplikacije.



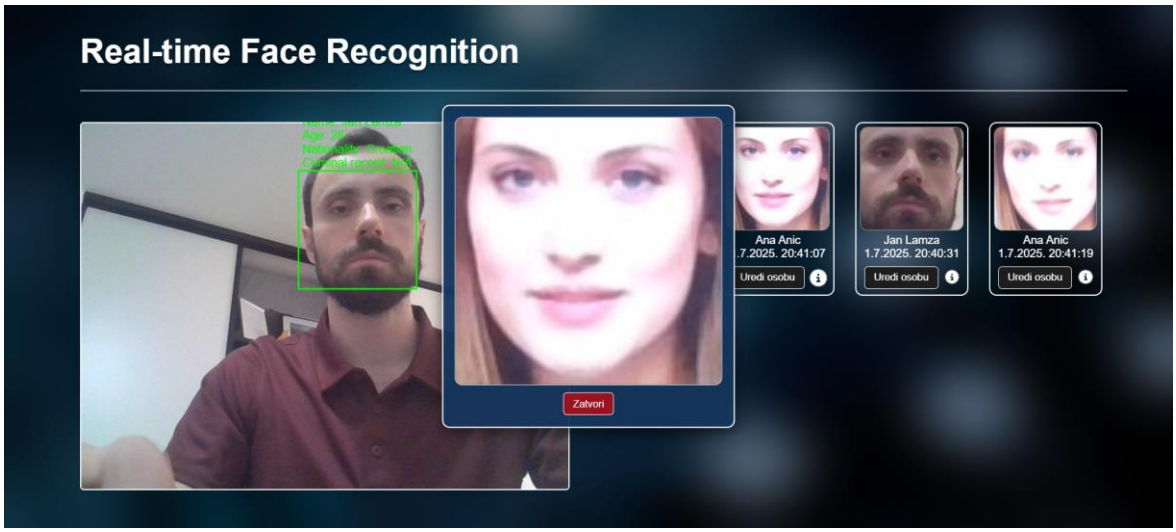

*Slika 10. Detekcija osobe Ana Anic nakon spremanja u bazu podataka*

S obzirom da je osoba Jan Lamza prethodno spremljena u bazi podataka kartica osobe ima gumb „uredi osobu“ i ikonu „i“. Klikom na gumb „Uredi osobu“, pojavljuje se skočni prozor s podacima osobe, kao što je prikazano na slici 11. Uređivanjem osobe njeni podaci su dinamički promijenjeni u lijevom prozoru video prikaza prilikom prvog sljedećeg prepoznavanja ali i odgovarajućoj kartici na desnoj strani.




*Slika 11. Skočni prozor za uređivanje osobe Jan Lamza*

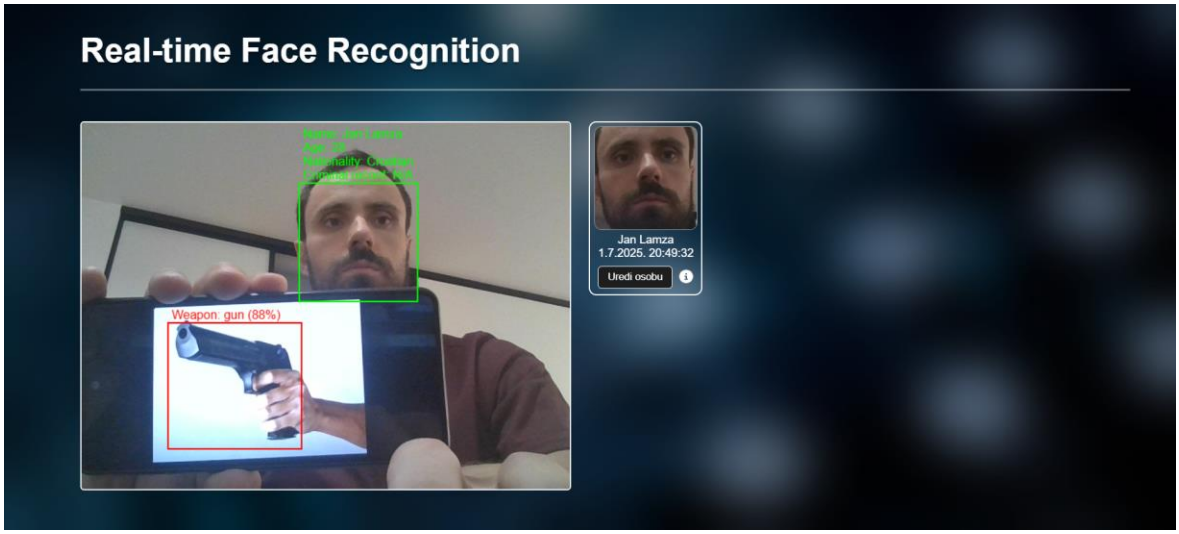
Klikom na ikonu „i“ otvara se referentna slika prepoznate osobe iz baze, kao što je prikazano na slici 12.




*Slika 12. Referentna slika po kojoj je prepoznata Ana Anic iz baze podataka*

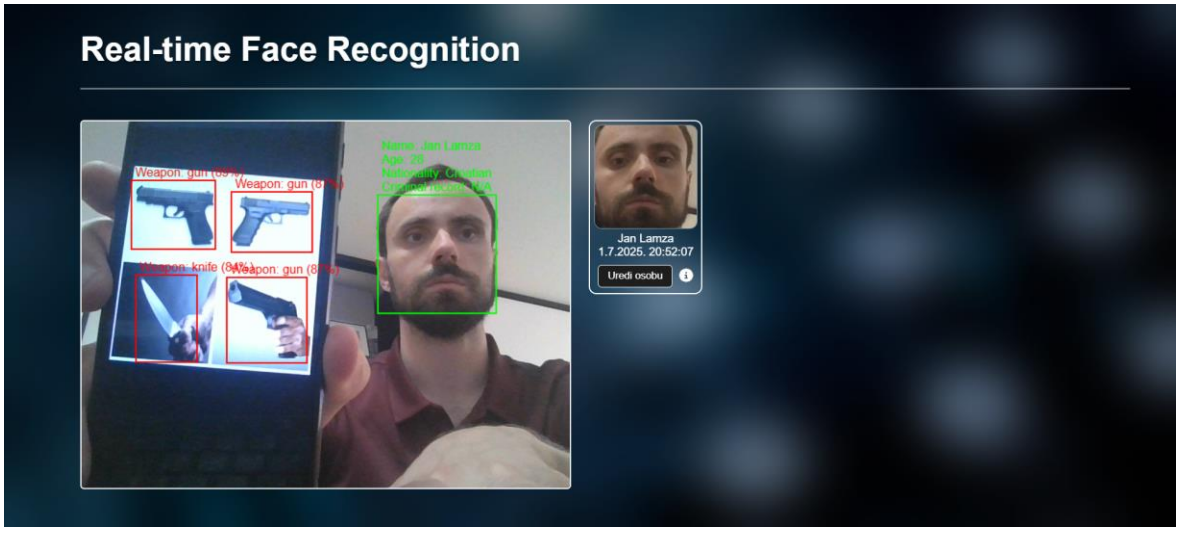
### 5.1.2. Prepoznavanje oružja

Prepoznavanje oružja omogućeno je u stvarnom vremenu u lijevom prozoru aplikacije. Model treniran za prepoznavanje oružja je zadnji model *YOLOv11* dotreniran na oružju poput noževa i pištolja. Slika 13 prikazuje detektirano oružje „*gun*“ (hrv. pištolj) uz prepoznatu osobu Jan Lamza u isto vrijeme.




*Slika 13. Prepoznata osoba Jan Lamza i detektirano oružje gun (pištolj)*

Detektirano oružje označeno je crvenim pravokutnikom oko oružja sa ispisanim tipom oružja i postotkom sigurnosti iznad pravokutnika. Na slici 14 prikazan je scenarij više detektiranog oružja s prepoznatom osobom u isto vrijeme.




*Slika 14. Četiri detektirana oružja i jedna prepoznata osoba*

Na slici je detektirano 4 tipa oružja s različitim postocima sigurnosti od kojih su tri pištolj a jedan nož. Oružja koja se mogu prepoznavati u aplikaciji sa trenutnim *YOLO* modelom su nož, puška i pištolj.

### 5.1.3. Okluzije i točnost prepoznavanja lica

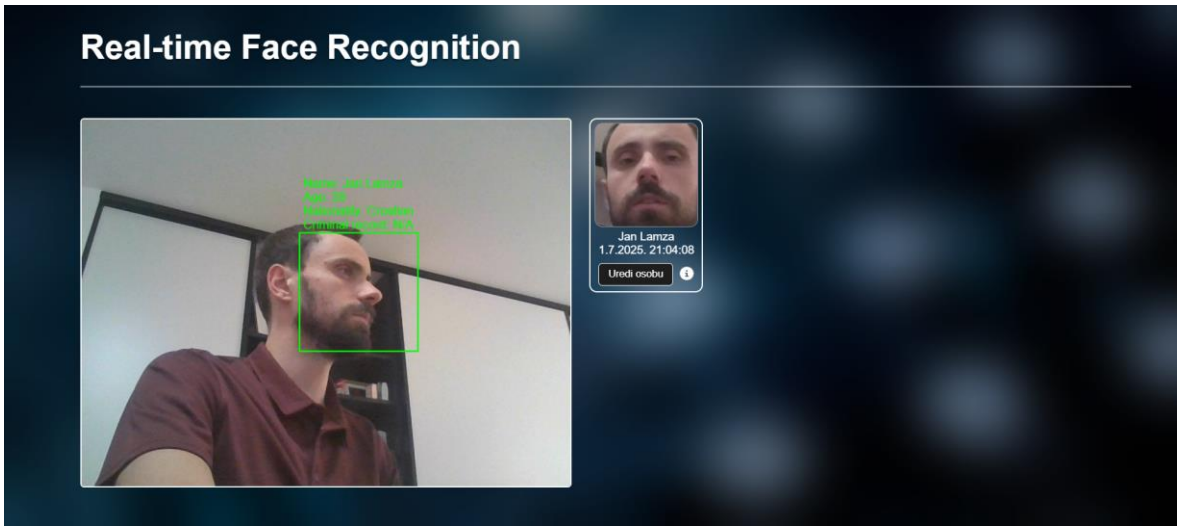
U kontekstu prepoznavanja lica, euklidska udaljenost se često koristi kao mjera sličnosti između dvaju vektora, no ona nije interpretabilna kao postotak točnosti. Euklidska udaljenost je metrička udaljenost u višedimenzionalnom prostoru, pa ne može označavati klasifikacijski ishod koji se može kvantificirati u postocima sigurnosti.

Za razliku od prave točnosti, koja se definira kao omjer točnih predikcija nad ukupnim brojem testnih uzoraka, euklidska udaljenost je neovisna o klasama i sama po sebi ne daje informaciju o stupnju uspješnosti prepoznavanja. Usporedbom udaljenosti s unaprijed definiranim pragom moguće je izvesti binarnu odluku o podudaranju, ali i tada rezultat nije postotak nego logički ishod (osoba prepoznata / osoba nije prepoznata).

Euklidska udaljenost je dakle, niskorazinska mjera sličnosti, koja se koristi kao temelj za odlučivanje, ali ne predstavlja semantičku mjeru točnosti sustava. Takva interpretacija može dovesti do pogrešnih zaključaka o točnosti algoritma i same aplikacije.

Okluzije s druge strane predstavljaju veliki izazov za algoritme prepoznavanja jer direktno utječu na samu točnost. Okluzija predstavlja elemente koji djelomično ili potpuno blokiraju korisnikovo lice poput: naočala, šala, položaja lica itd. U ovome poglavlju empirijska točnost aplikacije se mjeri mogućnošću detekcije i prepoznavanja lica s tri popularne okluzije: profil, naočale i ruka preko usta.

Slika 15 prikazuje uobičajeno težak položaj lica za detekciju i prepoznavanje; profil, odnosno skoro puni profil. Na slici je vidljivo da je u lijevom prozoru osoba detektirana i prepoznata. U slučaju punog profila osoba se nekada detektira kao nepoznata a u znatnom broju slučajeva ne detektira.



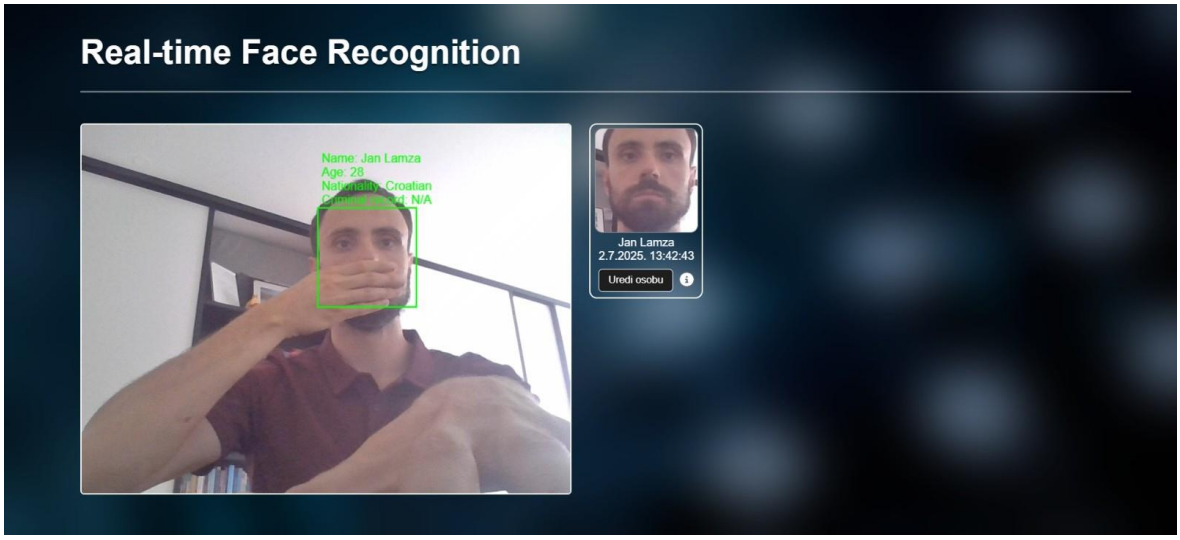

*Slika 15. Poluprofil prepoznate osobe*

Na slici broj 16 prikazana je okluzija sunčanim naočalama. Sunčane naočale smanjuju mogućnost detekcije jer čine referentne točke za izradu vektora koje se nalaze oko očiju nedostupne. Na slici je vidljivo da je algoritam za detekciju uspio detektirati lice a model za prepoznavanje upariti isti s referentnim licem iz baze.


*Slika 16. Okluzija sunčanim naočalama*

Slika 17 prikazuje okluziju rukom preko usta čime se simulira maska za lice. Maske za lice onemogućavaju detekciju referentnih točaka oko područja usta, čime se narušava kvaliteta detekcije lica i prepoznavanja. Na slici je vidljivo da je osoba prepoznata i detektirana s rukom stavljenom preko usta. U slučaju okluzije rukom

preko usta aplikacija ponekad detektira pojedinca ali ga ne prepoznaje, što nekada rezultira novom „*unknown*“ karticom osobe na desnoj strani prozora.



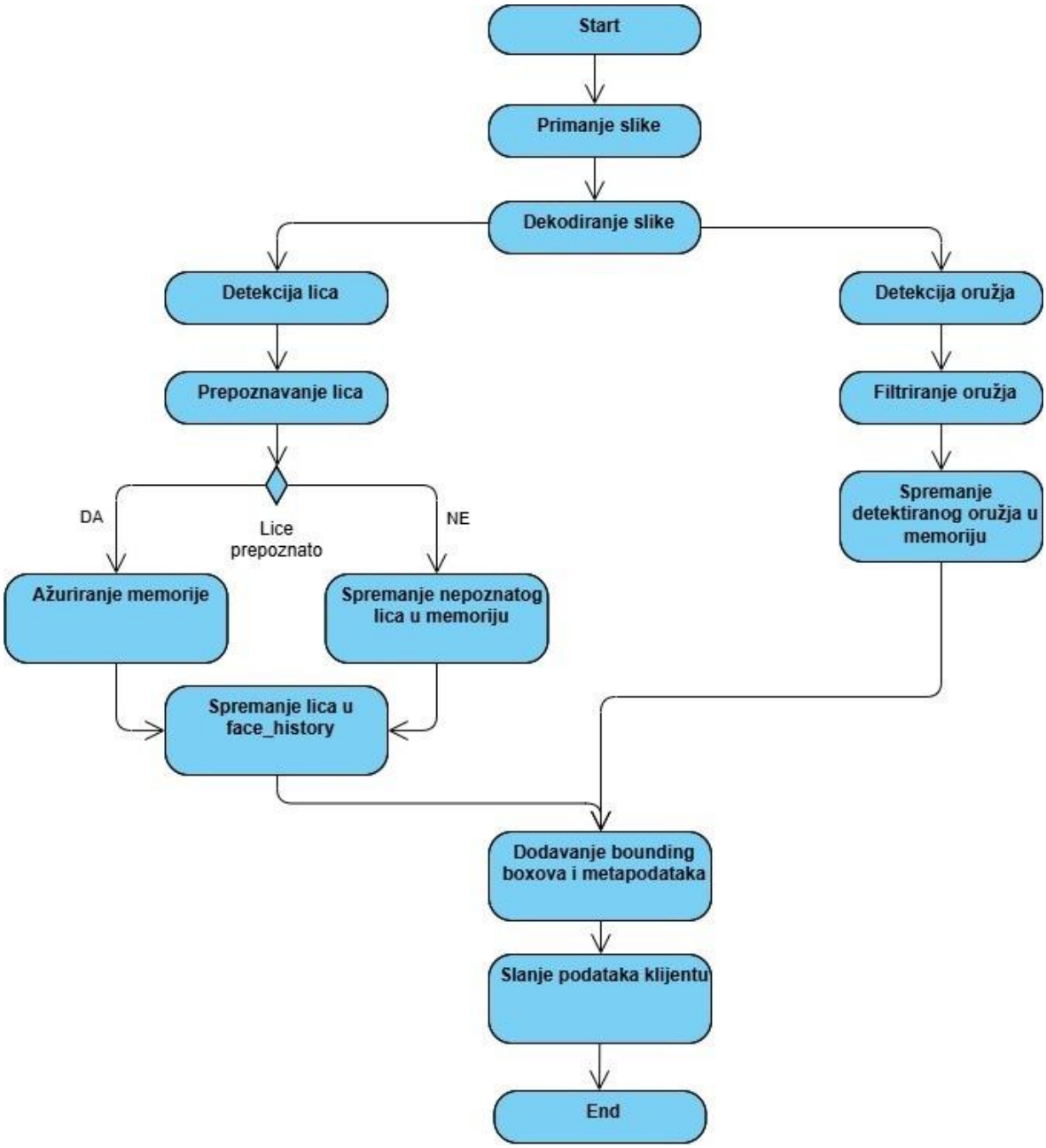

*Slika 17. Okluzija rukom preko usta*

## 5.2. UML dijagram aktivnosti

Na slici 18 prikazan je dijagram aktivnosti koji vizualno prikazuje sve ključne faze kroz koje prolazi jedna slika (eng. *frame*) unutar aplikacije prilikom prepoznavanja lica i detekcije oružja. Dijagram odgovara stvarnom tijeku obrade implementiranom u sustavu i temelji se na podkoracima koji su opisani u narednim poglavljima.

Nakon što se slika primi i dekodira, paralelno se odvijaju dvije grane obrade: prepoznavanje lica i detekcija oružja. Ako je lice prepoznato, ažurira se memorija s informacijama o osobi. Ako lice nije prepoznato, slika i *encoding* se također spremaju u memoriju kao "nepoznato" lice. Dakle, u oba slučaja lice se sprema u *face_history* listu. Oružje se detektira pomoću *YOLO* modela i rezultati se također spremaju u memoriju, ograničeno brojem i pouzdanošću detekcija, odnosno do četiri oružja i sigurnost veća od 80%.

Na kraju se svi rezultati kombiniraju, dodaju se *bounding box*-ovi i pripadajući metapodaci te se grupno šalju natrag klijentu za prikaz na *web* sučelju.




*Slika 18. UML dijagram toka obrade slike*

## 5.3. Funkcionalnosti na razini programskog koda

Aplikacija je podijeljena na poslužiteljsku i klijentsku stranu. Na poslužiteljskoj strani glavni dio programskog koda nalazi se u datoteci *main.py*. Na klijentskoj strani glavna logika se nalazi u datoteci *javascript.js*, dok se izgled same aplikacije nalazi u datotekama *index.html* i *style.css*. Ovo poglavlje ima za cilj objasniti glavne funkcionalnosti aplikacije kroz programske isječke *Main.py* datoteke na poslužitelju.

### 5.3.1. *Main.py* datoteka

*Main.py* datoteka sadržava logiku za spajanje na bazu podataka, funkcije za detekciju i prepoznavanje, kao i same *API* pozive, odnosno sadržava logiku poslužitelja.

Na početku datoteke pokreće se glavna *ASGI* aplikacija [49]. Ova instanca služi kao centar u kojem se registriraju rute (*HTTP i WebSocket*), *middleware* i *event-handleri*, kao što je prikazano u isječku koda 1. U isječku je također prikazano montiranje statičkih datoteka, odnosno sav promet na *URL* putanji */static/** poslužuje se direktno s diska iz mape *static/* (*CSS*, *JS,* slike, modeli). Prikazano je i spajanje na bazu podataka *PostgreSQL* knjižnicom *psycopg2*. Parametri za spajanje čuvaju se u varijablama okruženja. Veza prema bazi se ostvaruje jednom prilikom podizanja aplikacije, što je dovoljno za jednu procesnu instancu.

```
# ------------------------------------------------------------------------
# Inicijalizacija FastAPI aplikacije i montiranje statičkih datoteka
# ------------------------------------------------------------------------

app = FastAPI()
app.mount("/static", StaticFiles(directory="static"), name="static")

# ------------------------------------------------------------------------
# Inicijalizacija baze podataka i paralelnog izvršavanja
# ------------------------------------------------------------------------

# Povezivanje s PostgreSQL bazom putem psycopg2
conn = psycopg2.connect(
    dbname=os.getenv("DB_NAME"),
    user=os.getenv("DB_USER"),
    password=os.getenv("DB_PASS"),
    host=os.getenv("DB_HOST"),
    port=os.getenv("DB_PORT"),
)
```

*Isječak koda 1. Inicijalizacija aplikacije i spajanje na PostgreSQL bazu*

U isječku koda 2 prikazane su konfiguracije same aplikacije vezano za detekciju i prepoznavanje lica. Inicijalizira se mogućnost paralelne obrade više lica istovremeno pomoću četiri dretve. Na taj način svaka dretva zasebno računa *face embedding*, odnosno 128-dimenzionalni vektor lica.

U isječku su prikazane i varijable koje definiraju ponašanje aplikacije kao: minimalni razmak između prepoznavanja, te prag sličnosti s kojim se uspoređuje euklidska distanca detektirane i referentne slike osobe. U isječku je vidljiva inicijalizacija liste poznatih *encoding*-a iz baze podataka, te odgovarajućih meta-podataka (ime, prezime, dob, državljanstvo i kriminalni dosje). Inicijalizirane su i liste za koordinate i meta-podatke trenutno vidljivih lica u kadru što služi za crtanje pravokutnika preko videa u stvarnom vremenu. Također inicijaliziran je i kružni red tipa *deque* koji sadržava najviše 15 elemenata tipa rječnik, čime se omogućava distinkcija između zadnjih 15 detektiranih ali nepoznatih lica.

```
# Thread pool za paralelno računanje face embeddinga (4 CPU jezgre)
enc_executor = ThreadPoolExecutor(max_workers=4)

# ---------------------------------------------------------------------------
# Parametri i memorija za prepoznavanje lica
# ---------------------------------------------------------------------------

FACE_RECOGNITION_INTERVAL: float = 0.2        # Minimalni razmak (sekunde) iz
                                                među pokušaja prepoznavanja
UNKNOWN_FACE_MATCH_THRESHOLD: float = 0.6     # Prag sličnosti za praćenje ne
                                                poznatih osoba

face_busy = False                             # Oznaka da je obrada lica u ti
                                                jeku
known_face_encodings: List[np.ndarray] = []   # Encodinzi poznatih osoba iz
                                                 baze
known_face_metadata: List[Dict] = []          # Prateći metapodaci za poznata
                                                 lica

latest_face_boxes: List[Dict] = []            # Detektirani boxovi lica za o
                                                verlay
face_history: Deque[Dict] = deque(maxlen=15)  # Povijest nedavnih prepoznava
                                                nja za prikaz
```

*Isječak koda 2. Okruženje i parametri za prepoznavanje lica*

Isječak koda 3 prikazuje konfiguraciju, parametre i varijable vezane za detekciju oružja. Interval detekcije oružja stavljen je kao 0.2 sekunde s obzirom na hardverske sposobnosti. Pouzdanost podudaranja je postavljena na 0.8, naglašava se da ovo nije prag koji je postavljen u parametrima za prepoznavanje lica. Prag pouzdanosti ovdje označava odbacivanje svih detektiranih okvira s pouzdanošću manjom od 0.8, dok se prag kod prepoznavanja lica odnosi na euklidsku distancu, dakle, sve distance veće od 0.6 se odbacuju.

Definirana se klasa koja označava sve tipove oružja koje aplikacija želi detektirati, te maksimalan broj detekcija u isto vrijeme. Učitava se do-trenirani *yolov11m* (*Medium*) model za detekciju oružja nazvan *yolov11-weapon.pt*. U isječku koda se definira lista rječnika koja sadržava sve informacije potrebne za prikaz tog objekata (oružja) korisničkom sučelju.

```
# ------------------------------------------------------------------------
# Parametri i memorija za detekciju oružja
# ------------------------------------------------------------------------

YOLO_INTERVAL: float = 0.2                    # Minimalni razmak (sekunde) između YOLO
                                                detekcija
YOLO_CONFIDENCE_THRESHOLD: float = 0.8        # Minimalna pouzdanost detekcije da bi se
                                                prihvatila
WEAPON_WATCHED_CLASSES = {"pistol", "rifle", "knife", "gun"}  # Klase koje se promatraju
MAX_WEAPON_COUNT = 4                          # Maksimalan broj boxova koje se prikazuje

yolo_model = YOLO("models/yolo11-weapon.pt")  # Učitani YOLO model za detekciju oružja
yolo_busy = False                             # Oznaka da je YOLO detekcija u tijeku
latest_weapon_boxes: List[Dict] = []          # Detektirani boxovi oružja za overlay
```

*Isječak koda 3. Okruženje i parametri za detekciju oružja*

Isječak koda 4 prikazuje definiciju funkcije *load_known_faces_from_db* čija je svrha učitavanje poznatih osoba iz baze podataka i priprema njihovih vektora lica kako bi se kasnije u aplikaciji nepoznate osobe mogle uspoređivati sa istima. Funkcija otvara vezu prema bazi podataka, provjerava ako za svaku spremljenu osobu postoji putanja ka pripadajućoj slici, generira vektore lica te pohranjuje vektore u listu poznatih. Funkcija se odmah po definiciji i poziva u aplikaciji.

```python
# ------------------------------------------------------------------------
# Učitava poznate osobe iz baze i priprema encodirane vektore za prepoznavanje
# ------------------------------------------------------------------------
def load_known_faces_from_db() -> None:
    with conn.cursor() as cursor:
        cursor.execute("""
            SELECT id, image_path, name, surname, age, nationality, criminal_record
            FROM persons
        """)
        for row in cursor.fetchall():
            person_id, image_path, name, surname, age, nationality, criminal_record = row

            if not os.path.exists(image_path):
                print(f"[WARN] Slika ne postoji: {image_path}")
                continue

            image = face_recognition.load_image_file(image_path)
            encodings = face_recognition.face_encodings(image)
            if not encodings:
                print(f"[WARN] Nema detektiranog lica u slici: {image_path}")
                continue

            known_face_encodings.append(encodings[0])
            known_face_metadata.append({
                "id": person_id,
                "name": name,
                "surname": surname,
                "age": age,
                "nationality": nationality,
                "criminal_record": criminal_record,
            })
# Poziv funkcije odmah pri pokretanju aplikacije
load_known_faces_from_db()
```

*Isječak koda 4. Funkcija za obradu poznatih osoba za prepoznavanje*

U isječku koda 5 prikazana je funkcija *get_img* koja je ključna za obradu same slike prije daljnje upotrebe u aplikaciji. Svaka slika (eng. *Frame*) koja se šalje s klijenta preko *websocket*-a je enkodirana u *base64* formatu koji se treba dekodirati, te pretvoriti u *openCV* kompatibilan format *BGR*. Funkcija se koristi za predobradu slike za detekciju lica i oružja.

```python
# ------------------------------------------------------------------------
# Pretvara base64 Data-URI (npr. "....") u BGR sliku (ndarray)
# Vraća: NumPy sliku ili None ako dekodiranje ne uspije.
# ------------------------------------------------------------------------
def get_img(data_uri: str) -> Optional[np.ndarray]:
    """
    1) Odreže meta-dio prije zareza („”).
    2) Base64-dekodira ostatak u niz bajtova.
    3) Pretvori bajtove u NumPy vektor i OpenCV-om dekodira u BGR sliku.
    """
    try:
        header, b64_data = data_uri.split(",", 1)      # "data:..." | "base64..."
        nparr = np.frombuffer(base64.b64decode(b64_data), np.uint8)
        if nparr.size == 0:                            # zaštita od praznog dekodiranja
            raise ValueError("prazan niz bajtova")
        return cv2.imdecode(nparr, cv2.IMREAD_COLOR)   # BGR format
    except Exception as exc:                           # binascii.Error, ValueError
        print(f"[WARN] get_img() decoding failed: {exc}")
        return None
```

*Isječak koda 5. Obrada pojedine slike za openCV*

U isječku koda 6 prikazan je prvi dio funkcije *recognize_faces* za prepoznavanje lica na slici. Funkcija se poziva kasnije u kodu u *WebSocket* zahtjevu. Koraci same funkcije lijepo su dokumentirani u obliku komentara u kodu.

U drugom koraku vidljivo je korištenje *face_recognition_interval* varijable koja je postavljena na 0.2 kako bi se smanjilo opterećenje za *CPU*. U tom koraku se zapravo radi provjera ako je zadnje prepoznavanje bilo nedavno ili nema detektiranih lica, u kojem slučaju funkcija preskače obradu i vraća prazan rezultat.

```
# ------------------------------------------------------------------------
# Prepoznaje lica na slici, vodi evidenciju poznatih / nepoznatih i priprema
#     face_boxes  –  boxovi za overlay
#     new_captures –  izrezi novootkrivenih lica za galeriju
# Funkcija je optimizirana za brzinu:
#   1)  detektira sva lica (HOG/CNN ovisno o face_recognition konfigu)
#   2)  kontrolira učestalost obrade (FACE_RECOGNITION_INTERVAL)
#   3)  računa embeddinge paralelno u ThreadPool-u (enc_executor)
#   4)  uspoređuje s poznatima; nepoznate grupira i prati
#   5)  sprema rezultate u globalne strukture (face_boxes, face_history …)
# ------------------------------------------------------------------------
def recognize_faces(
    img: np.ndarray,
    seen_people: set,
    seen_unknown_encodings: List[np.ndarray],
    last_recognition_time: float
) -> Tuple[List[Dict], List[Dict], float]:

    face_boxes: List[Dict] = []
    new_captures: List[Dict] = []

    # ---------------------------------------------------------------
    # 1) Detekcija lokacija lica
    # ---------------------------------------------------------------
    rgb_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    face_locations = face_recognition.face_locations(rgb_img)

    # ---------------------------------------------------------------
    # 2) FPS kontrola – preskače ako je obrada prečesta ili nema lica
    # ---------------------------------------------------------------
    now              = time.time()
    do_recognition   = (now - last_recognition_time) > FACE_RECOGNITION_INTERVAL
    if not (do_recognition and face_locations):
        return face_boxes, [], last_recognition_time
```

*Isječak koda 6. Definicija funkcije recognize_faces i navod koraka kroz koje funkcija prolazi*

Treći korak prikazan je u isječku koda 7. U koraku se za svako detektirano lice pokreće poseban zadatak u *ThreadPoolExecutor*-u koji poziva funkciju za generiranje 128-dimenzionalnog vektora za svako lice. Na ovaj način omogućava se usporedna obrada četiri lica, od kojih je svakome posvećena zasebna jezgra procesora. Ovako se povećava brzina izvođenja zahtjevnih operacija za prepoznavanje i detekciju.

```
# ------------------------------------------------------------------
# 3) Paralelno računanje embeddinga (jedan future po licu)
# ------------------------------------------------------------------
futures = [
    enc_executor.submit(
        face_recognition.face_encodings,
        rgb_img,
        [location]              # ← svaki location: (top, right, bottom, left)
    )
    for location in face_locations
]

# Pairing location ↔ embedding, zadržavamo redoslijed i izbacujemo prazne
paired = [
    (loc, embedding[0])         # embedding lista → uzmi prvi vektor
    for loc, embedding in zip(face_locations, (f.result() for f in futures))
    if embedding
]
```

*Isječak koda 7. Paralelno računanje embedding-a do četiri detektirana lica*

Isječak koda 8 prikazuje četvrti korak funkcije *recognize_faces* koji se bavi obradom detektiranog lica za koje je u prošlom koraku izrađen *embedding*. Četvrti korak sadrži ukupno 4 podkoraka: usporedba s poznatim licima, grupiranje lica među nepoznatima, manipulacije nad galerijom te pravokutnik s koordinatama za lokaciju lica na klijentu. Isječak koda 8 prikazuje prvi podkorak označen s A koji radi usporedbu detektiranih lica s poznatim licima iz baze podataka. Tako funkcija *compare_faces* vraća niz *bool* vrijednosti prema zadanoj toleranciji (= 0.6). Funkcija *face_distance* daje euklidsku udaljenost svakog *embedding*-a do svih poznatih. Kada je pronađen odgovarajući par, meta-podaci osobe se kopiraju u lokalni meta rječnik a *person_id* se postavlja na odgovarajući *id*.

```
# ------------------------------------------------------------------
# 4) Obrada svakog lica: validacija boxa, usporedba, bilježenje
# ------------------------------------------------------------------
for (top, right, bottom, left), face_enc in paired:

    # Geometrijska provjera boxa
    if not is_valid_box(left, top, right, bottom):
        continue

    # ----- osnovni meta ----------------------------
    meta = {
        "name": "Unknown",
        "surname": "",
        "age": "",
        "nationality": "",
        "criminal_record": ""
    }
    person_id = None

    # ------------------------------------------------------
    # A) Usporedba s poznatim licima
    # ------------------------------------------------------
    if known_face_encodings:
        matches   = face_recognition.compare_faces(known_face_encodings, face_enc)
        distances = face_recognition.face_distance(known_face_encodings, face_enc)
        best_idx  = int(np.argmin(distances))
        if matches[best_idx]:
            meta      = known_face_metadata[best_idx]
            person_id = meta["id"]
```

*Isječak koda 8. Serijska obrada svakog para koordinata i embedding-a*

Isječak koda 9 prikazuje podkorake B i C. Podkorak B radi grupiranje nepoznatih lica. Na taj način detektirano lice se provjerava u bazi i ako *person_id* ne postoji lice se uspoređuje s ostalim nepoznatim a detektiranim (eng. *Unknown*) osobama. Tako se izbjegava scenarij u kojem se ponavljaju iste nepoznate osobe kao različite. Podkorak C bavi se funkcionalnostima kartica osoba na klijentu. Svaka osoba s poznatim *person_id*-jem ili nepoznata (eng. *Unknown*), dodaje se u *seen_people set*. Na taj način osobe (poznate i nepoznate) su zapamćene u kartici samo prilikom prve detekcije na slici. U suprotnom osobe bi imale broj kartica odgovarajućem broju *frame*-ova poslanih s klijenta na poslužitelj.

```
# ------------------------------------------------------
# B) Ako lice još nije poznato – grupiraj ga među "unknown"
# ------------------------------------------------------
if person_id is None:
    for idx, unk_enc in enumerate(seen_unknown_encodings):
        if np.linalg.norm(unk_enc - face_enc) < UNKNOWN_FACE_MATCH_THRESHOLD:
            person_id  = f"unknown {idx}"
            meta["name"] = person_id
            break
    else:
        seen_unknown_encodings.append(face_enc)
        person_id  = f"unknown {len(seen_unknown_encodings) - 1}"
        meta["name"] = person_id

# ------------------------------------------------------
# C) Galerija – novo lice pojavi se prvi put
# ------------------------------------------------------
if person_id not in seen_people:
    seen_people.add(person_id)
    crop = img[top:bottom, left:right]
    _, jpeg = cv2.imencode(".jpg", crop)
    b64_img = base64.b64encode(jpeg.tobytes()).decode()

    new_captures.append({
        "id": meta.get("id"),
        "name": meta["name"],
        "surname": meta["surname"],
        "time": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
        "image": f"{b64_img}",
        "criminal_record": meta.get("criminal_record", "")
    })
```

*Isječak koda 9. Rukovanje nepoznatim licima i funkcionalnosti galerije*

U isječku koda 10 prikazuje se priprema podataka za prikaz na klijentu i povrat same funkcije *recognize_faces*. Za svako obrađeno lice kreira se rječnik s koordinatama okvira *(Top, Right, Bottom, Left)* i pratećim meta-podacima. Rječnik se dodaje u listu *face_boxes* koja se *websocket*-om šalje klijentu gdje se obrađuje. U isječku koda je vidljivo da funkcija vraća i listu rječnika *new_captures* gdje se nalaze izrezana lica koja se pojavljuju prvi put a prikazuju se na klijentu u karticama. Funkcija na kraju vraća i *now* ažurirani vremenski pečat koji služi za kontrolu učestalosti poziva funkcije kroz varijablu *face_recognition_interval*.

```
            # ------------------------------------------------------
            # D) Spremi box za overlay
            # ------------------------------------------------------
            face_boxes.append({
                "top":          top,
                "right":        right,
                "bottom":       bottom,
                "left":         left,
                "id":           meta.get("id"),
                "name":         meta.get("name", "Unknown"),
                "surname":      meta.get("surname", ""),
                "age":          meta.get("age", ""),
                "nationality":  meta.get("nationality", ""),
                "criminal_record": meta.get("criminal_record", "")
            })

    # ------------------------------------------------------------------
    # 5) Vraćamo rezultat + ažurirani timestamp
    # ------------------------------------------------------------------
    return face_boxes, new_captures, now
```

*Isječak koda 10. Priprema podataka za prikaz i povrat funkcije*

U isječku koda 11 prikazana je funkcija za detekciju oružja. Na početku funkcija kreira praznu listu rječnika za spremanje koordinata za iscrtavanje pravokutnika, vrste oružja i postotka sigurnosti. Slika za obrađivanje prosljeđuje se učitanom modelu (*yolov11-weapon.pt*). Parametar *max_det* ograničava broj ukupnih detekcija kako bi se ubrzala obrada (četiri istovremeno), čim se postigne maksimalna dozvoljena vrijednost petlja se prekida. Funkcija zadržava samo detekcije čija klasa pripada skupu *weapon_detection_classes (Pistol, Rifle, Knife, Gun)* i čija je pouzdanost veća od zadanog praga *yolo_confidence_treshold* (odnosno 0.8). Za svaku valjanu detekciju oružja koja ima određeni postotak sigurnosti i pripada željenoj klasi puni se rječnik u listi i vraća kao izlaz funkcije.

```python
# ------------------------------------------------------------------------------
# Funkcija za detekciju oružja
# Koristi YOLO model s filtriranjem po nazivu klase, pouzdanosti i geometriji boxa.
# Broj detekcija ograničen s MAX_WEAPON_COUNT.
# Vraća listu bounding-boxova oružja za overlay.
# ------------------------------------------------------------------------------

def detect_weapon_boxes(img: np.ndarray) -> List[Dict]:
    boxes: List[Dict] = []

    for result in yolo_model(img, verbose=False, max_det=MAX_WEAPON_COUNT):
        for box in result.boxes:
            cls = int(box.cls[0])                       # numpy.int64 → int
            name = str(yolo_model.names[cls])           # osiguraj string
            confidence = float(box.conf[0])             # numpy.float32 → float

            if (
              name not in WEAPON_WATCHED_CLASSES
              or confidence < YOLO_CONFIDENCE_THRESHOLD
              ):
                continue

            if len(boxes) >= MAX_WEAPON_COUNT:
                return boxes

            left,  top, right, bottom = map(int, box.xyxy[0])

            if not is_valid_box(left, top, right, bottom):
                continue

            boxes.append(
                {
                    "top": int(top),
                    "right": int(right),
                    "bottom": int(bottom),
                    "left": int(left),
                    "label": name,
                    "confidence": confidence,
                    "weapon": True,
                }
            )

    return boxes
```

*Isječak koda 11. Funkcija za detekciju oružja*

Isječak koda 12 prikazuje prvi dio arhitekture prijenosa detektiranih / prepoznatih lica i oružja s poslužitelja na klijent i obratno. Na početku se definira *WebSocket* dekorator rutom */ws* s kojim klijent može otvoriti vezu sa poslužiteljem u stvarnom vremenu. Zatim se definiraju lokalni vremenski markeri za taktiranje samih pozivanih funkcija. Pokreće se beskonačna petlja koja svakim prolazom obrađuje jedan *frame* koji šalje klijent. Primljena slika dekodira se i priprema za obradu funkcijom *get_img*. Dohvaća se *event-loop* kako bi se delegirali *CPU* intenzivni zadaci u izdvojeni *ThreadPool*.

```
# ------------------------------------------------------------------------
# WebSocket endpoint  (ws://host:port/ws)
# • Prima Base64-encoded frameove s klijenta (script.js)       →  get_img()
# • Lansira dva paralelna zadatka:
#     1) prepoznavanje lica   (recognize_faces)  – u thread-poolu
#     2) detekciju oružja     (detect_weapon_boxes) – u thread-poolu,
#        ali rjeđe, prema YOLO_INTERVAL
# • Po završetku zadataka sprema globalne rezultate i šalje JSON klijentu.
# ------------------------------------------------------------------------
@app.websocket("/ws")
async def websocket_endpoint(websocket: WebSocket) -> None:
    await websocket.accept()

    # --- lokalni "state" po klijentu ---------------------------------------
    last_recognition_time: float = 0.0   # za kontrolu učestalosti face-recogna
    last_yolo_time:        float = 0.0   # za kontrolu učestalosti YOLO-a
    seen_people: set       = set()       # ID-jevi lica koja smo već prikazali
    seen_unknown_encodings: List[np.ndarray] = []  # encodinzi "unknown" lica

    global face_busy, yolo_busy, latest_face_boxes, latest_weapon_boxes

    try:
        while True:
            # ----------------------------------------------------------------
            # 1) Primi i dekodiraj sliku
            # ----------------------------------------------------------------
            data_uri = await websocket.receive_text()
            img = get_img(data_uri)
            if img is None:                          # loš frame → samo pauziraj
                await asyncio.sleep(0.01)
                continue

            loop = asyncio.get_running_loop()
```

*Isječak koda 12. WebSocket Endpoint*

U isječku koda 13 prikazuje se pokretanje funkcija za detekciju lica i oružja unutar *WebSocket*-a. U drugom koraku se provjerava zastavica *face_busy*, čime se doznaje ako je analiza lica već u tijeku. Zastavica *face_busy* sprječava preklapanje dva *CPU* skupih zadataka nad uzastopnim *frame*-ovima. Trenutni *frame* (hrv. slika) se kopira kako bi se original zaštitio od mogućih izmjena pri obradi. Poziva se funkcija *recognize_faces* te se prosljeđuju lokalni konteksti definirani u isječku koda 12. Delegira se zadatak *ThreadPoolExecutor*-u kako bi glavna petlja ostala

neblokirana. Završetkom dretve automatski se pozove *callback face_done*, čime globalni rezultati postaju dostupni sljedećem *WebSocket* odgovoru.

Funkcija za detekciju oružja poziva se vremenski definirano prema varijabli *yolo_interval* (0.2s). Detekcija se pokreće tek nakon proteka definiranog intervala i ako trenutno ne postoji drugi *YOLO* zadatak, čime se optimiziraju resursi procesora. Analogno detekciji i prepoznavanju lica, poziva se funkcija *detect_weapon_boxes* za detekciju oružja u pozadini. Završetkom dretve poziva se *callback yolo_done* čime se zastavica *yolo_busy* postavlja na vrijednost *false* za omogućavanje sljedećeg pozivanja. Rezultati se spremaju u *latest_weapon_boxes* za slanje klijentu.

Zastavice *face_busy* i *yolo_busy* ne utječu izravno na broj dretvi unutar *pool*-ova, već služe za regulaciju pokretanja novih instanci zadataka. Dakle, analiza video toka traje par desetinki sekunde. Da se svaka slika stavlja u *pool* stvorio bi se veliki red koji bi rezultirao kašnjenjem ali i zagrijavanjem hardvera. Zastavice omogućavaju čekanje dok se pojedini zadatak na slici ne obradi; detekcija i prepoznavanje lica ili detekcija oružja.

Unutar funkcije za prepoznavanje lica koristi se *enc_executor* s četiri dretve za četiri podzadatka računanja *embedding*-a lica istovremeno. Analogno tome ograničenje u funkciji za prepoznavanje oružja je *max_weapon_count* (4) što ograničava *YOLO* model da vrati eksplicitno ne više od 4 najpouzdanije detekcije oružja, kao što je prethodno objašnjeno.

```
# ---------------------------------------------------------------
# 2) Lansiraj zadatak prepoznavanja lica (ako nije već u tijeku)
# ---------------------------------------------------------------
if not face_busy:
    face_busy = True

    def face_task():
        # kopija slike jer ju prosljeđujemo u background-thread
        return recognize_faces(
            img.copy(),
            seen_people,
            seen_unknown_encodings,
            last_recognition_time
        )

    def face_done(fut):                  # callback na završetak
        nonlocal last_recognition_time
        global face_busy, latest_face_boxes

        face_busy = False
        boxes, caps, last = fut.result()
        last_recognition_time = last
        latest_face_boxes = boxes
        face_history.extend(caps)

    loop.run_in_executor(None, face_task).add_done_callback(face_done)

# ---------------------------------------------------------------
# 3) Lansiraj YOLO (detekciju oružja) rjeđe – prema YOLO_INTERVAL
# ---------------------------------------------------------------
now = time.time()
if (now - last_yolo_time > YOLO_INTERVAL) and not yolo_busy:
    yolo_busy = True
    last_yolo_time = now

    def yolo_task():
        return detect_weapon_boxes(img.copy())

    def yolo_done(fut):
        global yolo_busy, latest_weapon_boxes
        yolo_busy = False
        latest_weapon_boxes = fut.result()

    loop.run_in_executor(None, yolo_task).add_done_callback(yolo_done)
```

*Isječak koda 13. Pozivanje funkcija za detekciju lica i oružja iz WebSocket-a*

Isječak koda 14 prikazuje zadnji dio *WebSocket Endpoint*-a u kojem se trenutno raspoloživi podaci formatiraju u *JSON* format koji se asinkrono šalje klijentu. *Face_boxes* kolekcija rječnika sadrži koordinate i meta-podatke o svakoj osobi, klijent te podatke koristi za iscrtavanje okvira iznad videa. *Face_captures* sadržava povijest nedavnih prepoznavanja koja se iscrtavaju u karticama na desnoj strani *GUI*-a u klijentu. *Weapon_boxes* analogno prethodnome sadržava podatke *YOLO* detekcije oružja.

```
            # ------------------------------------------------------------
            # 4) Pošalji klijentu trenutno dostupne rezultate
            # ------------------------------------------------------------
            await websocket.send_json({
                "faces":        [f["name"] for f in latest_face_boxes],
                "face_boxes":   latest_face_boxes,
                "face_captures": list(face_history),
                "weapon_boxes": latest_weapon_boxes,
            })

            await asyncio.sleep(0.01)  # lagani "yield" da ne blokiramo petlju

    except WebSocketDisconnect:
        print("[INFO] Client disconnected")
```

*Isječak koda 14. Slanje podataka o prepoznatim licima i detektiranom oružjima klijentu*

# 6. Nadogradnja i optimizacije aplikacije

U ovom poglavlju izlažu se potencijalna nadogradnja i optimizacije kojima bi se postojeći sustav dodatno unaprijedio ili ubrzao. Naglasak je stavljen na proširenje mogućnosti aplikacije u vidu prepoznavanje nasilnog ponašanja te na inženjerske optimizacije koje bi ubrzale samu aplikaciju.

## 6.1. Nadogradnja

Jedna od mogućih nadogradnji aplikacije je detekcija nasilja iz pokreta ljudi. Detekcija bi se ostvarivala analizom vremenskih i prostornih uzoraka u video sekvencama. Za to se koriste modeli računalnog vida koji obrađuju niz uzastopnih *frame*-ova (slika) kako bi se identificirale agresivne ili abnormalne kretnje. Najčešće se primjenjuju konvolucijsko-rekurentne neuronske mreže (*CNN + LSTM*) ili *3D CNN* arhitekture koje istovremeno analiziraju prostornu strukturu i vremenski kontekst. Pokreti tijela se detektiraju putem metoda za procjenu položaja zglobova, npr. *OpenPose* [50] ili *MediaPipe* [51].

Dobiveni koordinatni vektori vremenski se prate i uspoređuju s unaprijed označenim obrascima nasilničkog ponašanja na kojima je model treniran. U realnom vremenu, detekcija se može sinkronizirati s postojećim sigurnosnim aspektima trenutne verzije aplikacije. Usporednom detekcijom oružja, nasilnog ponašanja i provjerom kriminalnog dosjea smanjila bi se stopa pogreške a potencijalno i omogućila predviđanja sigurnosnih rizika za društvo.

## 6.2. Optimizacije

U svrhu povećanja učinkovitosti aplikacije i smanjenja potrošnje računalnih resursa, moguće su ciljane nadogradnje u tri ključna segmenta: predobrada podataka, iskorištavanje hardverskih akceleracija te optimizacija prijenosa podataka između klijenta i poslužitelja. Svaka od implementiranih promjena temelji se na prethodno uočenim ograničenjima performansi i mogućnostima za poboljšanje aplikacije.

Prva značajna optimizacija moguća je u načinu rukovanja vektorima lica. U trenutnoj implementaciji, sve *encoding* vrijednosti poznatih lica računaju se pri svakom pokretanju aplikacije učitavanjem slika s diska i prolaskom kroz algoritam ekstrakcije značajki. Takav pristup rezultira nepotrebnim troškovima vremena i resursa *CPU*-a. To se može izbjeći ako se u bazi podataka unaprijed spremaju izračunati vektori *embedding*-a. Na taj način, prilikom inicijalizacije aplikacije, vektori se učitavaju iz baze u memoriju, čime se eliminira potreba za pozivom funkcije *face_encodings*. Na ovaj način su početna latencija i ukupno opterećenje sustava smanjeni, pogotovo u slučaju baze podataka koja sadrži veliki broj osoba.

Drugi aspekt optimizacije odnosi se na upotrebu grafičke procesne jedinice (*GPU*) za obradu podataka. Modeli poput *YOLO*, koji se temelje na dubokim konvolucijskim mrežama, zahtjevni su za izvođenje na centralnoj procesorskoj jedinici (*CPU*), osobito u stvarnom vremenu. U postojeću aplikaciju može se dodati podrška za izvršavanje *YOLO* modela na *GPU*-u, gdje bi se paralelne konvolucije značajno ubrzale. Za omogućavanje *GPU* izvršavanja aplikacije, *dlib* knjižnica mora biti ponovno izgrađena iz izvornog koda s uključenom podrškom za *CUDA*. Umjesto trenutnog *HOG* algoritma za detekciju lica, koristio bi se *dlib*-ov *CNN* model koji je optimiziran za *GPU*. Za *YOLO* model koristi se *PyTorch* s *CUDA* podrškom, koji automatski koristi *GPU* ako je dostupan. *YOLO* se može i eksplicitno postaviti da koristi *GPU* metode.

Treći segment optimizacije odnosi se na prijenos podataka između klijenta i poslužitelja putem *WebSocket* protokola. U trenutnoj implementaciji aplikacije, svaki *frame* dobiven s kamere odmah se pretvara u *Base64* format i šalje poslužitelju. Takav pristup za rezultat ima visoku mrežnu propusnost i nepotrebno opterećenje poslužitelja, budući da se većina *frame*-ova ne koristi zbog optimizacija za obradu detekcije lica i oružja. Promjenom logike na razini klijenta kojom se šalje svaki n-ti *frame*, mogu se dobiti znatne performanse. Treba se osigurati da se proces detekcije izvršava učestalije kako bi pravokutnik ažurno pratio lice, dok bi se proces prepoznavanja izvršavao svaki n-ti *frame* za provjeru ako je detektirana osoba i dalje ista.

# 7. Zaključak

Kroz rad je obrađena tema detekcije lica i oružja u stvarnom vremenu i demonstracija na izrađenoj aplikaciji. Rad pojašnjava teoriju iza računalnog vida umjetne inteligencije te opisuje najpoznatije modele za detekciju kao i korake prepoznavanja lica. Razložio se izbor tehnologija za izradu aplikacije kao i hardverski zahtjevi. Slikama su uprizorene funkcionalnosti aplikacije koje su opisane i kroz programski kod. Objašnjene su moguće optimizacije i nadogradnje aplikacije.

Glavni izazov razvoja aplikacije je izvršavanje na *CPU.* Kreativnim optimizacijama programskog koda dobila se responzivna aplikacija koja u stvarnom vremenu detektira lica i oružja s minimalnim zastajkivanjem. *Python* programski jezik pokazao se kao odličan odabir u radu zbog velikog broja podržanih knjižnica i opsežne dokumentacije. Iako skalabilnost nije bila tema ovog rada opisana aplikacija može se modularno nadograđivati u veći sustav ili korištenjem izrađenih *API*-ja integrirati kao dio postojećeg sustava.

Računalni vid izazovno je područje čija budućnost ima veliki potencijal. Brzina razvoja novih modela za prepoznavanje lica i oružja sigurno će povećavati točnost, dostupnost ali i same mogućnosti ovakvih aplikacija.

# 8. Literatura


[1] R. Johnson, The WebSockets Handbook: Seamless Communication for Web, Mobile, and IoT, HiTeX Press, 2025.

[2] S. Ramírez, »FastAPI,« tiangolo, 2018. [Mrežno]. Available: https://fastapi.tiangolo.com/. [Pokušaj pristupa 7 7 2025].

[3] G. v. Rossum, »Welcome to Python.org,« Python Software Foundation, 1980. [Mrežno]. Available: https://www.python.org/. [Pokušaj pristupa 7 7 2025].

[4] B. Jin, S. Saurabh i A. Shevat , Designing Web APIs: Building APIs That Developers Love, O'Reilly Media, 2018.

[5] J. Ingeno, Software Architect’s Handbook: Become a Successful Software Architect by Implementing Effective Architecture Concepts, Packt Publishing, 2018.

[6] T. Berners-Lee, »HTML,« Web Hypertext Application Technology Working Group, 1993. [Mrežno]. Available: https://html.com/. [Pokušaj pristupa 7 7 2025].

[7] K. J. Grant, CSS in Depth, Manning, 2018.

[8] B. Eich, »Javascript,« Oracle Corporation, 1995. [Mrežno]. Available: https://www.javascript.com/. [Pokušaj pristupa 7 7 2025].

[9] A. Geitgey, »face-recognition 1.3.0,« Adam Geitgey, 2017. [Mrežno]. Available: https://pypi.org/project/face-recognition/. [Pokušaj pristupa 7 7 2025].

[10] G. Jocher, »Ultralytics,« Ultralytics, 2024. [Mrežno]. Available: https://docs.ultralytics.com/models/yolo11/. [Pokušaj pristupa 7 7 2025].

[11] K. Suzuki i J. Machado, »AI-Based Computer Vision Techniques and Expert Systems,« 2023.

[12] A. Ivanov, »A COMPREHENSIVE REVIEW OF THE HISTORY AND METHODS OF COMPUTER VISION,« 2025.

[13] D. Bhatt, C. Patel, H. Talsania, J. Patel, R. Vaghela, S. Pandya, K. Modi i H. Ghayvat, »CNN Variants for Computer Vision: History, Architecture, Application, Challenges and Future Scope,« 2021.

[14] K. Sage i S. Young, »Security applications of computer vision,« 2002.

[15] I. Aydin i N. A. Othman, »A new IoT combined face detection of people by using computer vision for security application,« 2017.

[16] A. A. Abdulhussein, H. K. Kuba i A. N. A. Alanssari, »Computer Vision to Improve Security Surveillance through the Identification of Digital Patterns,« 2020.

[17] V. Bevia, S. Blanes, J.C. Cortés, N. Kopylov i R.J. Villanue, »A GPU-accelerated Lagrangian method for solving the Liouville equation in random differential equation systems,« 2024.

[18] T. Baji, »GPU: the biggest key processor for AI and parallel processing,« 2017.

[19] T. Marwala, »The Balancing Problem in the Governance of Artificial Intelligence,« 2024.

[20] R. Yu, »Application of CPU in AI and Machine Learning,« 2024.

[21] Y. Kortli, J. Maher , A. Al Falou i M. Atri , »Face Recognition Systems: A Survey,« 2020.

[22] T. H. Le, »Applying Artificial Neural Networks for Face Recognition,« 2011.

[23] B. Akinnuwesi, »Automated Students' Attendance Taking in Tertiary Institution using Facial Recognition Algorithm,« 2012.

[24] U. U. R. C. Tejashree Dhawle, »Face Detection and Recognition using OpenCV and Python,« 2020.

[25] I. Abimbola-Olulesi, »Hashnode,« Hashnode, 2023. [Mrežno]. Available: https://ifeolulesi.hashnode.dev/face-detection-with-python-deepface. [Pokušaj pristupa 7 7 2025].

[26] Afolabi I. Awodeyi, »Effective preprocessing techniques for improved facial recognition under variable conditions,« 2025.

[27] H. Cevikalp, »A comprehensive comparison of features and embedding methods for face recognition,« 2016.

[28] Y. C. H. W. a. Z. T. Hao Wu, »Face Recognition Based on Haar Like and Euclidean Distance,« 2021.

[29] L. Wang, Y. Zhang i J. Feng, »On the Euclidean distance of images,« pp. 1334 - 1339, 2005.

[30] O. Déniz, G. Bueno, J. Salido i F. De la Torre, »Face recognition using Histograms of Oriented Gradients,« 2009.

[31] S. P. S. A. Muhammed Jamshed, »Significant HOG-Histogram of Oriented Gradient Feature,« 2015.

[32] D.-M. C.-E. J.-A. R.-G. Juan Terven, »A Comprehensive Review of YOLO Architectures in Computer Vision: From YOLOv1 to YOLOv8 and YOLO-NAS,« 2023.

[33] J. L. a. H. S. Zhiming Xie, »A Face Recognition Method Based on CNN,« 2019.

[34] W. Z. Shuming Jiao, »High-resolution parallel phase-shifting digital holography using a low-resolution phase-shifting array device based on image inpainting,« pp. 482-485, 2017.

[35] D. E. King, »dlib,« Omnicom Group, 27 5 2025. [Mrežno]. Available: https://dlib.net/. [Pokušaj pristupa 8 7 2025].

[36] Shravan, »Medium,« Medium, 11 10 2023. [Mrežno]. Available: https://medium.com/@shravvv18/yolo-redefining-the-boundaries-of-computer-vision-405bea9dbb27. [Pokušaj pristupa 8 7 2025].

[37] S. Gollapudi, Learn Computer Vision Using OpenCV: With Deep Learning CNNs and RNNs, Apress, 2019.

[38] A. G. S. T. M. S. O. Mohil Maheshkumar Patel, »Trans-DF: A Transfer Learning-based end-to-end Deepfake Detector,« u *2020 IEEE 5th International Conference on*, Galgotias University, Greater Noida, UP, India, 2020.

[39] A. D. K. S. M. A. M. Muhammet Fatih Aslan, »CNN and HOG based comparison study for complete occlusion handling in human tracking,« *Measurement,* svez. 158, br. 107704, 2020.

[40] J. Ousterhout, »Python Documentation,« Python, 2025. [Mrežno]. Available: https://docs.python.org/3/library/tkinter.html. [Pokušaj pristupa 8 7 2025].

[41] R. Computing, »Py Qt,« Riverbank Computing, 1998. [Mrežno]. Available: https://doc.qt.io/qtforpython-6/. [Pokušaj pristupa 8 7 2025].

[42] T. K. organization, »Kivy Documentation,« The Kivy organization, 2012. [Mrežno]. Available: https://kivy.org/doc/stable/. [Pokušaj pristupa 8 7 2025].

[43] NVIDIA, »NVIDIA CUDA,« NVIDIA, 5 6 2025. [Mrežno]. Available: https://developer.nvidia.com/cuda-toolkit. [Pokušaj pristupa 8 7 2025].

[44] G. Bradsky, »OpenCV,« Intel, 2025. [Mrežno]. Available: https://opencv.org/. [Pokušaj pristupa 8 7 2025].

[45] C. Hattingh, »asyncio — Asynchronous I/O,« The Python community, 2025. [Mrežno]. Available: https://docs.python.org/3/library/asyncio.html. [Pokušaj pristupa 8 7 2025].

[46] P. G. D. Group, »PostgreSQL Documentation,« PostgreSQL Global Development Group, 8 5 2025. [Mrežno]. Available: https://www.postgresql.org/docs/. [Pokušaj pristupa 8 7 2025].

[47] TECHSLANG, »Techslang,« Techslang, 21 1 2024. [Mrežno]. Available: https://www.techslang.com/definition/what-is-dockerization/. [Pokušaj pristupa 8 7 2025].

[48] M. Masse, REST API Design Rulebook, O'Reilly Media, 2011.

[49] A. Team, »ASGI Documentation,« ASGI Team, 2018. [Mrežno]. Available: https://asgi.readthedocs.io/en/latest/. [Pokušaj pristupa 8 7 2025].

[50] O. Team, »OpenPose,« OpenPose Team, 2025. [Mrežno]. Available: https://cmu-perceptual-computing-lab.github.io/openpose/web/html/doc/index.html. [Pokušaj pristupa 8 7 2025].

[51] Google, »MediaPipe Solution Guide,« Google, 2025. [Mrežno]. Available: https://ai.google.dev/edge/mediapipe/solutions/guide. [Pokušaj pristupa 7 8 2025].

[52] »asyncio — Asynchronous I/O202,« [Mrežno]. Available: https://docs.python.org/3/library/asyncio.html.

# 9. Popis slika

# 10. Popis isječaka koda