

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike u Puli

Antonio Labinjan

COMPUTER VISION ATTENDANCE SYSTEM

Dokumentacija

Pula, 2024.

Sveučilište Jurja Dobrile u Puli

Fakultet Informatike u Puli

COMPUTER VISION ATTENDANCE SYSTEM

Dokumentacija

Antonio Labinjan

JMBAG: 0303106891 redovan student

Kolegij: **Web-aplikacije**

Mentor: **doc. dr. sc. Nikola Tanković**

Pula, prosinac 2024.

Tu ću hitit sadržaj kad ga buden napisa

1. Sažetak

Computer vision attendance system je aplikacija za evidentiranje prisutnosti pomoću prepoznavanja lica.

Ova aplikacija koristi computer vision za pojednostavljenje evidencije prisutnosti u obrazovnim institucijama. Studenti mogu bilježiti svoju prisutnost putem prepoznavanja lica u unaprijed definiranim intervalima, čime se uklanja potreba za ručnim ili papirnatim metodama. Profesori imaju intuitivno sučelje za upravljanje učenicima, predmetima i rasporedima te generiranje detaljnih izvještaja i statistika o prisutnosti kako bi pratili angažiranost učenika. Sustav osigurava točnost, smanjuje administrativni teret i pruža uvid u trendove prisutnosti. Razvijen s naglaskom na jednostavnost i učinkovitost, ovaj sustav integrira suvremenu tehnologiju kako bi unaprijedio iskustvo obrazovanja za studente i profesore.

Postoje 2 tipa korisnika; običan student koji unutar aplikacije može samo prijaviti prisutnost skeniranjem lica i profesor koji ima dodatne ovlasti koje će biti detaljnije razrađene u idućim poglavljima.

Omogućena je registracija profesora u sustav pomoću emaila i passworda. Nakon registracije, profesor može postaviti predmet koji se trenutno izvodi sa predviđenim datumom i vremenskim intervalom. Zatim, implementirane su funkcionalnosti dodavanja novih studenata u sustav pomoću ručnog uploada slika, ili pomoću izvlačenja frameova iz live snimke, ovisno o potrebi. Nadalje, omogućeno je automatizirano slanje obavijesti putem maila za svaku uspješnu prijavu prisustva korištenjem SendGrid API-ja.

Kod svake prijave prisustva također se provjerava se je li pred kamerom prava osoba ili slika pomoću jednostavne anti-spoofing mehanike. (postoje i neki napredniji API-ji za liveness check, ali su poprilično skupi).

Nakon postavljanja predmeta i dodavanja studenata, moguće je prijaviti prisutnost tako što aplikacija najprije provjeri koje je trenutno vrijeme i pripada li ono nekom od intervala za postavljene predmete. Zatim, kada student stane pred kameru, kreće složeni proces prepoznavanja koristeći OPENAI-jev CLIP model i Facebook-ov AI Similarity Search.

Što se tiče implementacije same aplikacije, korišten je Pythonov web-development framework Flask za backend, HTML za frontend te Sqlite za bazu podataka.

2. Uvod i motivacija

Ova aplikacija nastala je zbog želje za povezivanjem više različitih područja poput baza podataka, web-developmenta, računalnog vida i umjetne inteligencije u kompletan “proizvod” i zbog toga što me navedena područja izrazito zanimaju i definitivno ću se njima (bar se nadam) baviti u budućnosti. Odabran je Python kao jezik izrade aplikacije zato što se iz mog dosadašnjeg iskustva čini kao najbolji izbor za bilo kakve zadatke vezane uz AI. Također, posebno je motivirajuća činjenica da je ova aplikacija prilično inovativna. (Naravno, nije prva aplikacija koja implementira computer vision za prepoznavanje ljudi i evidenciju prisutnosti, ali je prva koja koristi kombinaciju CLIP-a i FAISS-a).

Nadalje, ono što je zanimljivo, su svakako prilično dobri rezultati koje ona postiže. Na iznimno dobrim referentnim slikama, postiže se točnost od oko 100%. Korištenjem nešto manje kvalitetnih slika, rezultati opadaju (~73%), no možemo tvrditi da je aplikacija pri primarnom use-caseu gotovo savršena. Ako bismo uzeli 30-ak ljudi (ekvivalent jednom školskom razredu) i par njihovih slika za feed-anje u model, aplikacija bi ih identificirala u 100% slučajeva.

Naravno, nije sve bilo savršeno iz prvog pokušaja i bilo je potrebno puno eksperimentiranja, istraživanja, pokušaja i pogrešaka, te u trenutku pisanja dokumentacije još uvijek tražimo idealnu kombinaciju parametara koja će dati maksimalne rezultate i za lošiji dataset slika.

2.1. SWOT analiza

Strengths:

- **Inovativna tehnologija:** Korištenje computer visiona i modela poput CLIP-a donosi modernu i naprednu tehnologiju u obrazovni sektor i ostale ustanove gdje je evidencija prisutnosti od ključne važnosti
- **Automatizacija procesa:** Eliminira ručno vođenje evidencije o prisutnosti, čime štedi vrijeme nastavnicima i administraciji
- **Preciznost:** Modeli mogu prepoznati učenike čak i u velikim grupama, uz minimalne greške
- **Jednostavna integracija:** Može se lako prilagoditi različitim školskim sustavima
- **Smanjenje varanja:** Prepoznavanje lica smanjuje mogućnost lažnog prijavljivanja prisutnosti

Weaknesses

- **Ovisnost o kvaliteti dataseta:** Loši podaci smanjuju preciznost modela, što može dovesti do misklasifikacija i frustrirati korisnike
- **Privatnost i sigurnost:** Mogući problemi s GDPR-om i zaštitom osobnih podataka učenika, usprkos činjenici da se podaci strogo čuvaju i teoretski ne bi trebali izlaziti izvan granica sustava
- **Infrastrukturni zahtjevi:** Potreba za kamerama visoke kvalitete i stabilnom mrežom u ustanovama što u nekim slučajevima može predstavljati velik problem
- **Ograničenja u nepovoljnim uvjetima:** Loše osvjetljenje, pokretni subjekti i slabe kamere mogu utjecati na točnost

- **Troškovi:** Ustanove s ograničenim budžetom mogu imati poteškoća s implementacijom sustava

Opportunities

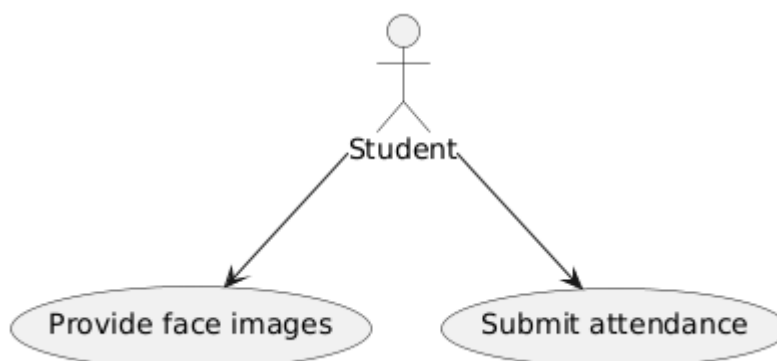
- **Širenje na druga tržišta:** Fakulteti, korporacije (praćenje zaposlenika), industrijske primjene (sigurnost i autorizacija)
- **Povećanje funkcionalnosti:** Integracija s dnevnicima, sustavima ocjenjivanja ili aplikacijama za roditelje
- **Prilagodba za druge namjene:** Npr. sigurnosne provjere, kontrola ulaza u objekte ili praćenje emocionalnog stanja učenika
- **Suradnja s vladinim institucijama:** Kao dio digitalizacije obrazovnog sustava
- **Razvoj freemium modela:** Osnovne funkcije besplatne, dok napredne zahtijevaju pretplatu

Threats

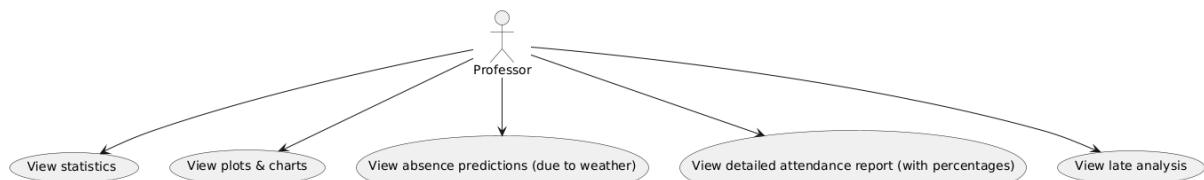
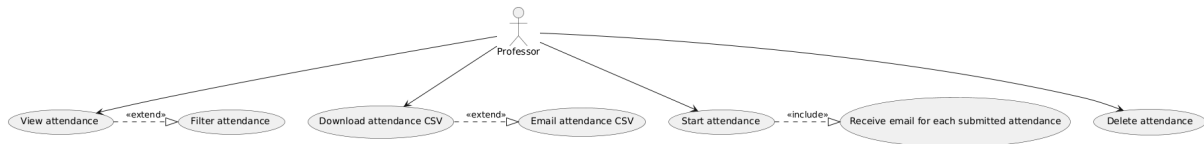
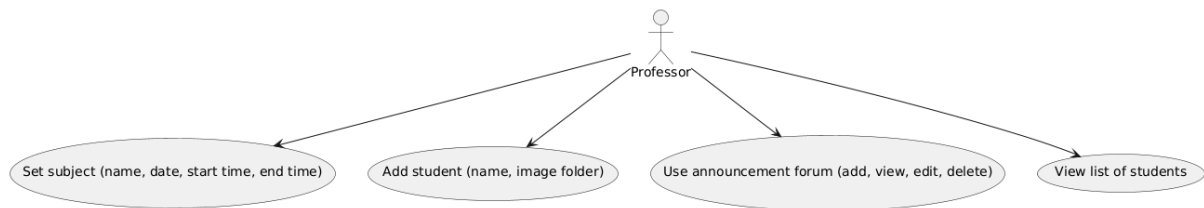
- **Regulacije i zakoni:** Stroge regulative o privatnosti podataka mogu ograničiti implementaciju
- **Ovisnost o tehnologiji:** Tehnički kvarovi, zastarijevanje modela ili sigurnosne rupe mogu imati loš utjecaj na prihvaćanje aplikacije
- **Otpor korisnika:** Strah ili skepticizam prema tehnologiji prepoznavanja lica, posebno kod roditelja
- **Troškovi održavanja:** Aplikacija zahtijeva stalna ažuriranja i održavanje kako bi bila konkurentna

3. Use Case dijagram sustava

Sustav ima 2 tipa korisnika: studenta koji nema praktički nikakve ovlasti unutar aplikacije osim prijavljivanja prisutnosti i profesora koji ima pune administratorske ovlasti, što je detaljnije razrađeno na sljedećim dijagramima (iako su i student i profesor dio istog sustava, razdvojeni su u 2 dijagrama radi preglednosti te je dodatno dijagram za profesora razdvojen u 3 dijela).



Use case za studenta - prilaže slike za evidenciju u sustav i prijavljuje prisutnost



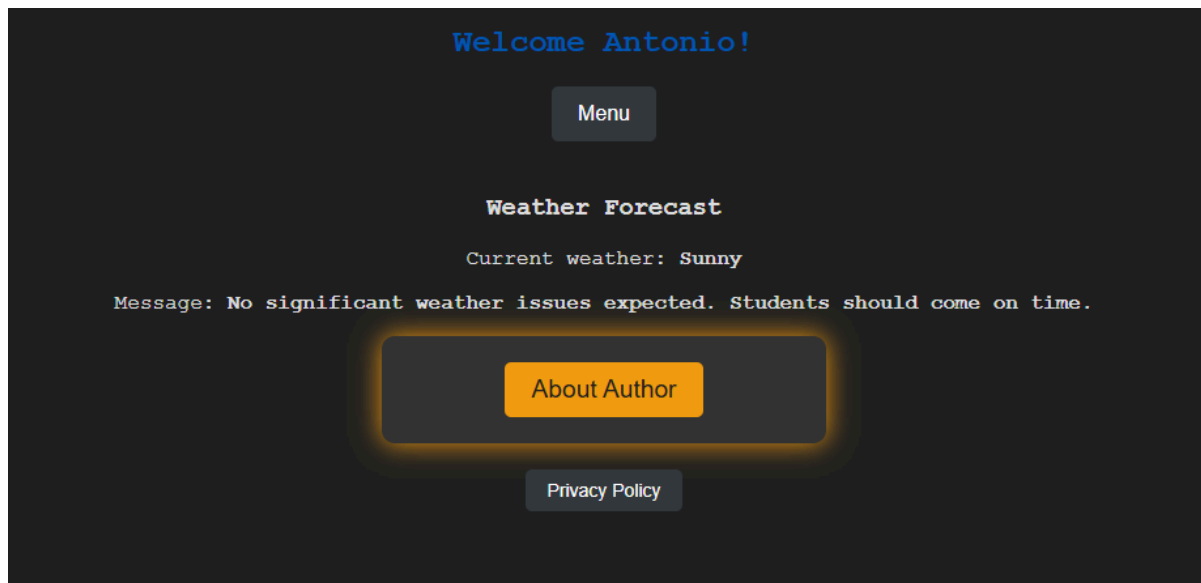
Use case za profesora - nakon prijave u sustav može postaviti trenutni predmet, njegov start time i end time, dodavati nove studente u sustav, uređivati objave na forumu, pregledavati popis svih studenata, pregledavati podatke o prisustvu s opcijom brisanja i filtriranja, preuzeti izvješće o prisustvima u csv formatu i dijeliti ga e-mailom, započeti interval bilježenja prisutnosti i zaprimiti email s potvrdom svake prijave i po potrebi obrisati neki od zapisa o prisutnosti. Može i pregledavati razne statističke analize, predikcije, izvještaje i grafove vezane uz podatke o prisutnostima.

BITNO=> NAKNADNO UBACIT CLASS DIJAGRAM

4. Razrada funkcionalnosti

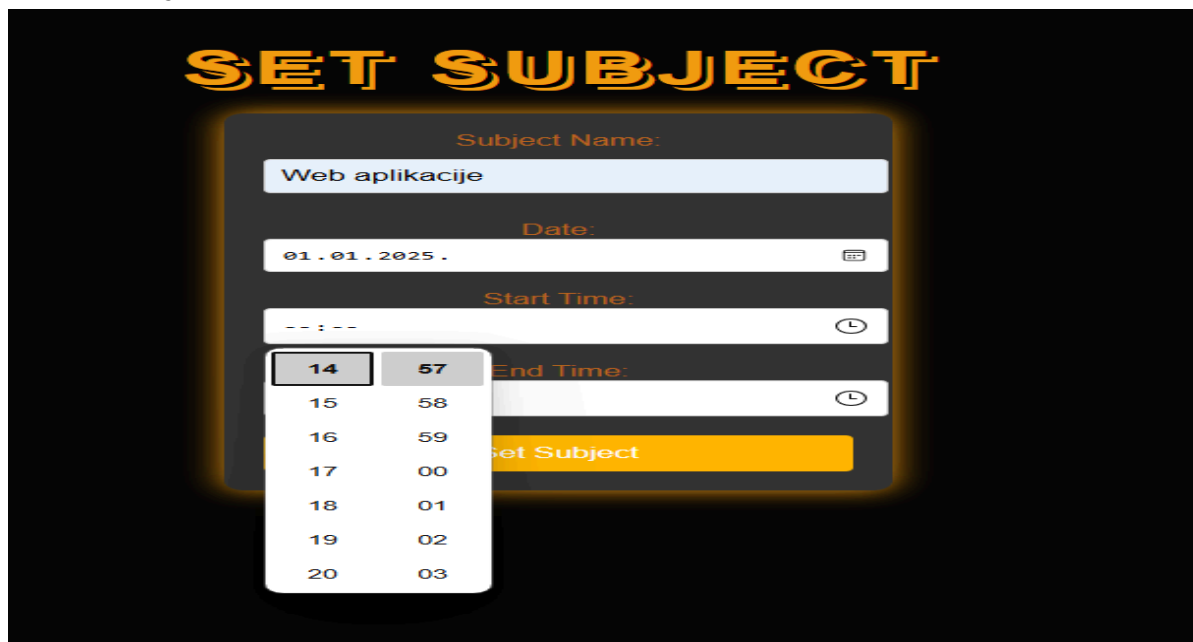
Aplikacija se sastoji od 2 dijela - frontenda napisanog u HTML-u i backenda napisanog u Flasku uz dodatak SQL baze podataka te CLIP modela i FAISS-a. U nastavku će biti detaljnije opisani svi dijelovi aplikacije uz napomenu da sposobnosti dizajna definitivno nisu nešto čime se mogu pohvaliti. Neki, jednostavniji i manje-više nebitni dijelovi bit će preskočeni (error redirectovi i slično).

4.1. Landing page



Nakon što se korisnik uspješno prijavi u sustav, dočeka ga landing page koji sadrži welcome poruku s dinamički dohvaćenim korisničkim imenom, tipkom za dropdown menu koja se aktivira pri hoveru miša, tipkom za rutiranje na about stranicu i tipkom za pregled privacy policy-ja koji objašnjava kako referentne slike korištene za prepoznavanje unutar aplikacije ni u kojem slučaju neće izaći izvan sustava. Također, prikazuje se i trenutna vremenska prognoza za Pulu dohvaćena putem vanjskog API-ja.

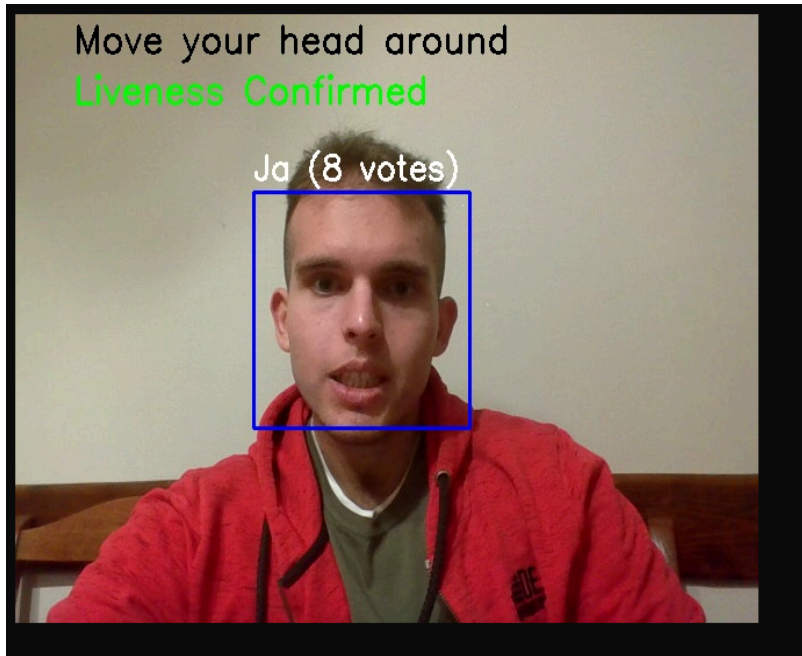
4.2. Set subject



Komponenta za postavljanje trenutnog predmeta je točka od koje kreće svaka evidencija prisutnosti. U tom dijelu aplikacije profesor definira za koji se predmet trenutno bilježi prisutnost. Nadalje, definira se datum evidentiranja prisutnosti koji se automatski postavlja

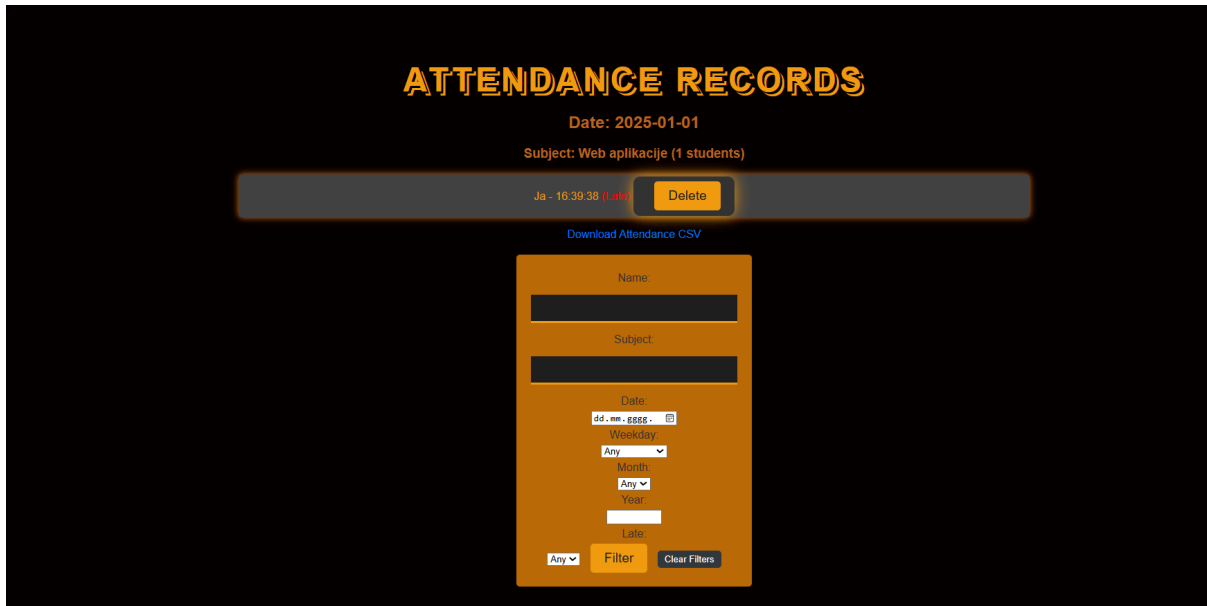
na današnji datum, no moguće ga je promijeniti ukoliko je to potrebno. Na kraju, još je potrebno definirati start time i end time za evidenciju kako bi se prijava prisutnosti ograničila na definirani vremenski interval.

4.3. Attendance logging



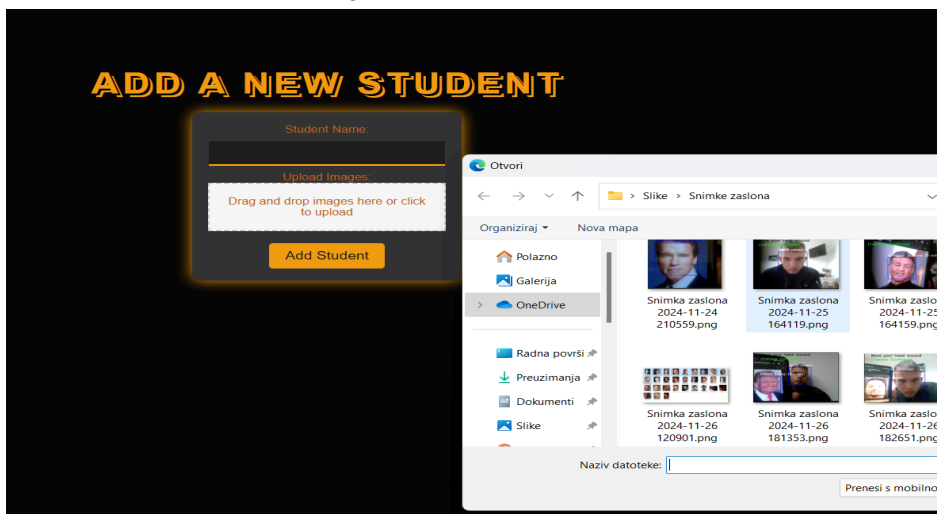
Najsloženiji dio aplikacije svakako se odnosi na modul za live prijavu prisustva. Pomoću cv2 biblioteke, otvara se kamera koja najprije traži lica u dohvaćenim frameovima koristeći haarcascade. Za svako prepoznato lice, kreira se bounding box. Nakon toga, provodi se jednostavna liveness detekcija koja provjerava kreće li se lice i registrira treptaje (kao što je već spomenuto, postoje bolja rješenja za liveness provjeru, ali naplaćuju se). Ukoliko lice prođe liveness check, kreće proces konverzije slike u embedding koristeći clip. Ono što se događa u pozadini je zapravo pretvorba featuresa slike u tensor. Nakon što se tensor dodatno normalizira, kreće usporedba dobivenog tensora sa tensorima svih poznatih lica koji su pohranjeni u faiss indexu. Faiss index omogućava indeksiranu pretragu na temelju sličnosti. Takva je potraga implementirana zato što su lica najprije bila pohranjivana u običan embedding space, no to se nije pokazalo funkcionalnim za veći broj poznatih lica jer je došlo do preklapanja embeddinga i misklasifikacije. Faiss je omogućio dohvat top-k rezultata, odnosno uzimanje input tensora sa kamere i dohvat k slika koje najviše sliče inputu. Nakon dohvata k slika, provodi se majority voting koji provjerava s kojom se klasom input najviše podudara. Radi dodatne optimizacije, koriste se k1 i k2 koji omogućuju da se ne pretražuju sva lica, već se na samom početku iz razmatranja izbace ona lica koja su previše različita. Također, postoji i parametar thresholda koji definira koliko % input mora sličiti nekom od lica kako bi se oni smatrali validnim parom.

4.4. Attendance overview



Nakon uspješne evidencije prisutnosti, u bazu podataka se pohranjuje zapis o prisutnosti koji se sastoji od trenutnog predmeta za koji bilježimo prisutnost, vremena evidentiranja, datuma, imena i prezimena studenta i late flaga. Late flag označava kasni li student ili je došao na vrijeme. Taj se dio evidentira provjeravajući koje je definirano početno vrijeme za postavljene predmet i koje je vrijeme evidentiranja prisutnosti. Ukoliko je prisutnost evidentirana 15 ili više minuta kasnije od početnog vremena, taj se zapis definira kao kašnjenje. (Postoji akademska četvrt). Ime i prezime se dohvaća preko naziva klase u koju je skenirano lice svrstano. Dodatno, postoji mogućnost brisanja zapisa i filtriranje po imenu studenta, predmetu, datumu, danu u tjednu, mjesecu, godini i late flagu. Na samom kraju, radi lakše obrade podataka omogućeno je preuzimanje zapisa o prisutnosti u csv formatu.

4.5. Add student manually



Moguće je dodati novog studenta u bazu poznatih lica ručno, koristeći slike iz lokalne pohrane. Potrebno je unijeti ime i odabrati slike iz računala ili ih drag&droppati na predviđeno mjesto. Minimalni unos je 1 slika, dok gornje granice nema. No, kako takav pristup nije naročito praktičan, omogućeno je i live dodavanje pomoću kamere.

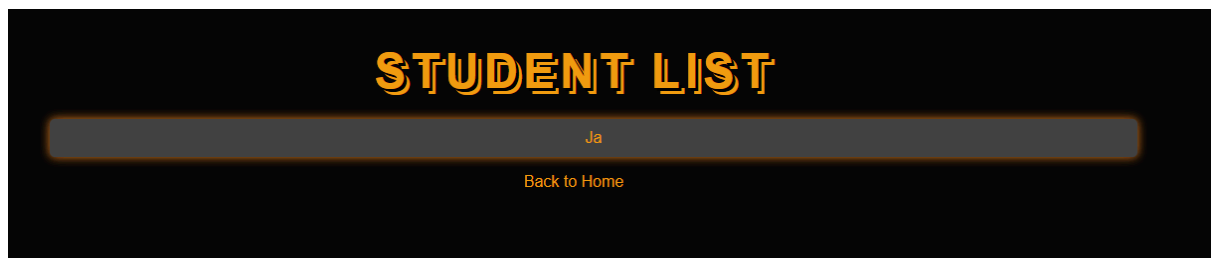
4.6. Add student using live feed

Add New Student via Live Feed

Student Name:

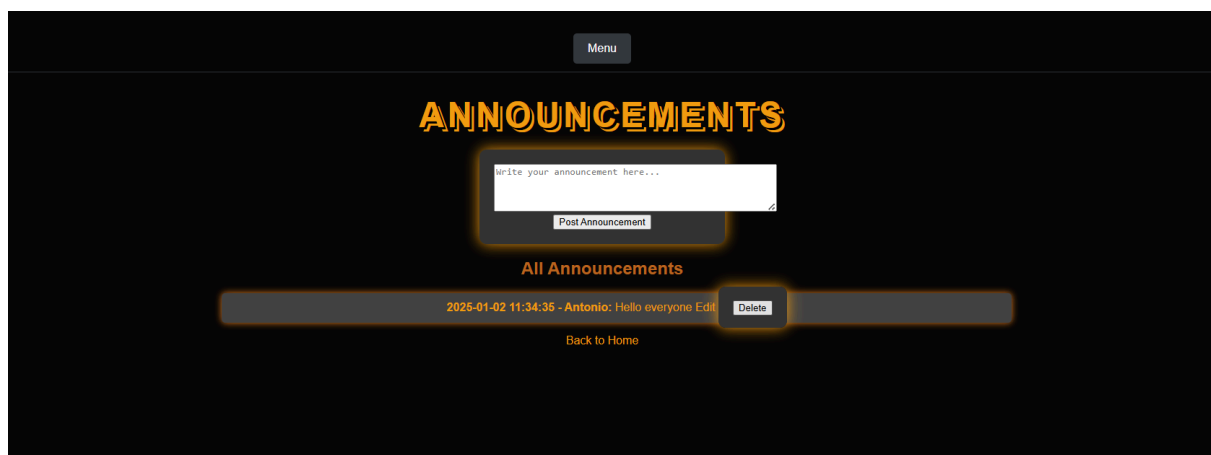
Dodavanje pomoću kamere funkcionira na prilično jednostavan način. Potrebno je unijeti ime i prezime studenta i samo pritisnuti na tipku. Tada student treba gledati u kameru nekoliko sekundi, kako bi se ulovilo 25 frameova i pohranilo ih se u novi folder s njegovim imenom. Ova mogućnost olakšava prikupljanje slika za proširivanje baze poznatih lica kako ne bi bilo potrebno za svakog studenta uploadati slike ručno i značajno ubrzava proces.

4.7. Students overview



Pri prvotnoj implementaciji spremanja prisutnosti došlo je do malih poteškoća u provjeri poznatih studenata, pa je tako nastala ova, inicijalno, debug ruta koja dohvaća sve studente iz svih zapisa o prisutnosti, neovisno o predmetu i datumu. Provodi se sql upit koji dohvaća sve unique studente koji su ikada spremljeni u bazu, odnosno evidentirani kao prisutni.

4.8. Announcement forum

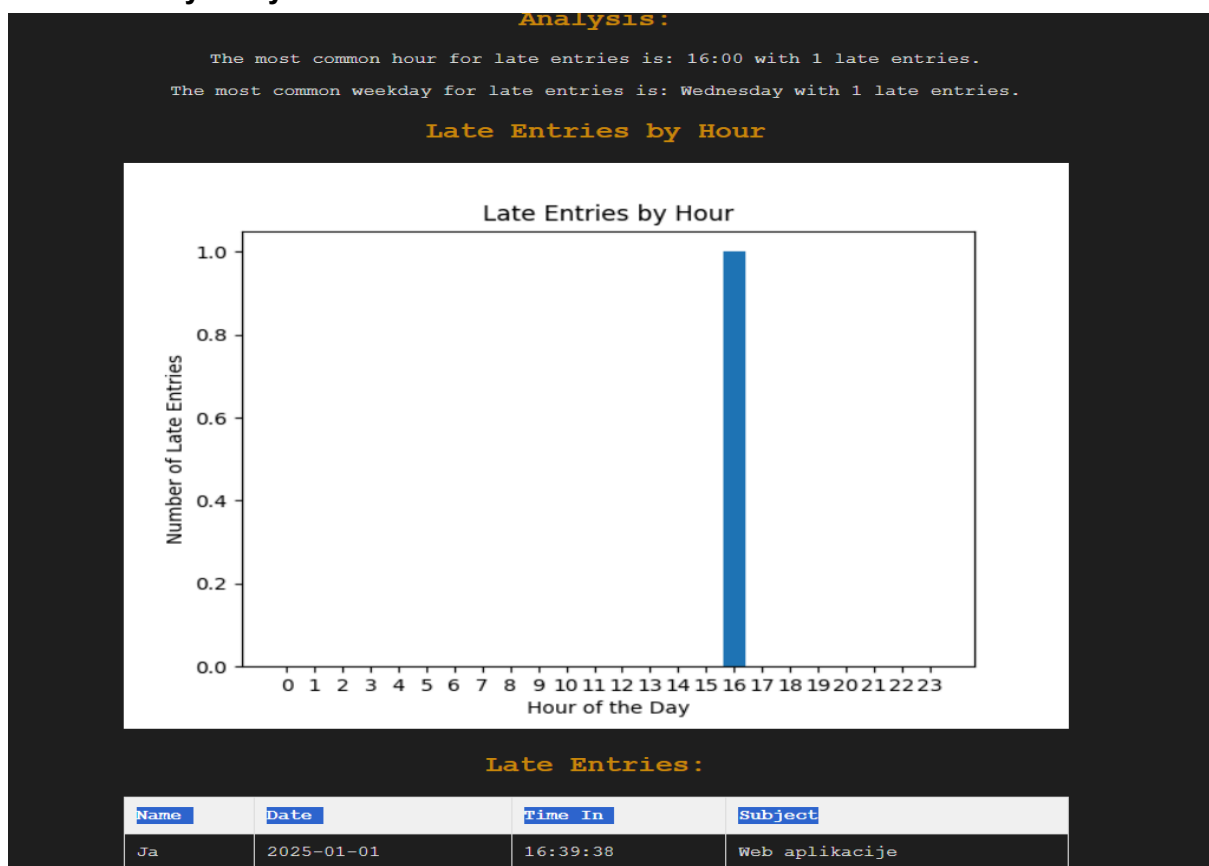


Announcement forum zamišljen je kao svojevrsni "interni chat" za korisnike aplikacije. Za svaku objavu vidljiv je korisnik koji ju je napisao te datum i vrijeme objave. Nakon objavljivanja, korisnik može obrisati i urediti svoju objavu. Također, implementiran je NLP model:

"badmatr11x/distilroberta-base-offensive-hateful-speech-text-multiclassification" koji pri pokušaju objavljivanja provjerava sadrži li objava neprimjerene riječi tako što najprije izvrši tokenizaciju objave, pretvori je u tensor i izračuna

“offensiveness-score”. Ukoliko je score veći od 30%, objava se flagira kao neprimjerena i onemogućuje se njezina objava uz prikladno upozorenje. Ista se provjera primjenjuje i pri editiranju objave.

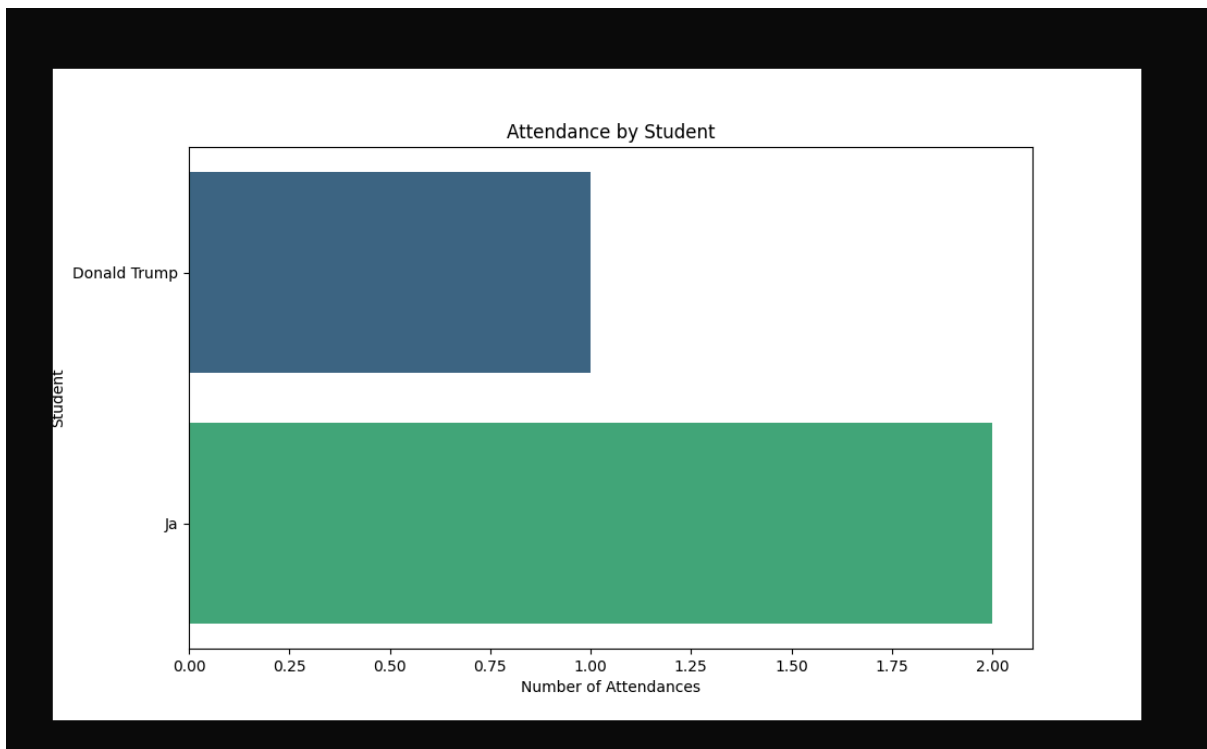
4.9. Late entry analysis



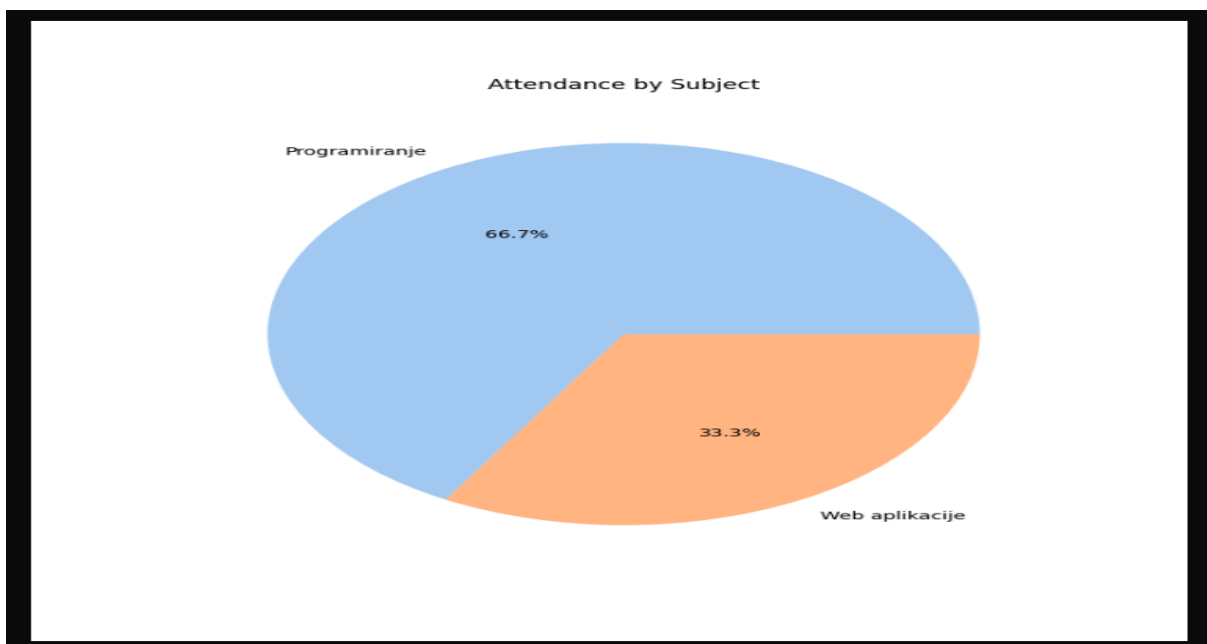
Za svako predavanje prati se vrijeme kada studenti dolaze i kasne li. Kašnjenje se evidentira ako je student prijavio prisutnost 15 ili više minuta nakon početka predavanja. Kašnjenja se dohvaćaju iz baze podataka pomoću late flaga, odnosno samo se select-aju sve zapisi gdje je late flag = True. Ti se podaci vizualiziraju pomoću grafikona koji pokazuje koliko kasnih ulazaka ima za pojedini sat u danu. Na taj se način može utvrditi koji sati u danu su posebno problematični i potencijalno se može optimizirati raspored na način da se minimiziraju kašnjenja.

4.10. Plot router & data visualization

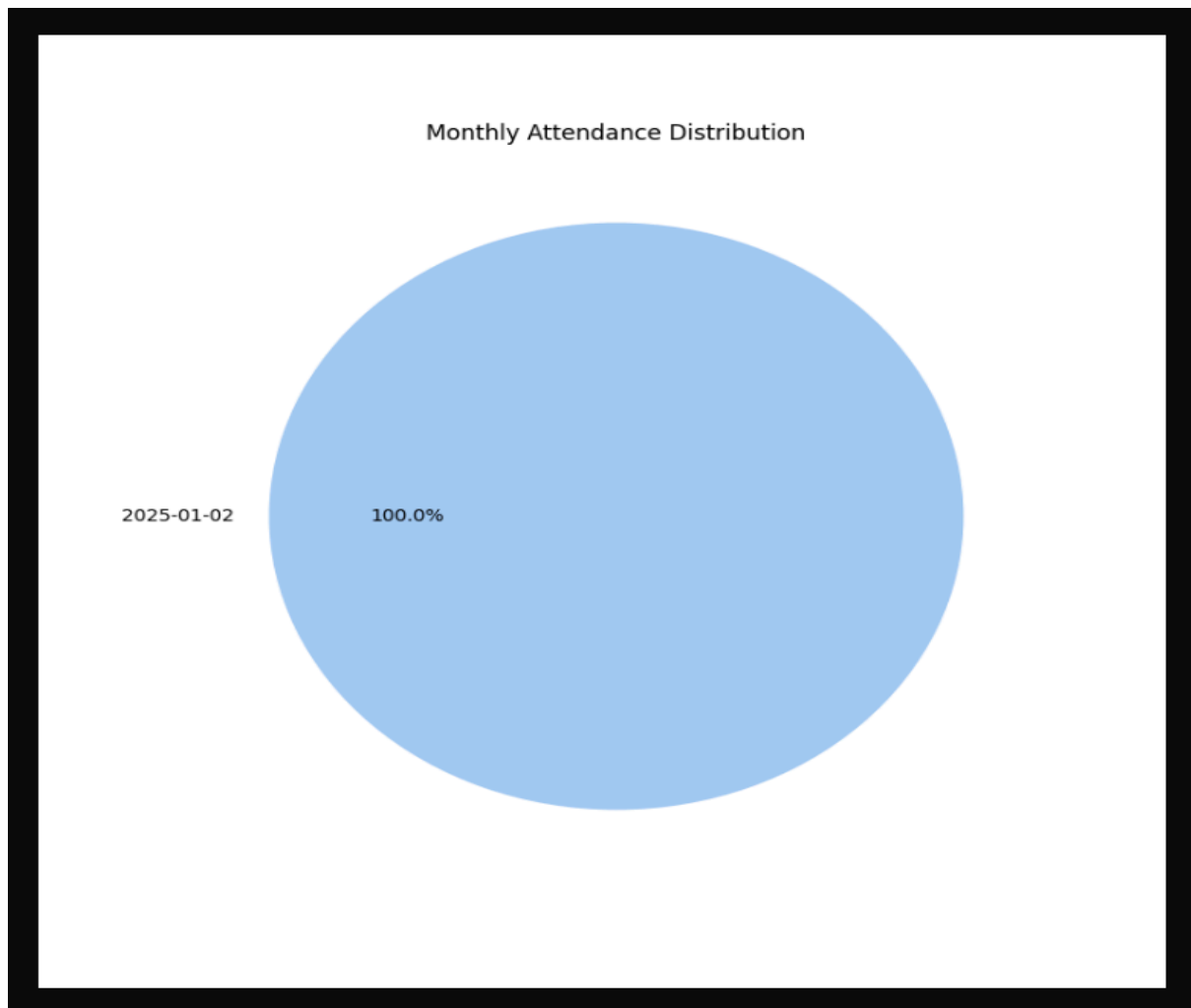
Definirana su 3 grafikona kojima možemo pristupiti pomoću posebnog rutera. Svaki od grafikona “puni se” određenim sql upitom koji dohvaća podatke koji nas zanimaju.



plot 1. - Prisutnost po studentu => vidljivo je na koliko je predavanja svaki pojedini student bio prisutan



plot 2. - Prisutnost po predmetu => vidljivo je koji je odnos prisutnosti među predmetima



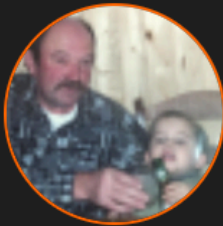
plot 3. - Mjesečna prisutnost => vidljivo je na koji datum je zabilježen koji udio prisutnosti, grupirano po mjesecu

4.11. Statistics

ATTENDANCE STATISTICS		
Student Attendance Counts		
Name	Subject	Attendance Count
Donald Trump	Matematika	1
Ja	Matematika	1
Ja	Web aplikacije	1
Subject Attendance Counts		
Subject	Attendance Count	
Matematika	2	
Web aplikacije	1	

Omogućena je i brza statistička analiza koja pomoću sql upita provjerava koliko je puta pojedini student prisustvovao nekom predmetu i koliko je ukupno unique studenata prisustvovalo pojedinom predmetu.

4.12. Profile



Antonio Labinjan

Bio: Student on FIPU Interested in: app development, databases, computer vision I love Python

Followers: 3

Repositories: 149

[Go to Home](#)

Profile, odnosno stranica o autoru (meni) dinamički dohvaća podatke s mojeg github accounta koristeći BeautifulSoup html web-parser za web scraping. Konkretni podaci koji su učitani s githuba su: profilna slika, username, bio, broj followera i broj **javnih** repozitorija na dan 2. siječnja 2025. Također, koristi se flaskova posebna funkcija render template string koja omogućava da izrenderiramo html template direktno iz Python koda bez da moramo pisati poseban template file.

4.13. Calendar

Non-Working Days 2024/2025

- 7.10. Početak nastavne godine u 1.11. Blagdan Svi sveti 18.12. Redovita sjednica Senata
- zimskom semestru i 18.11. Dan sjećanja na žrtve 25.12. Božić
- 24.10. Smotra Sveučilišta Vukovara i Škabrnje 23.-31. Zimski odmor
- 1.-3. Zimski odmor 3.2. Početak zimskog ispitnog 3.3. Početak nastavne godine
- 1.1. Nova Godina roka u ljetnom semestru
- 6.1. Sveta tri kralja 26.2. Redovita sjednica Senata 26.3. Redovita sjednica Senata
- 16.4. Dan Sveučilišta - nenastavni dan 1.5. Praznik rada 13.6. Završetak nastavne godine
- 20.4. Uskrs 28.5. Redovita sjednica Senata u ljetnom semestru
- 21.4. Uskršnji ponedjeljak 30.5. Dan državnosti 16.6. Početak ljetnog ispitnog roka
- 23.4. Redovita sjednica Senata 19.6. Tijelovo
- 11.7. Završetak ljetnog ispitnog 1.-29.8. Ljetni odmor 1.9. Početak jesenskog ispitnog roka
- roka 5.8. Dan domovinske zahvalnosti 24.9. Redovita sjednica Senata
- 16.7. Redovita sjednica Senata 15.8. Blagdan Velike Gospe 26.9. Završetak jesenskog ispitnog
- 14.-31.7. Ljetni odmor roka - Završetak akademske

Pomoću pdf scrapera dohvaćaju se neradni dani iz službenog unipu kalendara za akademsku godinu 2024/25 i prikazani su u html templateu koji se kreira pomoću već spomenute render template string funkcije.

4.14. Ostalo

Implementirane su još i komponente za autentifikaciju koristeći flaskove ugrađene funkcije iz biblioteke flask-login uz dodatnu validaciju passworda, success stranice za postavljanje

predmeta i dodavanje studenta i pop-up komponenta za notifikaciju nakon svake zabilježene prisutnosti. Kako se radi o prilično jednostavnim komponentama, nema smisla da ih opširno objašnjavam, pošto se nema što pretjerano objasniti.

5. Moduli aplikacije

Zbog specifičnog načina pisanja Flask aplikacija, teoretski je bilo moguće kompletnu aplikaciju (bar backend) napisati u jednom jedinom .py fajlu, što sam i radio na početku. No, kako je u jednom trenutku postalo nemoguće scrollati kroz više stotina linija koda i snalaziti se u njima, odvojio sam glavni app.py file na više modula radi lakšeg snalaženja i efikasnijeg pisanja koda.

Ti su moduli:

- imports.py
- model_loader.py
- load_post_check.py
- db_startup.py
- app.py
- također, postoji i cijeli niz helper datoteka koje se ne koriste u direktnom kontekstu aplikacije, već su kreirane s ciljem ubrzavanja određenih sub-taskova koje je bilo potrebno odraditi pri pisanju koda i testiranju

Imports.py

Imports.py je datoteka koja služi samo za importanje svih potrebnih biblioteka koje koristimo unutar aplikacije. Funkcionira na način da u njoj navedemo sve importove koji nam trebaju i onda u glavnom kodu samo pozovemo `from imports import *`.

U prijevodu, u glavni file uvozimo sve stvari navedene u imports.py.

Model_loader.py

Pošto se, teoretski, za isti use-case može koristiti više različitih AI modela, učitavanje modela je također odvojeno u poseban modul. Također, specifično je to što se model fine-tunira na trenutni dataset pri prvom pokretanju aplikacije i kreira nove weightove za taj dataset. Ukoliko bismo htjeli u nekom trenutku promijeniti model, samo bismo morali malo izmijeniti ovaj file. Konkretno, ovaj file učitava `openai/clip-vit-base-patch32`.

Load_post_check.py

Na sličan način funkcionira i ovaj modul. Učitava model za klasifikaciju teksta:

`badmatrl1x/distilroberta-base-offensive-hateful-speech-text-multiclassi
fication`.

Db_startup.py

I kreiranje baze podataka izdvojeno je u poseban modul. Koristi se sqlite3 baza.

```
conn = sqlite3.connect('attendance.db')  
c = conn.cursor()
```


Najprije kreiramo konekciju na bazu i onda pozivamo kursor za izvođenje upita te ga spremamo u varijablu koju možemo kasnije pozivati.

```
c.execute('''DROP TABLE IF EXISTS attendance''')
    c.execute('''CREATE TABLE IF NOT EXISTS attendance
                (name TEXT, date TEXT, time TEXT, subject TEXT, late
                BOOLEAN DEFAULT 0, UNIQUE(name, date, subject))''')

    c.execute('''DROP TABLE IF EXISTS announcements''')
    c.execute('''CREATE TABLE IF NOT EXISTS announcements
                (id INTEGER PRIMARY KEY AUTOINCREMENT,
                date_time TEXT,
                teacher_name TEXT,
                message TEXT)''')

    c.execute('''CREATE TABLE IF NOT EXISTS users
                (id INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT UNIQUE,
                password TEXT,
                email TEXT UNIQUE)''')

    c.execute('''
CREATE TABLE IF NOT EXISTS students (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    name TEXT NOT NULL
)
''')

    c.execute('''
CREATE TABLE IF NOT EXISTS embeddings (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    student_id INTEGER,
    embedding BLOB,
    FOREIGN KEY (student_id) REFERENCES students (id)
)
''')

    conn.commit()
    conn.close()
```

Nakon toga, koristeći kursor kreiramo tablice koje se koriste u aplikaciji. Attendance za spremanje zapisa o prisutnosti, announcements za objave, users za korisnike (profesore), students za studente i embeddings za spremanje embeddinga svakog studenta. Nakon toga, sve što je kursor odradio se commita i konekcija se zatvara kako u aplikaciji ne bi nastao kaos zbog slučajne interakcije s bazom podataka.

App.py

U app.py modulu odvija se glavni rad aplikacije. Najprije importamo sve što nam treba iz imports.py

```
from imports import *
```

Onda importamo funkcije iz ostalih datoteka.

```
from db_startup import create_db
create_db()

from model_loader import load_clip_model
from load_post_check import load_model_and_tokenizer
```

Učitavamo dotenv modul kako bismo dohvatili osjetljive podatke iz .env filea koje nije poželjno hardkodirati; poput secret-keya, api ključeva i sendgrid konfiguracije.

```
load_dotenv()
```

Inicijaliziramo flask aplikaciju i dohvaćamo secret key iz .env-a. Key ima nekoliko funkcija, ali najvažnija je svakako omogućavanje session managementa.

```
app = Flask(__name__)
app.secret_key = os.getenv('SECRET_KEY')
```

Konfiguriramo slanje e-mailova koristeći sendgrid. Također, radi sigurnosti, svi credentialsi se dohvaćaju iz .env. Dodatan razlog tome je i što sendgrid ne voli da se njegovi podaci commitaju na public github repozitorije i odmah suspendira account ukoliko slučajno commitamo api ključ ili nešto slično.

```
app.config['MAIL_SERVER'] = os.getenv('MAIL_SERVER')
app.config['MAIL_PORT'] = int(os.getenv('MAIL_PORT'))
app.config['MAIL_USE_TLS'] = os.getenv('MAIL_USE_TLS') == 'True' #
dohvaća bool varijablu iz .env i uspoređuje je s True. Ako je u .env
varijabla True, ovaj check će tornat True, i pičimo dalje
app.config['MAIL_USERNAME'] = os.getenv('MAIL_USERNAME')
app.config['MAIL_PASSWORD'] = os.getenv('MAIL_PASSWORD')

# SendGrid API Key
sendgrid_api_key = os.getenv('SENDGRID_API_KEY')
```

Konfiguriramo flask login paket kako bismo omogućili autentifikaciju korisnika. Koristimo LoginManagera i definiramo da će view, odnosno ruta na kojoj se login obavlja biti 'login'. Ukoliko bi se ruta zvala 'prijava', samo bismo ovdje morali redefinirati vrijednost login_view varijable.

```
login_manager = LoginManager()
login_manager.init_app(app)
login_manager.login_view = 'login'
```

Definiramo klasu User za kreiranje korisnika. Svaka instanca klase (a.k.a. svaki user) ima atribut id, username, password i email. Self dio klase služi kako bismo se referencirali na neki određeni objekt, a ne na klasu generalno. Koristeći self, Python zna da kreiramo novi objekt. Propertyji nisu bitni u ovoj aplikaciji, ali bez njih ne možemo instancirati korisnika. Postoje propertyji koji provjeravaju je li korisnik aktivan, autentificiran ili anoniman. Za naše potrebe, definirano je da je svaki korisnik po defaultu aktivan i autentificiran i nije anoniman. Također, postoji metoda koja vraća userov id kao string.

```
class User:
    # konstruktor
    def __init__(self, id, username, password, email):
        self.id = id
        self.username = username
        self.password = password
        self.email = email

    # Inače se moru dodat i složenije provjere, ali meni to ni toliko
    # bitno. Pretpostavljamo da je svaki user aktivan, autentificiran i da ni
    # anoniman (to su default properties čim se ulogira)
    # Flask-Login needs these properties to work correctly
    @property
    def is_active(self):
        # Return True if the user is active. You can modify this based
        # on user status.
        return True

    @property
    def is_authenticated(self):
        # Return True if the user is authenticated
        return True

    @property
    def is_anonymous(self):
        # Return False because this is not an anonymous user
        return False

    def get_id(self):
        # Return the user's ID as a string
        return str(self.id)
```

Zadnji dio autentifikacije na backendu je učitavanje korisnika preko id-ja. Korisnik unosi svoje credentials, provjerava se koji je id korisnika s tim credentialsima i ako je takav korisnik nađen, dozvolit će se ulazak u aplikaciju. Ako nema takvog korisnika, vratit će se None i onemogućiti ulazak u aplikaciju.

```
# prosljedimo id u funkciju, spajamo se na bazu, cursor izvršava query
na bazu tako da dohvaća onoga korisnika koji ima odgovarajući id
@login_manager.user_loader
def load_user(user_id):
    conn = sqlite3.connect('attendance.db')
    c = conn.cursor()
    c.execute("SELECT * FROM users WHERE id = ?", (user_id,))
    # kursor dohvaća prvega usera s odgovarajućim id-jem pomoću
fetchone (doduše, ne more imat 2 usera, isti id, ali čisto preventivno)
    user = c.fetchone()
    # zatvaramo konekciju da se sve ne zblesira
    conn.close()

    # ako smo našli korisnika s tim id-jem, dohvaćamo attribute
indeksirano po redu kako smo definirali u klasi
    if user:
        return User(id=user[0], username=user[1], password=user[2],
email=user[3])
    return None
```

Computer vision dio aplikacije počinje s učitavanjem poznatih lica u aplikaciju. Sama “baza” lica strukturirana je koristeći known_faces folder koji se sastoji od n subfoldera (n označava broj poznatih osoba u aplikaciji). U svakom subfolderu nalaze se slike na kojima je model, glupo rečeno, istreniran. Funkcija dodaje nova lica tako da prolazi kroz sve putanje do svih slika, provjerava što se tamo nalazi, učitava slike i prebacuje ih u RGB format. . Zatim se te slike prosljeđuju u clip processor koji ih pretvara u tensore koji će se kasnije koristiti kao input za detekciju/prepoznavanje unutar modela. Zatim se iz tih tensora izvlače featuresi, odnosno embeddingsi koji pomažu u “identifikaciji” pojedine klase (svaka klasa ima manje ili više unikatne embeddinge). Na samom kraju se ti embeddingsi pretvore u numpy array i normaliziraju koristeći linalg norm funkciju i dodaju u listu koja će se kasnije učitati.

```
# Function to add a known face.
# Dodaje sliku po sliku. Prosljeđuje putanju do slike i ime
klase(osobe)
def add_known_face(image_path, name):
    # učitavamo sliku iz putanje koju smo proslijedili
    image = cv2.imread(image_path)
    # ako je ni (None=> nema je; nismo je našli) dajemo value error
    if image is None:
        raise ValueError(f"Image at path {image_path} not found.")
```

```

    # ako je ima, konvertiramo je u drugi format da clip more delat s
njon
    image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    # processor uzima rgb slike i pretvara ih u tensore koji su input
za clip model
    inputs = processor(images=image_rgb, return_tensors="pt")
    with torch.no_grad():
        # uzimamo tensore i izvlačimo featurse iz njih tj. embeddinge
        outputs = model.get_image_features(**inputs)
    # pretvorimo u numpy array
    embedding = outputs.cpu().numpy().flatten()
    # normaliziranje za lakšu usporedbu
    known_face_encodings.append(embedding / np.linalg.norm(embedding))
    # appendamo u listu
    known_face_names.append(name)
    print(f"Added face for {name} from {image_path}")

```

Učitavamo poznata lica u aplikaciju tako što iteriramo kroz folder known_faces i sve njegove subfoldere. Definiramo o kojoj se klasi radi pomoću naziva subfoldera. U trenutnoj implementaciji postoje train i val subfolderi unutar svake klase, no to se može izbaciti i koristiti samo 1 set slika. Na kraju se samo ispisuje koliko se tensora i klasa ukupno učitalo.

```

# Load all known faces from the 'known_faces' directory
def load_known_faces():
    # Loop through each class (subfolder) in the 'known_faces' folder
    for student_name in os.listdir('known_faces'):
        # Path to the train subfolder for the current class =>
učitavamo samo train
        train_dir = os.path.join('known_faces', student_name, 'train')

        # Check if the train subfolder exists
        if os.path.isdir(train_dir):
            # Loop through all images in the train subfolder
            for filename in os.listdir(train_dir):
                image_path = os.path.join(train_dir, filename)
                try:
                    # Add the known face from the image
                    add_known_face(image_path, student_name)
                except ValueError as e:
                    print(e)

    # Debugging
    print(f"Loaded {len(known_face_encodings)} known face encodings")
    print(f"Known face names: {known_face_names}")

```

Unutar aplikacije se koristi i Facebook AI Similarity Search (FAISS). Služi za brzo pretraživanje i usporedbu embeddinga lica u bazi. Svaka klasa imao svoje embeddinge, koji predstavljaju jedinstvene karakteristike tog lica. Kada aplikacija prepozna lice, generira se embedding koji se zatim uspoređuje s postojećim embeddingsima u bazi podataka. FAISS omogućava brzo i efikasno pretraživanje tih embeddinga, čineći proces identifikacije lica bržim. Pogotovo u odnosu na prvotnu implementaciju gdje je korišteno linearno pretraživanje koje je postajalo sve sporije sa svakim novim dodanim licem. Osim toga, FAISS pomaže u grupiranju sličnih lica, što je korisno za organiziranje i analizu podataka. Uzimamo clip embeddinge, pretvorimo ih u array i spremamo to u FAISS indeks koji kasnije koristimo za pretragu.

```
# Build an index for Facebook AI Similarity Search (FAISS) using known
face encodings
def build_index(known_face_encodings):
    # Pretvaramo listu known_face_encodings u numpy array za rad s
    FAISS-om
    known_face_encodings = np.array(known_face_encodings)

    # Dobivamo dimenziju svakega embedding vektora (broj feturesa po
    vektoru)
    dimension = known_face_encodings.shape[1]

    # Kreiramo FAISS index koji koristi L2 (Euclidean distance) =>
    korijen od kvadrirane sume razlika podudarnih elemenata u 2 vektora
    faiss_index = faiss.IndexFlatL2(dimension)

    # Dodajemo sve poznate face encodings u FAISS index
    faiss_index.add(known_face_encodings)

    # Vraćamo izgrađeni FAISS index
    return faiss_index
```

Postavljamo basic setup za rad s kamerom koristeći biblioteku cv2. Nakon toga, koristimo haarcascade za osnovnu detekciju lica na kameri i crtanje bounding boxeva oko njih. Svako lice koje detektiramo na kameri, dobiva svoj bounding box.

```
# Initialize the webcam. Stavimo nulu za koristit defaultnu kameru (ako
smo toliko luksuzni da imamo više kamera, moremo staviti 1,2,3 itd. ;)
video_capture = cv2.VideoCapture(0)

# Face detection using Haar Cascade
# ovo ubacimo da detektira lica i crta bounding boxeve oko njih
face_cascade = cv2.CascadeClassifier(cv2.data.harcascades +
'haarcascade_frontalface_default.xml')
```

Svaki put kada započinjemo novo bilježenje prisutnosti, moramo definirati koji se predmet trenutno sluša. Trebaju nam atributi `current_subject`, `attendance_date`, `start_time` i `end_time`. Defaultna vrijednost im je `None` zato što varijabla treba biti inicijalizirana, a ne želimo postaviti neku smislenu vrijednost kako ne bi došlo do problema u kasnijem radu.

```
# Inicijaliziranje atributa za trenutnu prisutnost
current_subject = None
attendance_date = None
start_time = None
end_time = None
```

Implementirana je validacija passworda koja zahtjeva da ima bar 8 znakova, bar 1 malo i 1 veliko slovo, bar 1 broj i bar 1 poseban znak. Ukoliko neki od uvjeta nije ispunjen, funkcija će vratiti prikladni error message, a ako je sve OK, neće se vratiti ništa i password će biti odobren.

```
def validate_password(password):
    if len(password) < 8:
        return "Password must be at least 8 characters long."
    if not re.search(r"[A-Z]", password):
        return "Password must contain at least one uppercase letter."
    if not re.search(r"[a-z]", password):
        return "Password must contain at least one lowercase letter."
    if not re.search(r"\d", password):
        return "Password must contain at least one number."
    if not re.search(r"[!@#$%^&*(),.?\"':{}|<>/]", password):
        return "Password must contain at least one special character."
    return None
```

Ruta za signup koristi i GET i POST iz razloga što pomoću GET metode dohvaćamo formular za signup, dok pomoću POST-a šaljemo kreirane podatke o korisniku koji se dohvaćaju pomoću `request_form` (dohvat podataka koji su uneseni u html formular). Dodatno još provjerimo podudaraju li se prvi password i ponovljeni password i jesu li username i email unikatni. Ako je sve u redu, kreiramo novog korisnika.

```
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    # get je jer dohvaćamo formular, post je šaljemo zahtjev za signup
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        repeat_password = request.form['repeat_password']
        email = request.form['email']

        # standard validation
        if password != repeat_password:
            flash("Passwords do not match. Please try again.", "error")
```

```

        return redirect(url_for('signup'))

password_error = validate_password(password)
if password_error:
    flash(password_error, "error")
    return redirect(url_for('signup'))

try:
    conn = sqlite3.connect('attendance.db')
    c = conn.cursor()

    c.execute("SELECT * FROM users WHERE username = ? OR email
= ?", (username, email))
    if c.fetchone() is not None:
        flash("Username or email already taken. Please choose a
different one.", "error")
        conn.close()
        return redirect(url_for('signup'))

    # Hashiranje
    hashed_password = generate_password_hash(password,
method='pbkdf2:sha256')

    # Save to db
    c.execute("INSERT INTO users (username, password, email)
VALUES (?, ?, ?)",
              (username, hashed_password, email))
    conn.commit()
    conn.close()

    flash("Signup successful! Please log in.", "success")
    return redirect(url_for('login'))

except sqlite3.IntegrityError:
    flash("An error occurred during signup. Please try again.",
"error")

    return redirect(url_for('signup'))

return render_template('signup.html')

```


Iz istog razloga kao kod signupa i kod logina koriste se GET i POST metode. Pomoću GET dohvatimo template prijavu, a ako je metoda pristupa ruti POST, onda iz requesta dohvatimo poslani username i password. Spajamo se na bazu i instanciramo kursor koji pomoću SQL upita traži korisnika u bazi preko username-a. Ako nađe korisnika, onda prelazi na provjeru passworda. Ukoliko se hash passworda iz baze podudara sa hashem unesenog passworda (password je isti, pa za istu hash funkciju daje identičan hash), poziva se ugrađena funkcija login_user koja u trenutni session sprema usera koji je ulogiran.

```
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        conn = sqlite3.connect('attendance.db')
        c = conn.cursor()
        c.execute("SELECT * FROM users WHERE username = ?",
(username,))
        user = c.fetchone() # Fetches user row (dohvati 1 usera)
        conn.close()

        if user:
            # User is found
            user_id, db_username, db_password, db_email = user
            if check_password_hash(db_password, password):
                # Successful login
                login_user(User(id=user_id, username=db_username,
password=db_password, email=db_email))
                return redirect(url_for('index'))

            flash("Invalid username or password")
            return redirect(url_for('login'))

        return render_template('login.html')
```

Logout ruta je iznimno jednostavna, samo poziva logout_user ugrađenu funkciju i vraća nas na login rutu. (kad se korisnik izlogira, biti će preusmjeren na login formular)

```
@app.route('/logout')
@login_required
def logout():
    # samo call-a ugrađenu funkciju za logout
    logout_user()
    return redirect(url_for('login'))
```

Set subject služi za postavljanje trenutnog predmeta za kojeg se prijavljuje prisutnost. Ruta koristi login_required dekorator koji ograničava pristup ruti te dopušta da ju samo ulogirani korisnici vide. Definira se funkcija koja postavlja predmet tako što definira globalne varijable za predmet, datum, početak i kraj. Te varijable moraju biti globalne kako bi im bilo moguće pristupiti i izvan ove funkcije (koristimo predmet za prijavu prisutnosti u za to predviđenoj funkciji). Ako ruti pristupimo sa post zahtjevom, onda iz forma dohvaćamo podatke, postavljamo predmet i dinamički ispunimo set_subject_success template sa podacima koji su prosljeđeni kroz post zahtjev.

```
@app.route('/set_subject', methods=['GET', 'POST'])
@login_required # moramo bit ulogirani za pristupit ruti
def set_subject():
    # ove varijable su globalne jer želimo da ih se mora čitat/dohvaćat
    # i van funkcije (tribat će nan kad budemo logali attendance)
    global current_subject, attendance_date, start_time, end_time
    # ako šaljemo post request, spremamo u varijable ovo ča smo
    # prosljedili kroz req. form
    if request.method == 'POST':
        current_subject = request.form['subject']
        attendance_date = request.form['date']
        start_time = request.form['start_time']
        end_time = request.form['end_time']
        # renderiramo template s dodanima podacima
        return render_template('set_subject_success.html',
                               subject=current_subject,
                               date=attendance_date,
                               start_time=start_time,
                               end_time=end_time)
    return render_template('set_subject.html')
```

Pomoćna funkcija is_within_time_interval() služi kako bismo vidjeli koje je trenutno vrijeme i koji je trenutni datum. To dohvaćamo pomoću funkcije datetime.now() iz koje dodatno extractamo bitne djelove. Funkcija zatim provjeri podudaraju li se ti djelovi s odgovarajućim dijelovima iz onoga što smo definirali kroz set_subject. Vraća True ili False ovisno o podudaranju.

```
# provjera dali je trenutno vrijeme unutar intervala koji smo
# definirali za predavanje
def is_within_time_interval():
    # u now spremamo trenutni datum i vrime
    now = datetime.now()
    # iz now extractamo posebno datum i posebno vrime
    current_time = now.strftime("%H:%M")
    current_date = now.strftime("%Y-%m-%d")
    # vraća true ako smo unutar intervala, odnosno false ako nismo
```

```
return (current_date == attendance_date and
        start_time <= current_time <= end_time)
```

Add_student ruta služi za dodavanje novog poznatog studenta u aplikaciju. Ako šaljemo get zahtjev na rutu, samo se izrenderira template za dodavanje. Šaljemo li post zahtjev na rutu, onda kreće kreiranje novog studenta. Ima se dohvaća pomoću request_form (dohvati se ime koje smo unijeli u form field), a slike se dohvaćaju kao lista koja će se kasnije pretvoriti u embeddinge pomoću add_known_face. Taj se proces odradi za svaku sliku u listi i tako se doda novi poznati student. Minimum je dodavanje 1 slike, ali može ih biti i više. Ako se sve odradi i uspije, prikaže se success template.

```
# ruta za dodavanje studenta
@app.route('/add_student', methods=['GET', 'POST'])
# moramo bit ulogirani
@login_required
def add_student():
    # dohvaćamo ime iz requesta i izlistavamo sve slike koje smo
    poslali kroz request (1 je minimum, al more i više)
    if request.method == 'POST':
        name = request.form['name']
        images = request.files.getlist('images')

        # Novi subfolder za nove studente (4each student se dela
        folder)
        student_dir = os.path.join('known_faces', name)
        os.makedirs(student_dir, exist_ok=True)
        for image in images:
            # Dodajemo svaku sliku u studentov folder
            image_path = os.path.join(student_dir, image.filename)
            image.save(image_path)
            # pozivamo add_known_face za svaku sliku
            add_known_face(image_path, name)

        # dodajemo dinamički parametar name u template za success
        return render_template('add_student_success.html', name=name)

    return render_template('add_student.html')

# Route to confirm success => samo izrenderiramo stranicu ako uspije
@app.route('/add_student_success')
def add_student_success():
    return render_template('add_student_success.html')
```

Na vrlo sličan način, implementirana je i funkcija za live dodavanje gdje se ne zahtjeva manualni upload slika, već osoba samo treba gledati u kameru. Dohvaćaju se slike sa camera capture-a i pretvaraju se u embeddinge, normaliziraju i spremaju.

```
# Route to capture live feed images and add a new student
# Uglavnom, ovo je isti vrug kao ono static dodavanje, samo lovimo iz kamere

def add_known_face_from_frame(image_frame, name):
    # koristimo one iste encodinge i nameove od prije
    global known_face_encodings, known_face_names

    # Convert frame to RGB and process it
    image_rgb = cv2.cvtColor(image_frame, cv2.COLOR_BGR2RGB)
    inputs = processor(images=image_rgb, return_tensors="pt")

    # Generate the image feature embedding
    with torch.no_grad():
        outputs = model.get_image_features(**inputs)

    # Normalize the embedding
    embedding = outputs.cpu().numpy().flatten()
    normalized_embedding = embedding / np.linalg.norm(embedding)

    # If `known_face_encodings` is a list, append directly
    if isinstance(known_face_encodings, list):
        known_face_encodings.append(normalized_embedding)
    else:
        # If it's a numpy array, use numpy.vstack to add the new
embedding
        known_face_encodings = np.vstack([known_face_encodings,
normalized_embedding])

    # Append the name
    known_face_names.append(name)
    print(f"Added face for {name} from live capture")
```

Ruta za live dodavanje ponovno koristi get i post metode, ali u ovome slučaju se pri post zahtjevu otvara video capture. Definira se varijabla frame_count koja se inicijalizira na 0. Zatim se definira koliko frameova će biti dohvaćeno i započinje snimanje. Snimanje traje sve dok nije dohvaćeno i spremljeno dovoljno frameova ili ga korisnik prekine s q. Radi lakšeg snalaženja slike se nazivaju u formatu "name_timestamp_number".

```

@app.route('/add_student_live', methods=['GET', 'POST'])
def add_student_live():
    if request.method == 'POST':
        name = request.form['name']
        student_dir = os.path.join(dataset_path, name)
        os.makedirs(student_dir, exist_ok=True)

        # Start webcam capture
        cap = cv2.VideoCapture(0)
        frame_count = 0
        required_frames = 25 # Number of frames to capture => 25 po
klasi/osobi (more više, more manje)
        # otpri camera capture i lovi frameove sve dok ih više nema
(ugasimo kameru ili ima dovoljno frejmova); onda samo udri break
        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break

            # Display live video feed
            cv2.imshow("Capture Face - Press 'q' to Quit", frame)

            # Capture every frame => sve dok ih je manje od 25
(required)
            if frame_count < required_frames:
                # Save frame in student's directory
                timestamp = datetime.now().strftime("%Y%m%d_%H%M%S") #
=> dohvati datum i vrime za timestamp za naziv slike
                frame_path = os.path.join(student_dir,
f"{name}_{timestamp}_{frame_count}.jpg") # nazovi sliku
ime_timestamp_redni_broj_framea
                cv2.imwrite(frame_path, frame)

                # Process and save the embedding = > spremi tensor i
povećaj frame count za 1
                add_known_face_from_frame(frame, name)
                frame_count += 1

            # Killaj petlju ako ima dovoljno slika ili smo quitali s q
            if cv2.waitKey(1) & 0xFF == ord('q') or frame_count >=
required_frames:

```

```

        break

    # oslobodi kameru, zapri detection window
    cap.release()
    cv2.destroyAllWindows()

    # isto ko manualno dodavanje
    return redirect(url_for('add_student_success', name=name))

return render_template('add_student_live.html')

```

Za svaku uspješno prijavljenu prisutnost, šalje se email notifikacija koristeći sendgrid. Dohvaćaju se podaci o trenutnom predmetu koje smo postavili sa `set_subject`. Definira se sender email i proizvoljan broj receiver emailova. Kreira se mail sa definiranim sadržajem i šalje na odabrane adrese. Printa se response, kako bismo bili sigurni da sve radi dobro. API key dohvaća se iz `.env` datoteke.

```

def send_attendance_notification(name, date, time, subject): # => ovi
podaci nas zanimaju (postavili smo ih kroz set subject i ubacit ćemo ih
u mail)
    message = Mail(
        # izgled maila
        from_email='attendance.logged@gmail.com',
        to_emails='alabinjan6@gmail.com', # napraviti neki official mail
za ovaj app da ne koristim svoj mail
        subject=f'Attendance Logged for {name}',
        plain_text_content=f'Attendance for {name} in {subject} on
{date} at {time} was successfully logged.'
    )

    try:
        print("Attempting to send email...")
        #sendgrid_api_key = os.getenv('SENDGRID_API_KEY')
        sg = SendGridAPIClient(os.getenv('SENDGRID_API_KEY'))
        response = sg.send(message)
        # debug ako sendgrid ne dela
        print(f"Email sent: {response.status_code}")
        print(f"Response body: {response.body}")
    except Exception as e:
        print(f"Error sending email: {str(e)}")

```

REDAK 517