



UNIVERSITATEA DE VEST DIN TIMIȘOARA  
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ  
PROGRAMUL DE STUDII DE LICENȚĂ  
Informatică

# LUCRARE DE LICENȚĂ

**COORDONATOR:**

Lect. Dr. Popovici Adriana

**ABSOLVENT:**

Savu Rares Mihail

TIMIȘOARA

2023

UNIVERSITATEA DE VEST DIN TIMIȘOARA  
FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ  
PROGRAMUL DE STUDII DE LICENȚĂ  
Informatică

# POLICE PROFILING

**COORDONATOR:**

Lect. Dr. Popovici Adriana

**ABSOLVENT:**

Savu Rares Mihail

TIMIȘOARA

2023

# Cuprins

Abstract . . . . .	4
<b>1 Prezentarea problematicii abordate</b>	<b>5</b>
1.1 Abordările existente . . . . .	5
1.2 Facilitățile aplicației . . . . .	6
1.3 Arhitectura aplicației . . . . .	6
1.4 Modelul bazei de date . . . . .	8
1.5 Cazurile de utilizare . . . . .	8
1.5.1 Cazul de utilizare: Căutarea unui profil . . . . .	10
1.5.2 Cazul de utilizare: Căutarea într-un video . . . . .	11
<b>2 Prezentarea contribuției autorului</b>	<b>12</b>
2.1 Tehnologiile utilizate . . . . .	12
2.2 Detalii de implementare . . . . .	13
2.2.1 Baza de date . . . . .	13
2.2.2 Clasa Person . . . . .	14
2.2.3 Serverul în Flask . . . . .	15
2.2.4 Interfața grafică . . . . .	17
2.2.5 Recunoașterea facială . . . . .	23
2.3 Manual de utilizare . . . . .	27
2.3.1 Crearea și logarea unui cont . . . . .	27
2.3.2 Adăugarea unei persoane . . . . .	27
2.3.3 Căutarea fotografică și video . . . . .	28
2.4 Manual de implementare . . . . .	29
<b>3 Concluzii și direcții viitoare</b>	<b>30</b>
3.1 Lărgirea bazei de date . . . . .	30
3.2 Stocarea imaginilor . . . . .	30
3.3 Recunoașterea video . . . . .	31
3.4 Cautare în lista de persoane . . . . .	31
<b>Bibliografie</b>	<b>32</b>

## Abstract

The domain of computer vision and image processing is vast and complex, with lots of practical applications in our day to day lives. One application which would stand out as useful within Romania's police forces is a software with the capability of detecting faces in video footage, whether they be known or unknown compared to a database, while also having options for more advanced types of detection such as moving entities detection. The problem with this lies in storing profiles of people who are considered malicious or dangerous in a centralized way, therefore being able to input the image of someone's face into the software which is then designed to compare it in an efficient and accurate way to previously stored profiles. This type of recognition is designed to help in situations in which a perpetrator's face is obtained via a detailed sketch or a photo taken by an eye witness. Nevertheless, having only a picture to associate with a real person says little about the individual, making the storage of classic identification besides the picture a mandatory feature alongside others such as name, birth date, height, weight, hair color, eye color, address and place of birth. Further identification can be written manually in a comment section where one would include additional imagery such as clothing or birth defects. An equally important feature is that of video recognition in which the same process occurs live on a video file, using similar algorithms such as the image to image detection process mentioned before, however this time on multiple objects at once, and filtering out unnecessary contents that are not a human face. This proves itself useful when analyzing footage of crimes recorded by bystanders. Highlighting moving objects such as cars in certain scenes can also prove to be valuable in the context of speeding considering that once a car's shape is detected, obtaining the license plate or model of car would be certainly useful in finding the owner. Since most crimes or security related events happen to be witnessed and recorded, this functionality would find itself beneficial on security cameras, highways and as a standalone software for police to use when having video material as a starting point. Using background extraction as a method of separating moving entities from stationary ones can easily highlight people in a moving motion, which is most common for police related incidents as people standing still rarely commit a crime. Image processing also includes the idea of having an image which is sometimes blurred or shaky and stabilising it in order for it to be processed later on whether it be as a static comparison to another picture or for it to be searched within a video. One such way to achieve a clear image would be the Eigenfaces method which will be analyzed throughout this paper. Similar solutions to this problem exist such as a missing persons listing on the Romanian police website, phone's facial recognition software and fingerprint locks and perhaps confidential software used by the national security agency, however time and time again these have proved to be lacking in use and efficiency as witnessed in the public domain across news media.

# Capitolul 1

## Prezentarea problematicii abordate

### Introducere

Lucrarea își propune să ofere o soluție software pentru organele de investigații de stat sau private, dar și pentru entitățile private care doresc să recunoască persoane în mod automat, sau atunci când ochiul și memoria umană nu pot face asta. Recunoașterea facială ce urmează să fie elaborată este compusă în parte din imagini respectiv din materiale video.

Scopul este construirea unei baze de date cu cât mai multe persoane, spre exemplu toată populația unei țări, unde fiecărei persoane îi este utilizată poza de buletin și rularea recunoașterii faciale pentru a recunoaște persoanele care comit infracțiuni sau ilegalități în mod automat, având la bază doar profilul și poza persoanei.

Lucrarea este structurată prezentând modul în care funcționează arhitectura aplicației, cum a fost gândită aceasta și de ce, ce facilități oferă și cum au fost acestea implementate pentru a realiza obiectul propus, menționat anterior.

Am ales această temă din pasiune pentru domeniul de computer vision și dorința de a putea realiza ceva cu scop pur practic, pentru a ajuta organele de investigații să își realizeze obiectivele eficient și cât mai simplu, eliminând complet eroarea umană sau inabilitatea de a observa suspectul dintr-un material photo sau video.

### 1.1 Abordările existente

Abordările existente ale lucrării mele de licență se pot împărți în 2 categorii: publice și comerciale.

Prin aplicațiile din domeniul public înțelegem aplicațiile de recunoaștere facială prezente pe telefoane, oferind utilizatorului mai multă securitate din punct de vedere al accesului la dispozitiv, aplicațiile destinate publicului și sunt descărcabile gratis sau contra cost, care se ocupă cu același task însă fie în scop recreațional, fie în scop de investigații. Prin aplicațiile de uz comercial înțelegem aplicațiile dezvoltate în corporații cu scopul de a fi vândute mai departe sau utilizate intern. Câteva exemple relevante sunt cele din zona automotive unde recunoașterea facială are ca scop identificarea șoferului sau a persoanelor din vehicul cât și detectarea stării șoferului în funcție de expresia facială, a numerelor de înmatriculare, obiectelor care se mișcă cum ar fi pietonii sau condusul automat.

## 1.2 Facilitățile aplicației

Aplicația oferă utilizatorului mai multe opțiuni de analiză și identificare a imaginilor. După ce acesta și-a creat un cont, aplicația oferă 3 opțiuni: crearea de profile, căutarea unei poze printre acele profile și căutarea într-un material video al unei poze.

Prima opțiune, cea de creare de profile este reprezentată de simpla adăugare într-o bază de date a unui nume, prenume, CNP, și a unor date simple de identificare cum ar fi vârstă, sex, înălțime și greutate, urmate de o poză a persoanei pe care utilizatorul dorește să o adauge în aplicație. Acest pas este primul care trebuie urmat, întrucât celelalte funcționalități nu pot funcționa fără cel puțin o persoană în baza de date, fiind recomandate mai multe persoane.

A doua opțiune este cea de căutare a unei persoane prin încărcarea unei poze. Această opțiune are ca scop identificarea pozei din baza de date existentă, creată de utilizator prin opțiunea menționată anterior. Astfel se găsește poza care seamănă cel mai mult cu persoana încărcată la această opțiune, pe care utilizatorul dorește în mod ideal să afle cine este întrucât în baza de date fiecărei poze îi este atribuită o persoană.

A treia opțiune este cea de căutare în materiale video. Utilizatorul va trebui să încarce un material video, și având în prealabil o bază de date creată prin prima opțiune menționată, acesta va putea să observe cum aplicația încadrează persoanele din video într-un patrat și dacă persoanele din video sunt sau seamănă într-un procent mare cu cele din video, lângă patrat va fi menționat numele persoanei.[4] [9] [10]

## 1.3 Arhitectura aplicației

Aplicația este împărțită în două programe, primul reprezintă un flaskAPI care acționează drept server. Acesta primește requesturi de get și put pentru a actualiza o bază de date cu utilizatori care au voie să se logheze cu un cont în aplicația principală. Requesturile de get verifică dacă o logare transmisă din aplicația principală, mai precis numele de utilizator și parola, se regăsesc în tabel. Requestul de put are loc atunci când un utilizator decide să își creeze un cont, datele de logare create fiind adăugate în baza de date. [7]

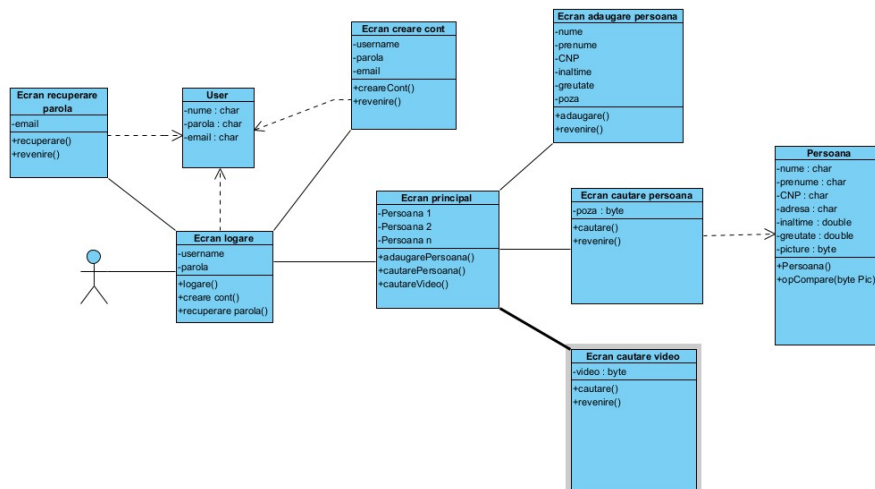


Figura 1.1: Diagrama de interfață

Dacă serverul acesta nu este pornit, aplicația principală nu va putea funcționa mai departe de ecranul de start, unde se introduc datele de logare. Motivul împărțirii aplicației în două părți constă în securitate, întrucât baza de date nu este stocată local pentru fiecare utilizator, astfel în cadrul unui departament de securitate, angajații vor primi doar partea principală a aplicației, serverul fiind păstrat în siguranță altundeva de către conducere. Aplicația serverului conține deasemenea o alta bază de date care se ocupă cu stocarea de persoane: datele lor de identificare și o poză per persoană. Aplicația de client, care conține interfața grafică cu pagina de logare, oferă posibilitate de logare, crearea unui cont și o opțiune pentru cazul în care cineva și-a uitat parola.

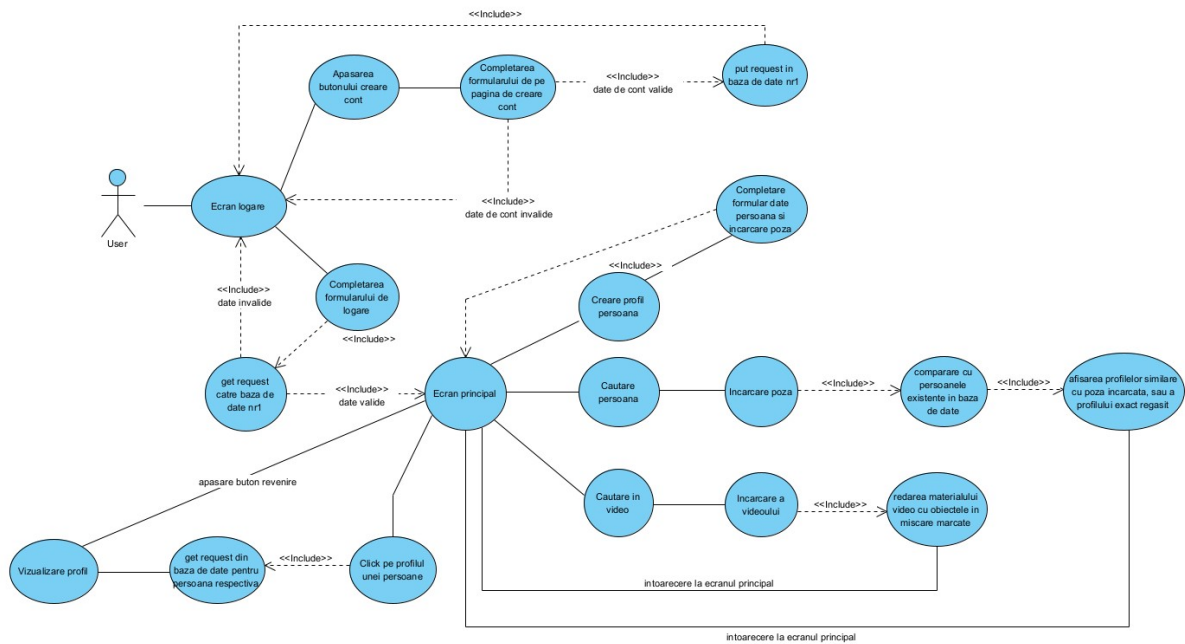


Figura 1.2: Diagrama sistemului

În cazul unei logări valide utilizatorul va fi trimis pe pagina principală a aplicației unde va putea avea acces la toate profilele stocate de către toți utilizatorii și opțiunile de a adăuga o persoană în baza de date împreună cu o poză, de a căuta o persoană oferind o poză prin baza de date și de a căuta într-un material video persoane deja existente.

## 1.4 Modelul bazei de date

Baza de date este realizată în MySQL și este împărțită în 2 tabele - unul care asigură logarea utilizatorilor în siguranță stocând usernameul, parola și emailul.

Al doilea tabel se ocupă cu stocarea profilelor persoanelor pentru a fi căutate sau analizate ulterior de către aplicație, acestea având nume, prenume, dată de naștere, CNP, înălțime și greutate, o secțiune pentru comentarii și o poză care va fi cea căutată ulterior sau analizată. Baza de date servește prin cereri de tip get și post utilizând flask către aplicația client, fie verificări în baza de date în cazul cererilor get, fie adăugări în baza de date în cazul cererilor post, pentru primul tabel.



id	username	password	email
1	rares	1234	rares.savu99@e-uvt.ro

Figura 1.3: Tabelul de logare cu un utilizator

Al doilea tabel servește către a stoca atât datele persoanelor cât și pozele acestora, care vor fi utilizate de către aplicația client în funcție de opțiunile alese, pentru a fi comparate cu o altă poză introdusă de utilizator sau vor fi verificate în cadrul unui material video.

## 1.5 Cazurile de utilizare

Un prim caz de utilizare este reprezentat de crearea unui cont în aplicație. Utilizatorul va porni aplicația client și va da click pe butonul create account. Acest buton îl va duce către o pagină nouă, unde va trebui să completeze câmpurile username, parola și email. La final acesta va apăsa pe butonul "Create!", care va trimite un post request în baza de date a aplicației server, adăugând în tabelul de autentificare datele utilizatorului. Dacă datele cu care acesta dorește să își creeze un cont deja există, atunci va primi o eroare anunțându-l acest lucru.

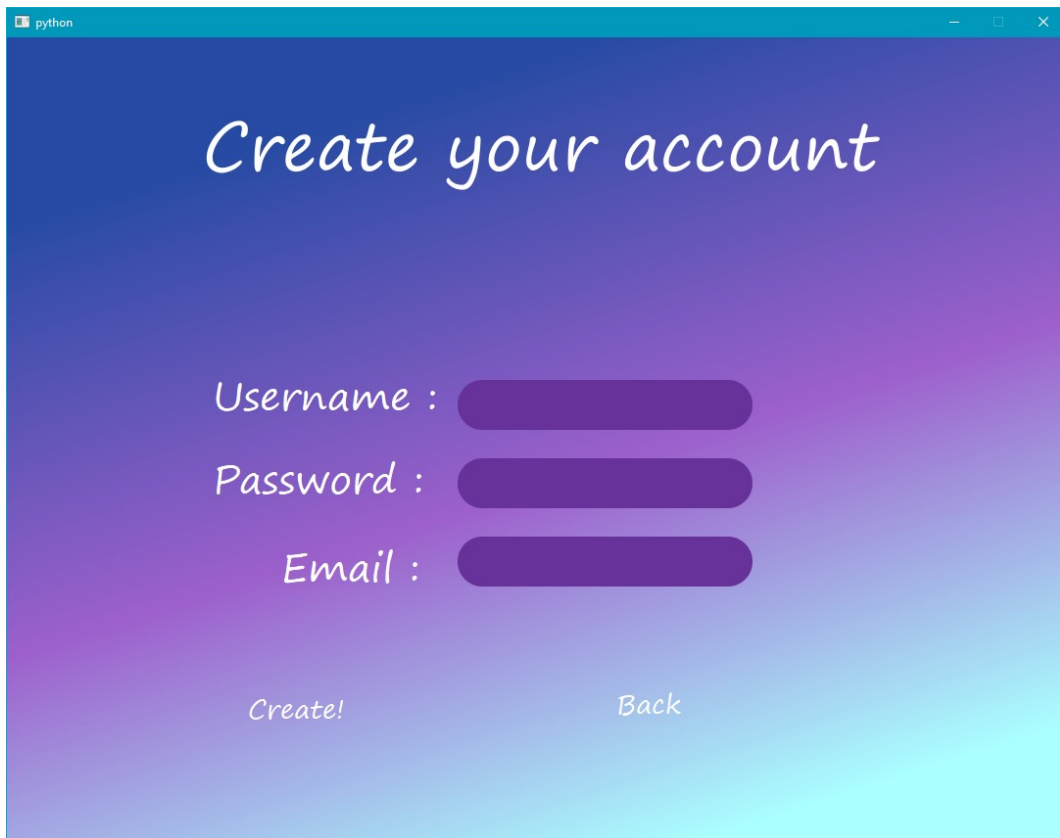
Un alt caz de utilizare constă în logarea utilizatorului în aplicație în cadrul căruia acesta va completa formularul de pe pagina de start, cu username și parolă.

Astfel apăsând pe logare, clientul va transmite către server cele două date, și le va verifica dacă există în baza de date. Dacă nu există, nu va permite logare și va transmite o eroare spunând că fie usernameul fie parola sunt gresite.

Dacă sunt corecte, utilizatorul va avea acces la pagina principală a aplicației, unde va avea opțiunile de a adăuga un profil al unei persoane, de a vedea o persoană, de a căuta o persoană după o imagine încărcată ulterior de acesta și de a căuta într-un video persoane, acesta din urmă fiind încărcat tot de utilizator. Opțiunea de "Forgot password?" va trimite utilizatorul la un alt ecran unde va trebui să își introducă emailul și să apese butonul de "Confirm". După acestea, aplicația client va trimite un get request la server care va verifica dacă există un cont înregistrat cu emailul specificat de utilizator. Dacă da, se va trimite un email la adresa respectivă cu usernameul și parola, altfel se va afișa mesajul de eroare "No account registered with this email".

Cazurile de utilizare care constituie soluția la problema menționată în introducere





A screenshot of a web application window titled 'python'. The background is a gradient of blue and purple. The main heading is 'Create your account' in a white, cursive font. Below the heading, there are three input fields for 'Username', 'Password', and 'Email', each with a white label and a dark purple rounded input box. At the bottom, there are two buttons: 'Create!' and 'Back', both in a white, cursive font.

python

# Create your account

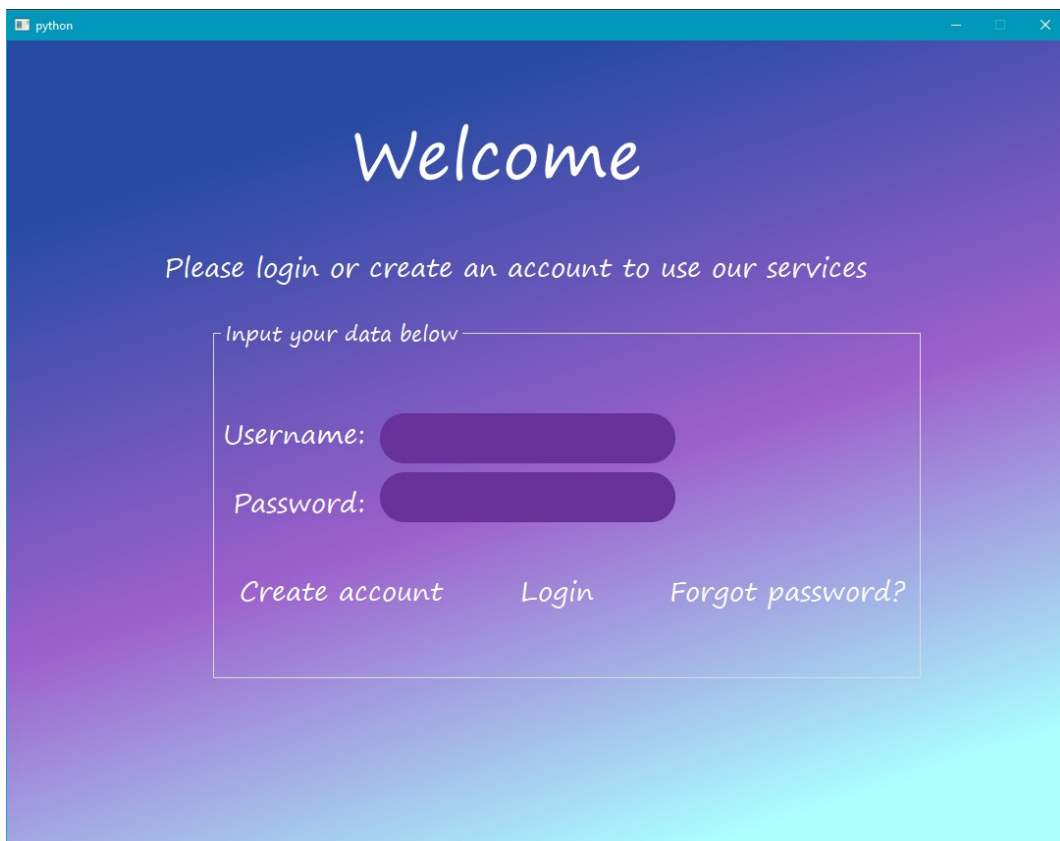
Username :

Password :

Email :

Create! Back

Figura 1.4: Ecranul de creare a unui cont



A screenshot of a web application window titled 'python'. The background is a gradient of blue and purple. The main heading is 'Welcome' in a white, cursive font. Below the heading, there is a line of text: 'Please login or create an account to use our services'. Underneath, there is a white-bordered box containing the text 'Input your data below'. Inside this box, there are two input fields for 'Username' and 'Password', each with a white label and a dark purple rounded input box. Below the input fields, there are three links: 'Create account', 'Login', and 'Forgot password?', all in a white, cursive font.

python

# Welcome

Please login or create an account to use our services

Input your data below

Username:

Password:

Create account Login Forgot password?

Figura 1.5: Ecranul de logare

reprezintă căutarea unei persoane în funcție de o poză oferită de utilizator și cautarea persoanelor într-un material video.

### 1.5.1 Cazul de utilizare: Căutarea unui profil

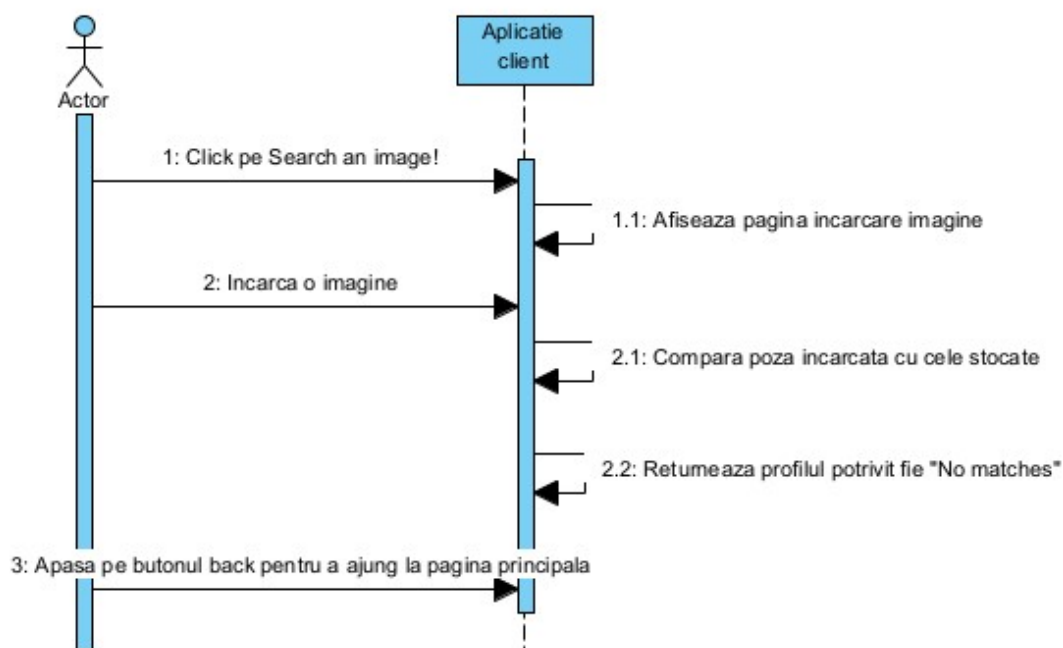


Figura 1.6: Căutarea unui profil

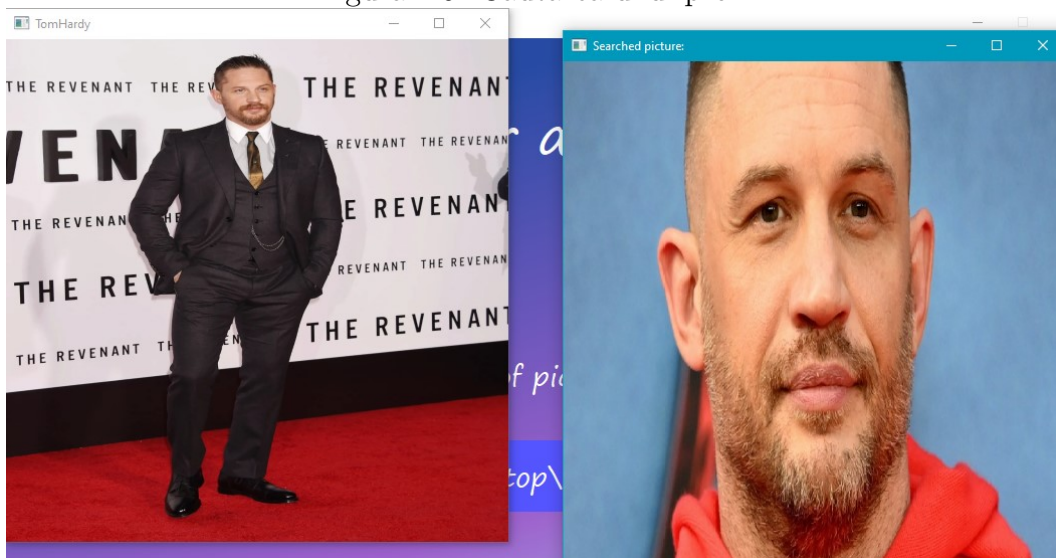


Figura 1.7: Actorul Tom Hardy căutat de aplicație

În cadrul acestui tip de căutare utilizatorul va fi nevoit să creeze unul sau mai multe profile în baza de date prin cazul de utilizare menționat anterior, întrucât scopul acestei funcționalități este de a recunoaște facial poze între ele, mai precis a face legătura între o poză neidentificată cu una care este cunoscută în prealabil.

### 1.5.2 Cazul de utilizare: Căutarea într-un video

În cadrul acestui tip de căutare utilizatorul trebuie să aleagă între a încarca un video și opțiunea de recunoaștere facială de pe webcam. Cea de a doua opțiune se accesează prin intermediul apăsării butonului de "Webcam", care va deschide webcamul dacă există. Prima opțiune se realizează prin introducerea pathului către un fișier video ( .mp4 de preferat ) în caseta de text, urmată de apăsarea butonului "From file". În cadrul ambelor opțiuni se vor detecta fețele persoanelor din materialul video, acestea fiind încadrate într-un dreptunghi care va fi etichetat cu numele persoanei dacă aceasta se regăsește în baza de date a serverului, iar dacă nu, cu numele "Unknown".

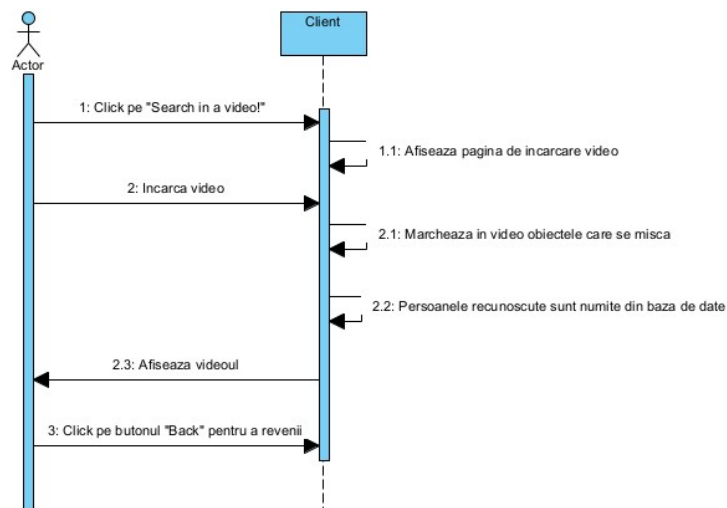


Figura 1.8: Diagrama căutare în video

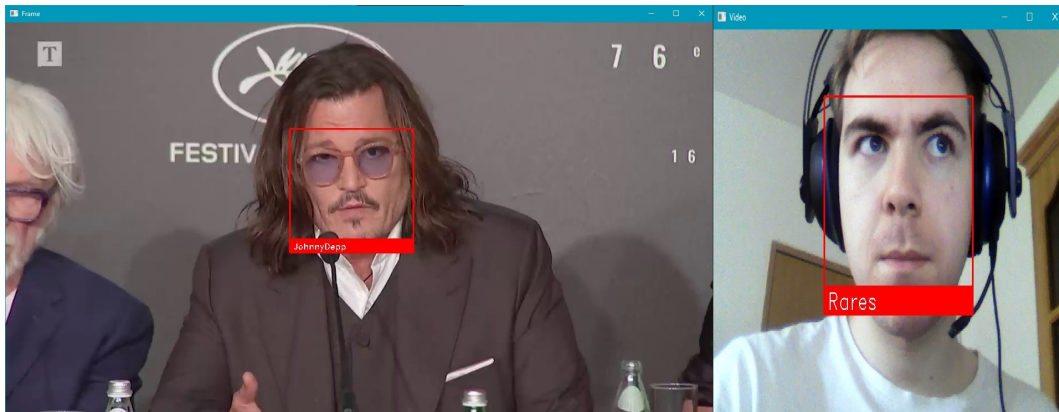


Figura 1.9: Căutare video stânga, Webcam dreapta

Pentru închiderea ușoară a ferestrelor indiferent dacă se caută prin webcam sau printr-un fișier se apasă tasta w.

# Capitolul 2

## Prezentarea contribuției autorului

### 2.1 Tehnologiile utilizate

Limbajul de programare ales este Python, întrucât oferă modelarea ușoară utilizând concepte de POO a claselor și suport pentru PyQt5 în care este realizată interfața grafică. Python de asemenea oferă suport pentru flaskAPI care este ușor de utilizat și oferă suficientă funcționalitate pentru partea de logare și autentificare în siguranță, scopul lucrării fiind preponderent reprezentat de partea de identificare și procesare facială. În realizarea interfeței grafice s-a utilizat Qt designer.

Pentru partea principală de procesare de imagini utilizez openCV datorită funcționalităților deja implementate și algoritmilor existenți care permit încărcarea, citirea, extragere de fundal și alte funcționalități cruciale pentru a identifica persoanele din imagini și conținut video. Deasemenea se utilizează extensiv biblioteca "facialregognition" întrucât aceasta dispune de manipulare mai ușoară a conținutului video de pe ecran sub formă de encodinguri faciale, care reprezintă o serie de trăsături ale fiecărei persoane și de funcții care calculează cea mai similară față sau distantă de asemănare între două fețe.

Serverul este realizat în flaskAPI pentru implementarea sistemului de autentificare și trimiterea de informații în baza de date prin endpointuri.

Interfața grafică este realizată în frameworkul QT pentru Python, PyQt5. În procesul de dezvoltare al interfeței grafice am utilizat aplicația Qt designer care permite implementarea cu ușurință a butoanelor și elementelor html, cât și designul css. Fișierele care conțin designul sunt stocate sub formatul ".ui". Pentru a edita codul și compila am utilizat PyCharm. [3] [2]

## 2.2 Detalii de implementare

### 2.2.1 Baza de date

Baza de date este creată în SQLAlchemy și conține două tabele: personModel și authModel. Primul are ca scop reținerea de date referitoare la persoanele pe care se va aplica reunoasterea facială, iar al doilea stocarea conturilor ce au acces la aplicație.[8]

```
1 class AuthModel(db.Model):
2     __tablename__ = 'auth_model'
3     id = db.Column(db.Integer, primary_key=True)
4     username = db.Column(db.String(20), nullable=False)
5     password = db.Column(db.String(20), nullable=False)
6     email = db.Column(db.String(20), nullable=False)
7
8     def __init__(self, username, password, email):
9         self.username = username
10        self.password = password
11        self.email = email
12
13    def __repr__(self):
14        return f"User( username={self.username}, password={self.
password}, email={self.email} )"
```

Listing 2.1: authModel code

```
1 class personModel(db.Model):
2     __tablename__ = 'person_model'
3     id = db.Column(db.Integer, primary_key=True)
4     CNP = db.Column(db.String, nullable=False)
5     name = db.Column(db.String(20), nullable=False)
6     age = db.Column(db.Integer, nullable=False)
7     address = db.Column(db.String(20), nullable=False)
8     sex = db.Column(db.String(10), nullable=False)
9     height = db.Column(db.Float, nullable=False)
10    imgPath = db.Column(db.String(50), nullable=False)
11
12    def __init__(self, name, CNP, age, address, sex, height, imgPath)
13    :
14        self.name = name
15        self.CNP = CNP
16        self.age = age
17        self.sex = sex
18        self.address = address
19        self.height = height
20        self.imgPath = imgPath
21
22    def __repr__(self):
23        return f"Person( name={self.name},CNP={self.CNP}, age={self.
age}, address={self.address},sex={self.sex}, height={self.height}
, imgPath={self.imgPath} )"
```

Listing 2.2: personModel code

### 2.2.2 Clasa Person

În cadrul aplicației transferul de date dintre baza de date și alte metode ale aplicației este realizat prin intermediul acestei clase intermediare care stochează informația relevantă pentru funcționalități. Am ales această implementare întrucât oferă posibilitatea apelării constructorului ca să creeze un obiect de tip `personProfile`, care ajută la plasarea datelor în cadrul cererilor de GET și POST trimise la endpointurile APIului. Dacă în viitoare se dorește extragerea de date cum ar fi doar 2 - 3 atribute, se pot utiliza funcțiile de `get` sau `set` membre ale clasei.

```
1 class personProfile:
2     name = ""
3     CNP = ""
4     Age = 0
5     Sex = ""
6     Address = ""
7     Height = 0
8     imgPath = ""
9
10    def __init__(self, name, CNP, Age, Address, Sex, Height, imgPath)
11    :
12        self.name = name
13        self.CNP = CNP
14        self.Age = Age
15        self.Sex = Sex
16        self.Address = Address
17        self.Height = Height
18        self.imgPath = imgPath
```

Listing 2.3: personClass

```
1 def toMainScreen(self):
2     login = MainScreen()
3     response = requests.get(BASEAddPerson, "")
4     people = response.json()
5     for person in people:
6         login.addItemToList(person)
7
8     widget.addWidget(login)
9     widget.setCurrentIndex(widget.currentIndex() + 1)
```

Listing 2.4: return to MainScreen

Funcționalitatea clasei de persoane ajută la repopularea listei cu persoane de pe ecranul principal, funcționalitate care permite vizual utilizatorului să observe clar ce persoane a adăugat și câte. Aceasta se apelează automat de fiecare dată când revenim la ecranul principal, de pe oricare altul.

### 2.2.3 Serverul în Flask

Aplicația este împărțită în două programare separate, prima parte având serverul care conține baza de date și end pointurile API realizate cu Flask, iar cea de a doua parte numită client. Aceasta conține interfața grafică și algoritmi de recunoaștere facială. Mai jos se regăsește codul pentru endpointuri.[6]

```
1 @app.route('/UserPass', methods=['GET', 'POST', 'PATCH', 'PUT', 'DELETE'])
2 def api_UserPass():
3     if request.method == 'GET':
4         username = request.args.get('username')
5         password = request.args.get('password')
6         alreadyExists = AuthModel.query.filter_by(username=username,
7 password=password).first()
8         print(alreadyExists)
9         if alreadyExists is None:
10             return "Does not exist!"
11         else:
12             return "Exists!"
13
14     elif request.method == 'POST':
15         return "ECHO: POST\n"
16
17     elif request.method == 'PATCH':
18         return "ECHO: PATCH\n"
19
20     elif request.method == 'PUT':
21         a = request.form.to_dict()
22         userList = list(a.values())
23         print(userList)
24         user = AuthModel(userList[0], userList[1], userList[2])
25         alreadyExists = AuthModel.query.filter_by(username=userList
26 [0]).first()
27         if alreadyExists is None:
28             db.session.add(user)
29             db.session.commit()
30         else:
31             return "Username already exists!"
32             return "Account created!"
33
34     elif request.method == 'DELETE':
35         return "ECHO: DELETE"
```

Listing 2.5: APIul UserPass

Endpointul UserPass se ocupă în principal cu metodele de GET și PUT. Metoda de GET realizează obținerea datelor din baza de date cu scopul verificării dacă există deja un utilizator cu numele și parola respective. Metoda de PUT trimite în baza de date, dar nu înainte de a verifica dacă deja există datele unui cont făcut nou pentru a avea acces la aplicație din client.

```

1 @app.route('/AddPerson', methods=['GET', 'POST', 'PATCH', 'PUT', '
    DELETE'])
2 def api_AddPerson():
3     if request.method == 'PUT':
4         a = request.form.to_dict()
5         addPersonList = list(a.values())
6         person = personModel(addPersonList[0], addPersonList[1],
7                               addPersonList[2], addPersonList[3], addPersonList[4],
8                               addPersonList[5], addPersonList[6])
9         alreadyExists = personModel.query.filter_by(name=
10        addPersonList[0]).first()
11         if alreadyExists is None:
12             db.session.add(person)
13             db.session.commit()
14             return "Added!"
15         else:
16             return "same name!"
17
18     if request.method == 'GET':
19         query = db.session.execute(db.select(personModel.name))
20         people = []
21         for i in query:
22             people.append(i[0])
23         return jsonify(people)

```

Listing 2.6: APIul AddPerson

Endpointul AddPerson lucrează cu metodele PUT și GET. Metoda PUT adaugă un nou candidat în tabelul "personModel" unde sunt stocate datele persoanelor care vor fi utilizate în recunoașterea facială, verificând după nume existența altei persoane. Metoda GET trimite către client în format json numele tuturor persoanelor din baza de date. Endpointul DBImages oferă doar funcționalitate pe GET cu scopul de a returna un dicționar unde cheia este numele iar valoarea este pathul către poza persoanelor din baza de date.

```

1 @app.route('/DBImages', methods=['GET', 'POST', 'PATCH', 'PUT', '
    DELETE'])
2 def api_DBImages():
3     if request.method == 'GET':
4         queryName = db.session.execute(db.select(personModel.name))
5         queryImg = db.session.execute(db.select(personModel.imgPath))
6
7         Names = []
8         for i in queryName:
9             Names.append(i[0])
10
11         Images = []
12         for i in queryImg:
13             Images.append(i[0])
14
15         nameImage = {}
16         for i in range(0, len(Names)):
17             nameImage[Names[i]] = Images[i]
18
19         return jsonify(nameImage)

```

Listing 2.7: APIul DBImages



## 2.2.4 Interfața grafică

Interfața grafică constă în mai multe ecrane care sunt implementate în QtDesigner atât cu elemente statice care rămân aceleași cât și elemente dinamice care se schimbă în funcție de ce face utilizatorul. Primul ecran este cel de start, implementat sub forma clasei "StartingScreen". Atributele clasei sunt casete pentru nume, parolă și butoane ale căror funcție de "onClick" este realizată drept funcție membră a clasei. Navigarea între pagini se face cu un widget array, la care se adaugă pagina nouă dacă se merge către o pagină nouă, sau se merge în spate cu un element pentru o pagină anterioară.

```
1 class StartingScreen(QDialog):
2     def __init__(self):
3         super(StartingScreen, self).__init__()
4         loadUi("startingScreen.ui", self)
5         self.lineEditPassword.setEchoMode(QLineEdit.Password)
6         self.pushButtonLogin.clicked.connect(self.ToLogin)
7         self.pushButtonCreateAccount.clicked.connect(self.ToCreate)
8         self.pushButtonForgotPass.clicked.connect(self.toRecoverPass)
9
10    def ToLogin(self):
11        global username
12        username = self.lineEditUsername.text()
13        password = self.lineEditPassword.text()
14        data = {"username": username, "password": password}
15        response = requests.get(BASELogin, data)
16
17        if response.text == "Exists!":
18            login = MainScreen()
19            response = requests.get(BASEAddPerson, "")
20            people = response.json()
21            for person in people:
22                login.addItemToList(person)
23            widget.addWidget(login)
24            widget.setCurrentIndex(widget.currentIndex() + 1)
25        elif response.text == "Does not exist!":
26            errmsg = QMessageBox()
27            errmsg.setWindowTitle("Error on login!")
28            errmsg.setText("Wrong username or password!")
29            errmsg.setIcon(QMessageBox.Critical)
30            errmsg.exec()
31
32    @staticmethod
33    def ToCreate():
34        createaccount = AccountCreateScreen()
35        widget.addWidget(createaccount)
36        widget.setCurrentIndex(widget.currentIndex() + 1)
37
38    @staticmethod
39    def toRecoverPass():
40        recPass = RecoverPassword()
41        widget.addWidget(recPass)
42        widget.setCurrentIndex(widget.currentIndex() + 1)
43
44    def getUsername(self):
45        return self.lineEditUsername.text()
```

Listing 2.8: StartingScreen code

Ecranul de start menționat anterior oferă posibilitatea utilizatorului de creare a unui cont personal care va fi stocat pe server și utilizat de acesta pentru logarea în client. Butonul "Create account" schimbă ecranul în cel de "Create Account". Funcția "createAccount" se ocupă cu luarea textului din câmpurile completate de utilizator, verifică validitatea lor și dacă un cont cu datele introduse există deja, se va afișa un mesaj, altfel contul va fi creat. Acest ecran poate doar să revină la cel de start prin apăsarea butonului "Back".

```
1 class AccountCreateScreen(QDialog):
2     def __init__(self):
3         super(AccountCreateScreen, self).__init__()
4         loadUi("createAccountScreen.ui", self)
5         self.buttonGoToStartFromCreate.clicked.connect(self.toStart)
6         self.buttonCreateAccount.clicked.connect(self.createAccount)
7
8     @staticmethod
9     def toStart():
10        startingScreen = StartingScreen()
11        widget.addWidget(startingScreen)
12        widget.setCurrentIndex(widget.currentIndex() + 1)
13
14    def createAccount(self):
15        usernameCreate = self.lineEditUsernameCreate.text()
16        passwordCreate = self.lineEditPasswordCreate.text()
17        emailCreate = self.lineEditEmailCreate.text()
18        if usernameCreate == "" or passwordCreate == "" or
emailCreate == "":
19            badMsg = QMessageBox()
20            badMsg.setWindowTitle("Not complete!")
21            badMsg.setText("Fill out both fields please!")
22            badMsg.setIcon(QMessageBox.Critical)
23            badMsg.exec()
24        else:
25            data = {"username": usernameCreate, "password":
passwordCreate, "email": emailCreate}
26            response = requests.put(BASELogin, data)
27
28            if response.text == "Account created!":
29                goodMsg = QMessageBox()
30                goodMsg.setWindowTitle("Account message!")
31                goodMsg.setText("Created successfully!")
32                goodMsg.setIcon(QMessageBox.NoIcon)
33                goodMsg.exec()
34            elif response.text == "Username already exists!":
35                badMsg = QMessageBox()
36                badMsg.setWindowTitle("Error on creation!")
37                badMsg.setText("Account already exists, please login!")
38
39                badMsg.setIcon(QMessageBox.Critical)
40                badMsg.exec()
```

Listing 2.9: AccountCreate code

În urma unei logării cu un cont existent, utilizatorul va fi trimis pe pagina principală a aplicației unde este întâmpinat cu o salutare cu numele acestuia, o listă cu toate persoanele din baza de date pe care se va efectua recunoașterea facială și opțiunile aplicației: adăugarea unei persoane, căutarea unei poze și căutarea într-un material video. Acest ecran este cel mai conectat din aplicație întrucât poate merge înapoi la cel de start, sau la toate funcționalitățile aplicației. La fiecare accesare a acestui ecran se va reîncarca lista cu persoane adăugate în aplicație în cazul că una sau mai multe persoane au fost adăugate relativ la ultima actualizare. Pentru această funcționalitate am ales utilizarea unui atribut denumit "listPerson", o listă în care sunt adăugate numele tuturor persoanelor luate din baza de date.

```

1 class MainScreen(QDialog):
2     def __init__(self):
3         super(MainScreen, self).__init__()
4         loadUi("mainScreen.ui", self)
5         self.mainScreenLabel.setText("Hi, " + username + " !")
6         self.buttonGoToStartFromMain.clicked.connect(self.toStart)
7         self.buttonAddProfile.clicked.connect(self.toAddProfile)
8         self.buttonSearchPerson.clicked.connect(self.toSearchPerson)
9         self.buttonSearchVideo.clicked.connect(self.toSearchVideo)
10        self.listPerson
11
12    def addItemToList(self, name):
13        self.listPerson.addItem(name)
14
15    @staticmethod
16    def toSearchVideo():
17        smh = VideoType()
18        widget.addWidget(smh)
19        widget.setCurrentIndex(widget.currentIndex() + 1)
20
21    @staticmethod
22    def toSearchPerson():
23        sP = searchPerson()
24        widget.addWidget(sP)
25        widget.setCurrentIndex(widget.currentIndex() + 1)
26
27    @staticmethod
28    def toStart():
29        startingScreen = StartingScreen()
30        widget.addWidget(startingScreen)
31        widget.setCurrentIndex(widget.currentIndex() + 1)
32
33    @staticmethod
34    def toAddProfile():
35        AddPersonScreen = addPersonScreen()
36        widget.addWidget(AddPersonScreen)
37        widget.setCurrentIndex(widget.currentIndex() + 1)

```

Listing 2.10: mainScreen code

Din ecranul principal putem naviga în ecranul de adăugare a unei persoane pentru a fi utilizată ulterior în recunoaștere facială.

```
1 class addPersonScreen(QDialog):
2     def __init__(self):
3         super(addPersonScreen, self).__init__()
4         self.name = ""
5         loadUi("addPersonScreen.ui", self)
6         self.pushButtonBackFromAdd.clicked.connect(self.toMainScreen)
7         self.pushButtonAddPerson.clicked.connect(self.addPerson)
8         self.pushButtonUploadButton.clicked.connect(self.
uploadPicture)
9
10    @staticmethod
11    def toMainScreen(self):
12        login = MainScreen()
13        response = requests.get(BASEAddPerson, "")
14        people = response.json()
15        for person in people:
16            login.addItemToList(person)
17
18        widget.addWidget(login)
19        widget.setCurrentIndex(widget.currentIndex() + 1)
20
21    def uploadPicture(self):
22        try:
23            path = self.lineEditPathToFile.text()
24            if path != "":
25                image = cv2.imread(path)
26                cv2.imshow('ceva', image)
27                cv2.waitKey()
28                cv2.destroyAllWindows()
29            else:
30                Msg = QMessageBox()
31                Msg.setWindowTitle("Error")
32                Msg.setText("Give a non empty file path!")
33                Msg.setIcon(QMessageBox.NoIcon)
34                Msg.exec()
35        except Exception:
36            Msg = QMessageBox()
37            Msg.setWindowTitle("Error")
38            Msg.setText("Give a valid file path!")
39            Msg.setIcon(QMessageBox.NoIcon)
40            Msg.exec()
41
42    def setName(self, Name):
43        self.name = Name
44
45    def addPerson(self):
46        x = personProfile(self.lineEditPersonName.text(), self.
lineEditCNP.text(), self.lineEditAge.text(),
47                        self.lineEditAddress.text(), self.
lineEditSex.text(), self.lineEditHeight.text(),
48                        self.lineEditPathToFile.text())
49        self.setName(x.getName())
50
51        data = {"name": x.getName(), "CNP": x.getCNP(), "age": x.
getAge(), "address": x.getAddress(), "sex": x.getSex(),
```

```

52         "height": x.getHeight(), "imgPath": x.getImgPath() }
53     if x.getName() == "" or x.getCNP() == "" or x.getAge() == "" or
    x.getAddress() == "" or x.getSex() == "" or x.getHeight() == ""
    or x.getImgPath() == "":
54         Msg = QMessageBox()
55         Msg.setWindowTitle("Hm")
56         Msg.setText("Fill out all the inputs!")
57         Msg.setIcon(QMessageBox.NoIcon)
58         Msg.exec()
59     else:
60         response = requests.put(BASEAddPerson, data)
61         if response.text == "Added!":
62             goodMsg = QMessageBox()
63             goodMsg.setWindowTitle("Great success!")
64             goodMsg.setText("Added successfully!")
65             goodMsg.setIcon(QMessageBox.NoIcon)
66             goodMsg.exec()
67             login = MainScreen()
68             response = requests.get(BASEAddPerson, "")
69             people = response.json()
70             for person in people:
71                 login.addItemToList(person)
72
73             widget.addWidget(login)
74             widget.setCurrentIndex(widget.currentIndex() + 1)
75         elif response.text == "same name!":
76             goodMsg = QMessageBox()
77             goodMsg.setWindowTitle("Not so great...")
78             goodMsg.setText("That name already exists!")
79             goodMsg.setIcon(QMessageBox.NoIcon)
80             goodMsg.exec()

```

Listing 2.11: addPersonScreen code

Această clasă permite reținerea de date cruciale pentru o persoană: nume, sex, înălțime, greutate cât și o poză a persoanei. De încărcarea pozei se ocupă funcția "uploadPicture" care verifică validitatea căii oferite către poză și ridică erori în funcție de caz. Funcția "addPerson" preia din casetele de text datele introduse de utilizator și le plasează într-un obiect de tip "personProfile". În cazul în care unul din inputuri lipsește sau există o persoană cu același nume vor apărea erori către utilizator, iar în cazul de adăugare cu succes un mesaj care să confirme acest lucru. În cazul adăugării cu succes datele persoanei: numele, sexul, poza etc vor fi trimise la server și adăugate în tabelul "personModel" unde vor fi obținute mereu când se revine la ecranul principal, pentru actualizarea listei cu persoanele deja adăugate în aplicație. Poza este reprezentată drept pathul către aceasta din calculatorul clientului, aceasta fiind deci locală. Am ales acest mod de stocare datorita ușurinței și lipsei de relevanță în cazul în care o persoană neautorizată vede calea către fișier. Persoanele dintr-o agenție sau companie pot să își trimită pozele între ei pe o aplicație de chatting și să le adauge local. Dacă agenția sau compania detine o bază de date cu o multitudine de poze atunci acestea pot fi arhivate și trimise în mod identic.

Aplicația conține un ecran pentru recuperarea parolei. Această funcționalitate este implementată în clasa `RecoverPassword` care conține butoane pentru navigarea înapoi la pagina de logare, o casetă în care se introduce emailul contului și un buton de recuperare. Metoda principală este `toRecoverPass` care obține textul din caseta de pe ecran și apelează endpointul serverului `"BASERecoverPass"` trimițând o cerere de `get` cu variabila `data`, care conține emailul introdus în caseta de utilizator. Serverul răspunde cu parola care corespunde emailului și cu numele de utilizator. Se crează un mesaj de forma `"Hello " + userData[0] + " your old password is " + userData[1] + ". To change it, contact a system administrator."` unde `userData[0]` este numele contului iar `userData[1]` este parola uitată. Acest mesaj este trimis cu ajutorul protocolului `smtp` de pe o adresă de email creată special pentru aplicație către emailul introdus de utilizator. În cazul unui email care nu există în baza de date, se afișează un mesaj.

```

1 class RecoverPassword(QDialog):
2     def __init__(self):
3         super(RecoverPassword, self).__init__()
4         loadUi("recoverPassword.ui", self)
5         self.buttonRecoverPass.clicked.connect(self.toRecoverPass)
6         self.buttonBackFromRecover.clicked.connect(self.
toStartFromRecover)
7
8     def toRecoverPass(self):
9         try:
10             emailReceiver = self.lineEditRecoverPassword.text()
11             data = {'email': emailReceiver}
12             response = requests.get(BASERecoverPass, data)
13             userData = response.json()
14
15             # sending email with something
16             emailSender = 'noreplypoliceapp@gmail.com'
17             emailPass = 'tavebryjbcdfnsxb'
18             subject = 'App password recovery'
19             body = "Hello " + userData[0] + " your old password is "
+ userData[
20                 1] + ". To change it, contact a system administrator.
"
21
22             em = EmailMessage()
23             em['From'] = emailSender
24             em['To'] = emailReceiver
25             em['Subject'] = subject
26             em.set_content(body)
27
28             context = ssl.create_default_context()
29             with smtplib.SMTP_SSL('smtp.gmail.com', 465, context=
context) as smtp:
30                 smtp.login(emailSender, emailPass)
31                 smtp.sendmail(emailSender, emailReceiver, em.
as_string())
32         except:
33             Msg = QMessageBox()
34             Msg.setWindowTitle("Error")
35             Msg.setText("Invalid input or email not found!")
36             Msg.setIcon(QMessageBox.NoIcon)
37             Msg.exec()

```

Listing 2.12: recoverPassword code

## 2.2.5 Recunoașterea facială

Căutarea unei persoane în baza de date cu ajutorul unei poze este realizată utilizând biblioteca facialrecognition. De aceste obiective se ocupă clasa searchPerson. Aceasta are ca atribute butoanele care se ocupă cu navigarea pe pagină și încărcarea pozei respective. Metoda findMatch are ca scop găsirea persoanei căutate în baza de date, în cazul pozitiv, adică persoana a fost găsită, se vor deschide 2 ferestre, una cu poza încărcată și una cu poza din baza de date a persoanei căutate. În cazul negativ, se va afișa un mesaj spunând că nici-o persoană similară nu a fost găsită. Pentru a realiza cele menționate, poza încărcată este redimensionată la 500x500 și se apelează endpointul intitulat BASEImgDB care răspunde cu un dicționar care este împărțit în două bucăți: o listă numită Images care ia valorile din dicționar, acestea reprezentând pozele din baza de date ale persoanelor existente, precum și o altă listă numită Names care reprezintă numele corespunzătoare pozelor de pe același index din lista Images. Cu ajutorul unei structuri de tip For se ia fiecare imagine din lista de imagini și se calculează un encoding pentru aceasta care conține caracteristicile imaginii. Dacă un encoding poate fi creat cu succes, se compară encodingul rezultat cu cel al pozei încărcate. Dacă există un rezultat acesta este arătat pe ecran, în cazul opus se afișează un mesaj care să menționeze că nu s-a găsit o potrivire.[1] [5]

```
1 class searchPerson(QDialog):
2     def __init__(self):
3         super(searchPerson, self).__init__()
4         loadUi("searchPerson.ui", self)
5         self.pushButtonToMainFromSearch.clicked.connect(self.
        backToMain)
6         self.pushButtonFindMatch.clicked.connect(self.findMatch)
7         self.pushButtonUploadToMatch.clicked.connect(self.
        uploadToMatch)
8
9     def findMatch(self):
10        findThisImagePath = self.lineEditPathToFile.text()
11        findThisImage = cv2.imread(findThisImagePath)
12        response = requests.get(BASEImgDB)
13        NameImages = response.json()
14        Images = list(NameImages.values())
15        Names = list(NameImages.keys())
16        c = 0
17        found = 0
18        searchedImage = face_recognition.load_image_file(
        findThisImagePath)
19        dim = (500, 500)
20        try:
21            for i in Images:
22                image = face_recognition.load_image_file(i)
23                facialEncodingSearched = face_recognition.
        face_encodings(searchedImage)[0]
24                facialEncodingCompare = face_recognition.
        face_encodings(image)[0]
25                # compare
26                results = face_recognition.compare_faces([
        facialEncodingSearched], facialEncodingCompare)
27                if results[0]:
28                    FromDB = cv2.imread(Images[c])
```

```

29         FromDB = cv2.resize(FromDB, dim, interpolation=
cv2.INTER_AREA)
30         findThisImage = cv2.resize(findThisImage, dim,
interpolation=cv2.INTER_AREA)
31         cv2.imshow("Searched picture: ", findThisImage)
32         cv2.imshow(Names[c], FromDB)
33         cv2.waitKey()
34         cv2.destroyAllWindows()
35         found = 1
36         break
37         c = c + 1
38     if found == 0:
39         Msg = QMessageBox()
40         Msg.setWindowTitle("I am sorry")
41         Msg.setText("No similar person found")
42         Msg.setIcon(QMessageBox.NoIcon)
43         Msg.exec()
44     except:
45         Msg = QMessageBox()
46         Msg.setWindowTitle("I am sorry")
47         Msg.setText("Only upload colored pictures with only 1
person!")
48         Msg.setIcon(QMessageBox.NoIcon)
49         Msg.exec()

```

Listing 2.13: searchPerson code 1

Clasa searchPerson conține și metodele uploadToMatch() și backToMain() care se ocupă cu încărcarea pozei și afișarea ei ca utilizatorul să poată verifica dacă a încărcat poza corectă, respectiv navigarea înapoi către pagina principală. [1]

```

1     def uploadToMatch(self):
2         try:
3             path = self.lineEditPathToFile.text()
4             if path != "":
5                 image = cv2.imread(path)
6                 dim = (500, 500)
7                 image = cv2.resize(image, dim, interpolation=cv2.
INTER_AREA)
8                 cv2.imshow('To be searched picture', image)
9                 cv2.waitKey(0)
10                cv2.destroyAllWindows()
11            else:
12                Msg = QMessageBox()
13                Msg.setWindowTitle("Error")
14                Msg.setText("Give a non empty file path!")
15                Msg.setIcon(QMessageBox.NoIcon)
16                Msg.exec()
17            except Exception:
18                Msg = QMessageBox()
19                Msg.setWindowTitle("Error")
20                Msg.setText("Give a valid file path!")
21                Msg.setIcon(QMessageBox.NoIcon)
22                Msg.exec()

```

Listing 2.14: searchPerson code 2



Căutarea video este implementată prin clasa VideoType. Aceasta are butoane pentru a oferi utilizatorului opțiunea de a căuta într-un video fie încărcat de acesta fie prin webcam. Dacă se alege opțiunea de fișier este necesară introducerea adresei fișierului video într-o casetă și apăsarea butonului care va declanșa funcția toVideoFile. Funcția începe prin a prelua adresa videoului și cheamă endpointul de pe server numit "BaseImgDB" care răspunde cu un dicționar în care există perechi alcătuite din imagine și numele persoanei din imagine. Acestea sunt împărțite ulterior în două liste, una cu poze și una cu nume de persoane. Se construiește o listă care conține encodingurile fiecărei poze din lista cu poze. [1]

```

1 def toVideoFile(self):
2     captureAdress = self.lineEditVideoPath.text()
3     capture = cv2.VideoCapture(captureAdress)
4     response = requests.get(BASEImgDB)
5     NameImages = response.json()
6     Images = list(NameImages.values())
7     Names = list(NameImages.keys())
8     knownFaces = []
9
10    for i in Images:
11        image = face_recognition.load_image_file(i)
12        imageEncode = face_recognition.face_encodings(image)[0]
13        knownFaces.append(imageEncode)
14    frame_number = 0
15
16    while capture.isOpened():
17        ret, frame = capture.read()
18        frame_number += 1
19        if ret:
20            face_locations = face_recognition.face_locations(frame)
21            face_encodings = face_recognition.face_encodings(frame,
22            face_locations)
23            face_names = []
24
25            for face_encoding in face_encodings:
26                matches = face_recognition.compare_faces(knownFaces,
27                face_encoding, tolerance=0.50)
28                name = "Unknown"
29                face_distances = face_recognition.face_distance(
30                knownFaces, face_encoding)
31                best_match_index = np.argmin(face_distances)
32
33                if matches[best_match_index]:
34                    name = Names[best_match_index]
35                    face_names.append(name)
36
37                for (top, right, bottom, left), name in zip(
38                face_locations, face_names):
39                    if not name:
40                        continue
41
42                    cv2.rectangle(frame, (left, top), (right, bottom),
43                    (0, 0, 255), 2)
44                    cv2.rectangle(frame, (left, bottom - 25), (right,
45                    bottom), (0, 0, 255), cv2.FILLED)
46                    font = cv2.FONT_HERSHEY_DUPLEX
47                    cv2.putText(frame, name, (left + 6, bottom - 6), font

```

```

    , 0.5, (255, 255, 255), 1)
42
    cv2.imshow('Frame', frame)
43
    if cv2.waitKey(25) & 0xFF == ord('w'):
44
        break
45

```

Listing 2.15: videoFile recognition

Cât timp captura din videoul încărcat este pornită se aplică urmatorul proces pe fiecare cadru al videoclipului: se obțin encoduri ale fiecărei persoane din cadrul curent împreună cu locațiile lor. Cu o structură de tip for se parcurge lista faceEncodings cu scopul de a lua fiecare encoding în parte și a găsi cel mai similar encoding din baza de date. Dacă se găsește o similaritate, în jurul feței utilizându-ne de locația obținută anterior se desenează un pătrat roșu cu o etichetă ce conține numele persoanei luată din lista cu nume de persoane.

Un proces similar este realizat și pentru căutarea video, sursa datelor fiind webcamul. Diferența majoră constă în modul în care sunt obținute cadrele, aici viteza fiind mai mică.

```

1 while True:
2     # frame
3     ret, frame = video_capture.read()
4     # encodes
5     faceLocations = face_recognition.face_locations(frame)
6     faceEncodings = face_recognition.face_encodings(frame,
7         faceLocations)
8
9     # loop
10    for (top, right, bottom, left), face_encoding in zip(
11        faceLocations, faceEncodings):
12        matches = face_recognition.compare_faces(knownFaceEncodings,
13            face_encoding)
14        name = "Unknown"
15        face_distances = face_recognition.face_distance(
16            knownFaceEncodings, face_encoding)
17        best_match_index = np.argmin(face_distances)
18
19        if matches[best_match_index]:
20            name = Names[best_match_index]

```

Listing 2.16: webcamVideo recognition

Similar cu recunoașterea dintr-un fișier video încărcat în prealabil, fețele găsite sunt marcate și încadrate într-un pătrat roșu și conțin o etichetă cu numele găsit, sau "Unknown person". Datorită vitezei mici de afișare, utilizatorul poate să închidă fereastra apăsând tasta 'W'.

```

1     if cv2.waitKey(1) & 0xFF == ord('w'):
2         break
3
4 video_capture.release()
5 cv2.destroyAllWindows()

```

Listing 2.17: exitVideo

## 2.3 Manual de utilizare

### 2.3.1 Crearea și logarea unui cont

Pentru utilizarea aplicației este nevoie de un cont. Utilizatorul trebuie să pornească aplicația client. Aplicația server trebuie să fie pornită deasemenea. Se apasă pe butonul "Create account!" și se introduce un nume de utilizator, o parolă și un email care poate fi accesat. După completare se apasă pe butonul "Create!". Dacă sunteți întâmpinat cu un mesaj de eroare care vă spune că deja există un utilizator cu acel nume sau email, introduceți alt nume și email. După ce contul a fost creat se poate reveni la pagina de start și introducerea datelor în formularul de logare. In cazul uitării parolei contului, aceasta poate fi recuperată apăsând butonul "Forgot password", și introducerea emailului asociat.

### 2.3.2 Adăugarea unei persoane

Pentru ca recunoașterea facială a aplicației să funcționeze este nevoie de o bază de date creată înainte de alegerea unei opțiuni de identificare facială. Odată logat în aplicație utilizatorul apasă pe butonul "Add a person!" și este întâmpinat cu un ecran unde poate completa câmpurile: Name, CNP, Age, Address, Sex si Height cu valorile care se cunosc pentru persoana, altfel se pune 0 la numere sau - pentru ceva care ar fi format din cuvinte. Sub textul "Path to file" se va insera locația de pe diskul utilizatorului la care se află poza persoanei care se dorește a fi adăugată. După inserare în caseta de sub textul menționat, se poate apăsa pe "Display picture!" pentru a se vedea poza încărcată - această opțiune servește drept o verificare. Când utilizatorul este mulțumit de datele introduse se apasă pe "Add!", altfel pe "Back". De fiecare dată când se revine la ecranul de "Hello, -nume utilizator- !", lista cu persoane este actualizată automat, nefiind nevoie de un restart al aplicației sau al serverului.

Input your data below

Name:

CNP:

Age:

Address:

Sex:

Height:

Path to file:

Display picture!

Add!

Back

Figura 2.1: Adăugarea unei persoane

### 2.3.3 Căutarea fotografică și video

Aceste două funcționalități pot fi utilizate oricând, însă este recomandat să fie utilizate doar după adăugarea în prealabil a mai multor persoane în baza de date pentru ca recunoașterea să aibă succes. Pentru a afla cine este într-o poză relativ la baza de date se apasă pe butonul "Search a person", care va arăta pe ecran o etichetă intitulată "Location of picture", sub aceasta o casetă și butonul "Display picture!". Utilizatorul trebuie să pună locația de pe disk a imaginii pe care dorește să o caute în casetă, ulterior putând a se verifica apăsând butonul menționat. Când acești pași au fost realizați se apasă pe butonul "Find!" iar după un timp de așteptare, aplicația va afișa două ferestre: una cu poza căutată care va avea drept titlu la bara ferestrei de sus "Searched picture" și o altă fereastră cu poza cu persoana găsită din baza de date, intitulată "Found picture". Apăsarea tastei W închide ambele ferestre. Pentru revenire la ecranul cu opțiunile principale se apasă pe "Back". Căutarea în video funcționează similar însă utilizatorul trebuie să apese pe butonul "Search in a video!", după care să aleagă între "Webcam" sau "From file". Dacă opțiunea de webcam este aleasă, acesta trebuie să aibă o cameră video conectată la dispozitiv și să permită aplicației să o utilizeze. Dacă aceste condiții sunt îndeplinite, se va deschide o fereastră cu inputul din webcam iar pe cadrele din materialul video vor fi marcate persoanele cunoscute. Rata de afișare a cadrelor este redusă datorită intensității procesului necesar ca aplicația să funcționeze. Dacă opțiunea "From file" este aleasă, trebuie să existe o cale validă către un fișier video, preferabil în format .mp4 în caseta din josul ecranului, după cum menționează și nota din subsolul paginii aplicației. Procesul de recunoaștere este identic cu cel cu sursa webcam, rata de afișare a cadrelor este redusă, iar pe ecran persoanele nerecunoscute sunt marcate drept "Unknown". Pentru ca să recunoască o persoană, mai întâi trebuie observată o față, astfel fețele parțiale văzute cum ar fi pe jumătate sau persoanele cu capul întors nu vor fi identificate dar vor fi detectate drept "Unknown".

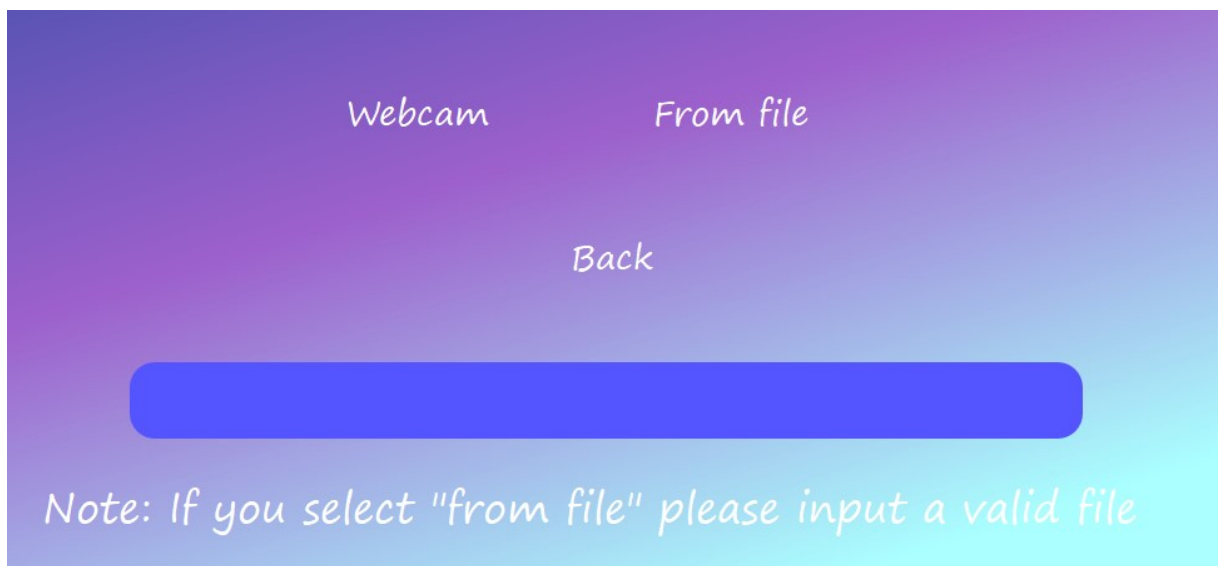


Figura 2.2: Adăugarea unei persoane

## 2.4 Manual de implementare

Funcționalitățile serverului se regăsesc în fișierul din folderul FlaskAPI. Funcționalitatea pentru cele două baze de date constă în endpointuri de tip GET și PUT care au fost implementate în funcție de unde a fost nevoie de acestea. Dacă se dorește funcția există însă nu returnează nimic și poate fi implementată în antetul deja plasat. Funcționalitatea pentru interfața grafică este recomandat să fie realizată în Qt5Designer, o aplicație software realizată pentru editarea cu ușurință a fișierelor de tip ".ui". Ecrane deja existente pot fi deschise și editate în aplicație. Funcționalitățile de recunoaștere facială pot fi modificate prin intermediul variabilei frameCounter. Dacă aceasta este incrementată mai rapid, default având o creștere de o unitate, viteza de procesare a cadrelor în materialele video va crește, însă va scădea acuratețea. Modificarea algoritmilor în sine poate fi realizată în funcția "toWebcam" și respectiv în "toVideoFile" în cadrul structurii repetitive while. Atât culoarea cadranelor poate fi schimbată, cât și mărimea acestuia. Pentru recunoașterea facială realizată pe materiale statice de tip .jpg, adică poze, rezoluția la care este procesată poza poate fi schimbată din 500x500 pentru acuratețe mai mare, însă o durată de lucru mai îndelungată. Lista cu adrese pentru API se află la începutul codului clientului și respectă formatul de nume "BASE" + " -funcționalitate- ".

```
1  # box
2  cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255),
3      2)
4
5  # label
6  cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0,
7      255), cv2.FILLED)
8  font = cv2.FONT_HERSHEY_DUPLEX
9  cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255,
10     255, 255), 1)
```

Listing 2.18: Edit frame - name

```
1  BASELogin = "http://127.0.0.1:5000/UserPass"
2  BASEAddPerson = "http://127.0.0.1:5000/AddPerson"
3  BASEImgDB = "http://127.0.0.1:5000/DBImages"
4  BASEEmailRequest = "http://127.0.0.1:5000/emailRequests"
5  BASERecoverPass = "http://127.0.0.1:5000/RecoverPass"
```

Listing 2.19: BASE name model

```
1  frame_number = 0
2  while capture.isOpened():
3
4      # get the frame
5      ret, frame = capture.read()
6      frame_number += 1
7
8      if ret:
9          # get loc & encods in current frame
10         face_locations = face_recognition.face_locations(frame)
11         face_encodings = face_recognition.face_encodings(frame,
12             face_locations)
```

Listing 2.20: Video recognition while loop

## Capitolul 3

### Concluzii și direcții viitoare

Recunoașterea facială realizată în cadrul lucrării de licență demonstrează posibilitatea dezvoltării unei aplicații care este capabilă să recunoască o gamă largă de persoane fără utilizarea de inteligență artificială în ciuda tendințelor curente din industria IT. Deasemenea algoritmi prezentați sunt capabili să lucreze doar cu o singură poză a unei persoane, iar rezultatele obținute demonstrează o acuratețe deosebită, în detrimentul memoriei și a duratei de procesare. Astfel, acest tip de recunoaștere facială se dovedește superioară în contextul practic de a recunoaște persoane având doar o poză ca punct de plecare, relativ la abordarea cu o rețea neuronală antrenată - întrucât aceasta din urmă are nevoie de o cantitate mare de date pe care să fie antrenată în prealabil, în timp ce abordarea din prezent are nevoie doar de o poză.

#### 3.1 Lărgirea bazei de date

Baza de date utilizată pentru recunoaștere este relativ mică, fiind compusă din aproximativ 50 de persoane în procesul de testare. O bază de date mai largă, care să conțină cât mai multe persoane ar ajuta instituții care se ocupă cu securitatea și ordinea publică în îndeplinirea sarcinilor acestora. În prezent fiecare țară dispune de o bază de date ce conține poza cărții de identitate a fiecărei persoane născute și înregistrate împreună cu date despre aceasta. Includerea unei asemenea baze de date ar extinde considerabil aplicațiile acestei lucrări.

#### 3.2 Stocarea imaginilor

Imaginile din modelul prezentat sunt stocate în baza de date drept stringuri către locația imaginii de pe diskul utilizatorului. Implementarea curentă consideră că programul server este rulat pe un calculator cu un hard disk de 1TB sau mai mare, unde pozele sunt stocate. Utilizatorii rulează diferite instanțe ale clientului și primesc poza propriu zisă prin intermediul unui third party sau prin email având-o local. Atunci ei trebuie să adauge poza manual în instanța lor de client. Acest proces este sigur întrucât aplicațiile third party ( whatsapp, gmail, yahoo, facebook messenger ) beneficiază de un nivel de securitate puternic, iar clientul nu comunică cu serverul decât transmițând stringuri care conțin calea relativă din calculatorul pe care se află serve-

rul. În viitor se poate adăuga un nivel de securitate mai puternic prin transmiterea acestor date utilizând un algoritm de criptare.

### 3.3 Recunoașterea video

Recunoașterea video are la bază următoarea implementare: mai întâi se găsesc zonele din cadrul curent care conțin fețe de persoane și sunt stocate drept encodinguri. Aceste encodinguri sunt apoi comparate cu o funcție predefinită care le calculează similaritatea cu encodingurile fiecărei poze a fiecărei persoane din baza de date. O modalitate de a îmbunătăți această metodă constă într-o detectare mai bună a zonei care reprezintă fața unei persoane în materialul video. Obiectele care se mișcă pot fi de asemenea detectate drept o altă capacitate a aplicației, însă detectarea unei persoane care nu se uita la cameră, sau se uită dar cu capul într-o parte ar ajuta la o recunoaștere mai riguroasă.

### 3.4 Cautare în lista de persoane

Pentru ușurința de lucru cu o bază de date cu un număr mare de persoane, de peste 100, se poate implementa o bară de căutare unde persoanele să fie căutate după nume. Acest lucru este momentan posibil în baza de date întrucât aplicația marchează în procesul de recunoaștere persoanele cu numele luat din baza de date, însă pentru utilizatorii fără experiență o bară de căutare în care să fie introdus numele găsit de program care să filtreze așadar lista de suspecti ar fi de folos. O proprietate de "onClick" pentru orice persoană din listă care să încarce profilul lor pe ecran, utilizând același template ca de la ecranul "addAPerson" poate fi adăugată pentru vizualizarea altor attribute legate de persoana găsită, înafară de nume.

# Bibliografie

- [1] ageitgey. facial recognition library, 2022. Read here: [https://github.com/ageitgey/face\\_recognition](https://github.com/ageitgey/face_recognition).
- [2] The QT Company. QT designer, 2023. Read here: <https://doc.qt.io/qt-6/qtdesigner-manual.html>.
- [3] Python Software Foundation. pyqt5, 2023. Read here: <https://pypi.org/project/PyQt5/>.
- [4] R. Thomas G. J. Awcock. Applied image processing, 1995. Read here: <https://link.springer.com/book/10.1007/978-1-349-13049-8>.
- [5] OpenCV. OpenCV computer vision library, 2023. Read here: [https://docs.opencv.org/3.4/d6/d00/tutorial\\_py\\_root.html](https://docs.opencv.org/3.4/d6/d00/tutorial_py_root.html).
- [6] Pallet Projects. Flask API, 2010. Read here: <https://flask.palletsprojects.com/en/2.3.x/>.
- [7] Himanshu Singh. For facial recognition, object detection, and pattern recognition using python, 2019. Read here: <https://link.springer.com/book/10.1007/978-1-4842-4149-3>.
- [8] SQLAlchemy. SQLAlchemy, 2023. Read here: <https://docs.sqlalchemy.org/en/20/>.
- [9] Gloria Bueno García Oscar Deniz Suarez. Learning image processing with opencv, 2013. Read here: <http://103.62.146.201:8081/jspui/handle/1/5411>.
- [10] Gloria Bueno García Oscar Deniz Suarez. Learning image processing with opencv, 2013. Read here: <http://103.62.146.201:8081/jspui/handle/1/5411>.