



Universitatea
Transilvania
din Brașov
FACULTATEA DE MATEMATICĂ
ȘI INFORMATICĂ

Lucrare de Licență

Real Time Monitoring Streaming Tool

Autor: Vlad-Mihail ȚACĂ

Coordonator: Lector Univ. Dr. Vlad MONESCU

Brașov, România
2021-2022

Cuprins

Scopul lucrării	3
Structura lucrării	4
1 Introducere	5
1.1 Alegerea temei de proiect	6
1.2 Soluții existente	6
1.2.1 Hik-Connect	6
1.2.2 Manything	7
1.2.3 Alfred Camera	7
1.3 Structura aplicației	8
2 Tehnologii utilizate	10
2.1 Java	10
2.2 Android	11
2.2.1 Arhitectura Android	12
2.3 Android Studio	13
2.4 Biblioteca <i>rtmp-rtsp-stream-client-java</i>	15
2.5 Ant Media Server	15
2.5.1 Arhitectura Serverului Ant Media	16
2.6 Python	17
2.7 Jupyter Notebook	17
2.8 OpenCV	18
2.8.1 Modulul DNN în OpenCV	18
2.9 Caffe	19
2.10 Modulul Python subprocess	19
2.11 FFmpeg	20
2.11.1 Procesare video	20
2.11.2 Compresie video	20
2.11.3 Suport pentru ambalarea video	21
2.11.4 Suport pentru containere audio și video	21
3 Noțiuni teoretice	22
3.1 Arhitectura MVVM în Android	22
3.2 Streaming-ul RTMP	23
3.2.1 RTMP	24
3.2.2 Funcționalitatea streaming-ului RTMP	24
3.2.3 Fluxul de lucru tipic livestream-ului RTMP	24
3.2.4 Advanced Audio Coding (AAC)	25
3.2.5 H.264	26

3.3	HTTP Live Streaming	26
3.4	Rețelele neurale convoluționale	28
4	Implementare	30
4.1	Aplicația Android	30
4.1.1	Versiunea de Android aleasă	30
4.1.2	Mediul de rulare	30
4.1.3	Debugging	30
4.1.4	Organizarea inițială a aplicației	31
4.1.5	Biblioteci	31
4.1.6	Activitatea principală	31
4.1.6.1	Organizarea UI	31
4.1.6.2	Metodele utilizate	33
4.1.6.3	Implementarea metodelor în cod	34
4.1.6.4	Gestionarea erorilor de conexiune	36
4.1.6.5	Testarea aplicației	36
4.2	Server-ul Media	38
4.2.1	Configurarea server-ului media	38
4.2.2	Configurarea server-ului pe computer-ul personal	39
4.2.2.1	Configurarea pe Windows Subsystem for Linux	39
4.2.2.2	Configurarea pe Mașina Virtuală	39
4.2.3	Accesarea panoului web Ant Media	40
4.2.4	Playeri media	42
4.3	Componenta de Procesare a Stream-ului	43
4.3.1	Organizarea inițială a Componentei de Procesare a Stream-ului	43
4.3.2	Mediul de rulare	43
4.3.3	Alegerea rețelelor neurale	43
4.3.3.1	Arhitectura rețelelor	43
4.3.4	Alegerea librăriei OpenCV-Python	44
4.3.5	Utilizarea librăriei OpenCV-Python	44
4.3.6	Logica modificării frame-urilor	46
4.3.7	Evidențierea pe frame	46
4.3.8	Transmiterea frame-urilor pe server-ul media	47
4.3.9	Finalitatea Componentei de Procesare a Stream-ului	48
5	Concluzii	50
6	Perspective de dezvoltare	51
	Bibliografie	54
	Cărți	54
	Articole	54
	Online	54

Scopul lucrării

„Real Time Monitoring Streaming Tool” are ca scop monitorizarea unui mediu/loc prin intermediul telefonului mobil. Utilizatorul se folosește de camera și microfonul telefonului pentru a captura data video și audio astfel încât aceasta să poată fi accesată și modificată ulterior.

Partajarea va fi realizată în așa fel încât utilizatorul să poată vedea cu ușurință ceea ce transmite telefonul de la distanță. Totodată, va fi creată o nouă partajare, accesibilă utilizatorului, prin care vor fi evidențiate prezența fețelor din spațiul cuprins de camera video și probabilitatea trăsăturilor majore ale acestora (vârsta și genul).

Astfel, proiectul conține trei componente principale:

- **Aplicația Android**

Implementează pe de-o parte posibilitatea de a înregistra și salva în telefon date provenite de la cameră și microfon, iar pe de altă parte posibilitatea de a coda în format AAC (pentru audio) și H.264 (pentru video) datele obținute de la microfon și cameră pentru a putea fi transmise unui server RTMP. Mai multe detalii despre această componentă pot fi consultate în Secțiunea 4.1.

- **Server-ul Media**

Server-ul media ales este Server-ul Ant Media descris în Secțiunea 2.5. Modul de utilizare al acestui server în aplicație este prezentat în Secțiunea 4.2.

- **Componenta de Procesare a Stream-ului**

Aceasta transformă output-ul celorlalte două componente în așa fel încât să prezinte detectarea feței, a vârstei aproximative și a genului persoanelor din mediul monitorizat. Mai multe detalii despre modul de implementare și funcționare pot fi consultate în Secțiunea 4.3.

Structura lucrării

Lucrarea de față este structurată pe șase capitole.

- Capitolul 1, intitulat **Introducere**, motivează alegerea temei de proiect, prezintă punctele tari și punctele slabe ale altor trei alte aplicații asemănătoare și oferă o schematizare a structurii aplicației personale.
- Capitolul 2, intitulat **Tehnologii utilizate**, amintește principalele tehnologii utilizate în dezvoltarea aplicației de monitorizare, enumerând limbajele de programare (Java, Python), principalele biblioteci (OpenCv, Caffe) și mediile de dezvoltare (Android Studio, Ant Media Server).
- Capitolul 3, intitulat **Noțiuni teoretice**, prezintă succint principalele noțiuni teoretice utilizate în dezvoltarea aplicației, evidențiind arhitectura MVVM, Streaming-ul RTMP și rețelele neurale convoluționale.
- Capitolul 4, intitulat **Implementare**, enumeră principalii pași urmați în implementarea aplicației Android și modul în care aceasta funcționează și interacționează cu utilizatorul.
- Capitolul 5, intitulat **Concluzii**, prezintă pe scurt concluziile lucrării.
- Capitolul 6, intitulat **Perspective de dezvoltare**, oferă posibile direcții de dezvoltare ale aplicației.

Capitolul 1: Introducere

În acest capitol oferim informații cu privire la conceptul de monitorizare, securitate, precum și motivația alegerii temei de proiect.

Monitorizarea reprezintă acțiunea de a supraveghea prin intermediul unui aparat specializat.

Securitatea este una dintre nevoile noastre de bază, evolutive. Majoritatea deciziilor și acțiunilor noastre se bazează pe menținerea sau îmbunătățirea circumstanțelor în care ne aflăm. Deși s-ar putea să nu fim în pericol constant ca în cazul omului primitiv, nevoia de siguranță pentru noi și cei dragi rămâne una prioritară.

Aceasta a fost recunoscută ca o nevoie umană de bază de către Abraham Maslow în „*Hierarchy of Needs: A Theory of Human Motivation*” ([1]). Nevoile de siguranță reprezintă al doilea nivel în ierarhia lui Maslow, cuprinzând securitatea corpului, a locului de muncă, a familiei, a sănătății, precum și confortul psihic necesar funcționării eficiente.



Figura 1.1: Piramida lui Maslow

Această nevoie este transferată și în comportamentul consumatorilor. Prin urmare, produsele și serviciile oferite acestora se folosesc de imagini, text sau orice alte elemente ce sugerează sau garantează siguranța, încurajând utilizatorii să aibă încredere în produsul respectiv și, ca rezultat direct, să finalizeze achiziția acestuia.

Proiectul intitulat „*Real Time Monitoring Streaming Tool*” are drept scop implementarea unui sistem de monitorizare ce realizează transmițeri live ușor accesibile utilizatorului, folosindu-se de camera telefonului mobil, transmițeri care, în urma publicării acestora pe server, pot fi modificate să includă outputul rețelelor neurale.

1.1 Alegerea temei de proiect

Ideea proiectului a luat naștere în urma a două aspecte, anume nevoia de a ști ce se întâmplă cu animalul de companie lăsat singur în momentul în care trebuie să lipsesc de acasă pentru o treime din zi, precum și dorința de a găsi o utilitate telefonului meu vechi, încă funcțional.

Totodată, având în vedere situația actuală a spargerilor și furturilor - „în anul 2021 au fost înregistrate 15.956 de furturi din locuințe” ([2]), un instrument de supraveghere al casei sau mașinii ar ajuta enorm, putând contribui atât la recuperarea bunurilor, cât și la detectarea celor ce comit infracțiunea.

Astfel spuse, curiozitatea funcționalității kiturilor de securitate disponibile în comerț, nevoia personală de siguranță și securitate, dorința de a “recicla” un telefon vechi, precum și dorul pentru animalul de companie au dat naștere temei proiectului de licență.

1.2 Soluții existente

În cele ce urmează prezentăm trei aplicații care au ca scop monitorizarea prin intermediul telefonului.

1.2.1 Hik-Connect

Hik-Connect (Figura 1.2) este o aplicație concepută pentru a funcționa cu camerele marcă Hikvision și alte echipamente de securitate, modul în care aplicația este utilizată depinzând de tipul de hardware pe care îl are clientul. Presupunând că acesta deține Hikvision acasă sau la birou, această aplicație excelează în ceea ce privește supravegherea și monitorizarea.



Figura 1.2: Aplicația Hik-Connect

Când se declanșează o alarmă pe sistemul de securitate, aplicația anunță user-ul instantaneu pentru a lua măsurile necesare.

Pentru un nivel suplimentar de securitate pe smartphone, aplicația permite conectarea prin intermediul amprentei digitale.

Deși Hikvision oferă servicii de stocare în cloud pentru întreprinderi, aplicația nu permite configurarea sau modificarea acestor setări.

Pentru mai multe detalii a se vedea [3].

1.2.2 Manything

Manything (Figura 1.3) este o aplicație gratuită care permite transformarea oricărui dispozitiv cu Android 4.2 + sau iOS 8.0 + într-o cameră de securitate. Video-clipurile pot fi redade din aplicație direct pe smartphone-ul sau tableta principală, aplicația alertând userul când este detectată mișcarea.



Figura 1.3: Aplicația Manything

O caracteristică utilă este capacitatea de a vorbi prin intermediul dispozitivului care funcționează drept cameră.

Stocarea în cloud și utilizarea mai multor dispozitive tip camere sunt plătite suplimentar.

Pentru mai multe detalii a se vedea [4].

1.2.3 Alfred Camera

Alfred Camera (Figura 1.4) este cea mai populară aplicație de tip cameră de securitate din lume pentru securitatea casei.

Alfred Camera permite oricui să transforme orice smartphone sau tabletă într-un sistem portabil de alarmă antiefracție prin intermediul detectorului de mișcare.



Figura 1.4: Aplicația Alfred Camera

Aplicația folosește camera dispozitivului ca senzor de mișcare pentru a capta videoclipuri în timp real ale intrușilor și orice activitate suspectă, alertând userul.

Videoclipurile înregistrate automat sunt păstrate în spațiu de stocare gratuit și nelimitat în cloud.

Pentru mai multe detalii a se vedea [5].

1.3 Structura aplicației

Aplicația Android a fost gândită să implementeze două mari elemente:

- Posibilitatea de a înregistra și de a salva în telefon capturile realizate de cameră și microfon;
- Posibilitatea de a coda (encoding) în format AAC (pentru audio) și H.264 (pentru video) captura celor două componente, pentru a fi transmisă mai departe unui server RTMP.

Structura aplicației este schematizată în Figura 1.5 și conține următoarele componente:

- **Server-ul media**

Are ca scop stocarea și partajarea conținuturilor media, atât cel transmis prin RTMP de către Aplicația Android, cât și cel modificat de componenta de procesare a stream-ului.

- **Componenta de Procesare a stream-ului**

Se folosește de stream-ul inițial de pe server pentru a adauga outputul inferențelor a trei rețele neurale (Detectarea Feței, Detectarea Vârstei, Detectarea Genului) și a transmite un alt stream în timp real serverului media cu frame-urile modificate.

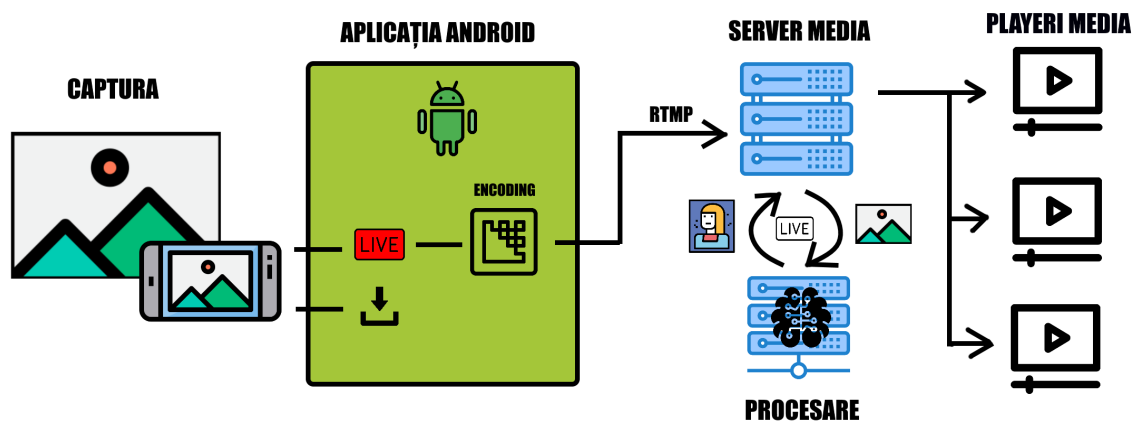


Figura 1.5: Real Time Monitoring Streaming Tool

Capitolul 2: Tehnologii utilizate

În acest capitol vor fi prezentate tehnologiile utilizate în dezvoltarea aplicației practice.

- Secțiunea 2.1 descrie particularitățile limbajului de programare Java.
- Secțiunea 2.2 prezintă sistemul de operare Android și structura acestuia.
- Secțiunea 2.3 vorbește despre Android Studio, mediul de dezvoltare al aplicațiilor Android.
- Secțiunea 2.4 descrie componentele bibliotecii Android utilizate pentru transmiterea de flux audio și video către servere media.
- Secțiunea 2.5 prezintă server-ul utilizat în aplicație pentru transmiterea datelor video și audio.
- Secțiunea 2.6 descrie particularitățile limbajului de programare Python.
- Secțiunea 2.7 prezintă mediul Jupyter Notebook utilizat pentru rularea programelor Python.
- Secțiunea 2.8 vorbește despre biblioteca OpenCV utilizată în Computer Vision, procesarea imaginilor și învățarea automată.
- Secțiunea 2.9 prezintă un framework de Deep Learning numit Caffe.
- Secțiunea 2.10 descrie modulul Python subprocess folosit pentru crearea de procese copil.
- Secțiunea 2.11 prezintă un proiect software pentru crearea de biblioteci și programe pentru manipularea datelor video și audio.

2.1 Java

Motto-ul Java este „scrie o singură dată, rulează oriunde” [6]. Codul Java poate fi găsit aproape oriunde. Este folosit în dezvoltarea de aplicații Android, dezvoltarea web, dezvoltarea jocurilor, dispozitivele inteligente și multe altele.

Java a fost dezvoltat de un grup de developeri de la Sun Microsystems în 1995, fiind acum deținut de Oracle. Unul dintre punctele forte ale Java este reprezentat de faptul că, odată scris codul, acesta poate fi rulat oriunde, nu doar pe fiecare sistem de operare, ci pe orice tip de hardware.

Dintre caracteristicile Java amintim următoarele:

- **Simplitatea**

Java a fost conceput pentru a fi ușor de utilizat, de scris, de compilat, de depanat și de învățat. Java se folosește de alocarea automată a memoriei și de garbage collector.

- **Orientat pe obiecte**

Programarea orientată pe obiecte este legată de concepte precum clasă, obiect, moștenire, încapsulare, abstractizare, polimorfism. Astfel, Java permite construirea de programe modulare și cod reutilizabil.

- **Independent de platformă**

Java oferă confortul de a scrie programe o singură dată, programe ce rulează pe orice platformă hardware și software și orice browser compatibil cu Java. Acest lucru oferă puterea codului de a fi manevrat cu ușurință de la un sistem de calcul la altul.

- **Securitate**

Java este primul limbaj de programare care a încorporat securitatea ca o parte integrală a sa. Compilatorul, interpretorul și mediul de rulare Java au fost fiecare dezvoltate cu securitatea în plan. Java Virtual Machine conține un identificator unic care recunoaște bytecode-ul și îl verifică înainte de al rula.

- **Alocare**

Java are caracteristica sistemului de alocare de tip Stack, urmărind structura LIFO (Last in First Out), ceea ce ajută informațiile să fie stocate și preluate cu ușurință.

- **Multithreaded**

Java este unul dintre limbajele de programare care acceptă Multithreading. Multithreading este capacitatea unui program de a efectua mai multe sarcini simultan.

- **API-uri bogate**

Java oferă o multitudine de API-uri pentru dezvoltarea aplicațiilor. API-urile Java (Application Programming Interface) reprezintă un set de comenzi sau metode de comunicare între activități, cum ar fi conexiunea la baze de date, rețea, I/O, XML parsing, utilități și multe altele.

- **Instrumente de dezvoltare Open source puternice**

Multe instrumente de dezvoltare open source, adică IDE-uri precum Eclipse și Netbeans, au la bază Java. IDE-urile (integrated development environment) simplifică dezvoltarea aplicațiilor prin intermediul caracteristicilor lor impresionante de codare și depanare.

Având caracteristici independente de platformă, Java este utilizat în Android.

2.2 Android

Android este un sistem de operare bazat pe Linux, proiectat în principal pentru dispozitive mobile cu ecran tactil, cum ar fi smartphone-uri și tablete.

Android este un sistem de operare puternic ce acceptă un număr mare de aplicații pe smartphone-uri. Aceste aplicații sunt concepute pentru confortul utilizatorilor.

Hardware-ul ce acceptă software-ul Android se bazează pe arhitectura ARM. Androidul este un sistem de operare open-source, însemnând că poate fi folosit de oricine, gratuit.

Android are milioane de aplicații disponibile ce ajută la gestionarea vieții de zi cu zi. Dezvoltarea Android favorizează limbajul de programare Java. Prima versiune 1.0 a kitului de dezvoltare Android (SDK) a fost lansată în 2008.

2.2.1 Arhitectura Android

Arhitectura Android este reprezentată de cinci componente de bază, pe care le amintim în continuare.

1. **Kernel Linux**

Androidul folosește puternicul kernel Linux și acceptă o gamă largă de drivere hardware. Kernel-ul este inima sistemului de operare care gestionează cererile de intrare și ieșire din software. Aceasta oferă funcționalități de bază ale sistemului, cum ar fi managementul proceselor, managementul memoriei și managementul dispozitivelor (camera, tastatura, afișajul). Kernel-ul în sine nu interacționează direct cu utilizatorul, ci mai degrabă interacționează cu shell-ul și alte programe, precum și cu dispozitivele hardware de pe sistem.

2. **Biblioteci**

Deasupra kernel-ului există un set de biblioteci implementate. Dintre acestea enumerăm biblioteci care includ browsere web open-source, cum ar fi *WebKit*, *library libc*, biblioteci folosite pentru redarea și înregistrarea audio și video, biblioteci pentru stocarea și partajarea datelor aplicației (spre exemplu *SQLite*), și bibliotecile SSL responsabile pentru securitatea internetului.

3. **Android Runtime**

Runtime-ul Android furnizează o componentă cheie numită *Dalvik Virtual Machine*, care este un fel de mașină virtuală Java. Este special conceput și optimizat pentru Android. Dalvik VM este mașina virtuală de proces din sistemul de operare Android, fiind un software care rulează aplicațiile pe dispozitivul Android. Dalvik VM folosește caracteristicile de bază Linux, cum ar fi gestionarea memoriei și multithreading, regăsite în limbajul Java. Dalvik VM permite fiecărei aplicații Android să ruleze propriul proces. De asemenea, acesta execută fișierele în format *.dex*.

4. **Framework-ul Aplicațiilor**

Framework-ul prevede servicii de nivel înalt aplicațiilor precum managerul de ferestre (eng. *windows manager*), sistemul de vizualizare, managerul de pachete și managerul de resurse. Dezvoltatorii de aplicații au acces la aceste servicii, putând fi utilizate în aplicațiile lor.

5. **Aplicații**

Toate aplicațiile Android se regăsesc în stratul superior al arhitecturii. Exemple de astfel de aplicații sunt contactele, cărțile, browserele, serviciile etc. Fiecare aplicație îndeplinește un rol diferit în contextul general al sistemului Android.

În Figura 2.1 este ilustrată arhitectura Android cu exemple reprezentative pentru fiecare componentă în parte.

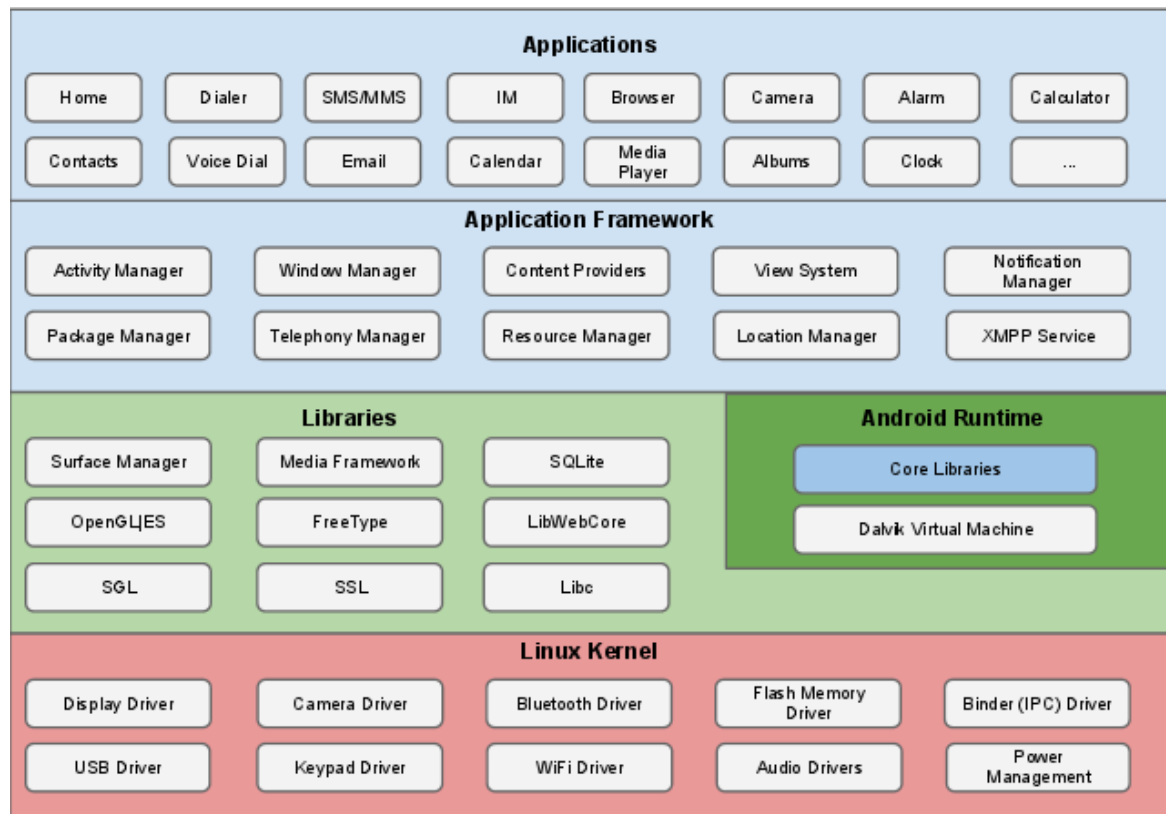


Figura 2.1: Diagrama Arhitecturii Android

2.3 Android Studio

Android Studio este un mediu de dezvoltare sofisticat, conceput pentru a dezvolta și testa aplicații Android. Acesta conține o colecție de instrumente și componente instalate și actualizate independent unele de altele.

Android Studio nu este singura modalitate de a crea aplicații Android; există și alte IDE-uri, cum ar fi *Eclipse* și *NetBeans*, fiind chiar posibil să fie dezvoltată o aplicație completă folosind doar *Notepad* și linia de comandă.

Există multe moduri în care Android Studio diferă de alte IDE-uri și instrumente de dezvoltare. Unele dintre aceste diferențe sunt destul de subtile, cum ar fi modul în care sunt instalate bibliotecile, dar altele, de exemplu procesul de construire al codului și designul UI (User Interface), sunt substanțial diferite.

Android Studio a atras un număr mare de plugin-uri third-party care oferă o gamă largă de funcții valoroase, indisponibile direct prin IDE. Acestea includ plugin-uri pentru accelerarea timpului de build al codului, depanarea unui proiect prin Wi-Fi și

multe altele.

Dintre caracteristicile Android Studio amintim:

- **Dezvoltarea UI**

Cea mai semnificativă diferență între Android Studio și alte IDE-uri este editorul său de aspect, care este cu mult superior oricărui competitor, oferind vizualizări ale textului, design și blueprint, și, cel mai important, instrumente de constrângere a aspectului pentru fiecare activitate sau fragment, cu un design funcțional *drag-and-drop*. Interfața descrisă anterior este prezentată în Figura 2.2.

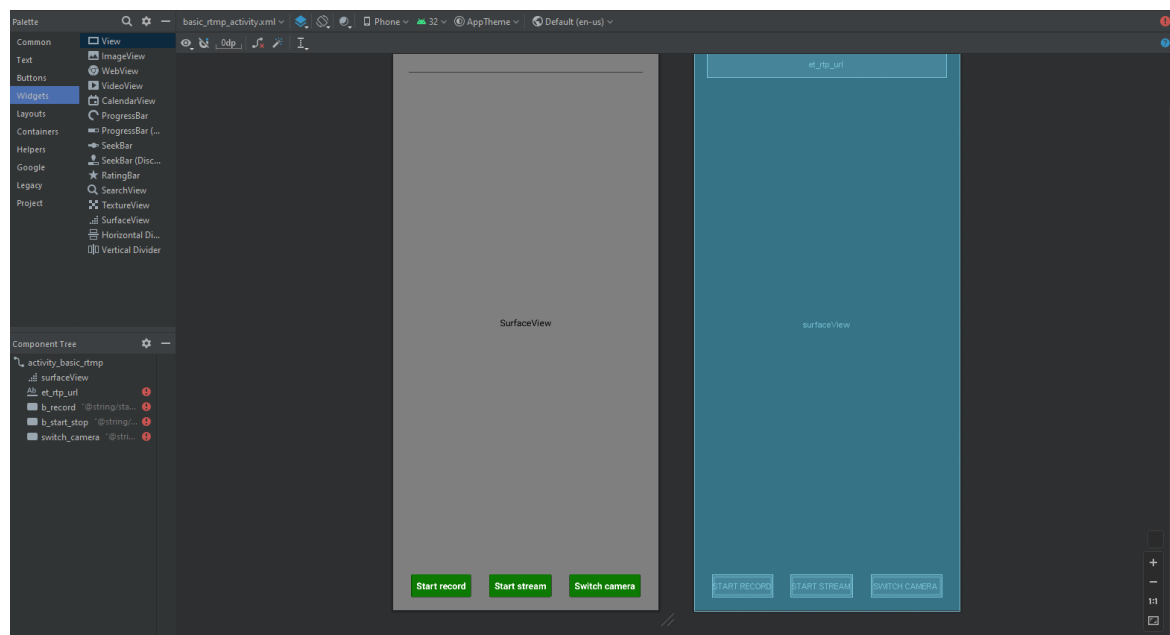


Figura 2.2: Interfața Android Studio

- **Structura Proiectelor**

Deși structura directorului de bază rămâne aceeași, modul în care Android Studio organizează fiecare proiect diferă considerabil de predecesorii săi. În loc să folosească spații de lucru (eng. *workspaces*) ca în Eclipse, Android Studio folosește module (eng. *modules*) în care se poate lucra mai ușor, fără a fi nevoie de a comuta între spațiile de lucru.

- **Emulatoare**

Android Studio este echipat cu un editor de dispozitive virtuale, permițând dezvoltatorilor să utilizeze emulatoare în locul unui dispozitiv Android fizic legat la calculator în depănarea codului. Emulatoarele sunt personalizabile, atât în ceea ce privește forma și dimensiunile, precum și în condițiile hardware ale acestora. Un exemplu de astfel de emulator este prezentat în Figura 2.3,

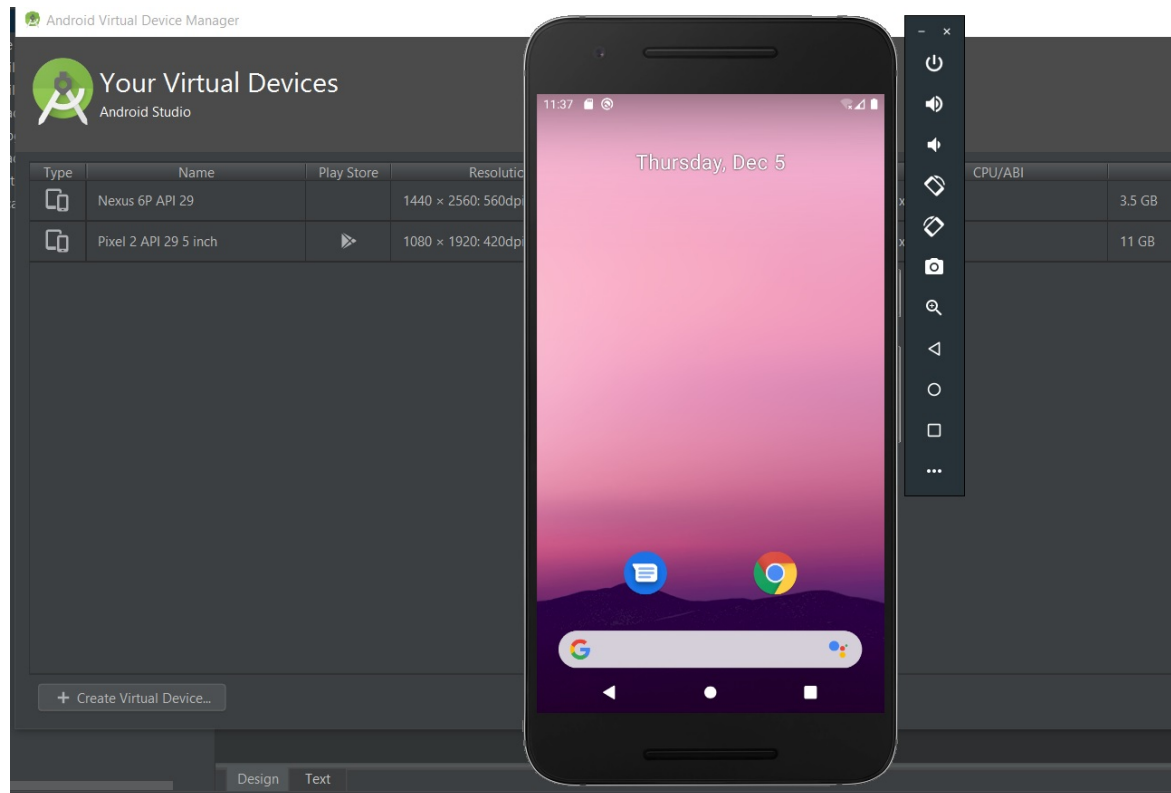


Figura 2.3: Emulator utilizat împreună cu Android Studio

2.4 Biblioteca *rtmp-rtsp-stream-client-java*

Rtmp-rtsp-stream-client-java ([7]) este o bibliotecă Android pentru transmiterea de flux audio și video către un server media în protocoale RTMP (Real-Time Messaging Protocol) sau RTSP (Real Time Streaming Protocol). Această bibliotecă primește date audio și video, le codifică în format AAC, respectiv H264 și le trimite modulului RTSP sau RTMP pentru a transmite fluxul de date.

În modulul RTSP se crează un pachet RTP (Real-time Transport Protocol) de video și audio care este apoi trimis către server. În modulul RTMP, se crează un pachet RTP de video și audio, se încapsulează în pachete tip FLV (Flash Video) și sunt trimise pe server.

Această bibliotecă folosește clasa *Android MediaCodec* pentru a face encodarea hardware.

2.5 Ant Media Server

Ant Media oferă soluții de streaming video în timp real gata de utilizare și extrem de scalabile.

Produsul principal al Ant Media, anume Ant Media Server, este o platformă de

video streaming care oferă soluții de streaming video de latență ultra scăzută (WebRTC) și latență scăzută (CMAF și HLS) extrem de scalabile, susținute cu utilități de management operaționale.

Ant Media Server oferă compatibilitate pentru orice browser web. În plus, SDK-urile pentru iOS, Android și JS sunt furnizate gratuit, lucru datorat funcției de streaming adaptivă, care permite redarea oricărui videoclip la orice lățime de bandă pe dispozitivele mobile.

2.5.1 Arhitectura Serverului Ant Media

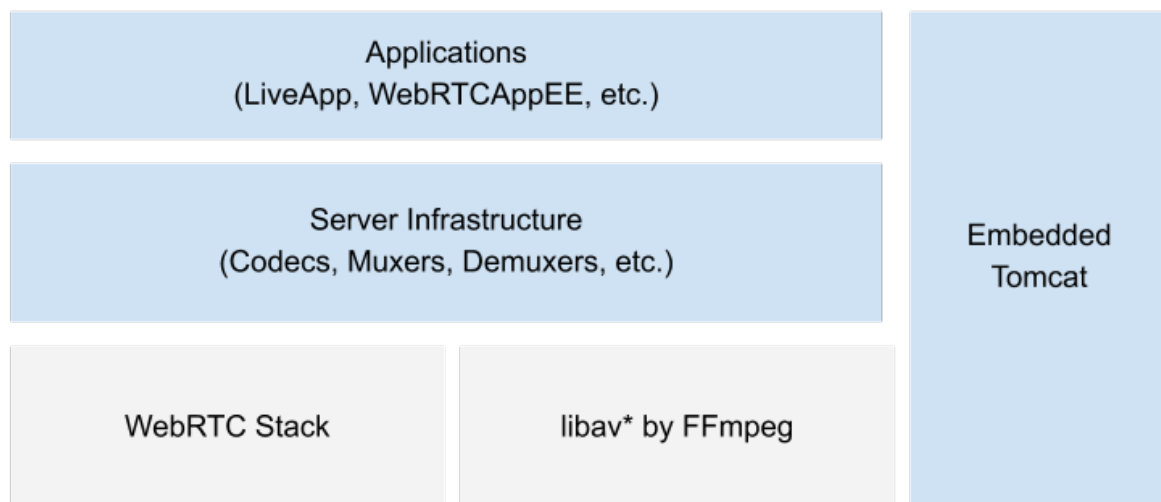


Figura 2.4: Diagrama Arhitecturii Server-ului Ant Media

Ant Media Server se află în mijlocul infrastructurii de streaming pentru procesarea și distribuția conținutului video către public, având potențialul de a gestiona sute de mii de spectatori concurenți fără a fi compromisă performanța.

Limbaajul de programare de bază al Ant Media este Java, codul fiind actualizat constant la cele mai noi versiuni pentru compatibilitate și optimizarea performanței. Independența platformei, portabilitatea, capabilitățile multi-threaded și optimizarea adaptivă se numără printre principalele motive pentru care Java este limbaajul de programare perfect pentru infrastructura serverului Ant Media.

Fiind o aplicație Java, Ant Media are nevoie de Apache Tomcat pentru a rula și a servi fluxurile de trafic HTTP. Serverul Ant Media acceptă trafic HTTP prin portul TCP 5080 (pentru trafic HTTP necriptat) și portul TCP 5443 (pentru trafic HTTPS criptat).

Ant Media Server rulează pe Linux (Ubuntu și CentOS) și acceptă doar arhitectura x64. Scripturile de implementare sunt furnizate pentru Ubuntu (începând cu 18.04) și CentOS 8. În plus, Ant Media Server poate fi folosit și pe distribuțiile MacOS, SuSE, Debian și Red Hat Enterprise Linux.

Arhitectura server-ului Ant Media este ilustrată în Figura 2.4.

2.6 Python

Python este un limbaj de programare la nivel înalt, ușor de interpretat, orientat pe obiecte, cu o sintaxă ușor de citit. Ideal pentru prototipuri și sarcini ad-hoc, Python are o utilizare largă în calculul științific, dezvoltarea web și automatizare. Fiind un limbaj de programare de uz general, prietenos pentru începători, Python sprijină mulți oameni de știință și dezvoltatori de aplicații la nivel global.

Python excelează atunci când este vorba o sarcină complexă care trebuie simplificată, un script scurt de rulat sau un set de date mare care trebuie manipulat. Este, de asemenea, o unealtă folosită de oamenii de știință și matematicieni, având o serie de biblioteci interne ideale pentru statistică și matematică complexă.

Limbajul de programare Python este utilizat activ în toate domeniile informaticii contemporane. Deoarece dezvoltarea Python este mai eficientă decât majoritatea celorlalte limbi, este o alegere populară pentru startup-uri, unde modificările la baza de cod trebuie făcute rapid, cu un cost cât mai redus.

Principalul punct forte al lui Python este în Data Science și Machine Learning. Codul Python poate fi creat utilizând atât IDE-uri specifice, precum *PyCharm*, *Eclipse*, *Jupyter Notebook*, *Spyder*, cât și într-un editor de text.

2.7 Jupyter Notebook

Jupyter Notebook (fost IPython Notebooks) este un mediu de calcul interactiv bazat pe web pentru crearea de documente notebook. Un document Jupyter Notebook este un browser-based REPL care conține o listă ordonată de celule de intrare/ieșire ce pot conține cod, text, matematică, diagrame sau conținut media. Sub interfață, un notebook este un document JSON, ce urmărește o schemă versionată, care se termină de obicei cu extensia *.ipynb*.

Aplicația Jupyter Notebook este o aplicație server-client care permite editarea și rularea documentelor de notebook printr-un browser web. Aplicația Jupyter Notebook poate fi executată pe un desktop local care nu necesită acces la internet sau poate fi instalată pe un server la distanță și accesată prin internet.

Pe lângă afișarea, editarea și rularea documentelor pentru notebook, aplicația Jupyter Notebook are un tablou de bord (eng. *Notebook Dashboard*) și un panou de control care arată fișierele locale și care permite deschiderea documentelor notebook sau închiderea kernel-urilor acestora.

2.8 OpenCV

OpenCV este o bibliotecă open-source specifică pentru Computer Vision, învățarea automată și procesarea imaginilor. OpenCV acceptă o mare varietate de limbaje de programare precum Python, C++, Java etc. Poate procesa imagini și videoclipuri pentru a identifica obiecte, fețe sau chiar scrisul de mână. Atunci când este integrat cu diferite biblioteci, cum ar fi *Numpy*, care este o bibliotecă foarte optimizată pentru operațiuni numerice, OpenCV oferă dezvoltatorilor o gamă largă de operații.

OpenCV-Python este o bibliotecă Python concepută pentru a rezolva problemele legate de Computer Vision. În comparație cu limbaje precum C/C++, Python este mai lent, însă poate fi extins cu ușurință prin C/C++, ceea ce permite scrierea de cod intensiv din punct de vedere computațional în C/C++, crearea de wrapper-uri Python care pot fi folosite drept module. Acest lucru ne oferă două avantaje:

- codul este la fel de rapid ca cel original scris în C/C++;
- este mai ușor de codat în Python decât în C/C++.

OpenCV-Python este un wrapper Python pentru implementarea originală OpenCV C++.

OpenCV-Python folosește Numpy, care este o bibliotecă foarte optimizată pentru operații numerice cu o sintaxă în stil *MATLAB*. Toate structurile matriceale OpenCV sunt convertite în și din matrice Numpy. Acest lucru facilitează, de asemenea, integrarea cu alte biblioteci care utilizează Numpy, cum ar fi *SciPy* și *Matplotlib*.

2.8.1 Modulul DNN în OpenCV

Modulul DNN (Deep Neural Network) din OpenCV care este responsabil pentru toate lucrurile legate de Deep Learning. A fost introdus în OpenCV versiunea 3, având de atunci o evoluție considerabilă. Acest modul permite utilizarea rețele neuronale pre-antrenate din cadre populare precum *TensorFlow*, *PyTorch* și utilizarea acestor modele direct în OpenCV (inferența se poate realiza folosind doar modulul OpenCV).

Folosind modulul DNN al OpenCV pentru inferență, codul final este mult mai compact și mai simplificat. Există cazuri în care utilizarea modulului DNN al OpenCV realizează inferența mai rapid pentru procesor. Pe lângă suportul pentru GPU-ul NVIDIA bazat pe CUDA, modulul DNN al OpenCV acceptă și GPU-uri Intel bazate pe OpenCL.

Acestea sunt cadrele suportate în prezent de modulul DNN:

- Caffe
- TensorFlow

- Torch
- Darknet
- ONNX

2.9 Caffe

Caffe este un framework de Deep Learning creat pentru viteză și modularitate. Este dezvoltat de Berkeley AI Research (BAIR) și de contributorii comunității. Yangqing Jia a creat proiectul în timpul doctoratului său la UC Berkeley ([8]). Este open source, scris în C++, cu interfață Python.

Caffe este utilizat în proiecte de cercetare academică, prototipuri de startup și chiar și aplicații industriale la scară largă în vision, speech și multimedia. Yahoo! a integrat de asemenea Caffe cu Apache Spark pentru a crea CaffeOnSpark, framework de Deep Learning distribuit.

2.10 Modulul Python subprocess

De fiecare data cand este utilizat un computer, userul interacționează cu programe. Un proces este abstractizarea de către sistemul de operare a unui program care rulează (Meniuri Start, bara de aplicații, interpreți de linie de comandă, editori de text, browsere).

Procesul care începe un alt proces este denumit **părinte**, iar noul proces este denumit **copil**. Procesele părinte și copil rulează în cea mai mare parte independent unul de celălalt. Uneori, copilul moștenește resurse sau contexte specifice de la părinte.

Informațiile despre procese sunt păstrate într-un tabel. Fiecare proces ține evidența părinților săi, ceea ce permite ca ierarhia proceselor să fie reprezentată ca un arbore, după cum se poate observa în Figura 2.5.

Când un proces a terminat de rulat, de obicei acesta se încheie. Fiecare proces la ieșire, ar trebui să returneze un număr întreg. Acest număr întreg este denumit cod de retur sau stare de ieșire. Zero este sinonim cu succes, în timp ce orice altă valoare este considerată un eșec. Pot fi folosite diferite numere întregi pentru a indica motivul pentru care un proces a eșuat.

Modulul Python subprocess este folosit pentru lansarea proceselor de tip copil (eng. *child processes*). Aceste procese pot fi orice: de la aplicații GUI la shell. Prefixul “sub” provine din relația părinte-copil a proceselor. Când este utilizat un subprocess, Python este părintele care creează un nou proces copil, cu specificațiile definite de către dezvoltator.

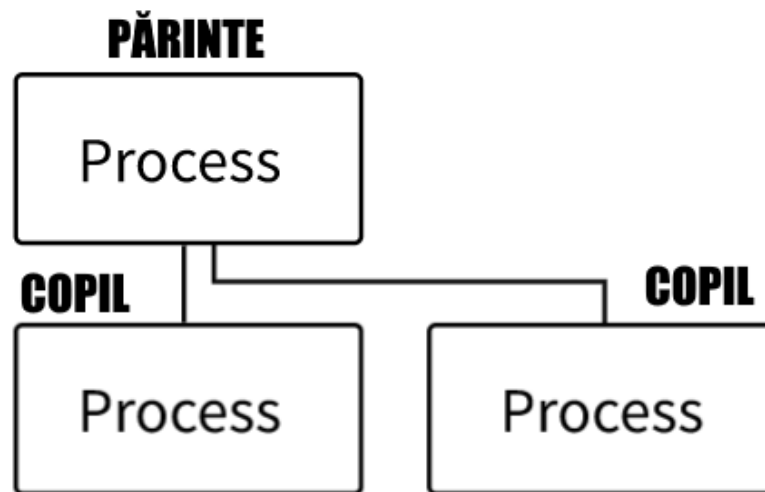


Figura 2.5: Ierarhizarea proceselor părinte-copil

Subprocesul Python a fost inițial propus și acceptat pentru Python 2.4 ca alternativă a utilizării modului `os`. O mare parte a interacțiunii cu modulul `subprocess` Python se realizează prin intermediul funcției `run()`. Această funcție de blocare va începe un proces și va aștepta până la terminarea acestuia înainte de a continua.

2.11 FFmpeg

FFmpeg este un proiect software gratuit care crează biblioteci și programe pentru manipularea datelor multimedia. FFmpeg poate gestiona întregul proces de transcodare, manipulare video și imagini (redimensionare, claritate etc.), ambalare, streaming și redare de conținut. Este cel mai popular software de procesare video și imagini din prezent ([9]).

2.11.1 Procesare video

FFmpeg poate fi folosit pentru diverse sarcini de procesare video, cum ar fi eliminarea zgomotului, estomparea, conversia culorilor, rotația, extragerea cadrelor sau letter-boxing, care sunt esențiale pentru majoritatea fluxurilor de lucru de procesare, compresie și livrare video.

2.11.2 Compresie video

Suportul FFmpeg pentru compresia video este fantastic și este un binecunoscut fapt că majoritatea companiilor de streaming folosesc sau au folosit FFmpeg pentru sistemele lor de producție.

FFmpeg conține biblioteci care oferă o interfață pentru diverse codec-uri pre-

cum JPEG, MPEG-1/2/4, H263+AAC (MPEG), Theora (Ogg Vorbis), AVS+, VP8 (WebM), H.264/AVC, HEVC, AV1, și multe altele ce pot fi folosite pentru a comprima, transcoda sau decoda videoclipuri după cum este necesar.

2.11.3 Suport pentru ambalarea video

Pentru furnizorii sau dezvoltatorii de sisteme OTT, FFmpeg oferă suport complet pentru ambalarea videoclipurilor (eng. *packaging*) atât în protocoalele HLS, cât și în protocoalele MPEG-DASH. De asemenea, poate fi configurat pentru a transmite videoclipuri folosind RTMP sau alte protocoale.

2.11.4 Suport pentru containere audio și video

FFmpeg are, de asemenea, suport extins pentru containere și poate fi folosit pentru a citi, a scrie și a converti containere precum *avi*, *mp4*, *mp3*, *wma*, *wav*, *ts*, *flv*, *mkv* și multe alte formate obscure.

Capitolul 3: Noțiuni teoretice

În acest capitol prezentăm principalele noțiuni teoretice utilizate în dezvoltarea aplicației.

- Secțiunea 3.1 descrie arhitectura MVVM și cele trei componente principale ale sale, care ajută la separarea clară dintre logica aplicației și interfața de utilizare.
- Secțiunea 3.2 prezintă streaming-ul RTMP evidențiind atât funcționalitatea și fluxul obișnuit de lucru al acestuia, cât și standardele de codare pentru compresia audio și video.
- Secțiunea 3.3 prezintă noțiunea de HTTP Live Streaming.
- Secțiunea 3.4 descrie rețelele neurale convoluționale și principalele straturi utilizate pentru o astfel de arhitectură.

3.1 Arhitectura MVVM în Android

Arhitectura MVVM are ca scop menținerea unei separări clare între logica aplicației și interfața de utilizare, ajutând la abordarea numeroaselor probleme de dezvoltare, aplicația fiind mai ușor de testat, întreținut și dezvoltat. De asemenea, arhitectura MVVM poate îmbunătăți considerabil reutilizabilitatea codului și permite dezvoltatorilor și designerilor UI să colaboreze mai ușor atunci când dezvoltă fiecare părți diferite ale unei aplicații.

Arhitectura MVVM conține trei componente principale, schematizate în Figura 3.1:

1. **Model**
Acesta deține datele și business logic-ul aplicației. Nu poate vorbi direct cu View.
2. **View**
Reprezintă interfața de utilizare a aplicației lipsită de orice logică de aplicație. Aceasta observă ViewModel.
3. **ViewModel**
Acționează ca o legătură între Model și View. Este responsabil pentru transformarea datelor din Model. Oferă fluxuri de date către View. De asemenea, utilizează hook-uri sau callback-uri pentru a actualiza vizualizarea. Va cere datele din Model.

Există două moduri principale de a implementa arhitectura MVVM în Android:

- **Data Binding**
În Android, Biblioteca Data Binding este o bibliotecă de tip support care permite legarea componentelor UI din aspecte la sursele de date din aplicație, folosind un format declarativ și nu programatic.

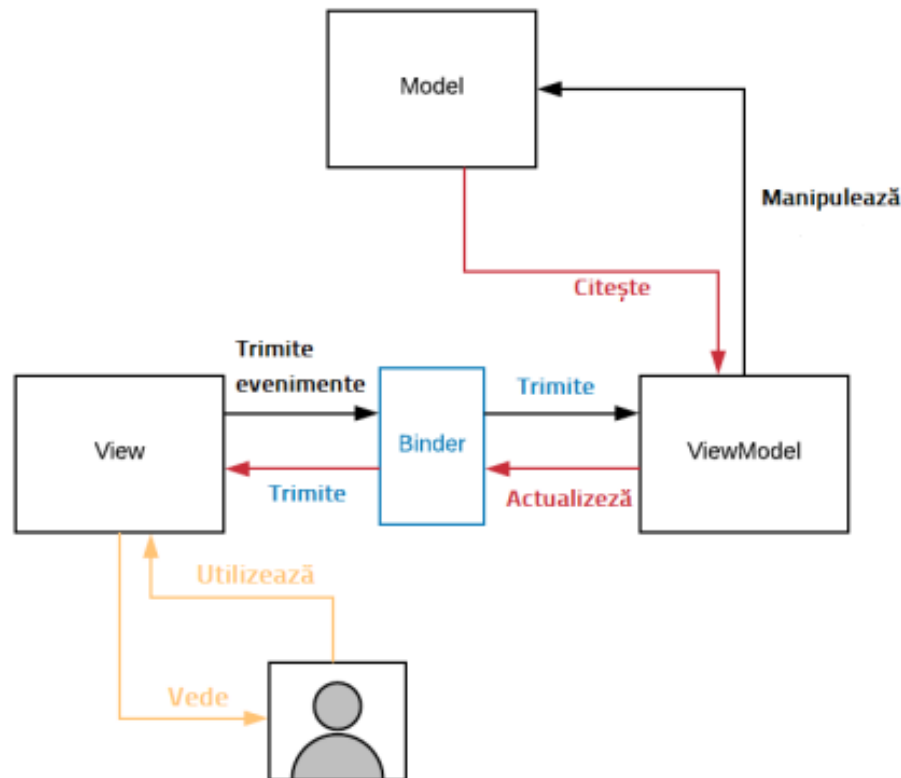


Figura 3.1: Diagrama Arhitecturii MVVM

- **RXJava**

Reactive Extensions (RX) sunt o colecție de metode și interfețe care ajută la economisirea timpului general de dezvoltare a unei aplicații mobile. Cu ajutorul extensiilor Reactive în Java, dezvoltatorii pot manipula acțiunile care sunt declanșate de evenimentele de sistem (eng. *system events*), permițând implementarea transformărilor funcționale peste fluxuri de evenimente.

3.2 Streaming-ul RTMP

În primele zile ale streaming-ului, protocolul de mesagerie în timp real (RTMP - Real-Time Messaging Protocol) a fost standard pentru transportul video prin internet (streaming). RTMP este un protocol bazat pe TCP (Transmission Control Protocol) conceput pentru a menține conexiuni persistente, cu latență scăzută și, prin extensie, experiențe de streaming fluide.

Protocolul a fost secretul din spatele streaming-ului live și on-demand din Adobe Flash Player. Deoarece acest plugin Flash popular a alimentat 98% din browserele de internet în perioada sa de glorie, RTMP a fost folosit în mod omniprezent ([10]).

Majoritatea codificatorilor (eng. *encoders*) de astăzi pot transmite RTMP, iar majoritatea serverelor media îl pot primi. Chiar rețelele sociale mari precum *Facebook*, *YouTube* și *Twitch* îl acceptă.

3.2.1 RTMP

Specificația RTMP este un protocol de streaming conceput inițial pentru transmiterea de date audio, video și alte date între un server de streaming dedicat și Adobe Flash Player. RTMP este acum disponibil tuturor.

Potrivit Adobe: „Protocolul de mesagerie în timp real (RTMP) de la Adobe oferă un serviciu de multiplexare a mesajelor bidirecționale printr-un transport de flux de încredere, cum ar fi TCP [RFC0793], destinat să transporte fluxuri paralele de mesaje video, audio și de date, cu informații de sincronizare asociate, între elementele unei perechi ce comunica unele cu altele” ([10]).

3.2.2 Funcționalitatea streaming-ului RTMP

Macromedia (Adobe Systems de astăzi) a dezvoltat specificația RTMP pentru transmisia de înaltă performanță a datelor audio și video. RTMP menține o conexiune constantă între clientul ce redă informația și server-ul, permițând protocolului să acționeze ca o conductă și să mute rapid datele video către vizualizator.

Deoarece RTMP se află deasupra protocolului de control al transmisiei (TCP), acesta folosește o relație în trei căi atunci când transportă date. Inițiatorul (clientul) îi cere acceptantului (server-ului) să înceapă o conexiune; cel care acceptă răspunde; apoi inițiatorul recunoaște răspunsul și menține o sesiune între ambele capete. Din acest motiv, RTMP este destul de fiabil.

Sfârșitul duratei de viață a Flash era de înțeles. Dar nu același lucru se poate spune despre RTMP. Codificatoarele RTMP sunt încă o alegere bună pentru mulți producători de conținut, după cum se poate observa în Figura 3.2 ([11]).

3.2.3 Fluxul de lucru tipic livestream-ului RTMP

Distribuitorii de conținut nu se limitează la un singur protocol de streaming de la captură până la redare. De fapt, reambalarea unui flux live în cât mai multe protocoale posibile ajută la asigurarea unei distribuții extinse.

Deoarece playerele HTML5 de astăzi necesită protocoale bazate pe HTTP, cum ar fi HLS (HTTP Live Streaming), un server media sau un serviciu de streaming poate fi folosit pentru a ingera un flux RTMP și a-l transcoda într-o alternativă mai prietenoasă cu redarea.

Cel mai comun flux de lucru în livestreaming este RTMP ([11]), după cum se poate observa și în Figura 3.3. Utilizarea unei combinații de RTMP și HLS ajută la maximizarea compatibilității fără a împinge latența prea mult. Această arhitectură permite, de asemenea, broadcaster-ilor să utilizeze protocoale bine susținute la fiecare

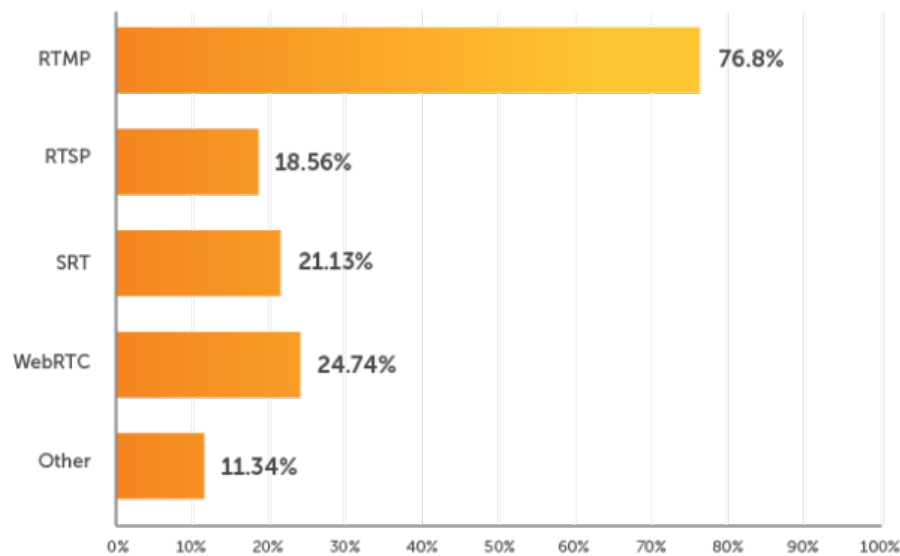


Figura 3.2: Cele mai utilizate formate de streaming ([11])

capăt al lanțului de livrare video în direct. Aproape toate codificatoarele software și hardware acceptă RTMP și același lucru se poate spune despre HLS din partea redării.

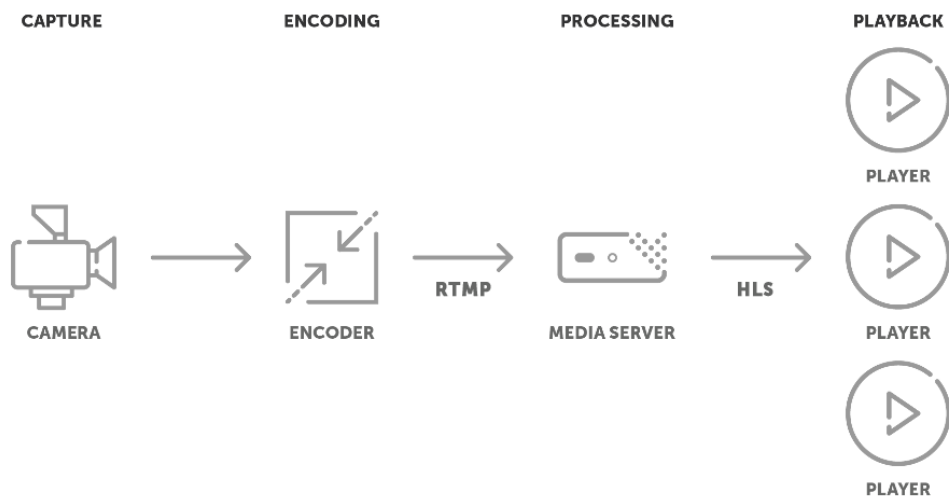


Figura 3.3: Fluxul de lucru tipic dintre RTMP și HLS ([11])

3.2.4 Advanced Audio Coding (AAC)

Advanced Audio Coding (AAC) este un standard de codare pentru compresia audio digitală cu pierderi. Conceput pentru a fi succesorul formatului MP3, AAC

oferă în general o calitate mai mare a sunetului decât codificatoarele MP3 la aceeași rată de biți (eng. *bitrate*).

3.2.5 H.264

H.264 este un standard de compresie video binecunoscut pentru videoul de înaltă definiție. Cunoscut și sub denumirea de MPEG-4 Part 10 sau Advanced Video Coding (MPEG-4 AVC), H.264 este definit ca un standard de compresie video orientat pe blocuri, bazat pe compensare, care definește mai multe profiluri (instrumente) și niveluri (rate și rezoluții maxime). Acest format suportă de la 4K până la 8K Ultra High-Definition.

3.3 HTTP Live Streaming

HTTP Live Streaming (HLS) este un protocol de streaming video care a fost lansat de Apple în 2009 odată cu lansarea iPhone 3.0. A fost creat în primul rând pentru a reduce dependența de Flash. Acest lucru se datorează faptului că HLS furnizează videoclipuri utilizând protocolul de transfer hipertext (HTTP) și, prin urmare, poate fi servit sau stocat în cache folosind orice server web obișnuit și poate fi redat de un player HTML5.

Este un protocol utilizat pe scară largă ([11]), în principal pentru că poate rula pe orice server web și poate fi redat pe orice dispozitiv. De asemenea, permite streaming adaptiv cu rata de biți, unde calitatea fluxului poate fi ajustată în funcție de condițiile de lățime de bandă în schimbare.

HLS este mult mai utilizat față de celelalte protocoale moderne de streaming, cum ar fi P2P (peer-to-peer), după cum se poate observa în Figura 3.4. De exemplu, streamingul P2P necesită o configurare specializată, deoarece dispozitivele utilizatorului final trebuie să mențină deschise porturile TCP (Transmission Control Protocol) sau UDP (User Datagram Protocol) prin NAT (Network address translation) și firewall-uri. Acesta nu este cazul cu HLS - este simplu și permite utilizatorilor să vizualizeze videoclipuri prin orice browser și orice dispozitiv.

HLS furnizează video folosind protocolul de transport TCP, care implică stabilirea unei conexiuni oficiale cu dispozitivul utilizatorului final înainte de a începe fluxul video. Acest lucru asigură faptul că niciun cadru video nu este pierdut în tranzit. Cu toate acestea, utilizarea TCP este unul dintre motivele pentru care videoul transmis prin HLS este mai lent, spre deosebire de protocolul de transport UDP, care este mai rapid, dar mai puțin sigur.

HLS este folosit atât pentru streaming video live, cât și pentru cel la cerere. În cazul live stream-urilor, videoclipurile sunt codificate în fluxuri în timp real și difuzate pe dispozitivele client. În fluxul la cerere, videoclipurile sunt stocate în formate pre-

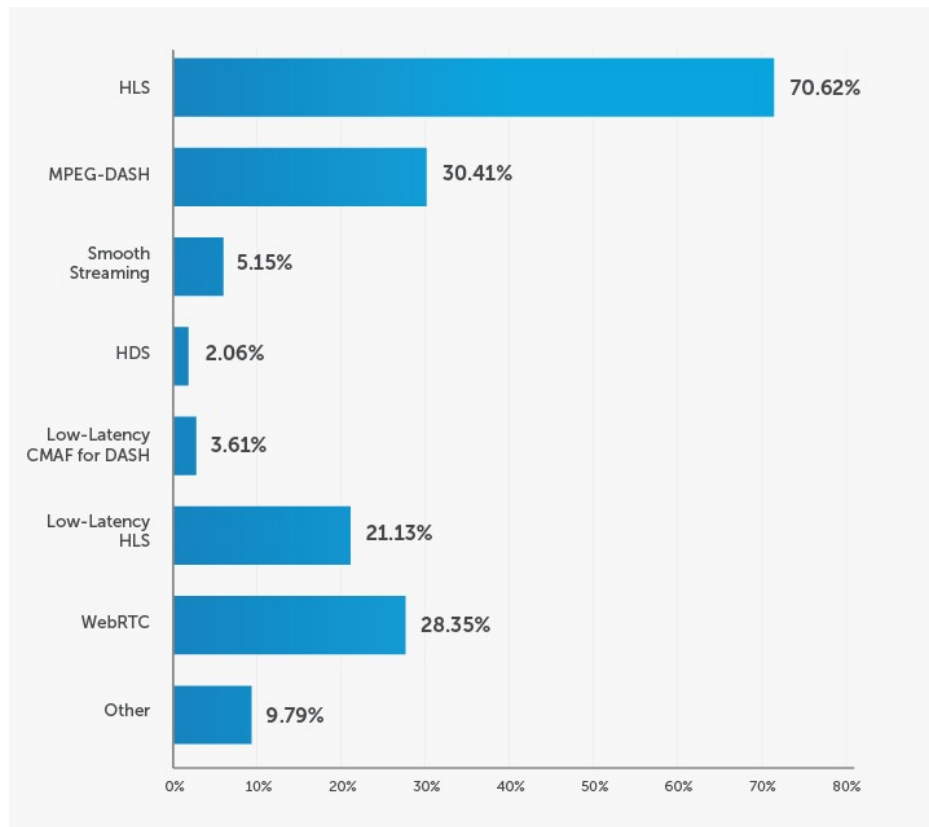


Figura 3.4: Cele mai utilizate de ingestare a stream-ului

codate și livrate printr-un server atunci când un dispozitiv client inițiază o solicitare de vizualizare.

HTTP Live Streaming poate fi asociat cu împărțirea videoclipurilor în bucăți mai mici. Fiecare dintre aceste bucăți de dimensiune mică poate fi apoi solicitată de către dispozitivul utilizatorului final de la server:

- Videoclipurile sunt mai întâi codificate într-un codec pe care dispozitivele utilizatorului final îl pot interpreta – H.264 pentru video și AAC sau MP3 pentru audio în cazul HLS.
- Videoclipurile sunt convertite în fluxuri duplicate de diferite calități (240p, 360p, 480p, 720p etc.).
- Videoclipurile sunt apoi segmentate în porțiuni mici de 10 secunde sau mai puțin.
- Este creat un fișier index (fișier *.M3U8* sau fișier manifest) care permite dispozitivului utilizatorului final să compileze bucățile în ordine.

Toate aceste procese au loc de obicei într-un server de streaming video sau în backend-ul platformelor de streaming video.

3.4 Rețelele neurale convoluționale

Rețelele neurale convoluționale (eng. *Convolutional Neural Network*) sunt un tip special de rețea neuronală artificială tip *feed-forward*. Legăturile dintre straturi sunt inspirate de cortexul vizual uman.

Rețelele neurale convoluționale, care sunt numite și *covnet*, nu sunt altceva decât rețele care își împărtășesc parametrii.

În general, o rețea neurală convoluțională are următoarele straturi:

- **Intrare**

Dacă imaginea este formată din 32 pixeli pe lățime și 32 pe înălțime, care cuprind trei canale R, G, B (Roșu, Verde și Albastru), atunci se vor păstra valorile pixelilor bruți ($[32 \times 32 \times 3]$) ale acestora.

- **Convoluția**

Calculează ieșirea acelor neuroni care sunt asociați cu regiunile locale de intrare, astfel încât fiecare neuron va calcula un produs între ponderi (eng. *weights*) și o regiune mică la care sunt legați în volumul de intrare. De exemplu, dacă alegem să incorporăm 12 filtre, atunci va rezulta un volum de $[32 \times 32 \times 12]$. În Figura 3.5 este ilustrat efectul unui strat Convoluție asupra neuronilor unei rețele neurale.

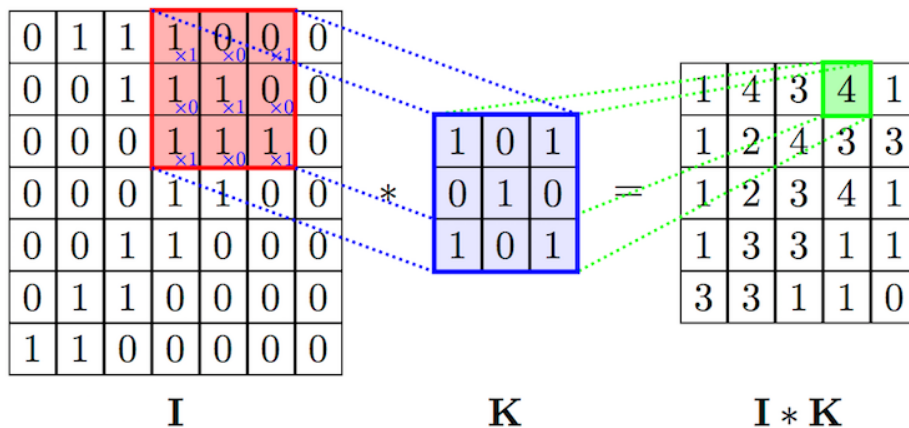


Figura 3.5: Stratul Convoluție

- **Stratul ReLU**

Este folosit special pentru a aplica o funcție de activare elementelor, și anume maximum dintre zero și valoarea neuronului. Rezultă $[32 \times 32 \times 12]$, adică dimensiune volumului rămâne neschimbată.

- **Pooling**

Acest strat este utilizat pentru a efectua o operație de subeșantionare de-a lungul dimensiunilor spațiale (lățime și înălțime) care are ca rezultat volumul $[16 \times 16 \times 12]$.

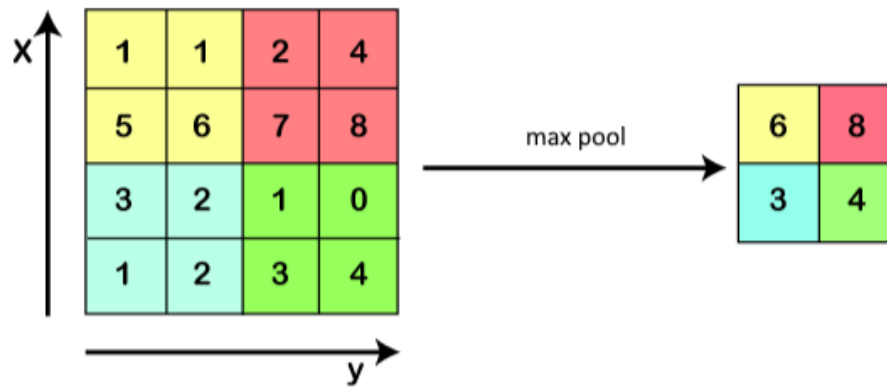


Figura 3.6: Stratul Pooling

12]. În Figura 3.6 este ilustrat modul de funcționare al stratului Pooling asupra neuronilor dintr-o rețea neurală.

- **Ieșirea**

Un strat obișnuit de rețea neurală care primește o intrare de la stratul precedent și calculează probabilitatea apartenenței claselor, rezultând o matrice 1-dimensională egală cu numărul de clase.

Capitolul 4: Implementare

În acest capitol prezentăm detaliile legate de implementarea aplicației de monitorizare.

- Secțiunea 4.1 prezintă pașii urmați în implementarea aplicației Android astfel încât aceasta să poată înregistra și salva date provenite de la camera și microfonul telefonului, date pe care le codifică în conformitate cu tipul acestora (ACC pentru audio și H.264 pentru video) pentru a putea fi transmise unui server RTMP.
- Secțiunea 4.2 descrie modul de utilizare al server-ului Ant Media în aplicația de monitorizare implementată.
- Secțiunea 4.3 evidențiază modul în care este transformat output-ul celor două componente descrise anterior astfel încât utilizatorul să primească detecția feței, a vârstei aproximative și a genului persoanelor din mediul monitorizat.

4.1 Aplicația Android

4.1.1 Versiunea de Android aleasă

Aplicația Android este realizată în IDE-ul Android Studio, aceasta fiind una dintre cele mai simple modalități de a dezvolta o aplicație mobilă.

În ceea ce privește versiunea Android, am ales să utilizez versiunea Android 5.1 (Lollipop, API 22) deoarece doresc ca aplicația să funcționeze și pe telefoanele cu versiuni de Android mai vechi. Conform SDK-ului aplicația va funcționa pe aproximativ 98,1% din dispozitivele Android.

4.1.2 Mediul de rulare

Pentru build și debug mă folosesc de telefonul personal, și anume Motorola One având versiunea Android 10. În continuare sunt enumerate specificațiile acestuia.

- Telefonul prezintă două camere: camera principală de 13 MP și camera secundară de 2 MP.
- Procesorul este un Octa-core pe 64 de biți și 2000 MHz.
- Ramul deține 4 GB de memorie.

4.1.3 Debugging

Prin intermediul unui cablu USB C conectat între telefon și calculatorul personal, nu a mai fost nevoie de un emulator suplimentar.

Am ales această variantă deoarece am putut vizualiza în mod direct timpul de procesare și aspectul aplicației, punându-mă în locul unui utilizator.

4.1.4 Organizarea inițială a aplicației

Aplicația Android a fost gândită să implementeze două mari elemente:

- Posibilitatea de a înregistra și de a salva în telefon capturile realizate de cameră și microfon;
- Posibilitatea de a coda (eng. *encoding*) în format AAC (pentru audio) și H.264 (pentru video) captura celor două componente, pentru a fi transmisă mai departe unui server RTMP.

Pentru a menține o structură sugestivă și confortabilă pentru utilizator, aplicația se va folosi de o singură activitate principală.

4.1.5 Biblioteci

Pentru a realiza cele propuse, aplicația se folosește de biblioteca *rtmp-rtsp-stream-client-java*. Această bibliotecă necesită adaugarea unor permisiuni în *Android-Manifest.xml*, după cum urmează:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.RECORD_AUDIO" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

De asemenea, biblioteca trebuie adăugată și în dependențele din *build.gradle*, cerință ilustrată în Figura 4.1.

```
dependencies {
    implementation 'com.github.pedroSG94.rtmp-rtsp-stream-client-java:rtpliblibrary:2.1.9'
}
```

Figura 4.1: Dependențe *build.gradle*

Această bibliotecă a fost aleasă pentru că oferă posibilitatea de a encoda și de a transmite fluxuri de date video și audio într-o manieră ușor de înțeles, intuitivă.

4.1.6 Activitatea principală

4.1.6.1 Organizarea UI

Pagina principală a aplicației cuprinde următoarele caracteristici:

- Pe fundal, un *SurfaceView* care conține preview-ul camerelor, astfel încât utilizatorul să poată vedea concret care este aria acoperită de camera telefonului;

- Un editor de text local (eng. *EditText*) prin care utilizatorul să poată tasta linkul endpoint-ului RTMP pentru conexiunea cu server-ul media;
- Un buton **Start/Stop Record** pe care utilizatorul îl poate folosi pentru a înregistra și salva în telefon captura;
- Un buton **Start/Stop Stream** pe care utilizatorul îl poate folosi pentru a începe sau opri conexiunea la serverul media și transmiterea de date;
- Un buton **Switch Camera** prin care utilizatorul poate schimba orientarea camerei

Caracteristicile descrise anterior pot fi vizualizate în Figura 4.2.

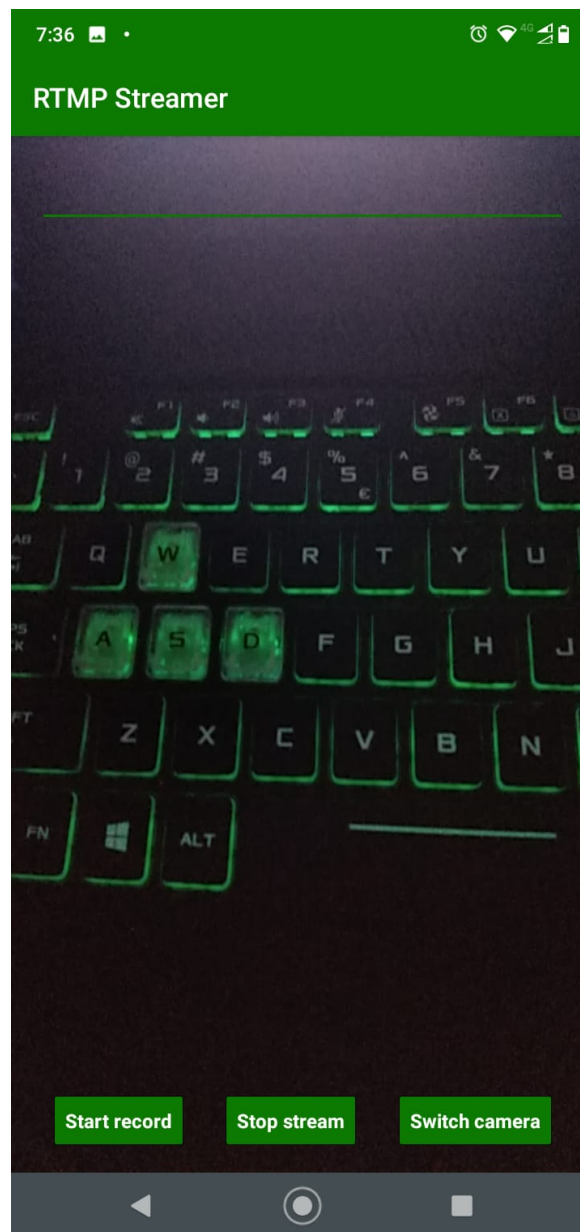


Figura 4.2: Design-ul UI al Aplicației

4.1.6.2 Metodele utilizate

Biblioteca inclusă oferă instrumentele necesare funcționării programului.

Prin intermediul clasei *RtmpCamera1* ce are la bază clasa *Camera1Base*, putem folosi următoarele metode:

- ***prepareVideo***
Se apelează această metodă înainte de a utiliza *startStream*. Se pot configura parametrii video pe care îi dorim precum: înălțimea, lățimea, bitrate-ul, rotația și numărul de frame-uri pe secundă.
- ***prepareAudio***
Se apelează această metodă înainte de a utiliza *startStream*. Se pot configura parametrii audio pe care îi dorim precum: bitrate-ul, sample rate-ul, mono/stereo etc..
- ***startRecord***
Începe înregistrarea video. Trebuie apelat după *prepareVideo* și *prepareAudio*. Ca parametru, metoda primește locația (eng. *path*) directorului în care dorim să realizăm înregistrarea.
- ***stopRecord***
Se oprește înregistrarea videoclipului MP4 început cu *startRecord*. Dacă nu este apelat, fișierul va fi ilizibil.
- ***startPreview***
Pornește previzualizarea camerei. Este ignorat dacă previzualizarea sau stream-ul sunt deja începute.
- ***stopPreview***
Oprește previzualizarea camerei. Este ignorat dacă difuzarea a fost deja oprită. Trebuie apelat după *stopStream* pentru a elibera camera corect în momentul în care se încheie activitatea.
- ***startStream***
Trebuie apelat după *prepareVideo* și/ sau *prepareAudio*. Această metodă înlocuiește rezoluția din *startPreview* la rezoluția din *prepareVideo*. Primește drept parametru url-ul endpoint-ului.
- ***stopStream***
Oprește stream-ul început de *startStream*.
- ***reTry***
Se încearcă reconectarea la server de atâtea ori cât este indicat în metoda *setRetries*.
- ***switchCamera***
Se realizează comutarea la camera nefolosită a telefonului. Poate fi apelată atât în timpul previzualizării cand și în timpul stream-ului.

4.1.6.3 Implementarea metodelor în cod

Pentru a începe previzualizarea, în momentul în care Surface View-ul este creat, apelăm metoda *startPreview*.

```
@Override
    public void surfaceCreated(@NonNull SurfaceHolder surfaceHolder) {
        rtmpCamera1.startPreview();
    }
```

Butoanele aplicației se folosesc de apelul ascultătorului de click folosind *View.OnClickListener*, astfel codul cuprins în *onClick(View view)* începe executarea după atingerea butonului. În funcție de butonul ales, *rtmpCamera1* execută metodele necesare.

Se poate alege dintre următoarele butoane disponibile:

- **Buton Start/Stop Stream** (Figura 4.3);

```
case R.id.b_start_stop:
    if (!rtmpCamera1.isStreaming()) {
        if (rtmpCamera1.isRecording())
            || rtmpCamera1.prepareAudio() && rtmpCamera1.prepareVideo() {
            button.setText(R.string.stop_button);
            rtmpCamera1.startStream(etUrl.getText().toString());
        } else {
            Toast.makeText(this, "Error preparing stream, This device cant do it",
                Toast.LENGTH_SHORT).show();
        }
    } else {
        button.setText(R.string.start_button);
        rtmpCamera1.stopStream();
    }
    break;
```

Figura 4.3: Cod buton Start/Stop Stream

- **Buton Switch Camera** (Figura 4.4);

```
case R.id.switch_camera:
    try {
        rtmpCamera1.switchCamera();
    } catch (CameraOpenException e) {
        Toast.makeText(this, e.getMessage(), Toast.LENGTH_SHORT).show();
    }
    break;
```

Figura 4.4: Cod buton Switch Camera

```

case R.id.b_record:
    if (!rtmpCameral.isRecording()) {
        try {
            if (!folder.exists()) {
                folder.mkdir();
                System.out.println("App folder");
            }
            SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMdd_HHmms", Locale.getDefault());
            currentDateAndTime = sdf.format(new Date());
            if (!rtmpCameral.isStreaming()) {
                if (rtmpCameral.prepareAudio() && rtmpCameral.prepareVideo()) {
                    rtmpCameral.startRecord(
                        folder.getAbsolutePath() + "/" + currentDateAndTime + ".mp4");
                    bRecord.setText(R.string.stop_record);
                    Toast.makeText(this, "Recording... ", Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(this, "Error preparing stream, This device cant do it",
                        Toast.LENGTH_SHORT).show();
                }
            } else {
                rtmpCameral.startRecord(
                    folder.getAbsolutePath() + "/" + currentDateAndTime + ".mp4");
                bRecord.setText(R.string.stop_record);
                Toast.makeText(this, "Recording... ", Toast.LENGTH_SHORT).show();
            }
        } catch (IOException e) {
            rtmpCameral.stopRecord();
            PathUtils.updateGallery(this, folder.getAbsolutePath() + "/" + currentDateAndTime + ".mp4");
            bRecord.setText(R.string.start_record);
            Toast.makeText(this, e.getMessage(), Toast.LENGTH_SHORT).show();
        }
    } else {
        rtmpCameral.stopRecord();
        PathUtils.updateGallery(this, folder.getAbsolutePath() + "/" + currentDateAndTime + ".mp4");
        bRecord.setText(R.string.start_record);
        Toast.makeText(this,
            "file " + currentDateAndTime + ".mp4 saved in " + folder.getAbsolutePath(),
            Toast.LENGTH_SHORT).show();
    }
    break;

```

Figura 4.5: Cod buton Start/Stop Record

- Buton Start/Stop Record (Figura 4.5).

De menționat este faptul că înregistrările se salvează în funcție de data și ora exactă la care a fost oprită înregistrarea, în formatul *yyyyMMdd-HHmms.mp4*, după cum se poate observa în Figura 4.6.

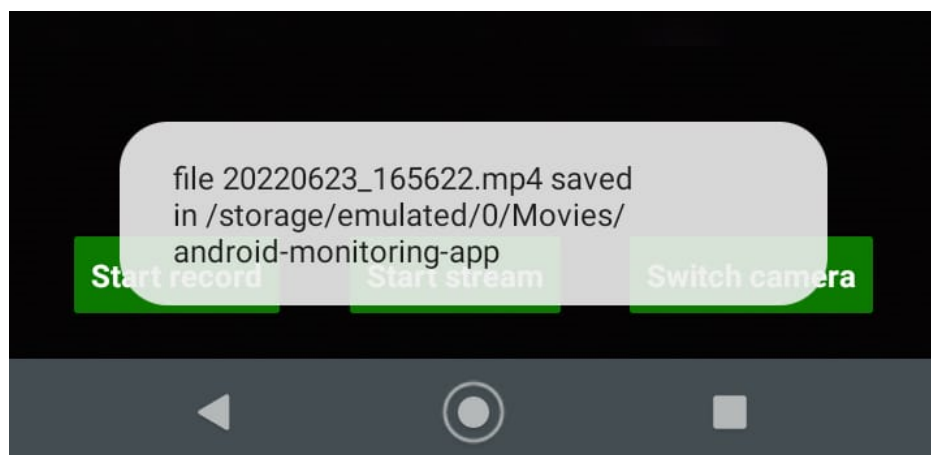


Figura 4.6: Fișier video salvat în aplicația Android

4.1.6.4 Gestionarea erorilor de conexiune

Erorile legate de conexiune se realizează folosind *runOnUiThread*, întrucât thread-ul curent fiind unul UI, runnable-ul se va executa imediat. Codul acestuia poate fi consultat în Figura 4.7.

```
@Override
public void onConnectionFailedRtmp(String s) {
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            if (rtmpCameral.retry(5000, s, null)) {
                Toast.makeText(BasicRtmpActivity.this, "Retry", Toast.LENGTH_SHORT)
                    .show();
            } else {
                Toast.makeText(BasicRtmpActivity.this, "Connection failed. " + s, Toast.LENGTH_SHORT)
                    .show();
                rtmpCameral.stopStream();
                button.setText(R.string.start_button);
            }
        }
    });
}
```

Figura 4.7: Cod Eroare de Conexiune

Prin intermediul vizualizărilor de tip *Toast*, acestea sunt vizibile utilizatorului. Figura 4.8 oferă un exemplu de astfel de mesaj de eroare de conexiune primit de către utilizator.

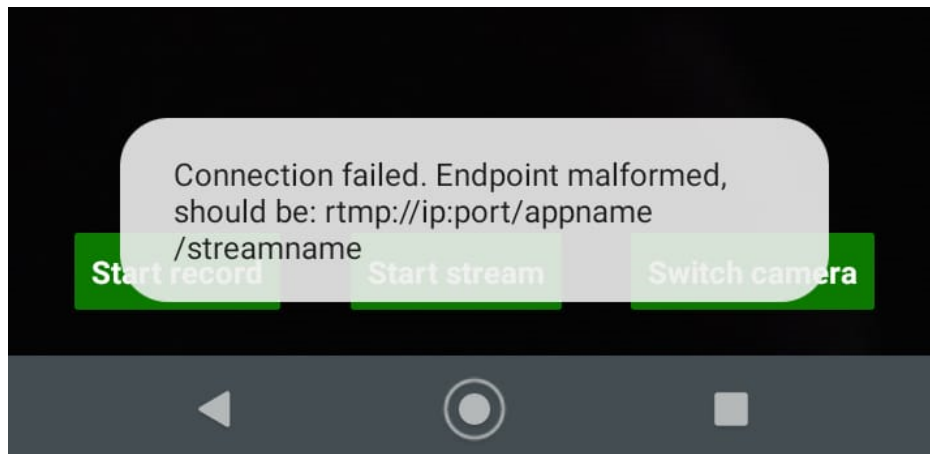


Figura 4.8: Eroare legată de conexiunea cu serverul în aplicația Android

4.1.6.5 Testarea aplicației

Cu ajutorul telefonului personal am recunstituit diverse scenarii de utilizare ale aplicației.

Unul dintre aceste experimente s-a realizat în mașină. Scenariul considerat a

fost monitorizate un băiețuș lăsat singur în mașină. Astfel, pe bancheta din spate a mașinii a fost instalat un scaun de băiețuș în care am așezat o păpușă, după cum se poate observa în Figura 4.9.

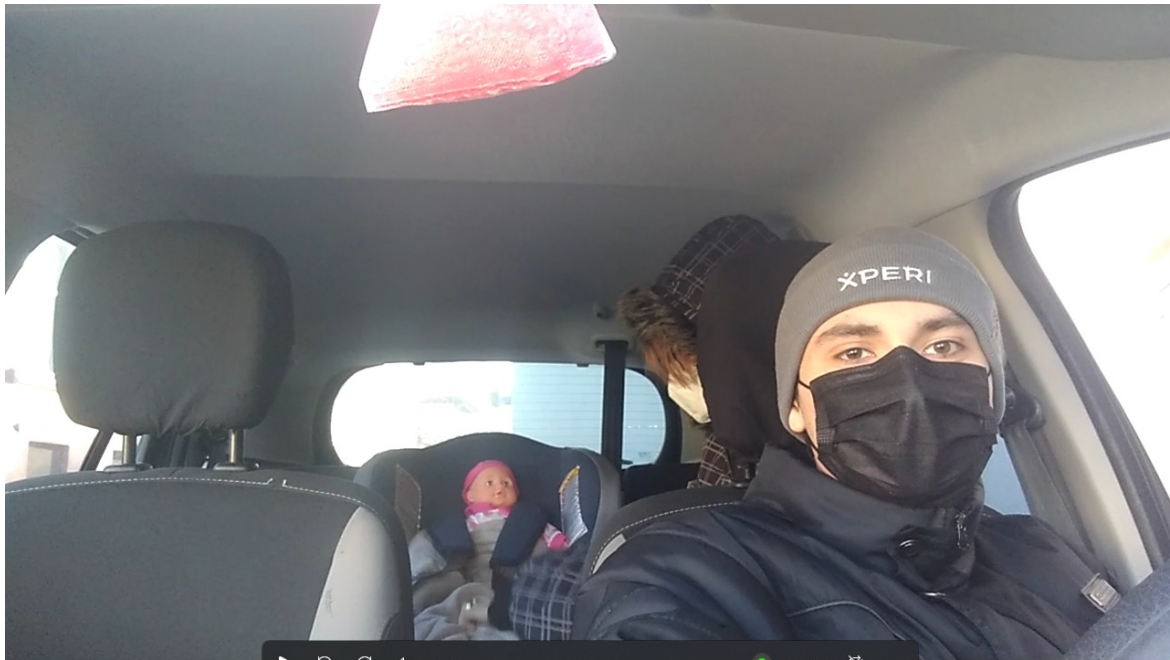


Figura 4.9: Frame din timpul experimentului

Pentru a fixa telefonul am folosit un suport obișnuit de telefon (pentru utilizarea GPS-ului). Datorită acestui suport, interiorul mașinii a fost înregistrat din mai multe unghiuri:

- lângă șofer;
- pe geamul pasagerului;
- pe geamul din spate;
- pe geamul portbagajului.

Ambele camere ale telefonului au fost utilizate în acest experiment, iar telefonul a fost poziționat atât în mod portret, cât și vedere.

Pe parcursul experimentului, am ajuns la concluzia că pentru telefonul personal, camera din spate oferă performanțe mai bune decât cea din față, atât din punct de vedere calitativ cât și din punctul de vedere al cantității de lumină receptate.

În plus, poziționarea vedere a telefonului lipit de parbriz a capturat cel mai bine interiorul mașinii.

4.2 Server-ul Media

Pentru server-ul media am ales să utilizez Server-ul Ant Media, deoarece printre funcțiile pe care le oferă se numără ingerarea RTMP (eng. *RTMP Ingest*) și suport pentru RTMP, RTSP, MP4 și HLS. Deoarece Ant Media Server rulează pe Linux, am creat o mașină virtuală dedicată cu sistem de operare Ubuntu.

4.2.1 Configurarea server-ului media

Prezentăm în cele ce urmează pașii necesari configurării server-ului Ant Media pentru aplicația practică.

- Se descarcă Serverul Ant Media (AMS) de pe repo-ul github ([12]).
- În terminal, se navighează către directorul unde s-a descărcat arhiva AMS.

```
cd directorul/ce/contine/ant-media-server....zip
```

- Se descarcă script-ul shell *install_ant-media-server.sh*.

```
wget
https://raw.githubusercontent.com/ant-media/Scripts/master/
install_ant-media-server.sh &&
chmod 755 install_ant-media-server.sh
```

- Se rulează scriptul de instalare.

```
sudo ./install_ant-media-server.sh -i [FISIER_SERVER_ANT_MEDIA]
```

- Pentru a actualiza de la o versiune anterioară se apelează:

```
sudo ./install_ant-media-server.sh -i [FISIER_SERVER_ANT_MEDIA] -r true
```

- Pentru mai multe opțiuni, se poate folosi comanda:

```
sudo ./install_ant-media-server.sh -h
```

- Se verifică dacă serviciul rulează, utilizand urmatoarea comandă in terminal:

```
sudo service antmedia status
```

- Pentru a porni/opri serviciul AMS se apelează:

```
sudo service antmedia stop
sudo service antmedia start
```



```
● antmedia.service - Ant Media Server
   Loaded: loaded (/etc/systemd/system/antmedia.service; enabled; vendor pre>
   Active: active (running) since Thu 2022-06-23 18:40:38 EEST; 44s ago
     Main PID: 649 (java)
       Tasks: 57 (limit: 36000)
      Memory: 347.7M
        CGroup: /system.slice/antmedia.service
                └─649 /usr/lib/jvm/java-11-openjdk-amd64/bin/java -Dlogback.Conte>
```

Figura 4.10: Statusul Server-ului Ant Media în Terminal

În Figura 4.10 este ilustrat modul de afișare al statusului server-ului Ant Media în terminalul sistemului de operare.

Serverul Ant Media suportă instalarea SSL. **Protocolul SSL** este o tehnologie de securitate standard utilizată pentru a stabili o legătură criptată între un server web și un client web. SSL facilitează comunicarea securizată în rețea prin identificarea și autentificarea serverului, precum și prin asigurarea confidențialității și integrității tuturor datelor transmise.

4.2.2 Configurarea server-ului pe computer-ul personal

Deoarece Ant Media funcționează strict pe sistemul de operare Linux, am avut de ales între două variante:

- Windows Subsystem for Linux (WSL);
- Mașină virtuală.

4.2.2.1 Configurarea pe Windows Subsystem for Linux

Pentru a putea folosi WSL în sistemul de operare Windows, acesta trebuie mai întâi activat și instalat, astfel fiind posibilă configurarea server-ului Ant Media fără a fi nevoie de schimbarea sistemului de operare.

Totuși, WSL funcționează doar pe sistemele de operare Windows 10, de la versiunea 2004 în sus sau direct pe Windows 11, pe varianta Developer.

4.2.2.2 Configurarea pe Mașina Virtuală

În mod evident, este posibilă instalarea și configurarea server-ului Ant Media pe o mașină virtuală cu sistemul de operare bazat pe Linux.

Această modalitate este de preferat din mai multe motive, dintre care amintim:

- WSL este instabil;
- Controlul asupra resurselor este mai mare într-o mașină virtuală;

- Mașina virtuală oferă o separare concretă între sistemul de operare de bază și cel disponibil pe mașina virtuală, ceea ce ar putea fi benefic în anumite situații.

În cazul meu, ideea WSL-ului mi-a stârnit curiozitatea, însă nu am putut duce la capăt configurarea acestuia pe calculatorul personal din cauza instabilității sale. Așadar, server-ul a fost configurat pe o mașină virtuală **Oracle VM Boc** cu sistemul de operare Ubuntu, versiunea 21.10.

4.2.3 Accesarea panoului web Ant Media

Pentru a utiliza server-ul Ant Media sunt necesari următorii pași:

- Panoul web al server-ului Ant Media, ilustrat în Figura 4.11, poate fi accesat din browser tastând <http://SERVER-IP-ADDRESS:5080>.

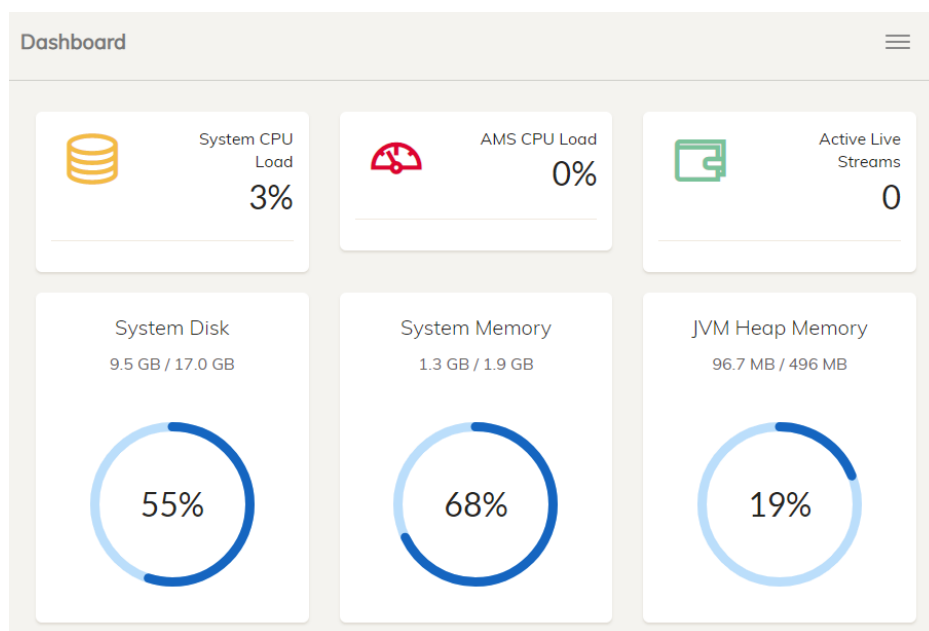
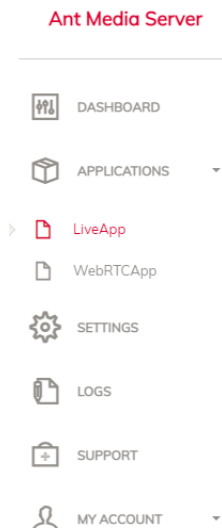


Figura 4.11: Panoul web al server-ului Ant Media

- Din aplicația *LiveApp* (Figura 4.12) se pot crea live stream-uri ce sunt accesate de aplicația Android.

Figura 4.12: Aplicația *LiveApp* a server-ului Ant Media

- Din *Actions*, găsim URL-ul de endpoint de care avem nevoie pentru a realiza conexiunea dintre aplicație și server (Figura 4.13).

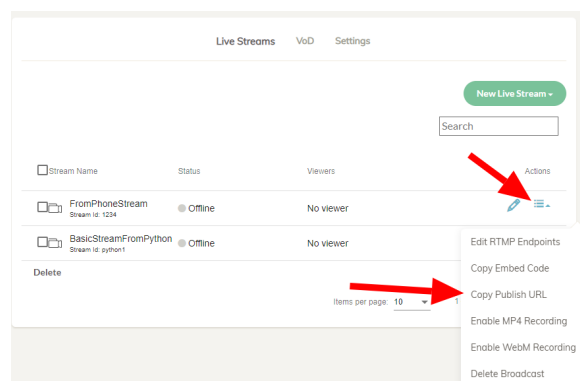


Figura 4.13: Localizarea URL-ului necesar realizării conexiunii dintre aplicație și server

- Când conexiunea se realizează cu succes, statusul stream-ului se schimbă. Tot din panoul web al serverului se poate vedea stream-ul nou creat, după cum se poate observa în Figura 4.14.

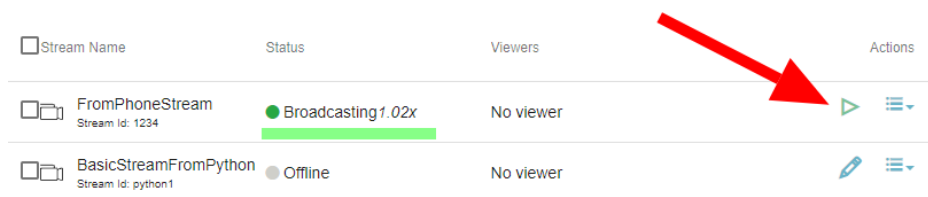


Figura 4.14: Statusul stream-ului la o conexiune realizată cu succes

În Figura 4.15 este prezentată comparația între stream-ul de pe server-ul Ant Media și preview-ul aplicației mobile.

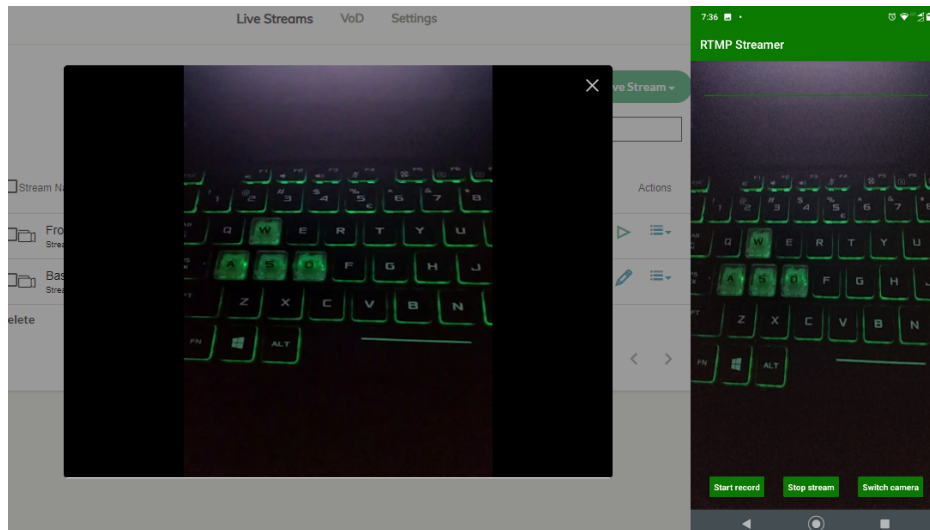


Figura 4.15: Comparație între stream-ul de pe server și preview-ul Aplicației

4.2.4 Playeri media

Stream-ul final poate fi transmis mai departe pentru realizarea de live stream-uri pe platforme precum *YouTube* și *Facebook*, prin adaugarea URL-ului stream-ului la endpoint-uri.

În Figura 4.16 se poate observa faptul că și platforma *Youtube* se folosește de protocolul RTMP pentru a transmite live stream-urile creatorilor de conținut.

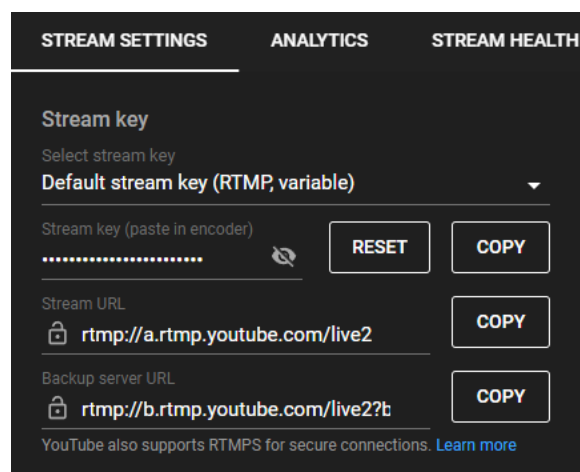


Figura 4.16: URL cu protocol RTMP al platformei Youtube

Majoritatea creatorilor de conținut de pe platforme precum *Youtube* și *Twitch*

se folosesc de aplicații precum *OBS*. Prin intermediul acestora, creatorii își pot personaliza live stream-urile.

Legătura pe care o fac cu server-ele backend ale companiilor se poate realiza prin URL de tipul RTMP, atât timp cât utilizatorul deține cheia stream-ului.

4.3 Componenta de Procesare a Stream-ului

4.3.1 Organizarea inițială a Componentei de Procesare a Stream-ului

Componenta de Procesare are în vedere transformarea stream-ului primit de către serverul media prin aplicația Android în așa fel încât să se evidențieze output-ul inferențelor a trei rețele neurale, anume detectarea feței, detectarea vârstei aproximative și detectarea genului persoanelor filmate prin intermediul camerei telefonului. Modificările vor fi transmise server-ului media ca un nou stream live, cu o întârziere cât mai scăzută.

4.3.2 Mediul de rulare

Acest lucru este realizat în Python, prin intermediul Jupyter Notebook (versiunea 6.4.5) de la Anaconda, utilizând versiunea de Python 3.

4.3.3 Alegerea rețelelor neurale

Rețelele folosite pentru detectările propuse sunt inspirate după proiectul lui Gil Levi și Tal Hassner din lucrarea "*Age and Gender Classification using Convolutional Neural Networks*" ([13]).

4.3.3.1 Arhitectura rețelelor

Aceste rețele sunt realizate pe arhitectura CNN (Convolutional Neural Network). Fiecare rețea conține trei straturi convoluționale, fiecare strat fiind urmat de un strat ReLu și de un strat pooling.

Primul strat convoluțional conține 96 de filtre de 7×7 pixeli. Al doilea strat convoluțional are 256 de filtre a câte 5×5 pixeli. Ultimul strat convoluțional conține 384 de filtre de 3×3 pixeli.

În final, două straturi complet conectate sunt adăugate, fiecare conținând 512 neuroni.

Arhitectura descrisă anterior poate fi consultată în Figura 4.17, iar vizualizarea straturilor este ilustrată în Figura 4.18.



Figura 4.17: Arhitectura Rețelelor CNN utilizate

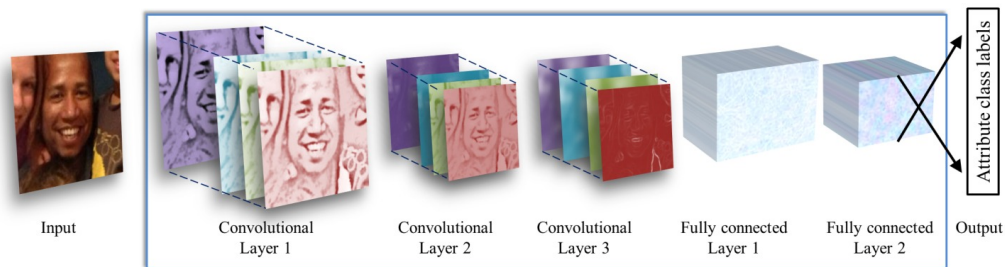


Figura 4.18: Ilustrarea Straturilor Rețelelor CNN utilizate

4.3.4 Alegerea librăriei OpenCV-Python

Această bibliotecă a fost aleasă deoarece OpenCV oferă posibilitatea de a accesa frame-urile stream-ului de pe serverul media, pentru input.

De asemenea, mulțumită modulului DNN (Deep Neural Network), inferența rețelelor se poate realiza în mod direct.

4.3.5 Utilizarea librăriei OpenCV-Python

Arhitectura modelelor pre-antrenate este de tipul Caffe, cu extensia *.caffemodel*.

```
face_net = cv2.dnn.readNetFromCaffe(FACE_MODEL, FACE_PROTO)
age_net = cv2.dnn.readNetFromCaffe(AGE_MODEL, AGE_PROTO)
gender_net = cv2.dnn.readNetFromCaffe(GENDER_MODEL, GENDER_PROTO)
```

Pentru detectarea feței, frame-ul trebuie modificat pentru a putea fi utilizat de rețea într-un blob. După ce aceasta este setată drept input pentru rețea, se realizează inferența și se primesc predicțiile. Lista rezultată este apoi parcursă pentru a reține coordonatele bidimensionale din planul frame-ului. Aceste coordonate sunt apoi convertite în tip întreg și mărite puțin pentru a lărgi chenarul în care se afla fața. Funcția implementată pentru a rezolva acest lucru este prezentată în Figura 4.19

```
def get_faces(frame, confidence_threshold=0.5):

    blob = cv2.dnn.blobFromImage(frame, 1.0, (300, 300), (104, 177.0, 123.0))
    face_net.setInput(blob)
    output = np.squeeze(face_net.forward())

    faces = []

    for i in range(output.shape[0]):
        confidence = output[i, 2]
        if confidence > confidence_threshold:
            box = output[i, 3:7] * \
                np.array([frame.shape[1], frame.shape[0],
                        frame.shape[1], frame.shape[0]])

            start_x, start_y, end_x, end_y = box.astype(np.int)
            start_x, start_y, end_x, end_y = start_x - \
                10, start_y - 10, end_x + 10, end_y + 10
            start_x = 0 if start_x < 0 else start_x
            start_y = 0 if start_y < 0 else start_y
            end_x = 0 if end_x < 0 else end_x
            end_y = 0 if end_y < 0 else end_y

            faces.append((start_x, start_y, end_x, end_y))

    return faces
```

Figura 4.19: Funcția *get_faces()*

Pentru predicțiile de vârstă și gen, frame-ul este de asemenea modificat pentru a putea fi utilizat de rețele într-un blob, însă output-ul este reprezentat de elementul cu ponderea cea mai ridicată de probabilitate, iar input-ul de chenarul determinat de detectarea feței. Cele două funcții sunt prezentate în Figura 4.20.

```
def get_gender_predictions(face_img):
    blob = cv2.dnn.blobFromImage(
        image=face_img, scalefactor=1.0, size=(227, 227),
        mean=MODEL_MEAN_VALUES, swapRB=False, crop=False
    )
    gender_net.setInput(blob)
    return gender_net.forward()

def get_age_predictions(face_img):
    blob = cv2.dnn.blobFromImage(
        image=face_img, scalefactor=1.0, size=(227, 227),
        mean=MODEL_MEAN_VALUES, swapRB=False
    )
    age_net.setInput(blob)
    return age_net.forward()
```

Figura 4.20: Funcțiile *get_gender_predictions()* și *get_age_predictions()*

4.3.6 Logica modificării frame-urilor

Componenta de Procesare a Stream-ului ia de pe server-ul media frame-urile stream-ului live. Scopul acesteia este de a modifica frame-ul astfel încât să accentueze output-ul rețelelor neurale implementate, frame-ul respectiv fiind apoi transmis aceluiași server într-un nou live stream, în așa fel încât diferența de timp între acest stream și stream-ul inițial să fie cât mai mică.

Pentru a realiza acest lucru trebuie ținut cont de faptul că în moment când se desenează pe un frame, procesul de desenare este costisitor din punct de vedere al timpului.

Pentru a micșora diferența de timp între stream-ul inițial și cel final, din cele 30 de frame-uri pe secundă transmise, componenta de procesare a stream-ului primește, modifică și încarcă doar o treime aleasă în mod arbitrar.

Astfel, prioritizăm o diferență minimală de timp în locul calității stream-ului final.

4.3.7 Evidențierea pe frame

Pentru a captura videoul stream-ului de pe server, se folosește *cv2.VideoCapture*, ce primește drept parametru URL-ul endpoint-ului RTMP. Pentru fiecare frame, se realizează inferențele rețelelor și se obțin output-urile, după cum se poate observa în Figura 4.21.

```
faces = get_faces(frame)
for i, (start_x, start_y, end_x, end_y) in enumerate(faces):
    face_img = frame[start_y: end_y, start_x: end_x]
    age_preds = get_age_predictions(face_img)
    gender_preds = get_gender_predictions(face_img)

    i = gender_preds[0].argmax()
    gender = GENDER_LIST[i]
    gender_confidence_score = gender_preds[0][i]

    i = age_preds[0].argmax()
    age = AGE_INTERVALS[i]
    age_confidence_score = age_preds[0][i]
```

Figura 4.21: Obținerea output-urilor

Aceste output-uri sunt apoi desenate pe frame, conform Figura 4.22.

Chenarul este desenat în funcție de coordonatele din output-ul rețelei de detectare de fețe (ușor mărite) prin intermediul funcției *cv2.rectangle()*.

Output-ul rețelei de detectare a genului va fi poziționat în funcție de acest chenar

(colțul din stânga sus). În plus, culoare chenarului, precum și a textului este aleasă în funcție de gen (roz pentru feminin și albastru pentru masculin).

Output-ul rețelei de aproximare a vârstei va fi poziționat în continuarea output-ului rețelei de detectare a genului, fiind evidențiat și procentajul de probabilitate.

Un exemplu poate fi consultat în Figura 4.25.

```
# Chenar
label = f"{gender}-{gender_confidence_score*100:.1f}%, {age}-{age_confidence_score*100:.1f}%"
yPos = start_y - 15
while yPos < 15:
    yPos += 15
box_color = (255, 0, 0) if gender == "Male" else (147, 20, 255)
cv2.rectangle(frame, (start_x, start_y), (end_x, end_y), box_color, 2)

# Label
font_scale = 0.54
cv2.putText(frame, label, (start_x, yPos),
            cv2.FONT_HERSHEY_SIMPLEX, font_scale, box_color, 2)
```

Figura 4.22: Desenarea pe frame a output-urilor

4.3.8 Transmiterea frame-urilor pe server-ul media

Transmiterea frame-urilor modificate catre server-ul media este realizată prin intermediul unui nou stream cu ajutorul bibliotecilor *ffmpeg* și *subprocess*. Astfel, putem crea un subproces pentru comanda *ffmpeg* care transmite frame-urile prin intermediul unui PIPE. **PIPE** este folosit pentru a trimite sau primi date de la un program care rulează ca subproces în Python. Comanda subprocesul descrisă anterior este ilustrată în Figura 4.23

```
command = ['ffmpeg',
           '-y',
           '-f', 'rawvideo',
           '-vcodec', 'rawvideo',
           '-pix_fmt', 'bgr24',
           '-flags', 'low_delay',
           '-s', "{}x{}".format(width, height),
           '-r', str(red_fps),
           '-i', '-',
           '-c:v', 'libx264',
           '-pix_fmt', 'yuv420p',
           '-preset', 'ultrafast',
           '-r', '25',
           '-f', 'flv',
           rtmp_url]

p = subprocess.Popen(command, stdin=subprocess.PIPE)
```

Figura 4.23: Commanda *ffmpeg* pentru PIPE

Lista atributelor comenzii subprocesului semnifică următoarele:

- **'-y'**
Suprascrie fișierele de ieșire fără a cere acordul.
- **'-f'**
Forțează formatarea fișierului de intrare sau de ieșire, în cazul nostru în format *rawvideo*. Formatul este în mod normal detectat automat pentru fișierele de intrare și ghicit din extensia fișierului pentru fișierele de ieșire, așa că această opțiune nu este necesară în majoritatea cazurilor.
- **'-vcodec'**
Setează codec-ul video-ului (decoder + codor) .
- **'-pix_fmt'**
Este un șir reprezentând formatul de pixeli al cadrelor video stocate. Poate fi un număr corespunzător unui format de pixeli sau un nume de format de pixeli.
- **'-flags', '-low_delay'**
Forțează codec-ul să aibă intarzieri mici.
- **'-s'**
Specifică dimensiunea imaginii, în cazul nostru înălțimea și lungimea stream-ului live.
- **'-r'**
Forțează rata frame-urilor inputului.
- **'-i'**
Reprezintă input-ul, în cazul nostru stream-ul RTMP.
- **'-c:v'**
Codifică toate fluxurile video (aici cu **libx264**) și copiază toate fluxurile audio.
- **'-preset'**
Setează configurația prestabilită. Aceasta face unele setări automate bazate pe tipul general al imaginii.

Frame-ul este transmis subprocesului prin intermediul comenzii
p.stdin.write(frame.tobytes())

4.3.9 Finalitatea Componentei de Procesare a Stream-ului

Astfel, server-ul media prezintă doua stream-uri live, cel realizat prin aplicația Android și cel modificat să includă output-ul rețelelor, după cum se poate observa în Figura 4.24.








<input type="checkbox"/> Stream Name	Status	Viewers	Actions
 FromPhoneStream Stream Id: 1234	● Broadcasting 1.00x	 RTMP : 1	 
 StreamFromPython Stream Id: python1	● Broadcasting 1.23x	No viewer	 

Figura 4.24: Cele două stream-uri live de pe server-ul media

În Figura 4.25 este prezentat stream-ul creat în urma modificărilor frame-urilor în raport cu output-ul rețelelor neurale.

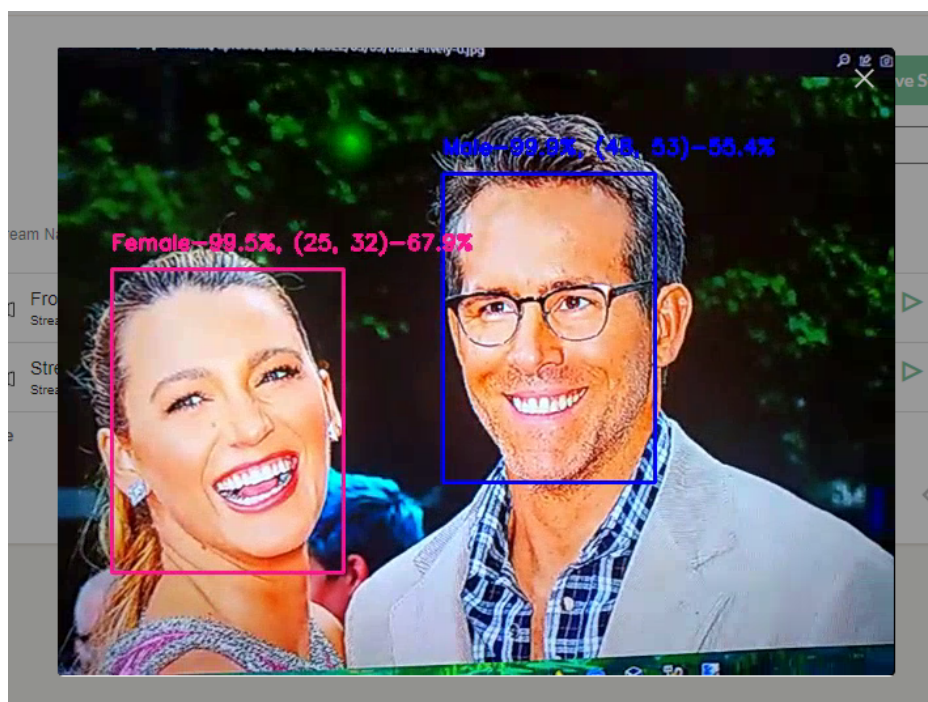


Figura 4.25: Stream-ul final de pe server-ul media

Capitolul 5: Concluzii

Lucrarea de față are drept scop descrierea tehnologiilor utilizate și a modului de implementare ale aplicației de monitorizare intitulată „*Real Time Monitoring Streaming Tool*”.

Tehnologiile utilizate au fost enumerate în Capitolul 2. Aici au fost prezentate particularitățile limbajelor de programare Java și Python, fiind evidențiate motivele pentru care au fost alese în implementarea aplicației. De asemenea, mediile de programare Android, Andoid Studio și Jupyter Notebook au fost descrise din punctul de vedere al accesibilității și ușurinței utilizării în dezvoltarea de aplicații mobile.

Au fost alese o serie de biblioteci reprezentative manipulării datelor audio și video pentru avantajele oferite în claritatea și performanța necesară unui live stream ce pune accentul pe siguranța persoanelor monitorizate. De asemenea, server-ul ales pentru streaming-ul video în timp real, Ant Media Server, a fost analizat prin descrierea arhitecturii acestuia și a modului de utilizare de către un număr considerabil de utilizatori simultan. Tot în Capitolul 2 au fost enumerate bibliotecile necesare lucrului cu rețele neurale, în special rețelele de tip Deep Learning utilizate pentru detectarea fețelor și a probabilităților trăsăturilor de vârstă și gen.

Noțiunile teoretice utilizate în realizarea acestei aplicații au fost prezentate în Capitolul 3. Arhitectura MVVM cu cele trei componente principale ale sale (Model, View și ViewModel) a fost aleasă pentru menținerea unei separări clare între logica aplicației și interfața de utilizare.

De asemenea, au fost evidențiate particularitățile streaming-ului RTMP, unul dintre cele mai utilizate formate de streaming potrivit ([11]). Totodată sunt prezentate și motivele pentru care această tehnologie a fost aleasă în implementarea aplicației de față. Teoria rețelelor neurale convoluționale încheie Capitolul 3 prin descrierea principalelor tipuri de straturi reprezentative dintre care amintim: stratul pooling și convoluția.

Capitolul 4 introduce cele trei componente principale ale aplicației: Aplicația Android, Server-ul media și Componenta de procesare a Stream-ului. Pentru fiecare componentă în parte au fost prezentate modul de implementare și interconectare, funcționalitățile principale și exemple reprezentative ale diverselor situații ce pot apărea în utilizarea aplicației.

Capitolul 6: Perspective de dezvoltare

În acest capitol oferim câteva idei de dezvoltare al aplicației prezentate în această lucrare.

Din moment ce RTMP nu este singurul protocol folosit pentru streaming, o primă idee de dezvoltare ar fi folosirea, integrarea și conexiunea cu RTSP (Real time Streaming Protocol), atât în partea de Aplicație Android, cât și pe partea de procesare din Componenta de procesare a streaming-ului.

Serverul Ant Media se dovedește a fi fiabil și confortabil de folosit. Adăugarea serverului media pe un domeniu public, cu propria adresă IP este o dorință de viitor, întrucât în momentul de față, acesta este instalat local (pe localhost).

Componenta de procesare a streaming-ului prezintă cel mai mare potențial de dezvoltare, ținând cont de posibilitatea integrării a noi rețele neurale, în așa fel încât pot fi satisfăcute nevoile cele mai specifice, chiar unice ale user-ilor. În acest sens, oferim următoarele exemple:

- Integrarea unei rețele de procesare a sunetului pentru a detecta diferite zgomote, de la alarme până la plânsul bebelușului;
- Integrarea unei rețele de detectare a animalelor de companie;
- Integrarea unei rețele de detectare a mișcării.

Totodată, implementarea unui sistem de notificări și alarme pentru Aplicația Android care să anunțe utilizatorul în cazul unei urgențe ar fi de folos.

Listă de Figuri

Figura 1.1	Piramida lui Maslow	5
Figura 1.2	Aplicația Hik-Connect	6
Figura 1.3	Aplicația Manything	7
Figura 1.4	Aplicația Alfred Camera	8
Figura 1.5	Real Time Monitoring Streaming Tool	9
Figura 2.1	Diagrama Arhitecturii Android	13
Figura 2.2	Interfața Android Studio	14
Figura 2.3	Emulator utilizat împreună cu Android Studio	15
Figura 2.4	Diagrama Arhitecturii Server-ului Ant Media	16
Figura 2.5	Ierarhizarea proceselor părinte-copil	20
Figura 3.1	Diagrama Arhitecturii MVVM	23
Figura 3.2	Cele mai utilizate formate de streaming ([11])	25
Figura 3.3	Fluxul de lucru tipic dintre RTMP și HLS ([11])	25
Figura 3.4	Cele mai utilizate de ingestare a stream-ului	27
Figura 3.5	Stratul Convoluție	28
Figura 3.6	Stratul Pooling	29
Figura 4.1	Dependințe <i>build.gradle</i>	31
Figura 4.2	Design-ul UI al Aplicației	32
Figura 4.3	Cod buton Start/Stop Stream	34
Figura 4.4	Cod buton Switch Camera	34
Figura 4.5	Cod buton Start/Stop Record	35
Figura 4.6	Fișier video salvat în aplicația Android	35
Figura 4.7	Cod Eroare de Conexiune	36
Figura 4.8	Eroare legată de conexiunea cu serverul în aplicația Android	36
Figura 4.9	Frame din timpul experimentului	37
Figura 4.10	Statusul Server-ului Ant Media în Terminal	39
Figura 4.11	Panoul web al server-ului Ant Media	40
Figura 4.12	Aplicația <i>LiveApp</i> a server-ului Ant Media	41
Figura 4.13	Localizarea URL-ului necesar realizării conexiunii dintre aplicație și server	41
Figura 4.14	Statusul stream-ului la o conexiune realizată cu succes	41
Figura 4.15	Comparatie între stream-ul de pe server și preview-ul Aplicației	42
Figura 4.16	URL cu protocol RTMP al platformei Youtube	42
Figura 4.17	Arhitectura Rețelelor CNN utilizate	44
Figura 4.18	Ilustrarea Straturilor Rețelelor CNN utilizate	44
Figura 4.19	Funcția <i>get_faces()</i>	45
Figura 4.20	Funcțiile <i>get_gender_predictions()</i> și <i>get_age_predictions()</i>	45
Figura 4.21	Obținerea output-urilor	46

Figura 4.22 Desenarea pe frame a output-urilor	47
Figura 4.23 Commanda <i>ffmpeg</i> pentru PIPE	47
Figura 4.24 Cele două stream-uri live de pe server-ul media	49
Figura 4.25 Stream-ul final de pe server-ul media	49
Figura 6.1 Raport Turnitin	55

Bibliografie

Cărți

- [1] Abraham H Maslow. *Hierarchy of needs: A theory of human motivation*. 2011.

Articole

- [8] Yangqing Jia et al. “Caffe: Convolutional Architecture for Fast Feature Embedding”. In: *arXiv preprint arXiv:1408.5093* (2014).

Online

- [2] Andreea Radu. *INFOGRAFIC: Cate locuinte au fost sparte de hoți in 2021?* 2022. URL: <https://www.1asig.ro/INFOGRAFIC-Cate-locuinte-au-fost-sparte-de-hoti-in-2021-articol-3,100-68020.htm>. (accesat mai 2022).
- [3] URL: [https://www.hik-connect.com/views/login/index.html?returnUrl=http://www.hik-connect.com/devices/page&r=9169861393332564327&host=www.hik-connect.com&from=c17392dc2e6c405a931b#/.](https://www.hik-connect.com/views/login/index.html?returnUrl=http://www.hik-connect.com/devices/page&r=9169861393332564327&host=www.hik-connect.com&from=c17392dc2e6c405a931b#/) (accesat iunie 2022).
- [4] URL: <http://www.manythings.org>. (accesat iunie 2022).
- [5] URL: <https://alfred.camera>. (accesat iunie 2022).
- [6] Strikingly. *What Are The Advantages and Disadvantages of Java?* 2020. URL: <https://cetpainfotech.mystrikingly.com/blog/what-are-the-advantages-and-disadvantages-of-java>. (accesat mai 2022).
- [7] URL: https://github.com/pedroSG94/rtmp-rtsp-stream-client-java/wiki?utm_source=android-arsenal.com&utm_medium=referral&utm_campaign=5333. (accesat februarie 2022).
- [9] Krishna Rao Vijayanagar. *What is FFmpeg? Usage, Benefits, and Installation Simplified*. 2021. URL: <https://ottverse.com/what-is-ffmpeg-installation-use-cases/>. (accesat mai 2022).
- [10] H. Parmar, H. Thornburgh, and Adobe. *Adobe’s Real Time Messaging Protocol*. 2012. URL: https://rtmp.veriskope.com/pdf/rtmp_specification_1.0.pdf. (accesat iunie 2022).
- [11] Traci Ruether. *2021 Video Streaming Latency Report*. 2021. URL: <https://www.wowza.com/blog/2021-video-streaming-latency-report#>. (accesat iunie 2022).
- [12] URL: <https://github.com/ant-media/Ant-Media-Server/releases>. (accesat mai 2022).



Figura 6.1: Raport Turnitin