Universitatea Tehnică de Construcții București

Bd. Lacul Tei 122-124, Sect. 2, cod poștal 020396, București, România
Tel.: +40-21-242.12.08, Tel./Fax: +40-21-242.07.81
secretariat@utcb.ro, www.utcb.ro

UNIVERSITATEA TEHNICĂ DE CONSTRUCȚII BUCUREȘTI
FACULTATEA DE HIDROTEHNICĂ
DOMENIUL INGINERIA SISTEMELOR
SPECIALIZAREA AUTOMATICĂ ȘI INFORMATICĂ APLICATĂ

LUCRARE DE LICENȚĂ

Sistem de securitate bazat pe recunoaștere facială

COORDONATOR: ABSOLVENT:

Șl. dr. ing. Ionela Halcu Tiberiu Marinică

București, 2020

Cuprins

Listă de figuri **iv**

1 Introducere **7**
1.1 Justificarea alegerii temei 7
1.2 Obiective generale 7
1.3 Descrierea problemei abordate și a metodei de rezolvare propuse 8
1.3.1 Problema 8
1.3.2 Soluția propusă 8
1.4 Structura lucrării 9

2 Fundamente teoretice **10**
2.1 Detecția facială folosind clasificatori în cascadă bazați pe caracteristici Haar 10
2.1.1 Descrierea problemei 10
2.1.2 Prezentarea generală a algoritmului și a caracteristicilor căutate . . 10
2.2 Multithreading 13
2.2.1 Ce este un Thread? 13
2.2.2 Threaduri vs. Procese 13
2.2.3 Prezentare concept multithreading 13
2.2.4 Avantajele multithreading-ului 13
2.3 Comunicarea pe socket 14
2.3.1 Modelul TCP/IP 14
2.3.2 Ce este un socket? 15
2.3.3 Modelul client-server 15
2.4 Advanced Message Queuing Protocol 17

3 Tehnologii hardware și software folosite în dezvoltarea proiectului **18**
3.1 Componente Hardware 18
3.1.1 Raspberry Pi 18
3.1.2 Raspberry Pi Camera 19
3.2 Tehnologii Software 21
3.2.1 Medii de dezvoltare 21
3.2.1.1 Eclipse 21
3.2.1.2 Notepad++ 22

3.2.1.3 Android Studio 22
3.2.2 Limbaje de programare 23
3.2.2.1 Java 23
3.2.2.2 Python 24
3.2.3 Biblioteci software 24
3.2.3.1 OpenCV 24
3.2.3.2 Spring 26
3.2.3.3 Maven 27
3.2.4 RabbitMQ 28
3.2.5 Oracle VM VirtualBox 29
3.2.6 VNC Viewer 31
3.2.7 Git 32
3.2.8 GitHub 32
3.2.9 Linux 33

4 Studiu de caz: ”Sistem de securitate bazat pe recunoaștere facială” 34
4.1 Sistemul de recunoaștere facială, supraveghere video și control de la distanță 35
4.1.1 Instalarea sistemului de operare 35
4.1.2 Conectarea automată la Wi-Fi 36
4.1.3 Acces de la distanță prin rețeaua wireless locală 36
4.1.4 Instalarea dependințelor software 39
4.1.5 Arhitectura internă 39
4.1.5.1 Diagrama de integrare cu restul componentelor 40
4.1.5.2 Scriptul de gestionare a comenzilor 40
4.1.5.3 Scriptul de gestionare a serviciilor 43
4.2 Sistemul de Live Streaming 47
4.2.1 Împachetare, distribuire și rulare 47
4.2.2 Arhitectură internă 48
4.3 Sistemul de comunicare între componente 52
4.3.1 Instalare 52
4.3.2 Configurare 52
4.3.2.1 Topologie 53
4.4 Aplicația Android 56
4.4.1 Ecranul principal 57
4.4.2 Primirea notificărilor 59
4.4.3 Ecranul de vizualizare a detecțiilor 61
4.4.4 Ecranul Live Stream 63
4.4.5 Ecranul de setări 66

5 Concluzii 67
5.1 Contribuții personale 67
5.2 Perspective de viitor 68

Bibliografie **69**

Listă de figuri

2.1 Exemplu de caracteristici rectangulare 11
2.2 Caracteristică Haar asemănătoare regiunii ochilor, aplicată pe o imagine ce conține o față 12
2.3 Caracteristică Haar asemănătoare regiunii nasului, aplicată pe o imagine ce conține o față 12
2.4 Descriere nivele TCP/IP 15
2.5 Legătura dintre Ip, Port și Socket 15
2.6 Exemplu de implementare model client-server 16

3.1 Raspberry Pi 4 19
3.2 Raspberry Pi Camera 20
3.3 Ciclul de viață al unui program Maven 28
3.4 Model de comunicare publish-subscribe în RabbitMQ 29

4.1 Configurare rețea wireless pentru Raspberry Pi 36
4.2 Configurare PuTTY pentru conectarea la Raspberry Pi 37
4.3 Configurare WinSCP pentru conectarea la Raspberry Pi 38
4.4 Arhitectură internă abstractă a sistemului de recunoastere facială, supraveghere video și control de la distanță 39
4.5 Diagrama de integrare Raspberry Pi cu restul sistemelor 40
4.6 Utilizare procesor Raspberry Pi cu ambele funcționalități pornite 41
4.7 Utilizare procesor Raspberry Pi doar cu funcționalitatea de detecție a feței pornită 42
4.8 Diagrama de clasă a scriptului de gestionare a serviciilor 43
4.9 Diagrama de activitate a funcției de detectare a feței 45
4.10 Captură ecran detectare corectă a feței 46
4.11 Diagrama de secvență a funcției de detectare a feței 47
4.12 Clasă ce modelează o comandă a serverului de streaming 48
4.13 Clasa prinicpală a serverului de streaming 49
4.14 Diagrama de secvență a serverului de Live Stream 50
4.15 Ecran de login RabbitMQ 53
4.16 Topologie de rutare a comenzilor către Raspberry Pi 53
4.17 Topologie de rutare a pozelor către telefoanele cu aplicația Android instalată 54

4.18 Implementare exchange de comenzi în RabbitMQ 55
4.19 Implementare coadă dinamică pentru poze în RabbitMQ 56
4.20 Diagrama de activitate a aplicației Android 57
4.21 Ecranul principal al Aplicației Android 58
4.22 Diagrama de clasă a ecranului principal 59
4.23 Diagrama de clasă a serviciului ce ascultă la mesaje RabbitMQ 60
4.24 Notificare față detectată 61
4.25 Ecranul de vizualizare a detecțiilor 62
4.26 Diagrama de clasă a ecranului de vizualizare a detecțiilor 63
4.27 Diagrama de clasă a ecranului de Live Stream 64
4.28 Ecranul de Live Stream 65
4.29 Ecranul de setări 66

Capitolul 1

Introducere

Inteligența artificială nu este un concept nou. Cercetarea în acest domeniu a început încă din 1955, dar nu a avut succesul meritat din cauza puterii de procesare slabe din acea vreme și din cauza seturilor de date mici, comparativ cu astăzi. Acest domeniu al informaticii a cunoscut o explozie și un interes considerabil în ultima decadă, alături de concepte precum *Machine Learning*, *Big Data*, *Cloud Computing*, realitate virtuală și augumentată, care încep să devină indispensabile în viața de zi cu zi.

Cu cât tehnoglogia avansează, ne dorim să o folosim astfel încât să eficientizăm și să automatizăm tot ce putem, rezultând conceptul de casă înteligentă (*Smart Home*). Această casă inteligentă este alcătuită dintr-o multitudine de dispozitive, capabile să comunice între ele, cât și să fie controlate de la distanță, într-un mod sigur, dar și eficient. Așa a luat naștere *Internet of Things*.

1.1 Justificarea alegerii temei

Am ales această temă deoarece conceptele menționate mai sus sunt în plină ascensiune, și folosirea lor în sisteme clasice (precum un sistem de supraveghere) nu doar va aduce noi funcționalități, dar va contribui la sporirea confortului și siguranței persoanelor, permițându-le astfel să câștige astfel timp prețios.

1.2 Obiective generale

Lucrarea de față își propune să îmbine inteligența artificială, conceptul de casă inteligentă, precum și securitatea într-un sistem ce are ca scop siguranța unei case, a unui apartament, a unui monument, etc.

Un alt obiectiv este acela de a depăși ca și funcționalități un sistem de siguranță și supraveghere clasic, aducând urmatoarele îmbunătățiri:

- detectarea persoanelor din preajmă folosind recunoașterea facială
- trimiterea notificărilor pe telefon în cazul persoanelor detectate

- live stream cu latență redusă
- posibilitate configurare sistem de pe telefon
- oprire și pornire funcționalități în funcție de utilizare, rezultând eficiența procesorului microcontrolerului

1.3 Descrierea problemei abordate și a metodei de rezolvare propuse

1.3.1 Problema

Un sistem de supraveghere clasic poate asigura înregistrarea și salvarea video, permițând utilizatorului fie să vadă în timp real faptele, fie să parcurgă arhivele, înregistrare cu înregistrare, secundă cu secundă și să tragă concluzii. Această parcurgere necesită timp prețios, de aceea este nevoie de o soluție mai eficientă la problema supravegherii.

1.3.2 Soluția propusă

Soluția prezentată în această lucrare elimină problema, întrucât utilizatorul nu este nevoit să urmarească imaginile video în timp real, deoarece algoritmul analizează cadru cu cadru imaginile video și dacă este detectată o persoană, sistemul notifică utilizatorul, acesta din urmă putând apoi, dacă dorește, să accesze funcționalitatea de live, si să ia decizii bazate pe concluziile trase (să încuie/descuie ușa, să pornească/oprească alarma - totul din aplicația de telefon ce însoțește sistemul).

Acest sistem se bazează pe microcontrolerul *Raspberry Pi*, care preia fluxul de imagini de la camera video atașată (*Raspberry Pi Camera*), și îl trimite către scriptul de procesare a imaginilor, scris în *Python*. Acest script se folosește de biblioteca *OpenCV*, care îi asigură utilizatorului, printre multe altele, funcționalitatea de detecție a feței. De îndată ce în imaginea curentă a fost detectată o față, transformă această imagine în text (pentru a putea fi trimisă cu ușurință pe internet) folosind encoding-ul *Base64*, iar apoi o trimite către brokerul de mesaje, *RabbitMQ*. Acesta din urmă se ocupă de multiplicarea imaginii și transmiterea către toți clienții conectați. (Un alt beneficiu al folosirii unui astfel de broker este acela ca permite conectarea indirectă a microcontrolerului și a aplicației de telefon, nefiind necesar ca acestea să fie în aceeași rețea, deoarece brokerul este gazduit public, pe internet, unde toate dispozitivele au acces). Imaginea ajunge în cele din urmă la client printr-o notificare pe telefon, urmând ca acesta să ia decizii bazate pe cele văzute.

Utilizatorul poate accesa imaginile live preluate de camera video direct din aplicația instalată pe telefonul personal. Atunci când această funcționalitate este accesată, aplicația de mobil trimite un mesaj către serverul de live streaming, care la rândul lui trimite un mesaj către *RabbitMQ*, care preia mesajul și îl trimite către microcontroler. Acesta pornește funcționalitatea de streaming live, direcționând fluxul către serverul de live stream, la care

este conectat telefonul, acesta din urmă primit imaginile în timp real. De îndată ce telefonul iese de pe funcționalitatea de live stream, este trimis un mesaj care îi spune microcontrolerului să oprească această funcționalitate. Tot din aplicația de mobil, utilizatorul poate alege calitatea streamingului live.

Pentru a oferi un plus de confort, dar și de siguranță, utilizatorul poate accesa de la distanță microcontrolerul (prin aplicația de mobil) și poate da comenzi acestuia, precum să încuie/descuie ușa sau să pornească/oprească alarma.

Centrul de comandă și vizualizare este aplicația de Android care însoțește sistemul. Din această aplicație utilizatorul poate controla microcontrolerul, poate vedea istoricul detecțiilor, poate vedea imagini în direct de la camera de luat vederi și puate lua decizii bazate pe informațiile acumulate.

1.4 Structura lucrării

Lucrarea este compusă din 4 capitole și un capitol de concluzii, unde sunt de asemenea prezentate contribuțiile personale și posibilități de dezvoltări ulterioare. Acest proiect reprezintă un sistem de securitate și streaming intelgient, bazat pe inteligență artificială, ce notifică utilizatorul iar acesta poate pe baza deciziilor să dea comenzi sistemului astfel încât să evite pericolul.

Al doilea capitol explică pe scurt fundamentele teoretice necesare întelegerii algoritmului de detectare facială bazată pe cascadele Haar, prezentând pas cu pas fiecare decizie a acestuia, și motivațiile din spate. Tot în acest capitol este prezentat conceptul de Multithreading (folosit la serverul de streaming live), fiind o tehnică de programare esențială când este vorba de programarea paralelă eficientă. În ultimul rând, este prezentată comunicarea pe socket, stackul TCP/IP, modelul de client-server cât și protocolul avansat de punerea mesajelor în cozi - AMQP (Advanced Message Queueing Protocol)

În al treilea capitol am prezentat tehnologiile folosite pentru dezvoltarea sistemelor , limbajele de programare folosite cât și echipamentele hardware utilizate. Printre acestea se numără Raspberri Pi, Python, Java, RabbitMQ, OpenCV, Spring, Linux și Eclipse.

Capitolul patru descrie în detaliu modul de implementare al sistemelor, cum comunică, motivarea deciziilor luate si arhitecturile interne și de integrare, diagramele de secvență, de clasă dar și diagramele cazurilor de utilizare.

Capitolul 2

Fundamente teoretice

În cele ce urmează va fi prezentat algoritmul care stă la baza detecției faciale, ce reprezintă un Thread și cum se pot folosi mai multe pentru a crește procesarea paralelă, cum pot două dispozitive să comunice în rețea prin Internet și protocolul care asigură comunicarea asincronă între componentele sistemului din prezenta lucrare.

2.1 Detecția facială folosind clasificatori în cascadă bazați pe caracteristici Haar

Algoritmul de detecție a obiectelor propus de Paul Viola și Michael Jones [25] este primul algoritm de acest timp care a reușit să crească rata de detecție în timp real a obiectelor, fiind astfel competitiv. Deși acest algoritm poate fi antrenat să detecteze o multitudine de clase de obiecte, motivul care a stat la baza concepției acestuia a fost problema detecției faciale. În continuare, se va prezenta algoritmul, la nivel general.

2.1.1 Descrierea problemei

Problema ce se cere a fi rezolvată este detectarea prezenței unei fețe într-o imagine. Deși un om poate face acest lucru cu ușurință, unui calculator îi este foarte greu deoarece îi sunt necesare instrucțiuni și constrângeri precise.

2.1.2 Prezentarea generală a algoritmului și a caracteristicilor căutate

Algoritmul are patru etape:

1. Selecția caracteristicii Haar
2. Creearea unei imagini integrale
3. Antrenarea Adaboost

4. Aplicarea clasificatorilor în cascadă

Caracteristicile căutate de algoritmul de detecție implică suma pixelilor dintr-o regine rectangulară a unei imagini. Imaginea următoare ilustrează patru tipuri diferite de caracteristici folosite de algoritm. Valoarea unei caracteristici arbitrare este diferența dintre suma pixelilor din regiunea deschisă și suma pixelilor din regiunea întunecată.

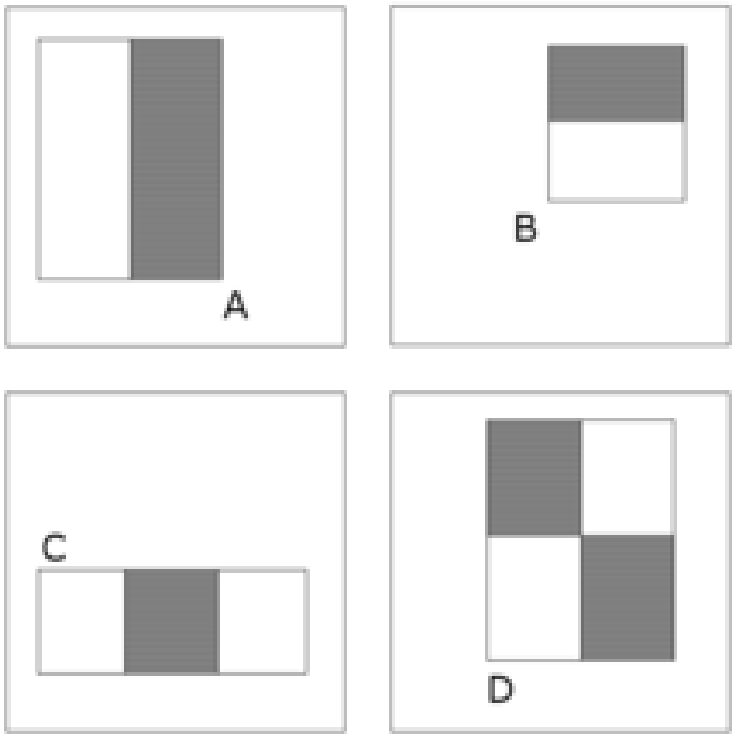

Figura 2.1: *Exemplu de caracteristici rectangulare*

Toate dimensiunile și locațiile posibile ale fiecărui kernel sunt utilizate pentru a calcula o mulțime de caracteristici. Pentru acest lucru este nevoie de o putere de calcul foarte mare. (De exemplu, pentru o fereastră de doar 24x24 pixeli, vor rezulta peste 160.000 caracteristici). Pentru calcului mai rapid al unei caracteristici, s-a introdus conceptul de *Integral image* [19], care simplifică calculul sumei de pixeli (indiferent cât de mare este fi numărul de pixeli), la o operație care implică doar patru pixeli.

Din toate caracteristicile calculate, majoritatea nu sunt relevante. În imaginile de mai jos, sunt prezentate anumite caracteristici prezente de obicei la majoritatea fețelor oamenilor. În prima imagine se observă că regiunea ochilor este mai întunecată decât regiunea obrajilor, iar în a doua imagine, regiunea obrajilor este mai întunecată decât regiunea nasului.

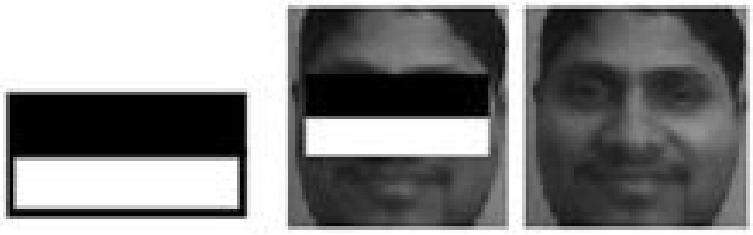

Figura 2.2: *Caracteristică Haar asemănătoare regiunii ochilor, aplicată pe o imagine ce conține o față*

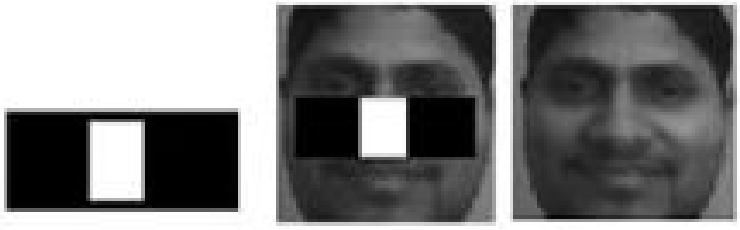

Figura 2.3: *Caracteristică Haar asemănătoare regiunii nasului, aplicată pe o imagine ce conține o față*

Aceste două caracteristici aplicate doar în aceste două zone sunt relevante, din orice altă mutare nu obținem nicio informație utilă. Pentru a selecta cele mai relevante caracteristici din cele peste 160.000, se foloseste algoritmul *Adaboost* [23].

Acest algoritm aplica fiecare caracteristică peste toate imaginile de antrenament. Pentru fiecare caracteristică, găsește cel mai bun prag care va clasifica imaginea curentă în pozitiv sau negativ (adică, conține sau nu o față). Se vor selecta caracteristicile cu eroare minimă. Clasificatorul final este o sumă ponderată a clasificatorilor slabi. Se numesc slabi deoarece nu pot clasifica singuri o imagine, dar împreună formează un clasificator puternic. Articolul [25] afirmă ca, chiar și cu 200 caracteristici se va obține o acuratețe de 95%. Algoritmul lor avea aproximativ 6000 caracteristici, reduse de la cele peste 160.000.

După ce s-au stabilit caracteristicile relevante, chiar dacă numărul lor a fost redus semnificativ, tot va fi nevoie de mult timp si putere de procesare pentru a trece prin toate ferestrele posibile. Pentru această problemă, soluția a fost o *Cascadă de Clasificatori* [25].

În loc să aplice toate cele peste 6000 caracteristici pe o anume fereastră, au fost grupate caracteristicile în diferite etape de clasificare și au fost aplicate una cate una. Dacă o fereastră nu va reuși să treacă de prima etapa de clasificatori, aceasta va fi înlaturată, și nu se vor lua în calcul restul caracteristicilor pentru ea. Dacă va trece, se vor aplica caracteristicile din cea de-a doua etapă, repetând procedura până când o anumită feareastră trece de toate etapele. Atunci când o fereastră obține acest rezultat, rezultă ca acea regiune este o regiune în care este prezentă o față. [1, 11]

2.2 Multithreading

2.2.1 Ce este un Thread?

Un *Thread* (sau, tradus, 'fir de execuție'), reprezintă cea mai mică secvență de instrucțiuni care poate fi gestionată de schedulerul sistemului de operare. Implementarea threadurilor și a proceselor diferă de la sistem de operare la sistem de operare, dar în majoritatea cazurilor, un thread este componentă a unui proces. Mai multe threaduri pot exista într-un anume proces, executând instructiuni concurent și împarțind memorie, în timp ce diferite procese nu pot împarți zone de memorie.

2.2.2 Threaduri vs. Procese

Thredurile diferă de procesele ce aparțin în general sistemelor de operare multitasking tradiționale, prin următoarele:

- procesele sunt independente, în timp ce threadurile există ca subseturi ale unui proces
- procesele dețin mult mai multă informație de stare decât threadurile, în timp ce acestea din urmă împart informațiile de stare și diverse alte resurse în cadrul procesului din care fac parte
- procesele au spații de adrese separate, în timp ce threadurile împart spațiul de adrese
- procesele interacționează doar prin mecanisme de comunicare inter-proces, oferite de sistemul de operare

2.2.3 Prezentare concept multithreading

Principiul de multithreading presupune execuția mai multor thread-uri în același pipeline, fiecare având propria secțiune de timp în care este menit să lucreze. Odată cu creșterea capabilităților procesoarelor au crescut și cererile de performanță, asta ducând la solicitarea la maxim a resurselor unui procesor. Necesitatea multithreading-ului a venit de la observația că unele procesoare puteau pierde timp prețios în așteptarea unui eveniment pentru o anumită sarcină.

Foarte repede a fost observat potențialul principiului de paralelizare a unui proces, atât la nivel de instrucțiune, cât și la nivel de fir de execuție. Firul de execuție sau thread-ul este un mic proces sau task, având propriile instrucțiuni și date.

2.2.4 Avantajele multithreading-ului

Ca și aplicabilitate, multithreading-ul poate fi folosit pentru sporirea eficienței atât în cadrul multiprogramării sau a sarcinilor de lucru pe mai multe fire de execuție, cât și în cadrul unui singur program. Astfel, un fir de execuție poate rula în timp ce alt fir de execuție

așteaptă un anumit eveniment. În ziua de azi, capacitățile oferite de acest tip de procesare sunt folosite până și la nivelul sistemului de operare.

Pentru a putea beneficia de avantajele multithreading-ului, un program trebuie să poată fi despărțit în secțiuni ce pot rula independent și în mod paralel, fiind foarte greu de utilizat în codul reprezentat de o înșiruire foarte lungă de instrucțiuni. Gestiunea firelor de execuție este controlată de sistemul de operare. Un exemplu de implementare a acestui concept este modalitatea prin care Microsoft Word repaginează un document în timp ce utilizatorul scrie.

Conceptul de multithreading poate fi folosit și pentru accelerarea unui singur thread prin utilizarea, atunci când procesorul nu este foarte solicitat, a execuției speculative pe mai multe căi și a firelor de execuție ajutătoare. [20, 21]

2.3 Comunicarea pe socket

2.3.1 Modelul TCP/IP

[24] TCP/IP (Transmission Control Protocol/Internet Protocol) este cel mai utilizat protocol folosit în reţelele locale cât şi pe Internet datorită flexibilităţi lui, având cel mai mare grad de corecţie al erorilor. TCP/IP permite comunicarea între calculatoarele din întreaga lume indiferent de sistemul de operare instalat.

Dezvoltarea acestui model a început în anii 1960 sub forma unui proiect finanţat de SUA prin DARPA.

Avantajele modelului TCP/IP:

- independența de producator
- nu este protejat prin legea copyright-ului
- se poate utiliza atât pentru retele locale (LAN), cât si pentru retele globale (WAN)
- se poate utiliza pe aproape orice tip de calculator
- asigură transferul pachetelor de date cu o rată foarte mică de eroare printr-o reţea neomogenă de calculatoare.

Modelul TCP/IP are patru niveluri (straturi, stive). În tabelul de mai jos este prezentată descrierea fiecărui nivel în parte:

Nivelul TCP/IP	Descriere
Aplicație	Prezintă și functionează protocoalele de nivel înalt, rezolva probleme de reprezentare, codificare şi control al dialogului.
Transport	Rezolvă problemele legate de performanţele sistemului, controlul fluxului şi corectarea erorilor.
Internet	Se realizează adresarea IP și transmiterea pachetelor de la sursă din orice reţea a interconectării şi sosirea lor la destinaţie independent de calea urmată. De asemenea, realizeaza determinarea căii (traseului) optime (optim) de la sursă la destinație.
Acces Rețea	Se realizeaza adresarea dupa MAC. Include detalii ale tehnologiei LAN si WAN si toate atributele caracteristice nivelului fizic si legatură de date din modelul OSI

Figura 2.4: *Descriere nivele TCP/IP*

2.3.2 Ce este un socket?

[24] Multiplexarea presupune ca două sau mai multe procese de pe același dispozitiv (calculator) să acceseze simultan resursele rețelei. Pentru aceasta se realizează o asociere între un număr de port și adresa IP. Această asociere se mai numeşte socket.În acest fel multiplexarea se realizează prin porturi.

În concluzie, un socket este o structură software folosită pentru a reprezenta fiecare din cele două capete" ale unei conexiuni între două procese ce rulează într-o rețea. Fiecare socket este atașat unui port astfel încât să poată identifica unic programul căruia îi sunt destinate datele.

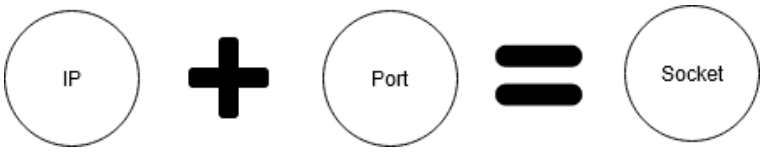

Figura 2.5: *Legătura dintre Ip, Port și Socket*

2.3.3 Modelul client-server

[2] Serverul este o aplicație care oferă servicii clienților sosiți prin rețea. Serverele oferă o gamă variată de servicii. Serverul cel mai cunoscut este serverul Web, care furnizează documentele cerute de către clienți. Un alt serviciu cunoscut este poșta electronică, care utilizează protocoalele SMPT (Simple Mail Transfer Protocol) și IMAP (Internet Mail Access Protocol). Pe principiul client-server funcționează și protocoalele NFS (Network File Service) și FTP (File Transfer Protocol) sau serviciul de nume de domenii DNS (Domain Name

Service) și serviciul de informatii despre retea NIS (Network Information Services). Trebuie să amintim și serviciul care permite logarea la un calculator aflat la distanță: TELNET și RLOGIN. Putem trage concluzia ca arhitectura client-server este instrumentul de bază în dezvoltarea aplicațiilor de rețea.

Clientul este o aplicație care utilizează serviciile oferite de către un server. Pentru a putea realiza acest lucru, clientul trebuie să cunoască unde se află serverul în rețea și cum trebuie comunicat cu acesta și ce servicii oferă. Deci dacă un client dorește o comunicare cu serverul, trebuie să cunoscă trei lucruri:

- adresa serverului
- portul serverului utilizat pentru comunicare
- protocolul de comunicație utilizat de server

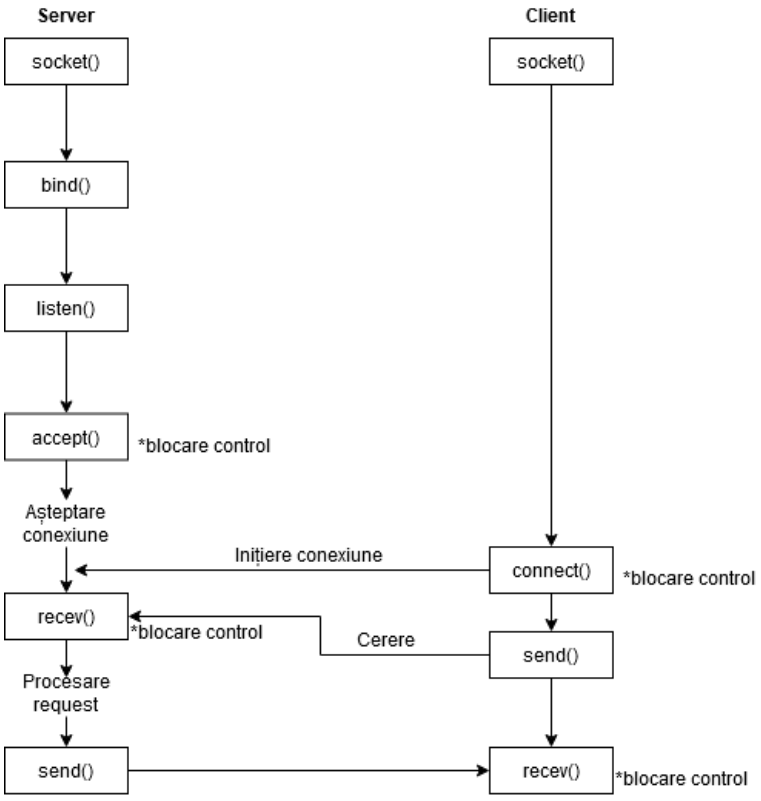

Figura 2.6: *Exemplu de implementare model client-server*

[22] Modelul client-server este o structură sau arhitectură aplicație distribuită care partajează procesarea între furnizorii de servicii numiți servere și elementele care solicită servicii, numite clienți. Clienții și serverele comunică printr-o rețea de calculatoare, de obicei prin Internet, având suporturi hardware diferite, dar pot rula și pe același sistem fizic. Un

server (fizic) rulează unul sau mai multe programe server, care partajează resursele existente cu clienții. Clientul nu partajează niciuna dintre resursele proprii, ci apelează la resursele serverului prin funcțiile server. Clienții inițiază comunicația cu serverele și așteaptă mesajele acestora. Pentru menținerea legăturii între cei doi, indiferent de pauzele care intervin, se folosește conceptul de sesiune, care de obicei este limitată în timp.

2.4 Advanced Message Queuing Protocol

AMQP (Advanced Message Queuing Protocol) [3] este un protocol deschis la nivel de aplicație care facilitează comunicarea componentelor în cadrul unui sistem distribuit. Caracteristicile acestui protocol sunt:

- orientarea pe mesaje
- procesare bazată pe cozi
- rutare (inclusiv point-to-point și publish-and-subscribe)
- siguranță ridicată

AMQP încearcă să creeze sisteme interoperabile, precum și protocoalele SMTP, HTTP, FTP fac acest lucru. Standardizările din trecut ale comunicării în cadrul sistemelor distribuite au avut loc la nivelul API (Application Programming Interface) - spre exemplu, Java Message Service (JMS), și erau orientate către standardizarea interacțiunii dintre programator și implementarea comunicării, în loc să asigure interoperabilitatea mutiplelor implementări. AMQP (spre deosebire de JMS) este un protocol la nivel de fir. Acest tip de protocol este o descriere a formatului datelor care sunt trimise prin rețea ca un flux de octeți.

Capitolul 3

Tehnologii hardware și software folosite în dezvoltarea proiectului

3.1 Componente Hardware

3.1.1 Raspberry Pi

Raspberry Pi este o serie de SBC (Single-board computer) de dimensiunile unui card de credit. Este inspirat de BBC Micro și produs în UK de către Raspberry Pi Foundation. Scopul a fost acela de a crea un dispozitiv cu costuri reduse care să îmbunătățească abilitățile de programare și înțelegerea hardware la nivel preuniversitar. Raspberry Pi este mai lent decât un laptop sau PC, dar care poate oferi majoritatea aplicațiilor acestora precum conectare la internet, procesare de text, redare de conținut video/audio, jocuri video, la un nivel de consum redus de energie. În plus, Raspberry Pi are o caracteristică specială pe care computerele nu o folosesc: port generic de intrare/ieșire (General-Purpose Input/Output)(GPIO). Acesta oferă posibilitatea de a conecta diverse componente electronice specifice sistemelor înglobate: senzori, butoane, ecran LCD, relee, și crearea de noi proiecte electronice. De la prima apariție pe piață în 2012, au fost vândute 18 milioane de plăci de Raspberry Pi, în diferite modele.

Sistemul de operare special optimizat pentru Raspberry Pi este Raspbian, preinstalat pe dispozitive. Există trei versiuni ale Raspbian: Wheezy, Jessie și Stretch. Derivat din Debian, acesta oferă pe lângă funcțiile de bază ale nucleului, aplicații cum ar fi browserul Chromium, Python, Scratch, Sonic Pi, RealVNC, NodeRED, Geany, Wolfram, Java și peste 35000 de alte pachete, software precompilat, toate aranjate într-o manieră ușor de instalat și utilizat.

Caracteristici Raspberry Pi 4:

- SoC Broadcom BCM2837B32
- Procesor 4 nuclee ARM Cortex-A53, 1,4 GHz (64/32-bit)
- 1GB de memorie RAM (folosită și ca memorie video, partajată cu procesorul grafic)

- Procesor grafic Broadcoam VideoCore IV 3D
- Ieșire digitală video/audio HDMI
- Ieșire analogică video (composite video)/audio mixtă prin intermediul unei mufe jack 3,5mm
- Mufă de rețea RJ45 Ethernet 10/100 Mbit/s
- WiFi 802.11n
- Bluetooth 4.1 BLE
- 4 porturi USB 2.0
- GPIO 40 de pini

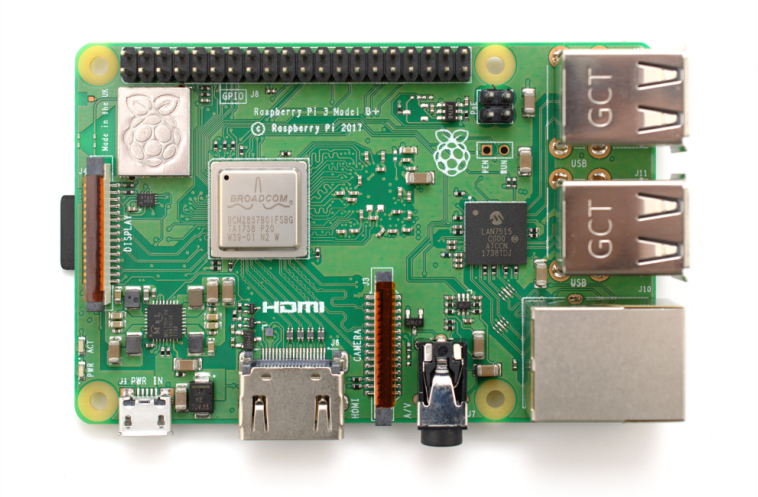

Figura 3.1: *Raspberry Pi 4*

3.1.2 Raspberry Pi Camera

Acest modul cu camera de 8 MP este un senzor de imagine Sony IMX219 capabil sa filmeze in format 1080p30, 720p60 și 640x480p60/90. Se conectează direct la Raspberry Pi prin portul CSI (Camera Serial Interface) al Raspberry Pi utilizând cablul panglică inclus. Spre deosebire de camera obișnuită, această cameră are filtrul infraroșu înlăturat, permițând înregistrarea în infraroșu.

Camera are dimensiuni de 25 mm x 23 mm x 9 mm și cântărește doar putin peste 3 g, fiind perfectă pentru aplicațiile mobile unde dimensiunile și greutatea sunt importante. Camera dispune de lentile încorporate cu focalizare fixă și un senzor cu rezoluție nativa de 8 MP,

fiind capabilă să captureze imagini statice de până la 3280 x 2464 pixeli și să filmeze în format 1080p30, 720p60 și 640x480p60/90. Este compatibila cu toate modelele de Raspberry Pi.

Caracteristici tehnice:

- Compatibil cu Raspberry Pi;
- Rezoluție foto: 8 MP;
- Rezoluție video: 1080p 30fps, 720p 60fps, 640x480p 60/90fps;
- Conexiune CSI (camera serial interface);

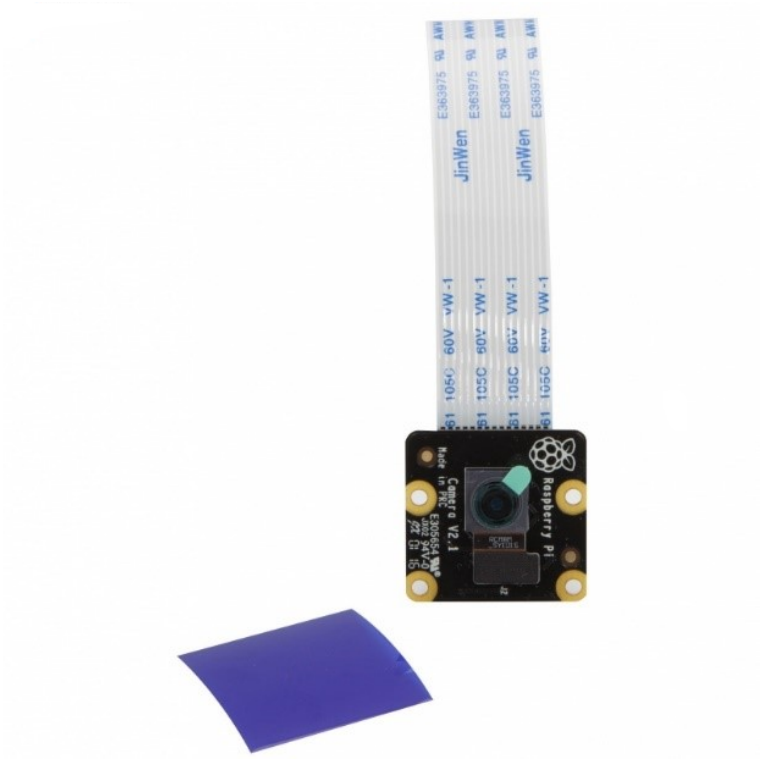

Figura 3.2: *Raspberry Pi Camera*

3.2 Tehnologii Software

3.2.1 Medii de dezvoltare

3.2.1.1 Eclipse

Eclipse [4] este un mediu de dezvoltare open-source scris preponderent în Java. Acesta poate fi folosit pentru a dezvolta aplicații Java și, prin intermediul unor plug-in-uri, în alte limbaje, cum ar fi C, C++, COBOL, Python, Perl și PHP. De dezvoltarea sa se ocupă Fundația Eclipse.

Eclipse este platforma extinsă a clientului (RCP - eng. rich client platform). Este compusă din următoarele componente:

- Nucleul platformei (încărcare Eclipse, module de lansare);
- OSGi (mediu standard de livrare)
- SWT (set de instrumente widget portabile)
- JFace (procesare de text, editoare de text)
- Eclipse mediu de lucru (panouri, editori, proiecții)

GUI în Eclipse este scris folosind setul de instrumente SWT. Acesta din urmă, spre deosebire de Swing (care emite independent controale grafice), utilizează componentele grafice ale sistemului de operare dat. Interfața de utilizator Eclipse depinde, de asemenea, de un strat intermediar GUI numit JFace, care simplifică construirea unei interfețe utilizator bazate pe SWT.

Eclipse utilizează plugin-uri pentru a furniza toate funcționalitățile din interiorul sistemului de rulare, cât și funcționalități ce se bazează pe el. Sistemul său de rulare se bazează pe Equinox, o implementare a specificației a nucleului OSGi.

Pe lângă faptul că permite extinderea platformei Eclipse folosind alte limbaje de programare, cum ar fi C și Python, cadrul de conectare permite platformei Eclipse să lucreze cu limbaje precum LaTeX și aplicații de rețea, cum ar fi sistemele de gestionare a bazelor de date și telnet. Arhitectura plug-in acceptă scrierea oricărei extensii dorite în mediu, cum ar fi extensiile pentru gestionarea configurației. Asistența Java și CVS este oferită în Eclipse SDK, cu suport pentru alte sisteme de control de versiuni furnizate de plugin-uri terțe.

Cu excepția unui mic nucleu de rulare, totul în Eclipse este un plug-in. Astfel, fiecare plug-in dezvoltat se integrează cu Eclipse în același mod ca și alte plugin-uri; în acest sens, toate caracteristicile sunt asemănătoare. Eclipse oferă plugin-uri pentru o mare varietate de caracteristici, dintre care unele sunt de la terți utilizând atât modele gratuite, cât și modele comerciale. Exemple de plugin-uri includ limbajul de modelare unificat (UML), un plugin pentru DB Explorer și multe altele.

Eclipse SDK include instrumentele de dezvoltare Eclipse Java (JDT), care oferă un IDE cu un compilator incremental Java încorporat și un model complet al fișierelor sursă Java.

Aceasta permite tehnici avansate de refactorizare și analiza codului. IDE folosește, de asemenea, un spațiu de lucru, în acest caz, un set de metadate peste un spațiu de fișier plat, care permite modificări de fișiere externe, atât timp cât resursa corespunzătoare a spațiului de lucru este actualizată ulterior.

Începând cu 2017, pachetele lingvistice dezvoltate de Proiectul Babel furnizează traduceri în peste 40 de limbi naturale.

3.2.1.2 Notepad++

Notepad++ [5] este un editor de text gratuit pentru Windows, găzduit la Sourceforge. Programul permite editarea codului sursă specific unui număr mare de limbaje de programare, precum C, C++, Pascal, Cobol, HTML, PHP.

Facilități remarcabile sunt:

- Colorare sintactică pentru 48 de limbaje de programare (se poate modifica de către utilizator)
- Tipărirea color a codului sursă (WYSIWYG);
- Autocompletare = deducerea și întregirea automată a cuvântului de cod din biblioteca utilizată
- Interfață cu mai multe ferestre (utilizează tab-uri);
- Suport pentru căutări/înlocuiri cu expresii regulate PERL;
- Suport pentru macroinstrucțiuni.

Notepad++ se bazează pe motorul Scintilla, fiind programat în C++ și utilizând Win32 API și biblioteca STL, asigurând astfel o execuție rapidă și un consum redus de resurse.

3.2.1.3 Android Studio

Android Studio [6] este un mediu de dezvoltare () pentru colaborarea cu platforma Android, anunțat pe data de 16 mai 2013 în cadrul conferinței I/O Google.

IDE-ul este disponibil gratuit începând cu versiunea 0.1, publicată în mai 2013, apoi a trecut la testarea beta, începând cu versiunea 0.8, care a fost lansată în iunie 2014. Prima versiune stabilă 1.0 a fost lansată în decembrie 2014, apoi suportul pentru pluginul Android Development Tools (ADT) pentru Eclipse a încetat.

Android Studio este bazat pe software-ul IntelliJ IDEA de la JetBrains, este instrumentul oficial de dezvoltare a aplicațiilor Android. Acest mediu de dezvoltare este disponibil pentru Windows, OS X și Linux. Pe 17 mai 2017, la conferința anuală Google I/O, Google a anunțat asistență pentru limbajul Kotlin utilizat de Android Studio ca limbaj de programare oficial pentru platforma Android, pe lângă Java și C++.

Pentru dezvoltarea și testarea aplicației de mobil, am folosit emulatorul Android. Dincolo de cerințele de bază ale aplicației Android Studio, Emulatorul Android are cerințe suplimentare care sunt descrise mai jos:

- SDK Tools 26.1.1 sau o versiune mai actuală
- Procesor cu arhitectura 64-bit
- Windows: CPU cu suport UG (unrestricted guest)
- HAXM 6.2.1 sau o versiune mai actuală (este recomandat HAXM 7.2.0 sau o versiune mai actuală)

Utilizarea accelerației hardware are cerințe suplimentare pentru Windows și Linux:

- Intel procesor în cazul sistemelor de operare Windows sau Linux: Intel procesor cu sprijin pentru Intel VT-x, Intel EM64T (Intel 64), și funcționalitatea Execute Disable (XD) Bit
- AMD procesor în cazul sistemei de operare Linux: AMD procesor cu sprijin pentru AMD Virtualization (AMD-V) și Supplemental Streaming SIMD Extensions 3 (SSSE3)
- AMD procesor în cazul sistemei de operare Windows: Android Studio 3.2 sau o versiune mai actuală și Windows 10 April 2018 release sau o versiune mai ctuală pentru funcționalitatea Windows Hypervisor Platform (WHPX)

Pentru a lucra cu Android 8.1 (API level 27) și cu imagini de sistem mai superioare, camera web atașată trebuie să aibă rezoluția de 720p.

3.2.2 Limbaje de programare

3.2.2.1 Java

Java [7] este un limbaj de programare orientat-obiect, puternic tipizat, conceput de către James Gosling la Sun Microsystems (acum filială Oracle) la începutul anilor 90, fiind lansat în 1995. Cele mai multe aplicații distribuite sunt scrise în Java, iar noile evoluții tehnologice permit utilizarea sa și pe dispozitive mobile, spre exemplu telefon, agenda electronică, palmtop etc. În felul acesta se creează o platformă unică, la nivelul programatorului, deasupra unui mediu eterogen extrem de diversificat. Acesta este utilizat în prezent cu succes și pentru programarea aplicațiilor destinate intranet-urilor.

Limbajul împrumută o mare parte din sintaxă de la C și C++, dar are un model al obiectelor mai simplu și prezintă mai puține facilități de nivel jos. Un program Java compilat, corect scris, poate fi rulat fără modificări pe orice platformă care e instalată o mașină virtuală

Java (engleză Java Virtual Machine, prescurtat JVM). Acest nivel de portabilitate (inexistent pentru limbaje mai vechi cum ar fi C) este posibil deoarece sursele Java sunt compilate într-un format standard numit cod de octeți (engleză byte-code) care este intermediar între codul mașină (dependent de tipul calculatorului) și codul sursă.

Mașina virtuală Java este mediul în care se execută programele Java. În prezent, există mai mulți furnizori de JVM, printre care Oracle, IBM, Bea, FSF. În 2006, Sun a anunțat că face disponibilă varianta sa de JVM ca open-source.

3.2.2.2 Python

Python [8] este un limbaj de programare dinamic multi-paradigmă, creat în 1989 de programatorul olandez Guido van Rossum. Van Rossum este și în ziua de astăzi un lider al comunității de dezvoltatori de software care lucrează la perfecționarea limbajul Python și implementarea de bază a acestuia, CPython, scrisă în C. Python este un limbaj multifuncțional folosit de exemplu de către companii ca Google sau Yahoo! pentru programarea aplicațiilor web, însă există și o serie de aplicații științifice sau de divertisment programate parțial sau în întregime în Python.

Popularitatea în creștere, dar și puterea limbajului de programare Python au dus la adoptarea sa ca limbaj principal de dezvoltare de către programatori specializați și chiar și la predarea limbajului în unele medii universitare. Din aceleași motive, multe sisteme bazate pe Unix, inclusiv Linux, BSD și Mac OS X includ din start interpretatorul CPython.

Python pune accentul pe curățenia și simplitatea codului, iar sintaxa sa le permite dezvoltatorilor să exprime unele idei programatice într-o manieră mai clară și mai concisă decât în alte limbaje de programare ca C. În ceea ce privește paradigma de programare, Python poate servi ca limbaj pentru software de tipul object-oriented, dar permite și programarea imperativă, funcțională sau procedurală. Sistemul de tipizare este dinamic iar administrarea memoriei decurge automat prin intermediul unui serviciu gunoier" (garbage collector). Alt avantaj al limbajului este existența unei ample biblioteci standard de metode.

3.2.3 Biblioteci software

3.2.3.1 OpenCV

OpenCV [9] (Open Source Computer Vision Library) este o bibliotecă software de viziune computerizată și software de învățare automată. OpenCV a fost construit pentru a furniza o infrastructură comună pentru aplicațiile de viziune computerizată și pentru a accelera utilizarea percepției mașinilor în produsele comerciale. Fiind un produs autorizat BSD, OpenCV facilitează utilizarea și modificarea codului pentru întreprinderi.

Biblioteca are mai mult de 2500 de algoritmi optimizați, care include un set cuprinzător de algoritmi de viziune computerizată de ultimă generație și de învățare automată. Acești algoritmi pot fi folosiți pentru detectarea și recunoașterea fețelor, identificarea obiectelor, clasificarea acțiunilor umane în videoclipuri, urmărirea mișcărilor camerei, urmărirea obiectelor în mișcare, extragerea de modele 3D ale obiectelor, producerea unor nori de puncte 3D din

camerele stereo, cusături de imagini pentru a produce o imagine de rezoluție înaltă a unei scene întregi; găsire imagini similare dintr-o bază de date de imagini, eliminare ochi roșii din imaginile luate cu ajutorul flash-ului, urmărirea mișcării ochilor, recunoașterea peisajului și stabilirea markerilor pentru a-l suprapune cu realitatea augmentată etc.

OpenCV are peste 47 de mii de utilizatori în comunitate și numărul estimat de descărcări depășesc 18 milioane. Biblioteca este utilizată pe scară largă în companii, grupuri de cercetare și de către organisme guvernamentale.

Alături de companii bine înființate precum Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota care utilizează biblioteca, există multe startup-uri precum Applied Minds, VideoSurf și Zeitera, care utilizează pe scară largă OpenCV. Utilizările ”în teren” se îndind de la cusut imagini de streetview împreună, detectând intruziuni în imaginile de supraveghere în Israel, monitorizarea echipamentelor miniere din China, ajutarea roboților să navigheze și să ridice obiecte la Willow Garage, detectarea accidentelor de înnec în piscină în Europa, rularea artelor interactive în Spania și New York, verificarea pistele pentru resturi în Turcia, inspectarea etichetele produselor din fabricile din întreaga lume, până la detectarea rapidă a feței în Japonia.

În concluzie, OpenCV este folosit la:

- Instrumente pentru detalii 2D și 3D
- Estimarea ego-mișcării
- Sisteme de recunoaștere facială
- Recunoașterea gesturilor
- Interacțiune om-calculator (Human Computer Interaction)
- Înțelegerea mișcării
- Identificarea obiectelor
- Segmentarea și recunoașterea imaginilor
- Vederea stereoscopică: percepția adâncimii cu 2 camere video
- Structură din mișcare (Structure From Motion)
- Urmărirea mișcării
- Realitate augmentată

3.2.3.2 Spring

Spring Framework [10] este o bibliotecă ce ușurează dezvoltarea aplicațiilor Java foarte mari, dar și un container de inversare a controlului. Caracteristicile de bază ale bibliotecii pot fi utilizate de orice aplicație Java, dar există extensii pentru construirea aplicațiilor web peste platforme Java EE (Enterprise Edition). Deși Spring nu impune niciun model de programare specific, el a devenit popular în comunitatea Java ca o completare sau chiar înlocuire a modelului Enterprise JavaBeans (EJB). Spring este open source.

Spring include multiple module care asigură o multitudine de servicii, precum:

- Spring Core Container: acesta este modulul de bază al Spring și furnizează containere (BeanFactory și ApplicationContext)
- Programare orientată pe Aspecte
- Autentificare și Autorizare: procese de securitate configurabile care acceptă o serie de standarde, protocoale, instrumente și practici prin intermediul subproiectului Spring Security
- Convenții utile care elimină configurările uzuale
- Acces la baze de date: lucrul cu sisteme de gestionare a bazelor de date relaționale pe platforma Java utilizând Java Database Connectivity (JDBC) și instrumente de mapare relațională cu obiecte și cu baze de date NoSQL
- Container de Inversiune a Controlului: configurarea componentelor aplicației și gestionarea ciclului de viață a obiectelor Java, realizată în principal prin injecția de dependințe
- Gestionarea tranzacțiilor: unifică mai multe API-uri de gestionare a tranzacțiilor și coordonează tranzacțiile pentru obiecte Java
- Testare: clase ajutătoare pentru scrierea de teste unitare și teste de integrare

Elementul central al Spring Framework este containerul de Inversare a Controlului (Inversion of Control), care oferă un mijloc consistent de configurare și gestionare a obiectelor Java cu ajutorul reflecției. Containerul este responsabil de gestionarea ciclurilor de viață ale obiectelor specifice: crearea acestor obiecte, apelarea metodelor lor de inițializare și configurarea acestor obiecte prin legarea lor.

Obiectele create de container se mai numesc obiecte gestionate sau 'beans'. Containerul poate fi configurat încărcând fișiere XML (Extensible Markup Language) sau detectând anumite adnotări Java la clasele de configurare. Aceste surse de date conțin definițiile de beanurilor care furnizează informațiile necesare pentru a crea beanuri.

Obiectele pot fi obținute fie prin căutarea dependinței, fie prin injecția dependinței. Căutarea dependințelor este un model în care un apelant solicită obiectului container un obiect cu un nume specific sau un tip specific. Injecția de dependință este un model în care containerul

transmite obiecte către alte obiecte, fie prin constructori, proprietăți, fie prin metode de fabrică.

În multe cazuri, nu este necesar să folosiți containerul atunci când utilizați alte părți din Spring Framework, deși utilizarea acestuia probabil va face o aplicație mai ușor de configurat și personalizat. Containerul Spring oferă un mecanism consistent de configurare a aplicațiilor și se integrează cu aproape toate mediile Java, de la aplicații la scară mică până la aplicații mari pentru întreprinderi.

3.2.3.3 Maven

Maven [12] este un sistem de build și administrare a proiectelor, scris în Java. Face parte din proiectele găzduite de Apache Software Foundation. Funcționalitățile sale principale sunt descrierea procesului de build al software-ului și descrierea dependențelor acestuia. Proiectele sunt descrise printr-unul sau mai multe fișier XML, denumite POM-uri (Project Object Model), dar au o structură implicită, ceea ce încurajează structurarea similară a proiectelor. POM-ul principal conține informații despre module, precum și despre dependențele proiectului (alte proiecte). Ordinea operațiunilor de build este definită prin declararea unor pluginuri folosite, din cadrul cărora unele goaluri sunt plasate și configurate în diferitele faze predefinite din ciclul de viață al unui build. Maven descarcă dinamic bibliotecile Java si pluginurile, din unul sau mai multe repository-uri.

Conceptul central în Maven este acela de POM (Project Object Model), un document XML care descrie un proiect sau un modul al acestuia. Proiectele multimodulare au un POM pentru fiecare modul, precum și un POM general care le agregă. În mod minimal, POMul unui proiect îi definește o denumire unică (formată din groupId și artifactId) și o versiune. Componentele denumirilor trebuie să asigure unicitate proiectului, astfel că s-a răspândit pe scară largă convenția ca groupId să fie inversul unui nume complet calificat de domeniu, la care se adaugă elemente comune unui set de proiecte ale persoanei sau firmei care le dezvoltă, similar cu numele package-ului în care sunt grupate clasele Java. groupId poate fi sau nu identic cu un package părinte al claselor din proiect.

Un proiect care definite aceste denumiri și versiunea este un proiect minimal pe care Maven îl poate builda fără alte elemente de bază, inferând toate celelalte configurații necesare din convenții, implementate într-un așa-numit Super POM". Proiectele din viața reală mai au însă și alte elemente incluse în POM, care completează sau suprascriu unele configurații ale Super POM-ului.

Pentru un proiect Java, convențiile sunt acelea de a compila toate sursele din subdirectorul src/main/java, de a alătura fișierelor bytecode rezultate toate fișierele din src/main/resources și apoi de a le arhiva într-un fișier cu extensia jar. Se compilează apoi testele automate definite în src/test/java și se rulează având în classpath și resursele statice (fișierele) din src/test/resources. Dacă toți pașii au reușit, la final, în subdirectorul target al directorului proiectului va apărea fișierul artifactId-version.jar.

Artifactele se produc, în urma compilării și apoi împachetării, implicit în subdirectorul target al directorului de build, și ulterior se depun într-un repository, o structură de directoare din care aceste artifacte pot fi recuperate automat pe baza unor reguli convenționale. Maven

depune apoi artifactele într-un astfel de repository local (creând, dacă este nevoie, calea directorului în care se pune el), iar maven va folosi acest artifact atunci când este adresat prin groupId, artifactId și versiune, de alte builduri Maven sau alte software-uri capabile să navigheze structura unui repository Maven. Fișierele pot fi însă depuse și în repository-uri comune mai mari, stocate în rețea și accesibile prin HTTP, ca de exemplu un repository al unei companii de dezvoltare sau al unei organizații.

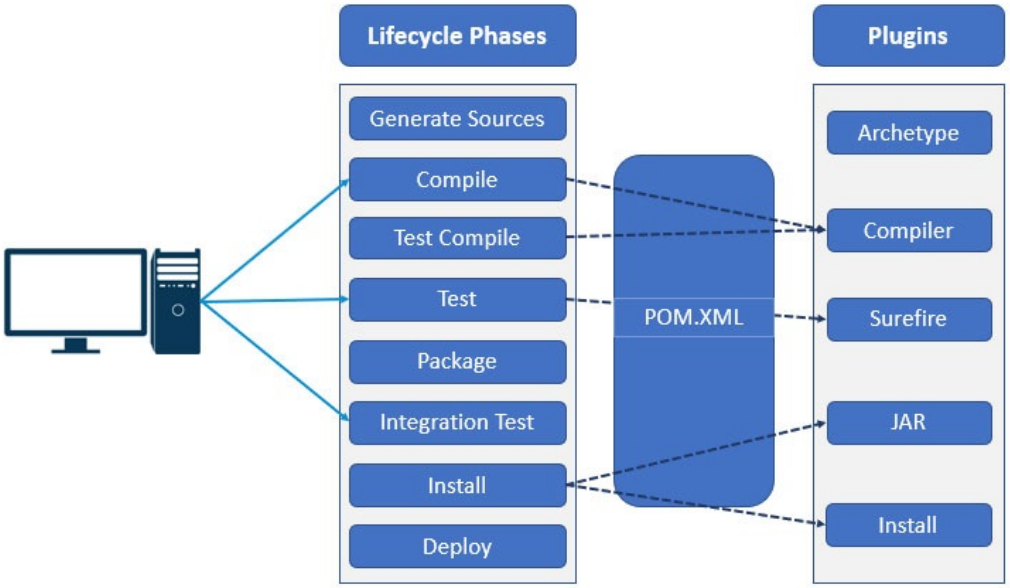

Figura 3.3: *Ciclul de viață al unui program Maven*

3.2.4 RabbitMQ

RabbitMQ este un de broker de mesaje cu sursă deschisă (numit uneori middleware orientat spre mesaje) care inițial a implementat protocolul Advanced Message Queuing Protocol (AMQP) și care, de atunci, a fost extins cu o arhitectură plug-in pentru a sprijini protocolul Streaming Text Oriented Messaging Protocol (STOMP), MQ Telemetry Transport (MQTT) și alte protocoale.

Programul serverului RabbitMQ este scris în limbajul de programare Erlang și este construit pe cadrul platformei Open Telecom pentru clustering și failover. Bibliotecile clienților care vor interfața cu brokerul sunt disponibile pentru toate limbajele de programare majore.

Caracteristici RabbitMQ: [13]

- Fiabilitate: RabbitMQ oferă o varietate de funcții care vă permit să schimbați performanța cu fiabilitate, inclusiv persistență, confirmări de livrare, confirmări ale publisherului și disponibilitate ridicată.

- Rutare flexibilă: Mesajele sunt dirijate prin schimburi înainte de a ajunge la cozi. RabbitMQ dispune de mai multe tipuri de schimb încorporate pentru logica de rutare tipică. Pentru o rutare mai complexă, puteți lega schimburile împreună sau chiar scrie propriul tip de schimb ca un plugin.
- Clustering: Mai multe servere RabbitMQ dintr-o rețea locală pot fi grupate împreună, formând un singur broker logic.
- Federație: Pentru serverele care trebuie să fie mai puțin conectate și mai puțin fiabile decât le permite clusteringul, RabbitMQ oferă un model de federație.
- Cozi disponibile mereu: Cozile pot fi reflectate pe mai multe mașini dintr-un cluster, asigurându-vă că, chiar și în cazul unei defecțiuni hardware, mesajele dvs. sunt în siguranță.
- Multi-protocol: RabbitMQ acceptă mesageria printr-o varietate de protocoale de mesagerie
- Consolă de administrare: RabbitMQ este livrat cu o consolă de administrare ce este ușor de utilizat, care vă permite să monitorizați și să controlați fiecare aspect al brokerului de mesaje.

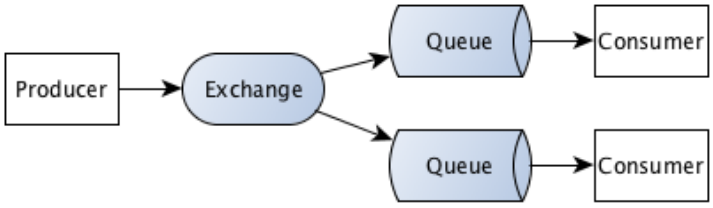

Figura 3.4: *Model de comunicare publish-subscribe în RabbitMQ*

3.2.5 Oracle VM VirtualBox

Oracle VM VirtualBox [14] (anterior Sun VirtualBox, Sun xVM VirtualBox și Innotek VirtualBox) este un hipervizor găzduit gratuit și open-source pentru virtualizare x86, dezvoltat de Oracle Corporation. Creat de Innotek, a fost achiziționat de Sun Microsystems în 2008, care a fost, la rândul său, achiziționat de Oracle în 2010.

VirtualBox poate fi instalat pe Windows, macOS, Linux, Solaris și OpenSolaris. Există, de asemenea, porturi către FreeBSD și Genode. VirtualBox sprijină crearea și gestionarea mașinilor virtuale guest care rulează Windows, Linux, BSD, OS/2, Solaris, Haiku și OSx86, precum și virtualizarea limitată a clienților macOS pe hardware-ul Apple. Pentru unele sisteme de operare guest, este disponibil un pachet Adăugări de oaspeți" de drivere de dispozitiv și aplicații de sistem, care îmbunătățește de obicei performanța, în special cea a graficii.

Utilizatorii VirtualBox pot încărca mai multe sisteme de operare guest sub un singur sistem de operare gazdă. Fiecare guest poate fi pornit, întrerupt și oprit independent în

propria mașină virtuală (VM). Utilizatorul poate configura în mod independent fiecare VM și o poate rula sub o alegere de virtualizare bazată pe software sau de asistare hardware asistată dacă hardware-ul de bază suportă acest lucru. Sistemul de operare gazdă și sistemul de operare guest pot comunica între ele printr-o serie de mecanisme, inclusiv un clipboard comun și o facilitate de rețea virtualizată. VM-urile guest pot comunica, de asemenea, direct între ele, dacă sunt configurate pentru a face acest lucru.

VirtualBox acceptă atât virtualizarea Intel-VT-x cât și AMD-V. Utilizând aceste facilități, VirtualBox poate rula fiecare VM guest în propriul spațiu de adresă separat; codul de apel 0 al sistemului de operare guest rulează pe gazdă la inelul 0 în modul non-root VMX, mai degrabă decât în inelul 1.

Începând cu versiunea 6.1, VirtualBox acceptă doar această metodă. Până atunci, VirtualBox a sprijinit în mod specific unii guest (inclusiv guest pe 64 de biți, invitați SMP și anumite sisteme de operare proprietare) doar pe gazdele cu virtualizare asistată de hardware.

Sistemul emulează hard disk-uri într-unul dintre cele trei formate de imagini pe disc:

- VDI: Acest format este imaginea de disc virtual specifică VirtualBox și stochează date în fișierele care au o extensie de fișier ”.vdi”.
- VMDK: Acest format deschis este folosit de produsele VMware, cum ar fi VMware Workstation și VMware Player. Stochează datele într-unul sau mai multe fișiere care poartă extensii de nume de fișier ”.vmdk”. Un singur hard disk virtual poate cuprinde mai multe fișiere.
- VHD: Acest format este utilizat de Windows Virtual PC și Hyper-V și este formatul discului virtual nativ al sistemului de operare Microsoft Windows, începând cu Windows 7 și Windows Server 2008 R2. Datele din acest format sunt stocate într-un singur fișier care are extensia de nume .vhd”.

Prin urmare, o mașină virtuală VirtualBox poate folosi discuri create anterior în VMware sau Microsoft Virtual PC, precum și propriul format nativ. VirtualBox se poate conecta, de asemenea, la ținte iSCSI și la partiții brute de pe gazdă, folosind fie discuri hard virtuale. VirtualBox emulează controloarele IDE (PIIX4 și ICH6), SCSI, SATA (controler ICH8M) și controlere SAS la care pot fi atașate hard disk-uri.

Pentru un adaptor de rețea Ethernet, VirtualBox virtualizează aceste carduri de interfață de rețea:

- AMD PCnet PCI II (Am79C970A)
- AMD PCnet-Fast III (Am79C973)
- Intel Pro/1000 MT Desktop (82540EM)
- Intel Pro/1000 MT Server (82545EM)
- Intel Pro/1000 T Server (82543GC)

- Paravirtualized network adapter (virtio-net)

Cardurile de rețea emulate permit rularea majorității sistemelor de operare pentru guests, fără a fi nevoie să găsească și să instaleze drivere pentru rețelele hardware, deoarece sunt livrate ca parte a sistemului de operare guest. De asemenea, este disponibil un adaptor de rețea paravirtualizat special, care îmbunătățește performanța rețelei prin eliminarea necesității de a se potrivi cu o anumită interfață hardware, dar necesită asistență specială a driverului la guest. (Multe distribuții ale Linux cu acest driver sunt incluse.) În mod implicit, VirtualBox folosește NAT prin care pot utiliza software-ul de internet pentru utilizatorii finali, cum ar fi Firefox sau ssh. De asemenea, pot fi configurate rețele cu punte printr-un adaptor de rețea gazdă sau rețele virtuale între guests. Până la 36 de adaptoare de rețea pot fi atașate simultan, dar doar patru sunt configurabile prin interfața grafică.

3.2.6 VNC Viewer

În informatică, Virtual Network Computing (VNC) [15] este un sistem grafic de partajare a desktop-ului care folosește protocolul Remote Frame Buffer (RFB) pentru a controla de la distanță un alt computer. Acesta transmite evenimentele de la tastatură și mouse de la un computer la altul, redând actualizările ecranului grafic înapoi în cealaltă direcție, printr-o rețea.

VNC este independent de platformă - există clienți și servere pentru multe sisteme de operare bazate pe GUI și pentru Java. Mai mulți clienți se pot conecta la un server VNC în același timp. Utilizările populare pentru această tehnologie includ asistență tehnică de la distanță și accesarea fișierelor de pe computerul de lucru de la computerul de acasă sau invers.

Există o serie de variante de VNC care oferă propria lor funcționalitate; de exemplu, unele optimizate pentru Microsoft Windows sau care oferă transfer de fișiere (care nu fac parte din VNC propriu-zis), etc. Multe sunt compatibile (fără funcțiile adăugate) cu VNC propriu-zis, în sensul că un vizualizator se poate conecta cu un server al altuia. Altele se bazează pe codul VNC, dar nu sunt compatibile cu VNC standard.

Componente:

- Server VNC: program pe aparatul care partajează un anumit ecran (și este posibil să nu fie legat de un afișaj fizic - serverul poate fi headless") și permite clientului să partajeze controlul acestuia.

- Client VNC: program care reprezintă datele ecranului provenite de la server, primește actualizări de la acesta și, probabil, îl controlează informând serverul despre intrarea locală colectată.

- Protocol VNC: protocol foarte simplu, bazat pe transmiterea unei primitive grafice de la server la client (Pune un dreptunghi de date de pixeli la poziția X, Y specificată") și mesaje de evenimente de la client la server

În metoda normală de operare, un vizualizator se conectează la un port pe server (port implicit: 5900). Alternativ (în funcție de implementare) un browser se poate conecta la server (portul implicit: 5800). Și un server se poate conecta la un vizualizator în ”modul de ascultare” din portul 5500. Un avantaj al modului de ascultare este că site-ul serverului nu trebuie să-și configureze firewallul pentru a permite accesul pe portul 5900 (sau 5800); datoria este asupra privitorului, ceea ce este util dacă site-ul serverului nu are cunoștințe informatice și utilizatorul vizualizatorului este mai informat.

3.2.7 Git

Git [16] este un sistem revision control care rulează pe majoritatea platformelor, inclusiv Linux, POSIX, Windows și OS X. Ca și Mercurial, Git este un sistem distribuit și nu întreține o bază de date comună. Este folosit în echipe de dezvoltare mari, în care membrii echipei acționează oarecum independent și sunt răspândiți pe o arie geografică mare.

Git este dezvoltat și întreținut de Junio Hamano, fiind publicat sub licență GPL și este considerat software liber.

Dintre proiectele majore care folosesc Git amintim Amarok, Android, Arch Linux, Btrfs, Debian, DragonFly BSD, Eclipse, Fedora, FFmpeg, GIMP, GNOME, GTK+, Hurd, Linux kernel, Linux Mint, openSUSE, Perl, phpBB, Qt, rsync, Ruby on Rails, Samba.

Dezvoltarea Git a început după ce mai mulți developeri ai nucleului Linux au ales să renunțe la sistemul de revision control proprietar BitKeeper. Posibilitatea de a utiliza BitKeeper gratuit a fost retrasă după ce titularul drepturilor de autor a afirmat că Andrew Tridgell a încălcat licența BitKeeper prin acțiunile sale de inginerie inversă. La conferința Linux.Conf.Au 2005, Tridgell a demonstrat în timpul discursului său că procesul de inginerie inversă pe care l-a folosit a fost pur și simplu o sesiune telnet pe portul corespunzător al serverului BitKeeper și rularea comenzii help pe server.

Controversa a dus la o renunțarea rapidă la sistemul BitKeeper care a fost înlocuit cu un nou sistem intitulat Git construit special pentru scopul de revision control în cadrul proiectului Linux kernel. Dezvoltarea noului sistem a fost începută de Linus Torvalds în 3 aprilie 2005 pentru a fi anunțat câteva zile mai târziu (aprilie 6) pe lista de email a proiectului Linux kernel. O zi mai târziu, noul sistem a început să fie folosit pentru dezvoltarea actuală de cod pentru proiectul Git. Primele operații merge a avut loc pe data de 18 aprilie. În data de 16 iunie, versiunea 2.6.12 Linux kernel a fost pusă în Git care continuă și în ziua de azi să fie sistemul revision control folosit de proiectul Linux kernel.

Tot în această perioadă, și tot cu scopul de a înlocui BitKeeper, a fost creat sistemul Mercurial.

3.2.8 GitHub

GitHub este un serviciu de găzduire web pentru proiecte de dezvoltare a software-ului care utilizează sistemul de control al versiunilor Git. GitHub oferă planuri tarifare pentru depozite private, și conturi gratuite pentru proiecte open source. Site-ul a fost lansat în 2008

de către Tom Preston-Werner, Chris Wanstrath, și PJ Hyett. În 2018 Microsoft a cumpărat Github pentru 7.5 miliarde de dolari.

3.2.9 Linux

Linux [16] este o familie de sisteme de operare de tip Unix care folosesc Nucleul Linux (în engleză kernel). Linux poate fi instalat pe o varietate largă de hardware, începând cu telefoane mobile, tablete, console video, continuând cu calculatoare personale până la supercomputere. Linux este cunoscut în principal pentru utilizarea sa ca server, în 2009 i se estima o cotă de piață între 20-40%. Cota de piață de desktop este estimată între 1-2% și 4.8%. În ultimii ani, Linux a început să devină tot mai popular atât datorită unor distribuții precum Ubuntu, openSUSE, Fedora, precum și datorită apariției netbook-urilor și a noii generații de telefoane inteligente (în engleză smart phone) care rulează o versiune embedded de Linux.

Nucleul Linux a fost dezvoltat inițial pentru microprocesorul Intel 386, dar în prezent rulează pe o mare gamă de microprocesoare și arhitecturi de calculatoare. Este folosit pe calculatoare de tip personal, servere, pe supercomputere, dar și pe sisteme înglobate (embedded), cum ar fi unele telefoane mobile sau recordere video.

Nucleul (kernel-ul) Linux este un nucleu monolitic. Cu toate acestea, spre deosebire de multe alte nuclee monolitice, driverele se pot încărca în memoria de lucru la utilizare, și se pot șterge de acolo ulterior, eliberând resursele utilizate, fără a necesita resetarea sistemului sau recompilarea nucleului. Facilitățile oferite de nucleu includ, printre alele:

- multitasking real și complet,
- suport pentru memorie virtuală,
- distribuția executabilelor la scriere,
- management avansat al memoriei,
- suport avansat pentru TCP/IP (inclusiv rutare și filtrare),
- până la un miliard de procese simultane,
- sistem de sunet modularizat (OSS sau ALSA).

Nucleul este scris integral în C și poate fi compilat folosind compilatorul GCC.

Capitolul 4

Studiu de caz: ”Sistem de securitate bazat pe recunoaștere facială”

În lucrarea de față, este prezentat un sistem de supraveghere și securitate inteligent, bazat pe recunoaștere facială. Acest sistem poate detecta doar din analiza imaginilor prezența unui potențial infractor, iar apoi va trimite o notificare utilizatorului, care poate intra să vadă imagini live sau să dea anumite comenzi de la distanță - spre exemplu, încuiere/descuiere uși, pornire/oprire alarmă.

În aplicația de mobil, utilizatorul poate vedea un istoric în care sunt prezentate toate notificările primite, care pot fi șterse. Se mai poate, de asemenea, seta frecvența notificărilor, spre exemplu - 1 notificare / secundă, sau 1 notificare / 30 secunde, etc.

Pentru realizarea acestor funcționalități, au fost implementate următoarele subsisteme:

1. Sistemul de recunoaștere facială, supraveghere video și control de la distanță
2. Sistemul de Live Streaming
3. Sistemul de comunicare între componente
4. Aplicația Android

În cele ce urmează, vor fi prezentate pe rând cele patru subsisteme, explicând pentru fiecare în parte motivele din spatele alegerilor, raționamentul, arhitectura internă, problemele întâmpinate, rezolvările problemelor (inclusiv anumite compromisuri făcute) și rezultatele final.

4.1 Sistemul de recunoaștere facială, supraveghere video și control de la distanță

Acest sistem este format dintr-un microcontroler Raspberry Pi 4B, căruia îi este atașată o cameră Raspberry Pi Camera prin intermediul căreia primește un flux de imagini. Acest flux este procesat cadru cu cadru de un script scris in Python, iar dacă este detectată o față, imaginea respectivă este trimisă catre sistemul de comunicare, care acționează ca un releu, în cele din urma imaginea ajungând la aplicația Android.

Microcontrolerul mai conține un script care ascultă, gestionează și răspunde la comenzile primite de la serverul de comunicare, și ia decizii în concordanță. (pornire/oprire funcționalitate live stream, încuiere/descuiere ușă, oprire/pornire alarmă).

4.1.1 Instalarea sistemului de operare

Sistemul de operare instalat pe Raspberry Pi este Raspbian, un sistem de operare bazat pe versiuni Debian pentru arhitectura ARM (armhf), adaptat pentru utilizare pe dispozitivele Raspberry Pi. Numele Raspbian reprezintă o combinație dintre Raspberry și Debian.

Pentru a instala sistemul de operare, este nevoie de un alt computer care poate citi un card SD, deoarece acest sistem de operare va fi instalat pe un card SD, iar apoi acest card va fi introdus în Raspberry Pi. Cerințe minime card SD:

1. Capacitate stocare
   (a) Dacă se va instala versiunea completă cu desktop Raspberry Pi OS prin NOOBS, se recomandă minim 16 GB
   (b) Dacă se va instala versiunea completă cu desktop Raspberry Pi OS, prin altă modalitate, se recomandă minim 8 GB
   (c) Dacă se va instala versiunea Raspberry Pi OS Lite, se recomandă minim 4 GB
2. Clasa cardului SD
   (a) Pentru clasa 4 - viteze de scriere de minim 4 MB/s
   (b) Pentru clasa 10 - viteze de scriere de minim 10 MB/s

Apoi, este nevoie de un software special care poate instala sistemul de operare pe cardul SD mai sus menționat. În acest caz, a fost folosit Raspberry Pi Imager.

Inițial a fost folosit sistemul de operare fără GUI (Graphical User Interface) dar pe parcurs s-a observat că scriptul de Python care generează o fereastră GUI nu poate funcționa fără un astfel de sistem. Din această cauză, a fost facută schimbarea către sistemul de operare cu interfață grafică, făcând astfel un compromis pe partea de putere de procesare.

4.1.2 Conectarea automată la Wi-Fi

Pentru a conecta automat microcontrolerul Raspberry Pi la rețeaua locală wireless, trebuie urmăriți următorii pași:

- Introducerea cardului SD pe care este instalat sistemul de operare într-un alt calculator
- Crearea unui fișier cu numele

  `'wpa_supplicant.conf'`

  în directorul rădăcină al sistemului de operare
- Configurarea fișierului, punând valorile proprii în câmpurile 'ssid' și 'psk' (numele rețelei și parola rețelei), urmărind imaginea de mai jos:

```
country=US
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
    ssid="NETWORK-NAME"
    psk="NETWORK-PASSWORD"
}
```

Figura 4.1: *Configurare rețea wireless pentru Raspberry Pi*

După această configurare, se scoate cardul SD din calculator, se mută în Raspberry Pi, se pornește și se încearcă conectarea prin rețea, folosind PuTTY.

4.1.3 Acces de la distanță prin rețeaua wireless locală

Acest lucru este necesar deoarece se dorește configurarea microcontrolerului fără a conecta periferice direct, făcând accesul foarte ușor, putând asftel să se instaleze, modifice, steargă programe sau să se configureze sistemul.

Sistemul de operare al Raspberry Pi vine preinstalat cu server Secure Shell. Secure Shell (SSH) este un protocol de rețea criptografic ce permite ca datele să fie transferate folosind un canal securizat intre dispozitive de rețea. Cele două mari versiuni ale protocolului sunt SSH1 sau SSH-1 și SSH2 sau SSH-2. Folosit cu precădere în sistemele de operare multiutilizator linux și unix, SSH a fost dezvoltat ca un înlocuitor al Telnet-ului și al altor protocoale nesigure de acces de la distanță, care trimit informatia, în special parola, în clar, făcând posibilă descoperirea ei prin analiza traficului. Criptarea folosită de SSH intenționează să asigure confidențialitatea și integritatea datelor transmise printr-o rețea nesigură cum este Internetul. [17]

Pentru activarea acestui server (care este dezactivat în mod predefinit) se face astfel:

- Introducerea cardului SD într-un alt calculator
- Crearea unui fișier gol cu numele

  `'ssh'`

  în directorul rădăcină al sistemului de operare

După ce acest feature a fost activat, se poate încerca conectarea prin rețea la Raspberry Pi, folosind PuTTY. Adresa IP a microcontrolerului se poate afla fie din routerul personal (se poate observa în lista dispozitivelor conectate la rețea), fie trimițând o comandă 'ping' către hostname-ul 'raspberrypi.local'. După aflarea adresei IP, se poate configura clientul ssh PuTTY, precum în imaginea de mai jos:

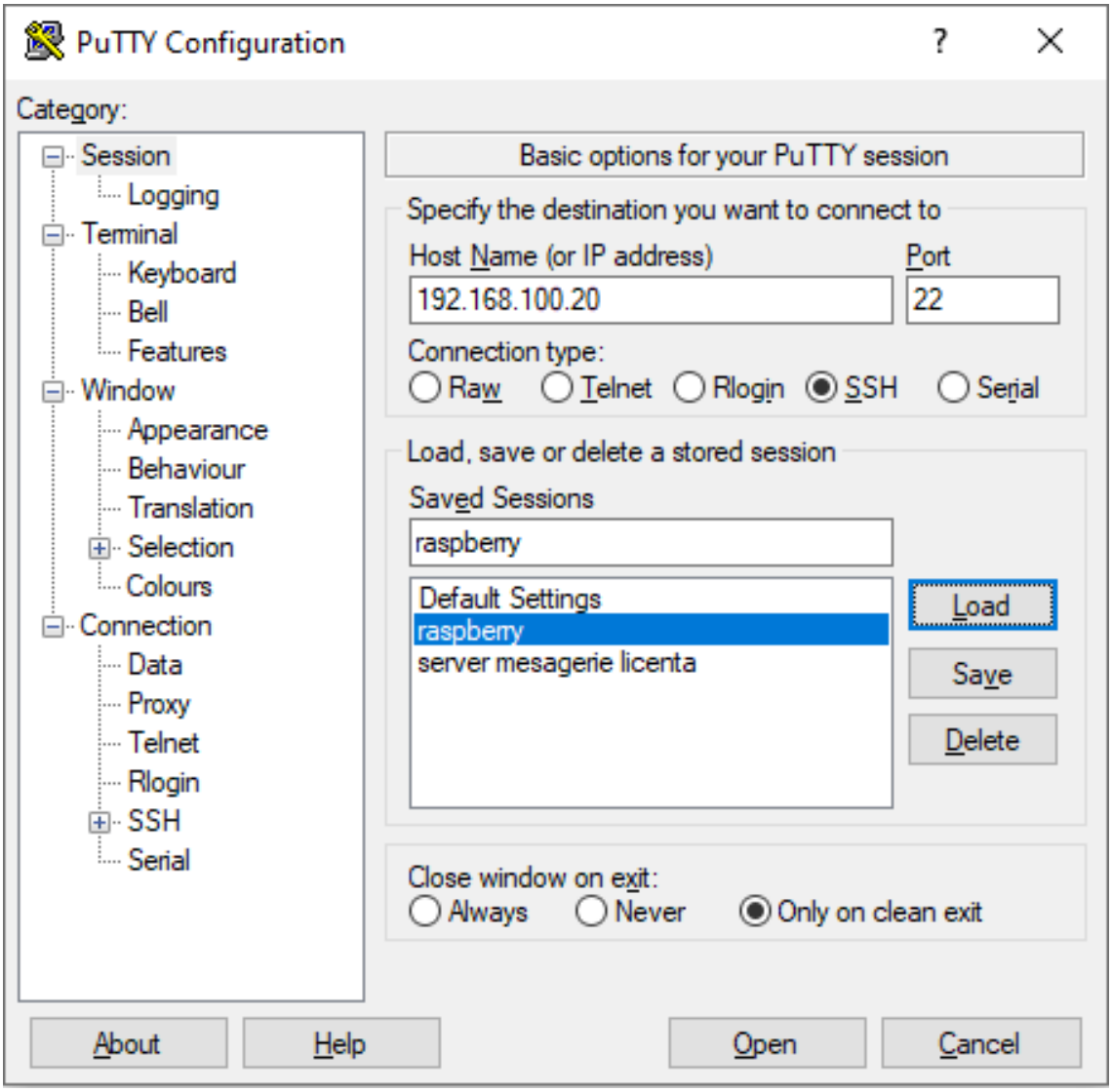

Figura 4.2: *Configurare PuTTY pentru conectarea la Raspberry Pi*

Pentru copierea de fișiere, este nevoie de un software special care poate transfera fișiere pe rețea, între sisteme de operare diferite. Pentru acest lucru, este folosit protocolul SCP, împreună cu WinSCP. WinSCP (Windows Secure CoPy) este un utilitar, gratuit și Open Souce, client FTP, SFTP și SSH ce ruează sub Microsoft Windows. Funcția lui principală este transferul de fișiere între calculatorul local și alte calculatoare din LAN sau WAN. De asemenea WinSCP are și alte funcționalități: Manager fișiere, Sincronizare fișiere, editare fișiere, etc. La transferul de fișiere se utilizează protocoalele SSH, SCP sau SFTP. [18]

WinSCP se configurează analog cu PuTTY, precum în imaginea următoare:

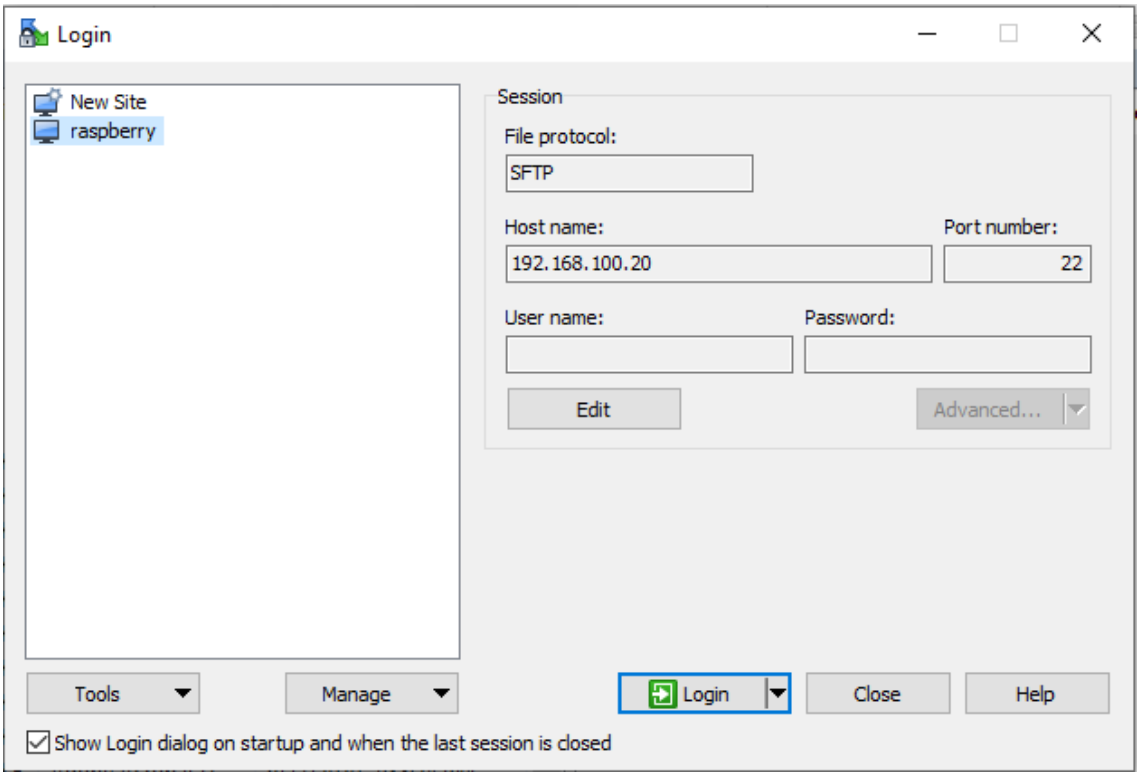

Figura 4.3: *Configurare WinSCP pentru conectarea la Raspberry Pi*

Acum se poate lucra cu microcontrolerul de la distanță, în același mod cum am lucra conectând perifericele direct la Raspberry Pi.

4.1.4 Instalarea dependințelor software

Programul care analizează fluxul de imagini primite de la cameră, aplică algoritmul de detecție facială și trimite mesaje către serverul de mesagerie are dependințe terțe. Pentru a le instala, este nevoie în primul rând de un manager de biblioteci software. Pentru Python, acest manager este implicit 'pip'.

Folosind 'pip', se vor instala următoarele dependințe, urmărind instrucțiunile de mai jos:

1. OpenCV - libraria ce conține algoritmul pentru detecție facială

```
$ sudo pip install opencv-contrib-python
```

2. Picamera - libraria ce permite accesul la Raspberry Pi Camera dintr-un script Python

```
$ sudo apt-get install python-picamera python3-picamera
```

3. Pika - libraria ce permite conectarea la un server RabbitMQ (server care stă la baza sistemului de comunicare între componente)

```
$ sudo pip install pika
```

4.1.5 Arhitectura internă

Software-ul care se ocupă de detecția facială, trimiterea notificărilor, sau trimiterea imaginilor ca și flux către serverul de Live Streaming are următoarea arhitectură, prezentată la nivel înalt:

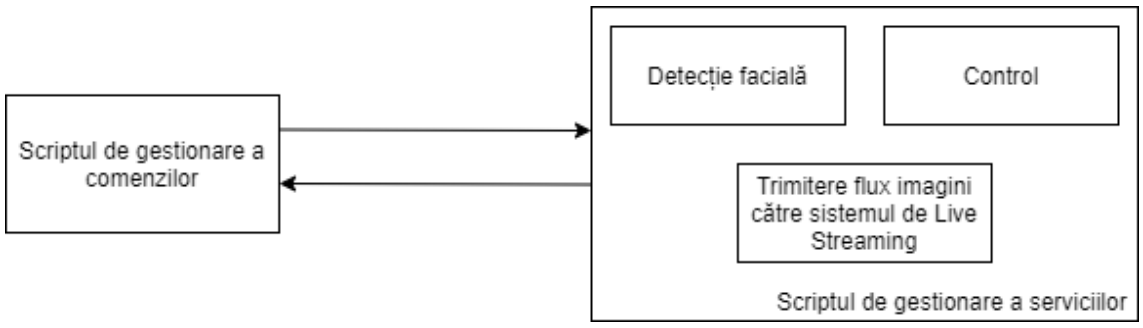

Figura 4.4: *Arhitectură internă abstractă a sistemului de recunoastere facială, supraveghere video și control de la distanță*

4.1.5.1 Diagrama de integrare cu restul componentelor

Scripturile instalate pe Raspberry Pi comunică în permanență cu celelalte componente ale sistemului, deoarece se dorește un sistem fiabil, eficient și disponibil mereu, care poate fi controlat de la distanță, chiar daca utilizatorul nu este în aceeași rețea cu microcontrolerul. Pentru a realiza aceste funcționalități, scriptul de gestionare a comenziulor se leagă la internet, conectându-se la sistemul de comunicare între componente.

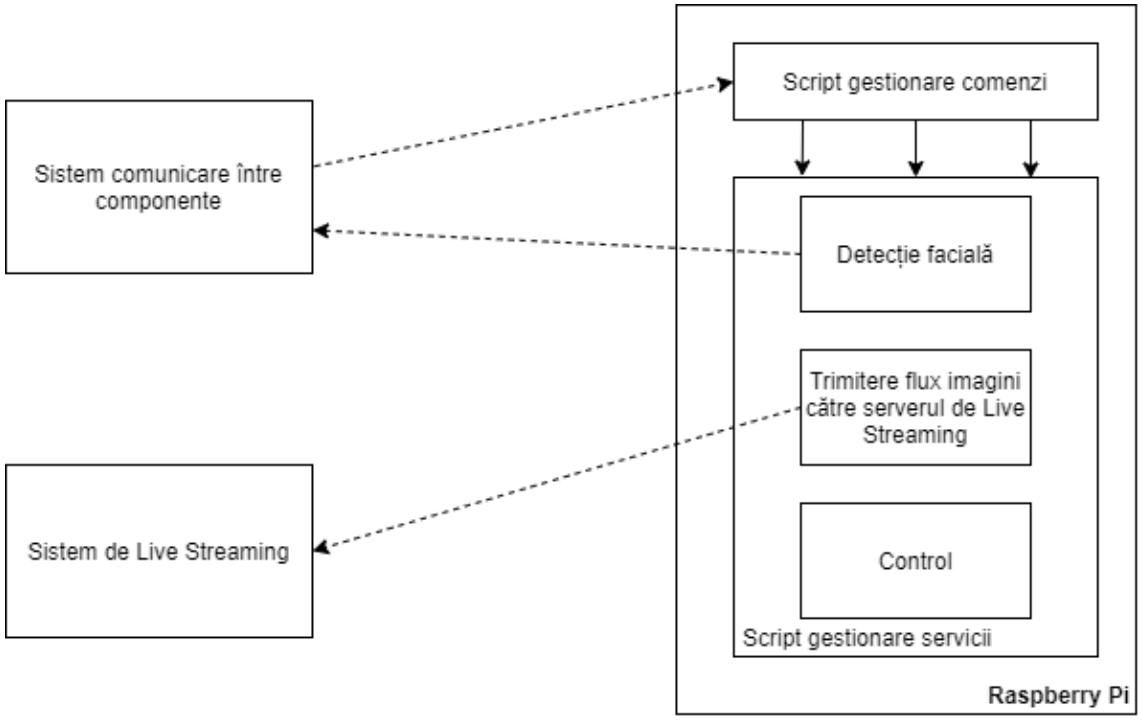

Figura 4.5: *Diagrama de integrare Raspberry Pi cu restul sistemelor*

4.1.5.2 Scriptul de gestionare a comenzilor

Acest script se conecteaza la sistemul de comunicare între componente și, pe firul de execuție principal, ascultă la comenzi din partea acestuia. Pe un fir paralel de execuție, se pornește scriptul de gestionare a serviciilor, cu care acesta vorbește prin intermediul valorii unor variabile.

Comenzile înțelese de acest script sunt următoarele:

- START_LIVE_STREAM
- STOP_LIVE_STREAM
- CHANGE_LIVE_STREAM_QUALITY
- LOCK_DOOR

- UNLOCK_DOOR
- START_ALARM
- STOP_ALARM

Comanda 'CHANGE_LIVE_STREAM_QUALITY' poate avea și un parametru, care va determina calitatea streamului. Valorile posibile pentru calitate sunt:

- LOW
- MEDIUM
- HIGH

Atunci când primește comanda 'START_LIVE_STREAM', scriptul setează variabila 'startLiveStream' a scriptului de gestionare a serviciilor la 'True', iar când primește comanda 'STOP_LIVE_STREAM' seteaza valoarea la 'False'. Acest lucru determină oprirea sau pornirea funcționalității de live stream.

Acest lucru determină o creștere a eficienței din punct de vedere al procesorului, deoarece atunci când ambele funcționalități sunt pornite, load-ul este de aproximativ 5.37, cele 4 procesoare rulând aproape la capacitate maximă (aproximativ 97% pe fiecare procesor). Atunci când doar funcționalitate de detectare a feței este pornită, load-ul este de 4.08, deci rezultă o scădere în folosirea procesorului. Deși pare puțin, orice eficientizare contează întrucât se lucrează cu un microcontroler cu capacitate de procesare mică:

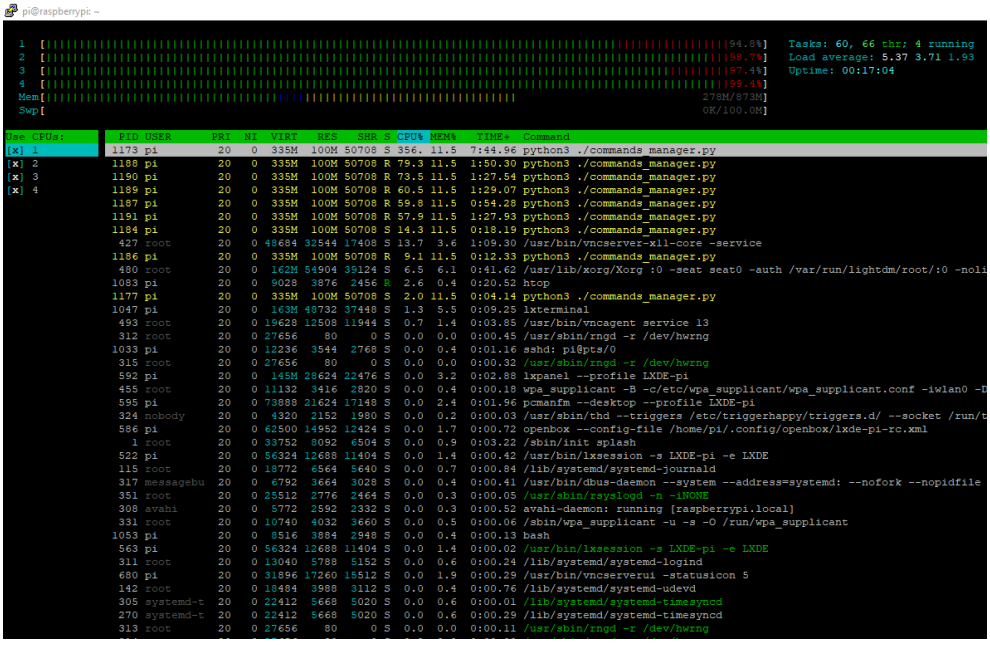

Figura 4.6: *Utilizare procesor Raspberry Pi cu ambele funcționalități pornite*

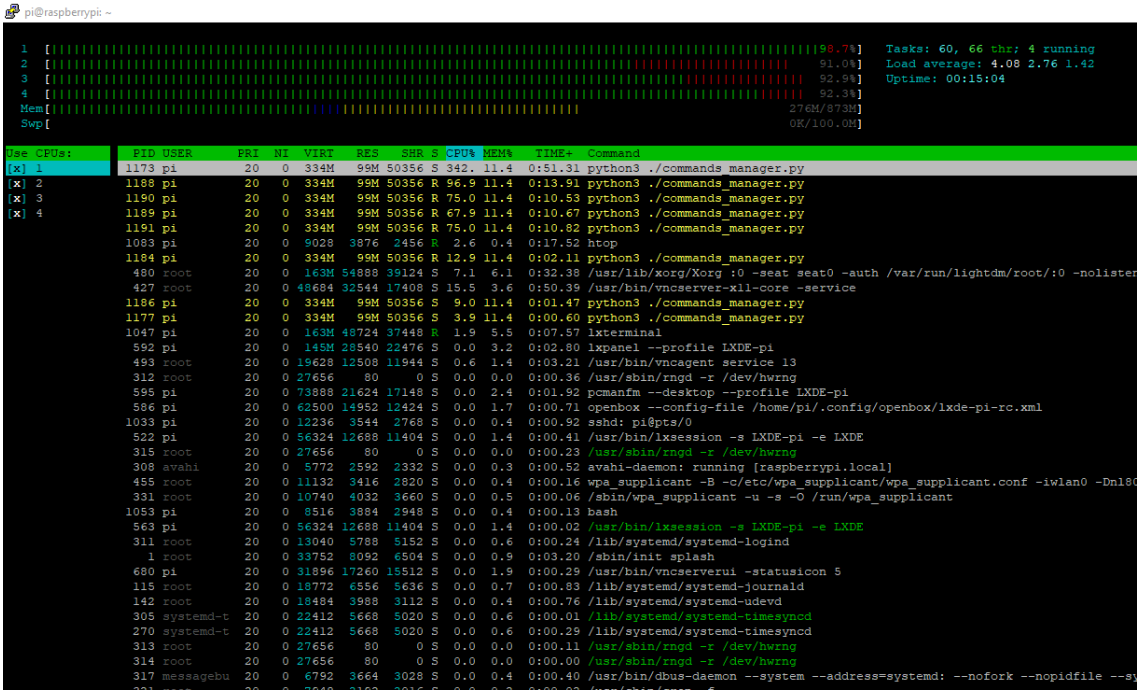

Figura 4.7: *Utilizare procesor Raspberry Pi doar cu funcționalitatea de detecție a feței pornită*

[H]

Funcționalitate activă	Load procesor
Detectarea feței și Live Stream simultan	5.37
Doar detectarea feței	4.08

Se observă că prin oprirea funcționalității de Live Stream atunci când nu este folosită, scădem stresul de pe procesor cu **31.62%**.

4.1.5.3 Scriptul de gestionare a serviciilor

Acest script este un program complex, care înglobează logica necesară tutoror serviciilor: trimiterea fluxului de imagini către Sistemul de Live Streaming, detectarea facială și trimiterea imaginilor către Sistemul de comunicare între componente și controlul ușii și a alarmei.

Diagrama UML din figura următoare prezintă câmpurile scriptului, precum și metodele care acționează fiecare funcționalitate:

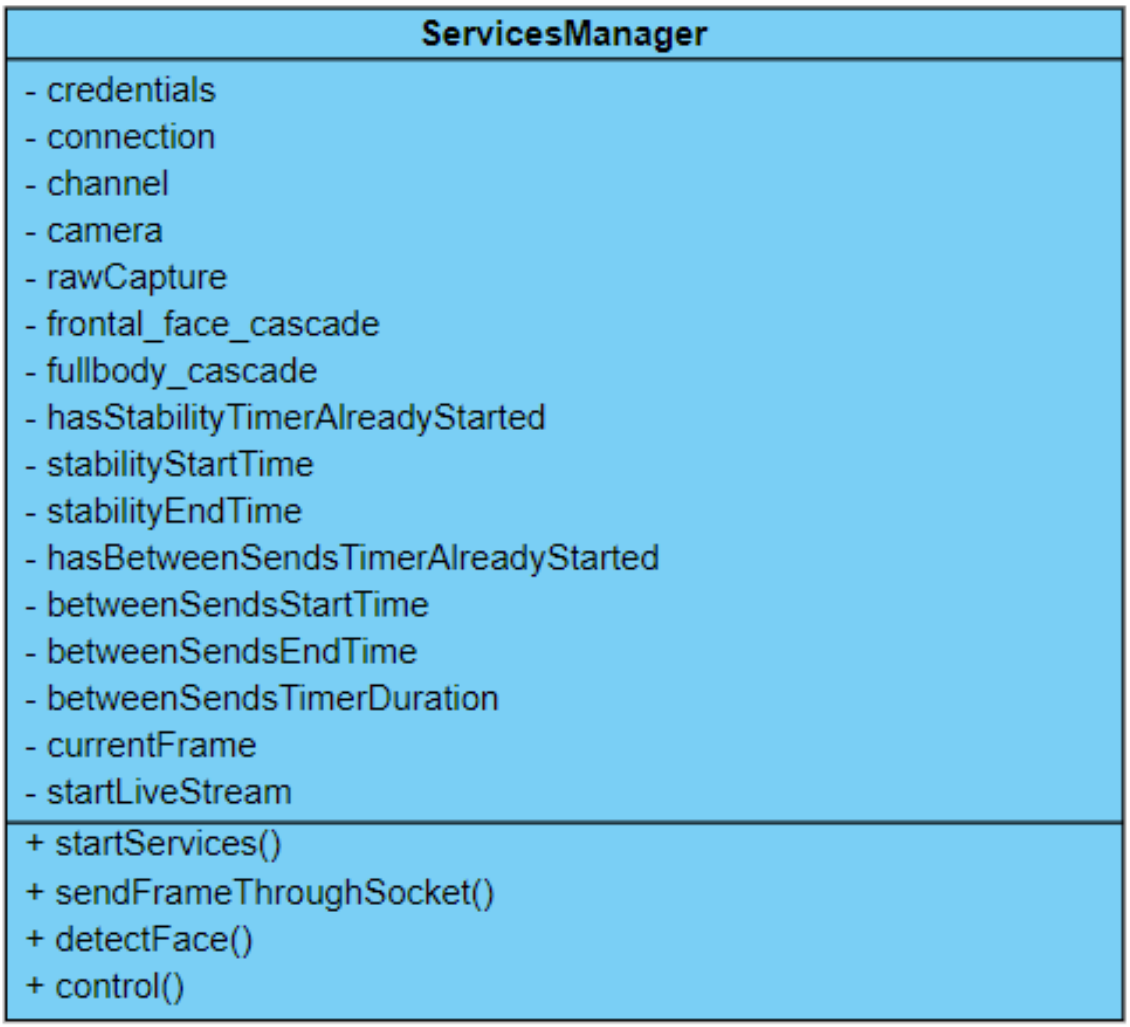

Figura 4.8: *Diagrama de clasă a scriptului de gestionare a serviciilor*

Metoda **startServices()** are următoarele întrebuințări:

- se conectează folosind protocolul TCP la Sistemul de Live Streaming, folosind adresa Ip si portul corespunzător. Socketul este inițializat cu atributul 'SOCK_STREAM' deoarece se dorește trimiterea cadrelor tinând conexiunea deschisă
- se folosește de capabilitățile de multithreading ale limbajului de programare Python și pornește metodele 'sendFrameThroughSocket()' și 'detectFace()' fiecare pe firul propriu de execuție, deoarece acestea trebuie să ruleze în paralel, nu secvențial
- se conectează la fluxul de imagini primit de la Raspberry Pi Camera, într-o buclă infinită, setând cadrul curent ca fiind fiecare cadru primit din acest flux

Metoda **sendFrameThroughSocket()** funcționează astfel:

- începe o buclă infinită, în care:
  - verifică dacă variabila *startLiveStream* este *False*. Dacă da, oprește firul curent de execuție pentru o secundă. Dacă nu, continuă.
  - verifică dacă cadrul curent este null. Dacă este, continuă la următoare iterație a buclei
  - setează calitatea imaginii la parametrul setat
  - encodează imaginea folosind encodingul *base64*, pentru a putea facilita transmitea imaginii prin socket
  - trimite imaginea prin socket, urmată de un delimitator ce are rolul de a separa imaginile, deoarece acestea sunt trimise una după alta. Daca delimitatorul nu ar exista, Sistemul de Live Streaming care primeste aceste imagini nu ar putea face ști unde începe și unde se termină o imagine

Metoda **detectFace()** are următorul comportament:

- pornește o buclă infinită în care sunt executate toate instrucțiuniile următoare
- setează cadrul curent la valoarea cadrului de la nivelul clasei
- dacă cadrul curent este null, continuă la următoarea iterație a buclei infinite
- transformă imaginea curentă în alb-negru, pentru a fi mai ușor de procesat de algoritmul de detectare facială din OpenCV
- apelează metoda *detectMultiScale* pe imaginea curentă. Această metodă aplică algoritmul de detectare facială
- dacă s-a detectat o față în imaginea curentă, setează valoarea variabilei *stabilityStartTime* la secunda curentă. Această variabilă verifică dacă s-a detectat o față continuu timp de cel puțin o secundă, pentru a crește fiabilitatea algoritmului

- dacă detecția a fost stabilă pentru cel puțin o secundă:
  - se encodează imaginea folosind encodingul *base64*
  - se trimite către Sistemul de comunicare între componente, folosind protocolul AMQP
  - se resetează cronometrul de stabilitate
- dacă a fost detectată o față în cadrul curent, se desenează un dreptunghi în jurul zonei feței
- pornește o feareastră în care se va afișa imaginea curentă, cu dreptunghiul aferent, dacă este detectată o față

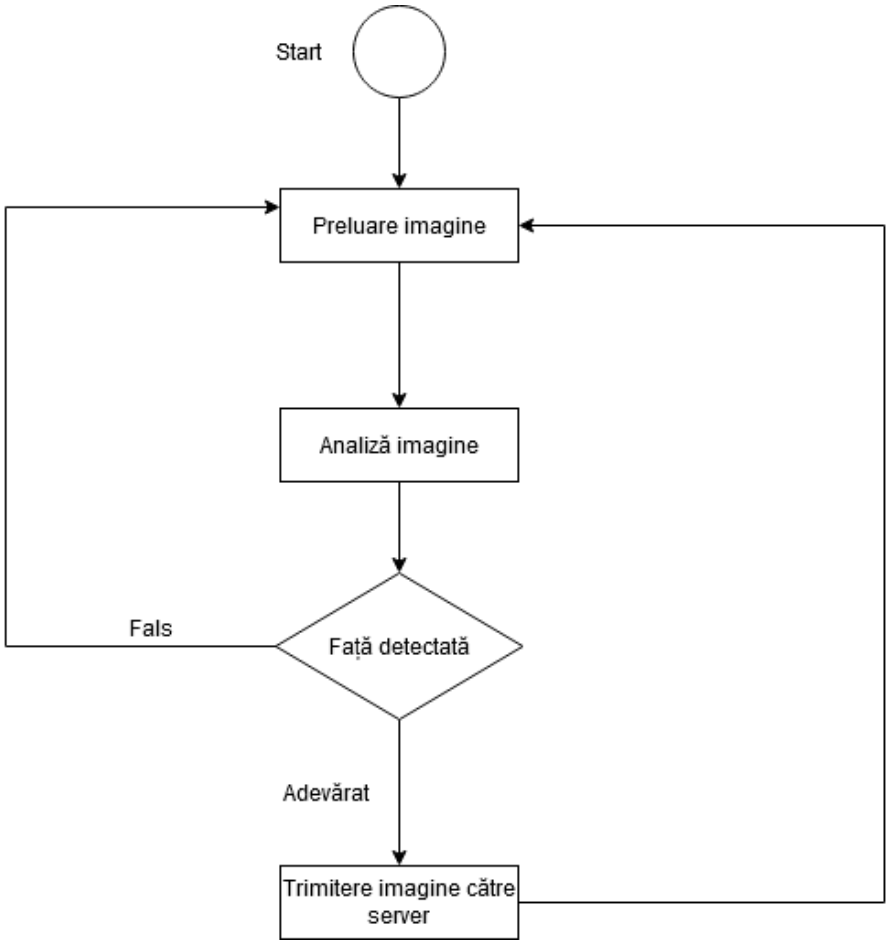

Figura 4.9: *Diagrama de activitate a funcției de detectare a feței*

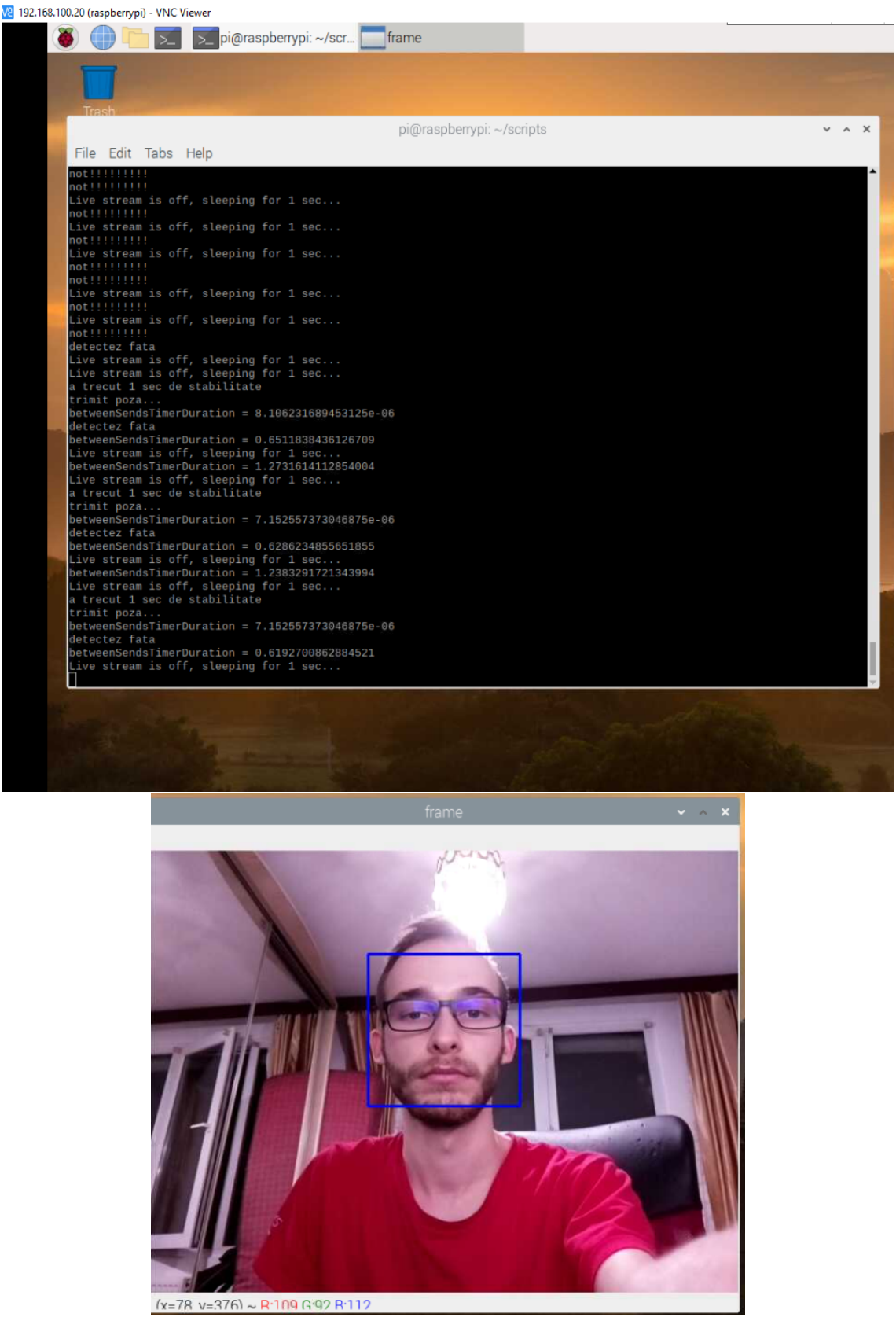

Figura 4.10: *Captură ecran detectare corectă a feței*

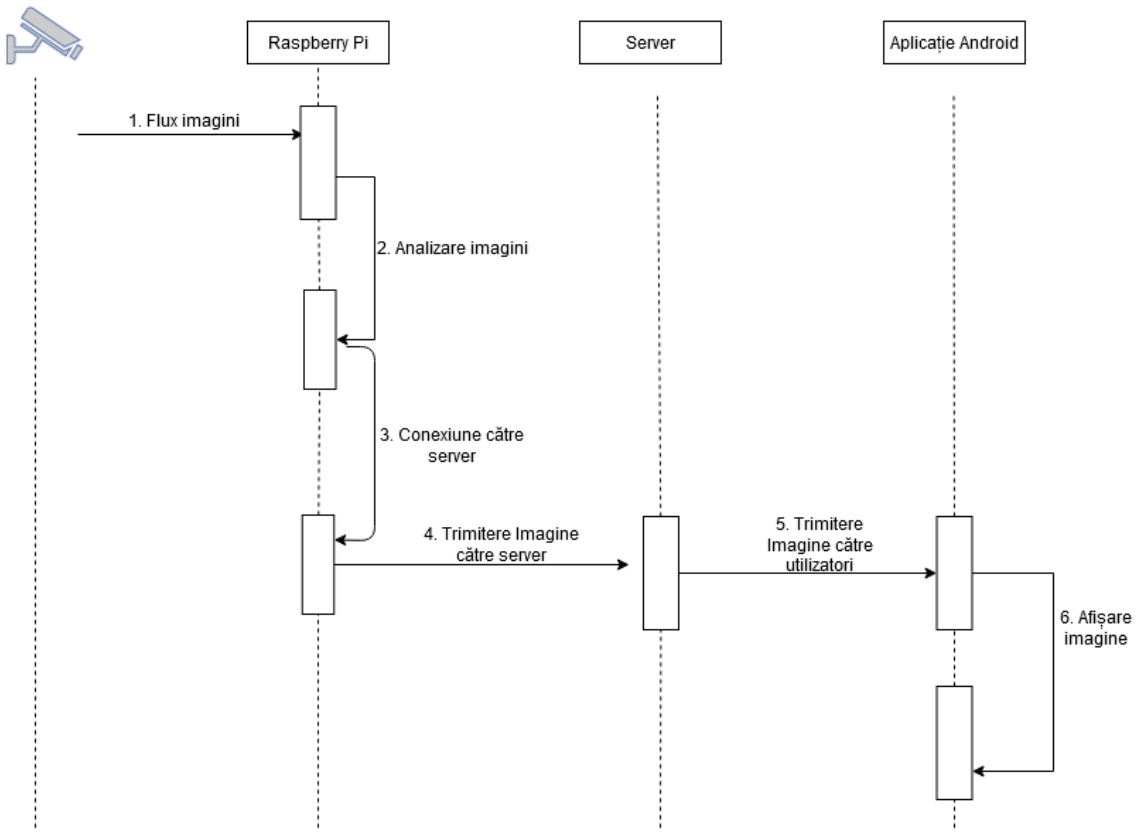

Figura 4.11: *Diagrama de secvență a funcției de detectare a feței*

4.2 Sistemul de Live Streaming

Sistemul de Live Streaming este un server web scris în Java, folosind concepte precum Multithreading și comunicare la nivel de socket. Pentru gestionarea dependințelor am folosit *Maven*, iar pentru injectarea dependințelor și configurarea lor, cât și pentru ușurarea scrierii serverului am folosit *Spring Boot*

4.2.1 Împachetare, distribuire și rulare

Pentru a compila și împacheta serverul, se folosește *Maven*. Rezultatul va fi un fișier cu extensia *.jar*, (Java ARchive), care, datorită Spring Boot, vine cu un servlet container Tomcat embedded.

```
c:\Users\Tiberiu Marinica\LiveStreamServerProject>mvn clean package
```

Pentru a porni serverul pe o mașină Linux direct din arhiva *.jar*, se ruleaza următoarea comandă:

```
$ sudo java -jar ./live-stream-server.jar
```

După ce această comandă este rulată, pe mașina locală va porni serverul web, ce deschide porturile 8000 (pentru a primi de la Raspberry Pi streamul de imagini) și 8001 (pentru a primi conexiuni de la aplicațiile Android)

4.2.2 Arhitectură internă

Sistemul de Live Streaming este un server web, ce se folosește de Multithreading pentru a putea asculta în paralel la streamul de imagini venit de la Raspberry Pi, și să multiplice acest stream pentru fiecare din clienții conectați.

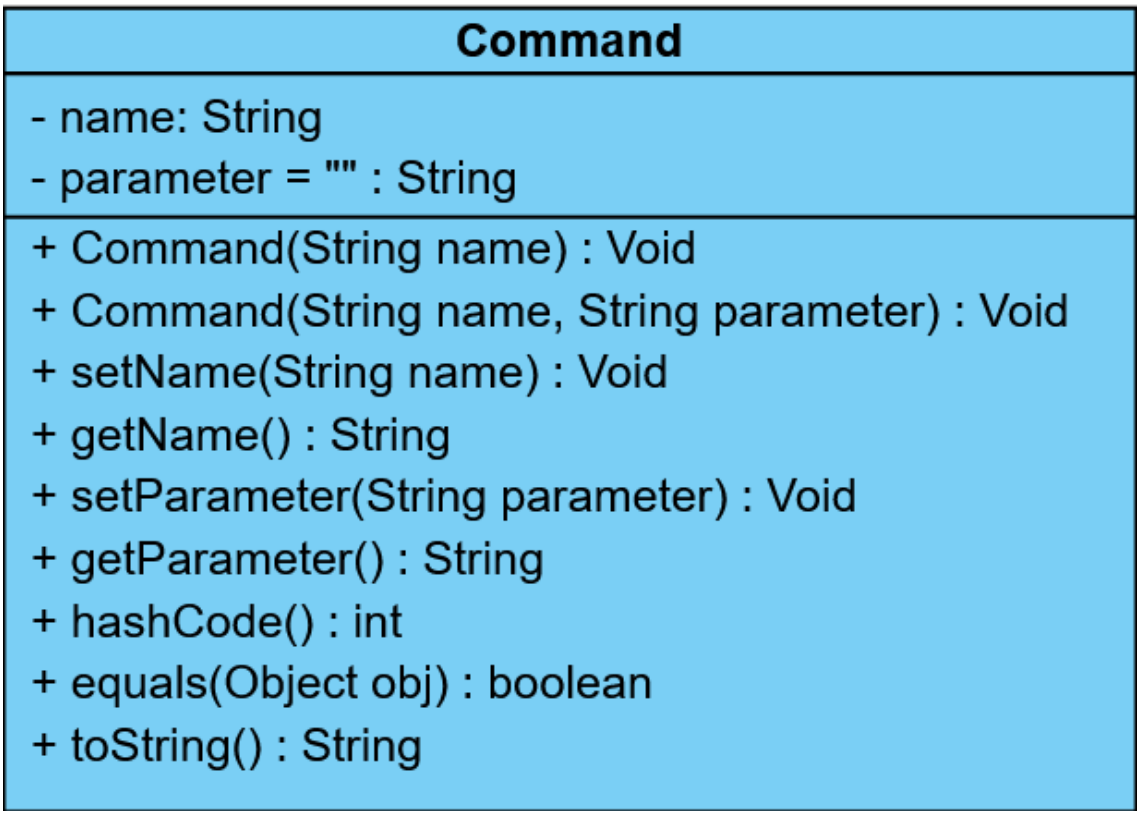

Figura 4.12: *Clasă ce modelează o comandă a serverului de streaming*

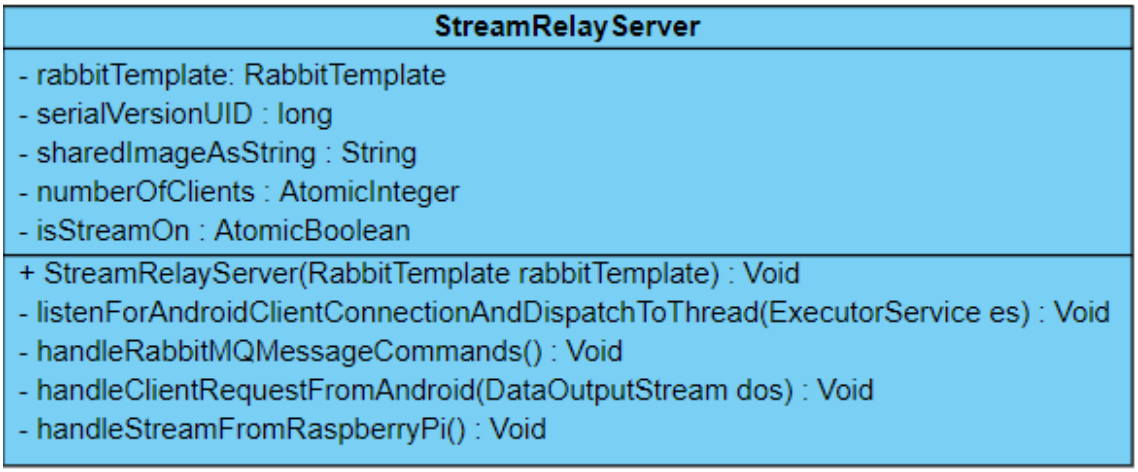

Figura 4.13: *Clasa prinicpală a serverului de streaming*

Atunci când serverul pornește, se va invoca o nouă instanță a serverului, care în constructor va rula următoarele instrucțiuni:

- va creea o nouă instanță a ExecutorService, bibliotecă ce ajută la gestionarea Thread-urilor și a programelor Multithreaded. Această instanță va:
  - creea un nou Thread în care se va rula metoda *handleStreamFromRaspberryPi*
  - creea un nou Thread în care se va rula metoda *handleRabbitMQMessageCommands*
- apelează metoda *listenForAndroidClientConnectionAndDispatchToThread*, dând ca parametru instanța de ExecutorService mai sus creată

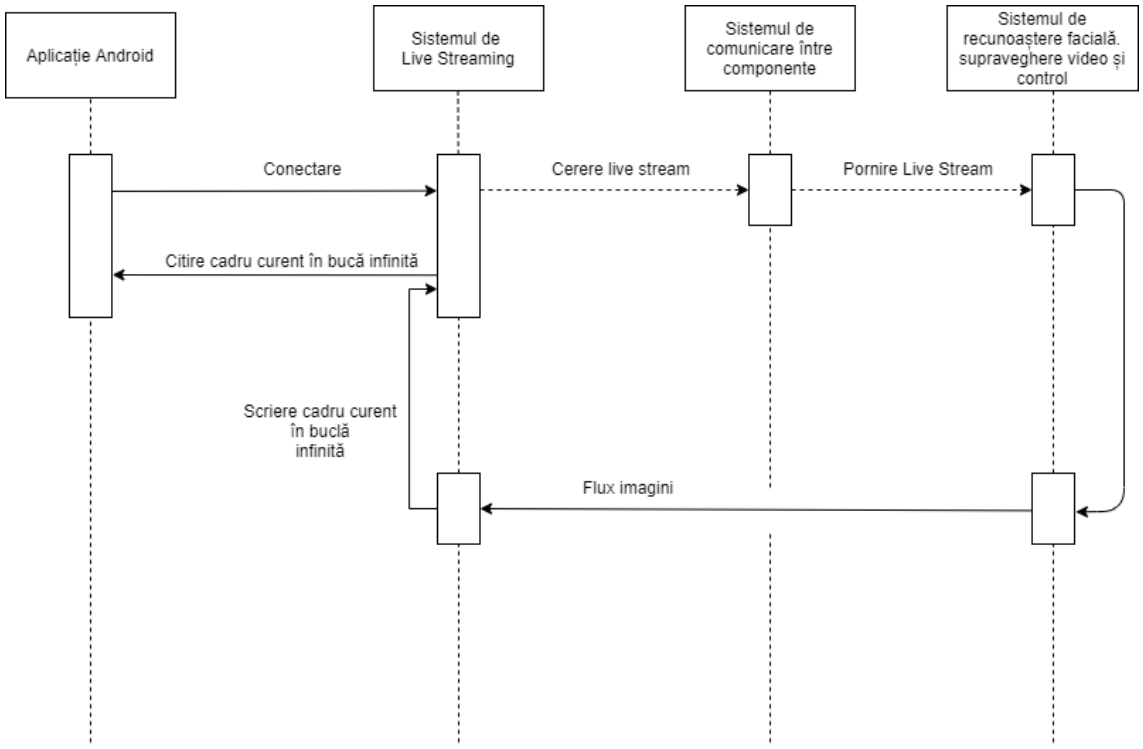

Figura 4.14: *Diagrama de secvență a serverului de Live Stream*

În continuare, vor fi prezentate și explicate instrucțiunile fiecărei proceduri din cadrul serverului:

- *listenForAndroidClientConnectionAndDispatchToThread(ExecutorService es)*: ascultă pe un socket la conexiuni, iar când se înițializează o conexiune se transmite controlul pe un alt thread, care trimite valoare imaginii curente către client, la intervale de 200ms, până ce acesta se deconectează. Instrucțiuni implementare:
  - inițializează un socket care pe portul 8001
  - pornește o buclă infinită care:
    - blochează Threadul curent până când un client se conectează pe socketul mai sus definit
    - de îndată ce se conectează un client, ia fluxul de date și îl trimite ca parametru, pe un Thread diferit, pentru fiecare client
- *handleClientRequestFromAndroid(DataOutputStream dos)*: procedură ce primește un canal de output în care se transmite informația către client
  - incremetează numărul de clienți, folosind o variabilă atomică (fiind un mediu multithreaded, pot exista situații în care două threaduri modifică în același timp aceeasi variabilă, rezultând valori eronate; o variabilă atomică rezolvă aceste probleme

  - pornește o buclă infinită care:
    * citește datele din variabila globală de clasă care conține imaginea curentă și o scrie pe canalul de output către client
    * blochează threadul timp de 100 ms

- *sendRabbitMQMessageCommands()*: metodă care, în funcție de numărul de clienți conectați la serverului de Live Streaming și de starea streamului (oprit/pornit) trimite comenzi de pornire/oprire a funcționalității de Live Streaming către sistemul de comunicare între componente, care va trimite aceste comenzi către Raspberry Pi. Modalitate implementare:

  - pornește o buclă infinită, care:
    * verifică dacă numarul curent de clienți conectați la serverul de Live Streaming este mai pozitiv (deci dacă sunt clienți conectați) și dacă deja streamingul nu este pornit.
      · dacă condiția de mai sus este adevarată, se serializează un obiect de tip Command, cu mesajul "START_LIVE_STREAM", și se trimite către serverul de comunicare între componente; se setează variabila care indică starea streamului la *true*
    * verifică dacă numărul de clienți este egal cu 0 și dacă streamul este deja pornit
      · dacă da, se serializează un obiect de tip Command, cu mesajul "STOP_LIVE_STREAM", și se trimite către serverul de comunicare între componente; se setează variabila care indică starea streamului la *false*
    * blochează threadul timp de 2s

- *handleStreamFromRaspberryPi()*: funcție care ascultă pe un socket la streamul de imagini venit de la Raspberry Pi, procesând caracter cu caracter datele primite, iar când se întâlnește caracterul delimitator, se copiază datele din bufferul temporar într-o variabilă globală, de unde citesc toate celelalte funcții. Mod implementare:

  - inițializare socket pe portul 8000
  - pornire buclă infinită, care, ascultă deschide socketul si blochează threadul până când se conectează Raspberry Pi-ul. De îndată ce acesta s-a conectat, se pornește altă buclă infinită, care citește, caracter cu caracter, datele primite, astfel:
    * verifică dacă caracterul curent este delimitator ('*'). Dacă este:
      · copiază datele transmise până în acest moment din buffer într-o variabilă locala din care se va forma imaginea curenta
      · se golește bufferul
      · se setează indexul curent la 0
    * dacă caracterul curent nu este delimitator, atunci:

- se pune caracterul curent în buffer
- se incrementează indexul curent

4.3 Sistemul de comunicare între componente

Sistemul de comunicare între componente este un server Linux, pe care este instalat brokerul de mesaje ce implementează protocolul AMQP, RabbitMQ. Acest broker acționează ca un releu, în sensul în care primește mesaje de la anumiți clienți și acele mesaje le redirecționează către alți clienți, folosindu-se de diverse tehnici de rutare.

4.3.1 Instalare

Serverul RabbitMQ este instalat pe o mașină virtuală ce rulează Ubuntu Server, folosindu-se Virtual Box pentru virtualizarea sistemului de operare Linux. După instalarea mașinii virtuale, se folosesc următoarele comenzi pentru instalarea serverului RabbitMQ:

Aducerea la zi a intregului sistem este recomandată înainte de orice instalare:

```
$ sudo apt-get update
```

Apoi instalarea propriu-zisă a serverului:

```
$ sudo apt-get install rabbitmq-server
```

Pentru a face acest server să ruleze ca și serviciu, trebuie rulate următoarele comenzi:

```
$ sudo systemctl start rabbitmq-server.service
$ sudo systemctl enable rabbitmq-server.service
```

4.3.2 Configurare

Implicit, RabbitMQ vine cu un user numit 'guest', ce are ca și parolă 'guest'. Pentru a elimina aceast user și implicit această problemă majoră de securitate (aceea de a lăsa pentru un server utilizatorul implicit), se va creea în continuare un utilizator cu drepturi de administrator, cu o nouă parolă, sigură. Comenzile de mai jos adaugă un user 'admin', cu parola '48kZuUdf' căruia i se atribuie tagul 'administrator', si toate permisiunile:

```
$ sudo rabbitmqctl add_user admin 48kZuUdf
$ sudo rabbitmqctl set_user_tags admin administrator
$ sudo rabbitmqctl set_permissions -p / admin ".*" ".*" ".*"
```

Consola de administrare și configurare web este în mod implicit dezactivată. Pentru a o putea activa, trebuie rulate următoarele comenzi:

```
$ sudo rabbitmq-plugins enable rabbitmq_management
$ sudo chown -R rabbitmq:rabbitmq /var/lib/rabbitmq/
```

Se poate verifica funcționarea consolei accesând Ip-ul mașinii ce ruleaza RabbitMQ, pe portul 15672, ca în captura de mai jos:

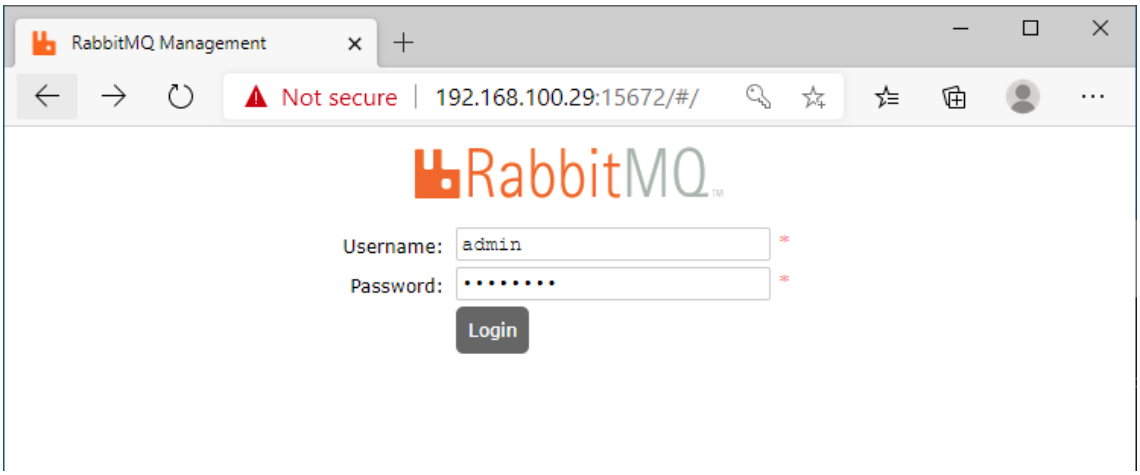

Figura 4.15: *Ecran de login RabbitMQ*

4.3.2.1 Topologie

Configurarea folosită din punct de vedere al topologiei RabbitMQ folosește două Exchange-uri:

- comenzi
- poze

Pentru exchange-ul 'comenzi' este creată o coadă cu același nume, folosind o legatură de rutare directă bazată pe o cheie de rutare cu același nume, astfel încât fiecare mesaj trimis de un publisher pe exchange-ul 'comenzi', având cheia de rutare 'comenzi' va ajunge în coada cu același nume. Fiecare utilizator conectat la această coadă va putea primi mesajul, pe principiul primul venit, primul servit (FIFO - first in first out).

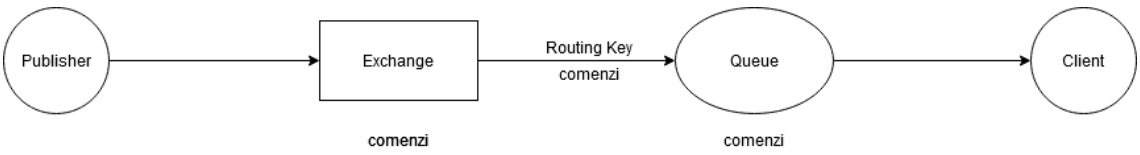

Figura 4.16: *Topologie de rutare a comenzilor către Raspberry Pi*

Pentru exchange-ul 'poze', rutarea este diferită, întrucât este de tipul 'fanout', ceea ce înseamnă că orice coadă legată la exchange-ul 'poze' va primi o copie a mesajului trimis pe exchange de către un publisher. Spre deosebire de celălalt caz, nu există un queue definit manual, deoarece se va creea în mod dinamic atunci când un telefon ce deține aplicația

Android se va conecta la server. Pentru acest tip de rutare, nu este nevoie de o cheie de rutare, deoarece mesajul se va duce automat către fiecare coadă legată la exchange.

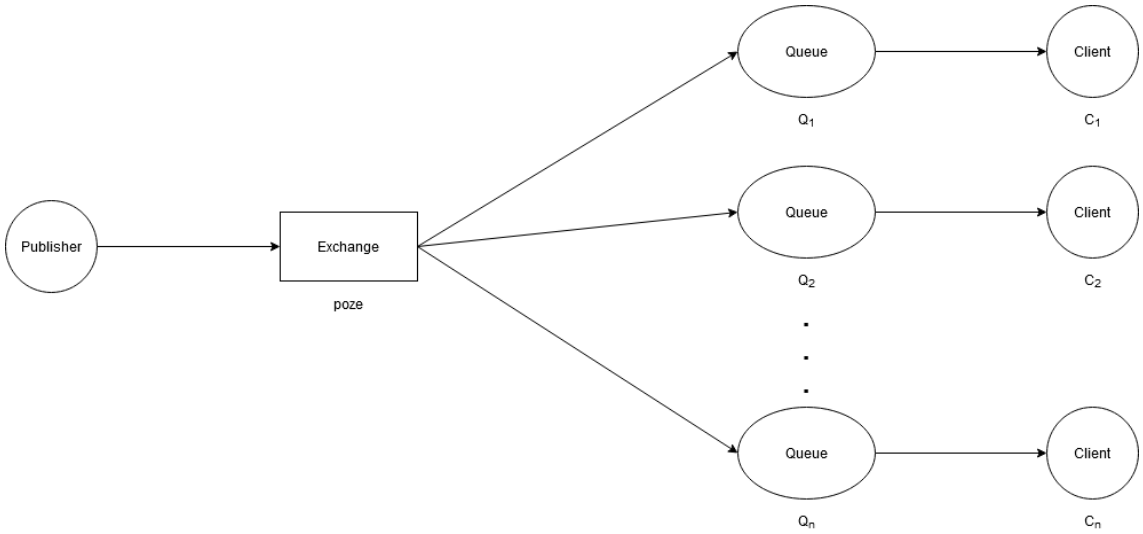

Figura 4.17: *Topologie de rutare a pozelor către telefoanele cu aplicația Android instalată*

Exemplu de implementare pentru exchange-ul de comenzi și coada de poze în care apar persoanele detectate:

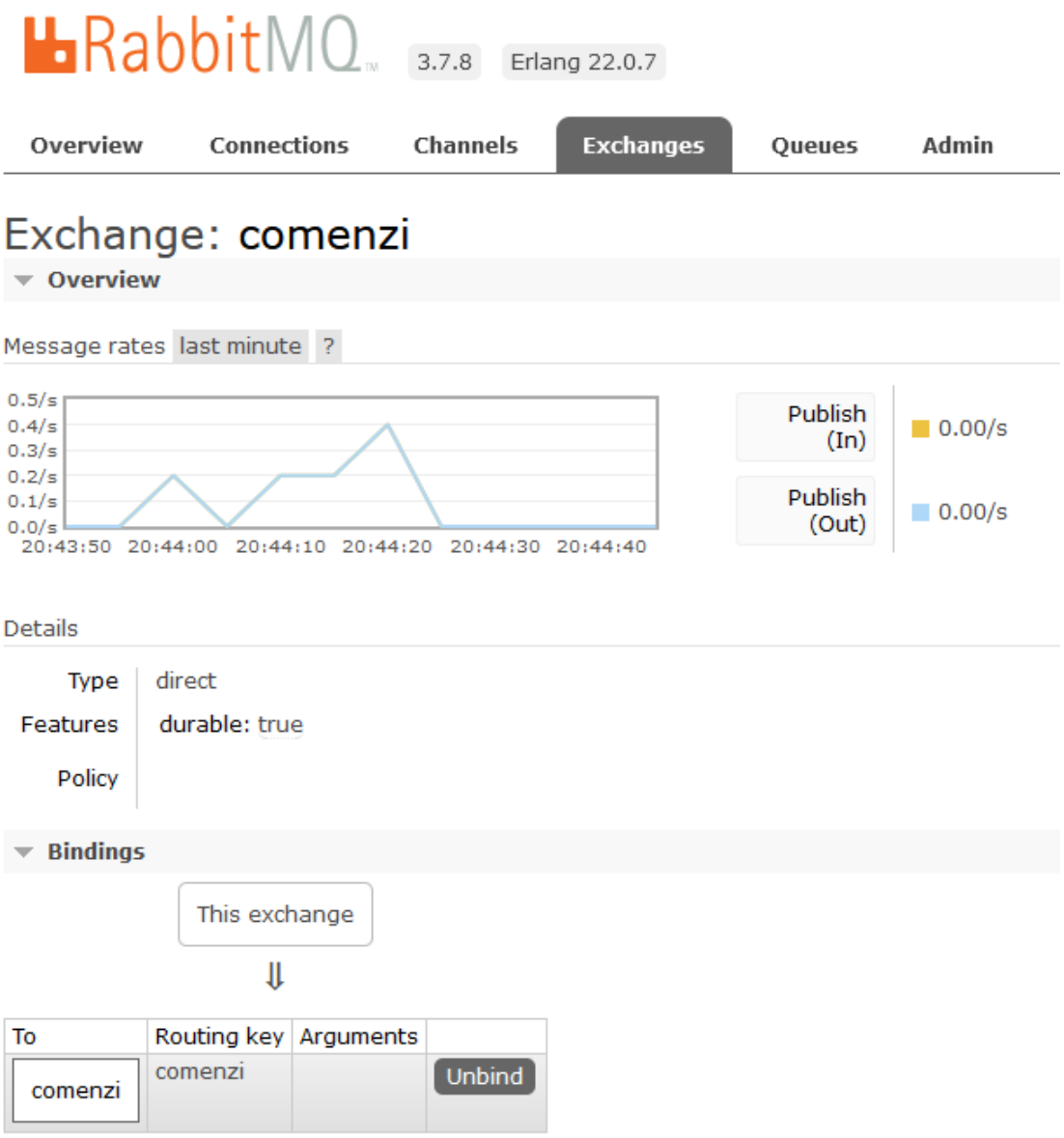

Figura 4.18: *Implementare exchange de comenzi în RabbitMQ*

Figura 4.19: *Implementare coadă dinamică pentru poze în RabbitMQ*

4.4 Aplicația Android

Centrul de alertare și comandă îl constituie aplicația nativă Android, în care utilizatorul este notificat de fiecare detectare realizată cu succes, poate vedea istoricul detecțiilor și poate șterge pozele vechi. Frecvența notificărilor se poate seta dinamic, utilizatorul având la dispoziție trei optiuni:

1. frecvență mică - se va trimite o notificare la fiecare minut
2. frecvență medie - se va trimite o notificare la fiecare 30 secunde
3. frecvență medie - se va trimite o notificare pe secundă

Tot în aplicația Android utilizatorul poate intra să vadă ce vede microcontrolerul, adica să acceseze fluxul în timp real de imagini. De aici, se poate alege și calitatea streamingului, având următoarele alegeri:

- calitate mică
- calitate medie

- calitate mare

Mai jos este prezentată diagrama de activitate, ce arată schema logică și prezintă pașii de decizie ai aplicației Android:

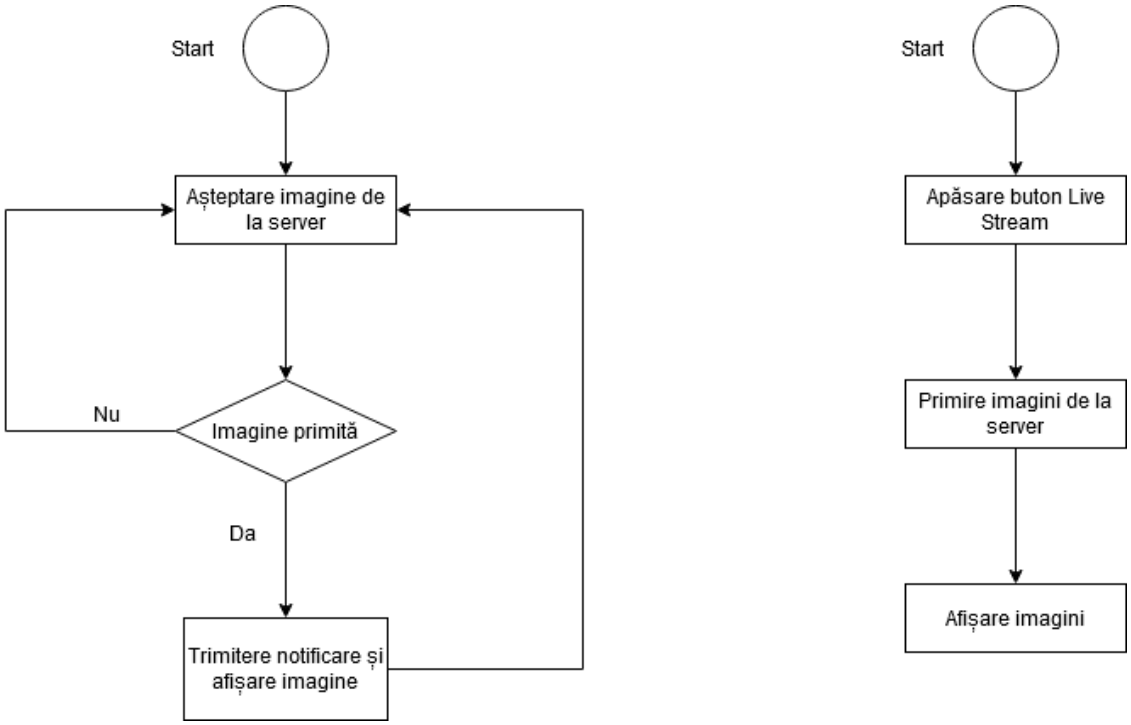

Figura 4.20: *Diagrama de activitate a aplicației Android*

De notat este faptul că pe măsură ce calitatea streamului crește, va încetini rata de transmitere a imaginilor (framerate-ul); rezultă că viteza de transmitere a imaginilor este invers proporțională cu calitatea acestora.

O importantă funcționalitate este aceea de comandă a microcontrolerului de la distanță, utilizatorul putând să încuie/descuie ușa sau să pornească/oprească alarma.

4.4.1 Ecranul principal

În acest ecran, utilizatorul poate alege să intre pe următoarele ecrane, ce vor fi detaliate în continuare: ’PERSOANE DETECTATE’, ’LIVE STREAM’, SETĂRI’.

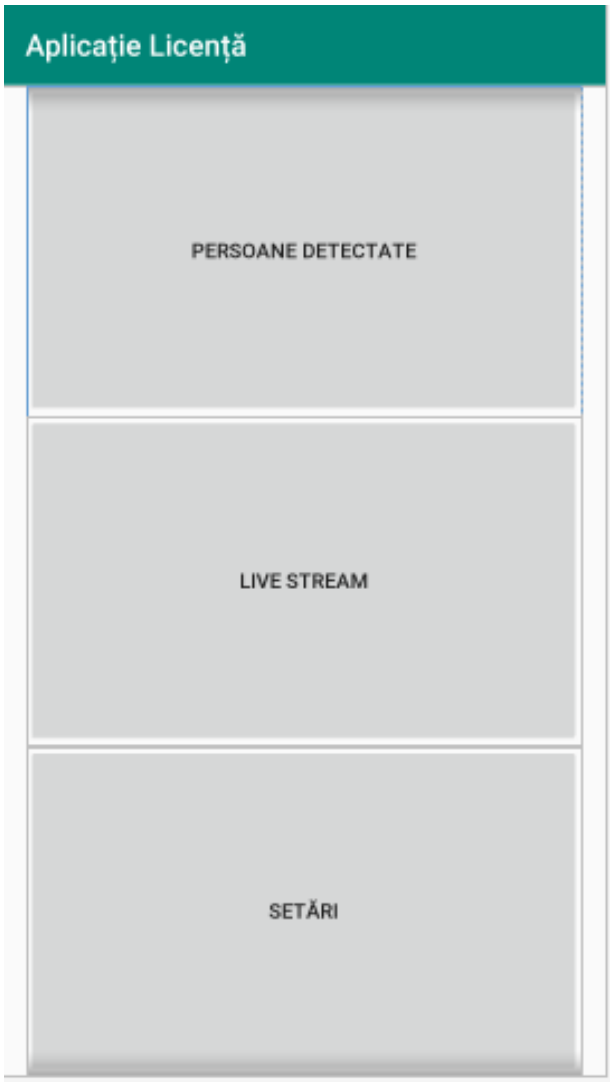

Figura 4.21: *Ecranul principal al Aplicației Android*

Pentru asigurarea funcționalității acestui ecran, este creată o clasă Java ce extinde clasa ’AppCompatActivit’, precum în următoarea diagramă UML de clasă:

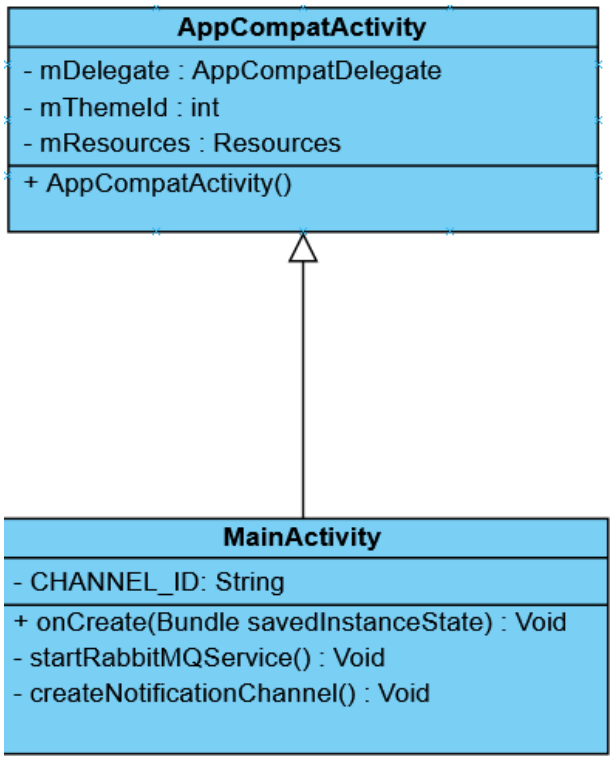

Figura 4.22: *Diagrama de clasă a ecranului principal*

În metoda *onCreate* a clasei *MainActivity* sunt atașați listeneri pentru fiecare dintre butoanele ecranului, astfel încât atunci când utilizatorul apasă pe buton, este trimis la ecranul corespunzător prin intermediul unui *Intent*. Metoda *startRabbitMQService* pornește serviciul care va rula într-un fir de execuție din fundal, așteptând mesaje de la RabbitMQ.

4.4.2 Primirea notificărilor

Pentru asigurarea funcționalității de notificări automate în cazul detectări fețelor, este nevoie de un serviciu ce rulează în fundal, pe un Thread separat, care așteaptă și interceptează mesajele de la RabbitMQ, iar apoi le trimite sub formă de notificare; acest lucru este asigurat de clasa *RabbitMQListenerService*.

Această clasă extinde clasa *Service*, deci moștenește astfel toate funcționalitățile unui serviciu Android. Atunci când este instanțiată, inițiază o conexiune către RabbitMQ, și atașează listenerii corespunzători pentru mesajele trimise cu succes, dar și pentru cele de eroare.

Procedura ce se conectează la RabbitMQ este ea însăși rulată într-un fir de execuție separat, rezultând nevoia ca în cazul închiderii serviciului, să se trimită închidă Threadul. (acest lucru este făcut în metoda *onDestroy*)

Atunci când un mesaj este primit, se verifică distanța între primirea mesajului curent și primirea mesajului anterior, și se compară apoi cu valoarea setată în setările aplicației, astfel încât un mesaj este concretizat într-o notificare doar daca distanța între două mesaje

consecutive este mai mare decât setarea aleasă. Dacă acesta este cazul, se trimite un mesaj către utilizator sub formă de notificare, și se setează data ultimului mesaj primit la data curentă.

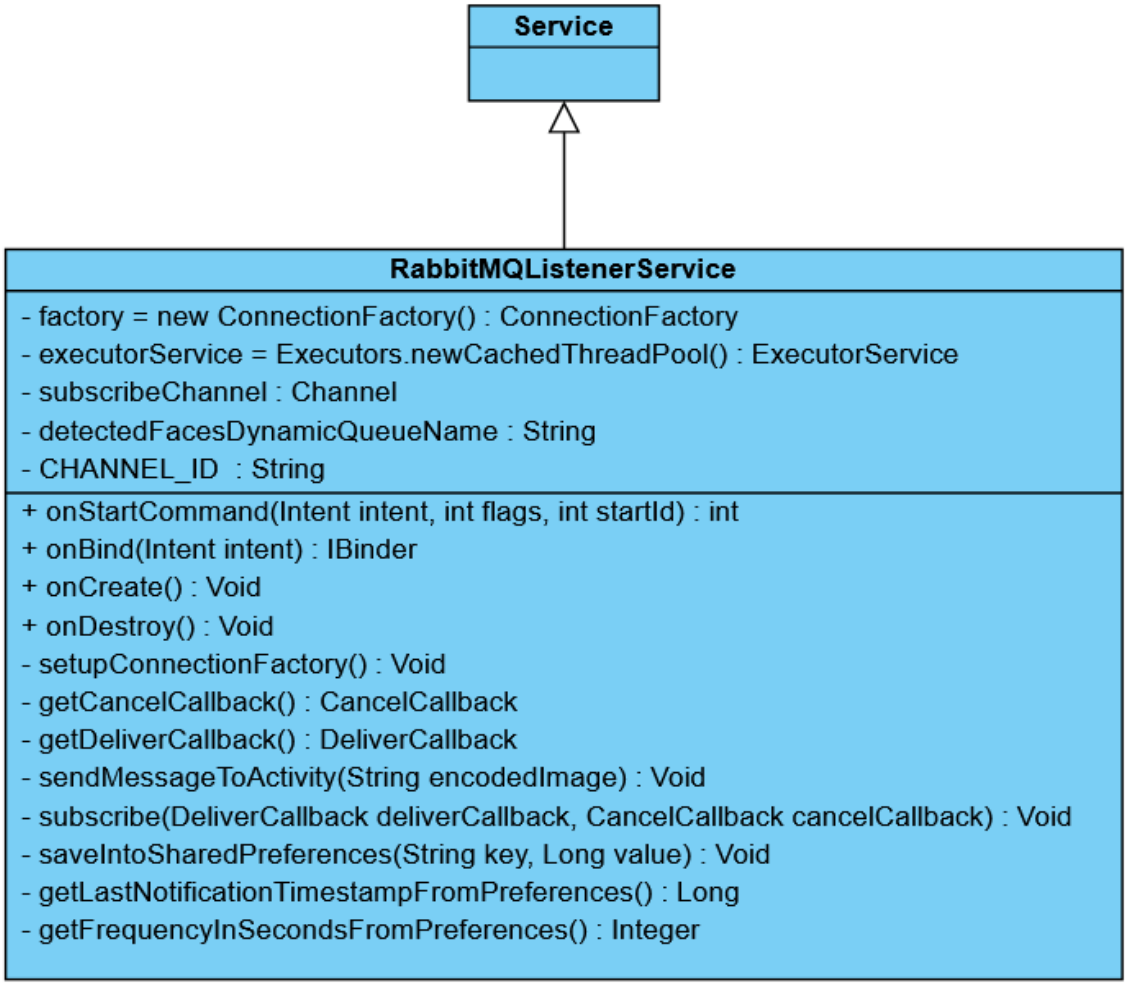

Figura 4.23: *Diagrama de clasă a serviciului ce ascultă la mesaje RabbitMQ*

Dacă utilizatorul apasă pe notificare, va fi trimis către ectranul de vizualizare a detecțiilor feței. De reținut este că utilizatorul poate vedea poza cu persoana detectată direct din notificare, fără a fi nevoie să intre în aplicație, după cum este prezentat în imaginea de mai jos:

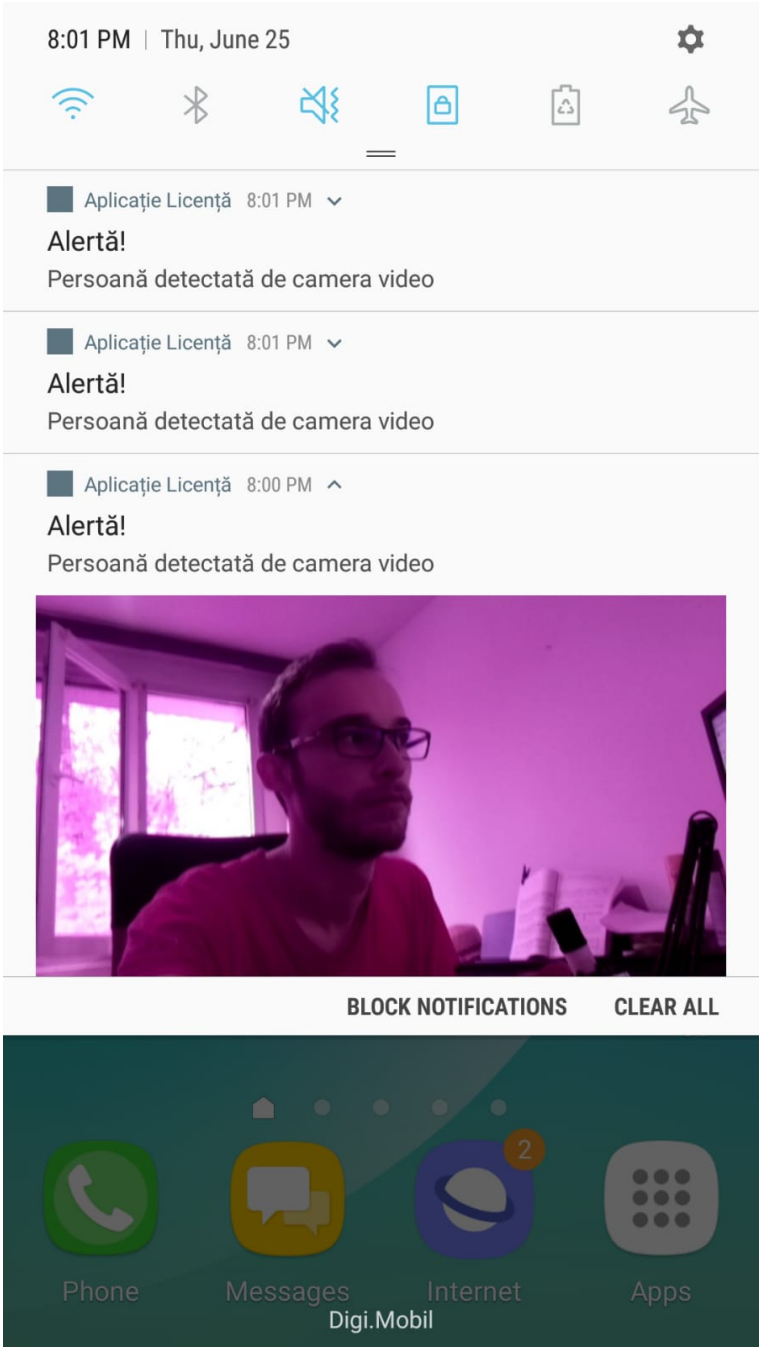

Figura 4.24: *Notificare față detectată*

4.4.3 Ecranul de vizualizare a detecțiilor

În acest ecran utilizatorul poate vedea cea mai recentă detecție făcută de sistemul de detecție, dar și istoricul detecțiilor, putând să șteargă sau nu imaginile vechi.

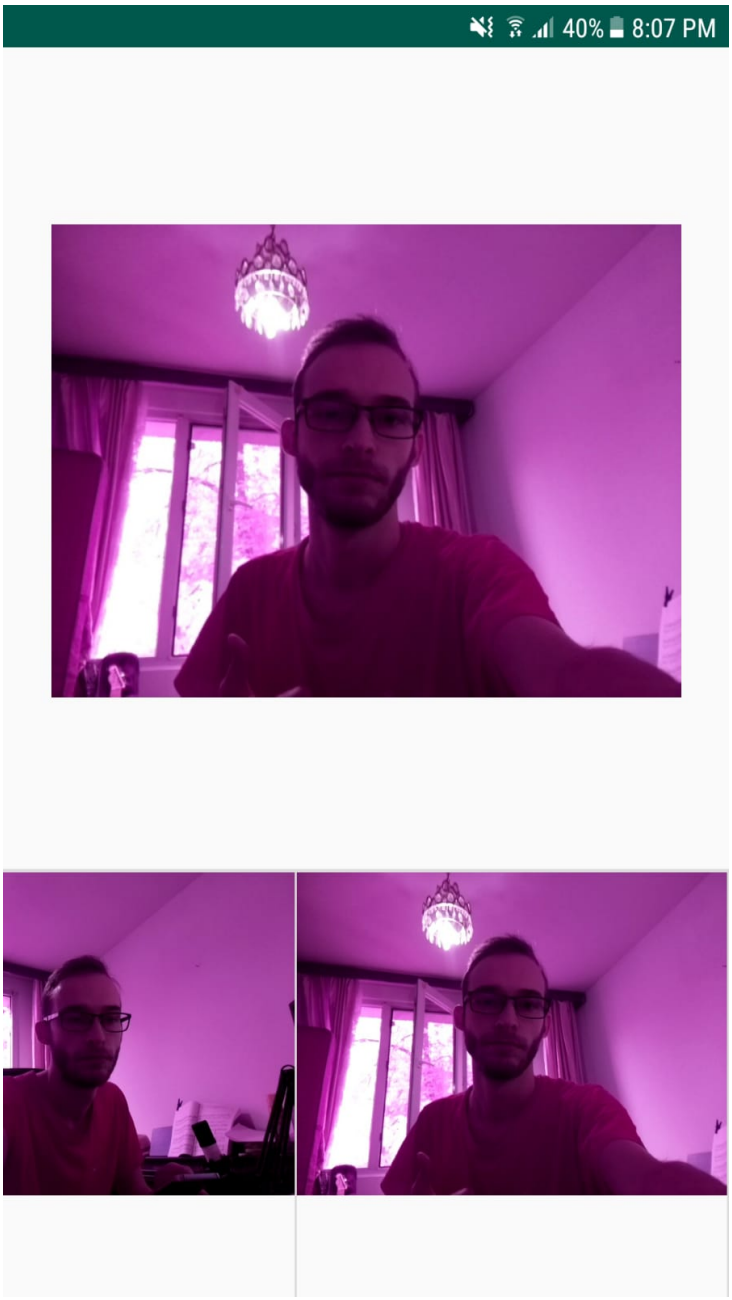

Figura 4.25: *Ecranul de vizualizare a detecțiilor*

Pentru asigurarea acestei funcționalități, este folosită activitatea *DetectedFacesFeedActivity* care ascultă la mesajele de la serviciul menționat în subcapitolul anterior. Atunci când primește un mesaj, schimbă imaginea principală, salvează în memoria telefonului poza curentă, și o adaugă în lista istoricului. Atunci când utilizatorul dă swipe up sau down pe o poză din lista istoricului, aceasta este ștearsă.

Diagrama de clasă aferentă ecranului de vizualizare a detecției fețelor este următoarea:

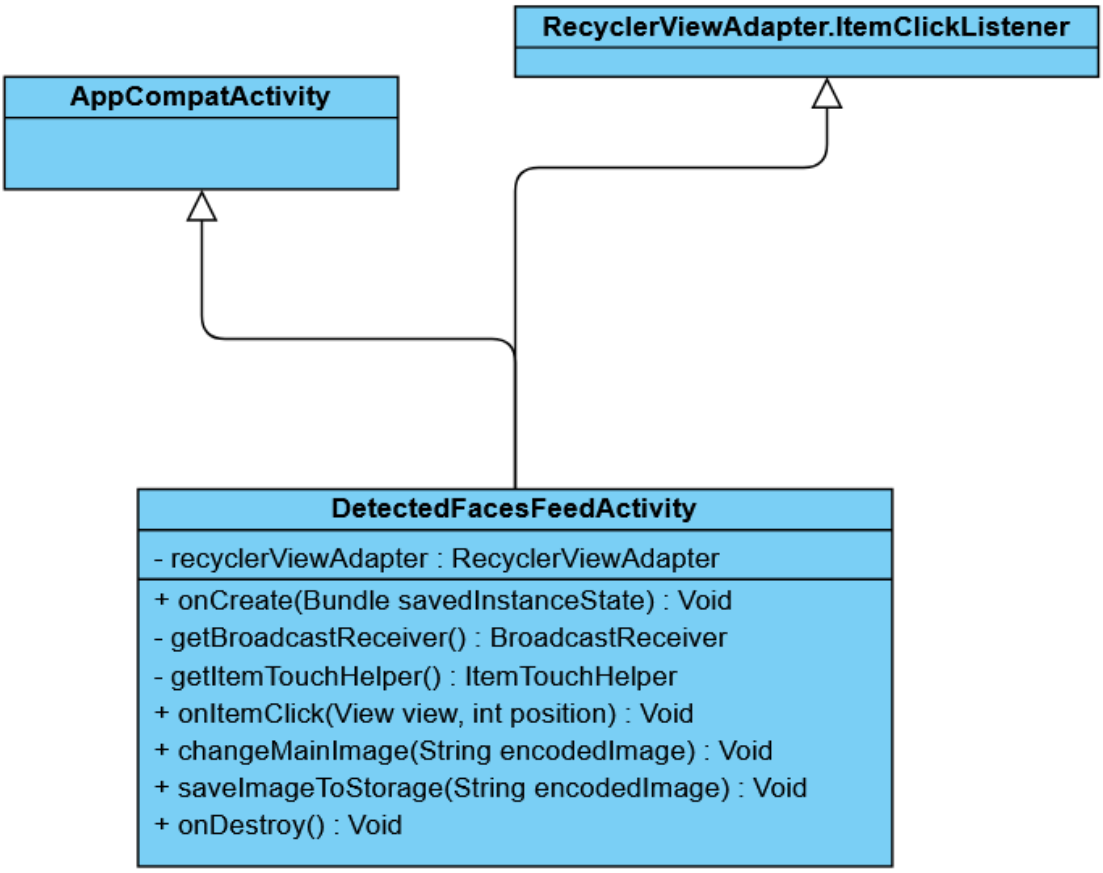

Figura 4.26: *Diagrama de clasă a ecranului de vizualizare a detecțiilor*

4.4.4 Ecranul Live Stream

Ecranul Live Stream este un ecran ce conține o mai multe funcționalități:

- vizualizarea în timp real a fluxului de imagini venite de la Raspberry Pi
- Alegerea calității fluxului
- Trimiterea comenzii de încuiere/descuiere a ușii locuinței
- Trimiterea comenzii de pornire/oprire a alarmei conectate la Raspberry Pi

În continuare vor fi explicate metodele prin care au fost obținute aceaste funcționalități, plecând de la diagrama de clasă a ecranului:

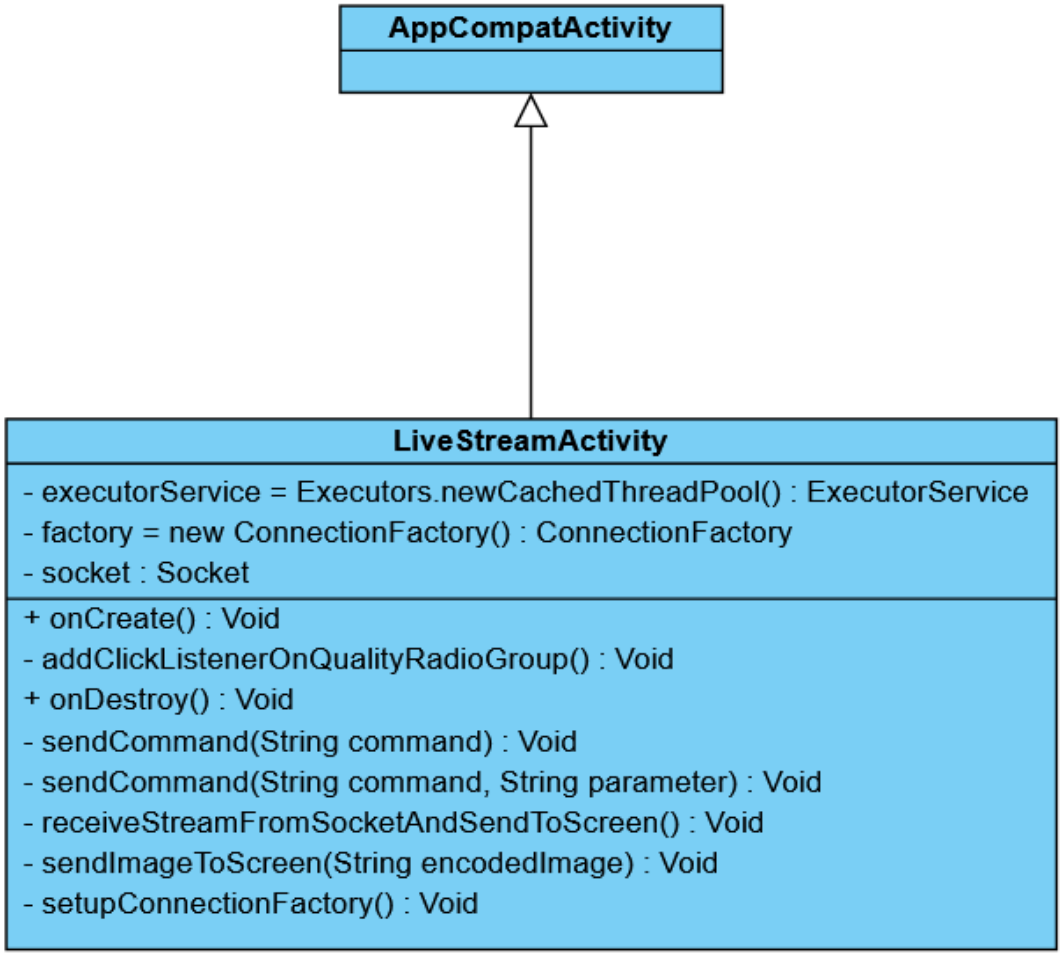

Figura 4.27: *Diagrama de clasă a ecranului de Live Stream*

Explicarea în linii mari a funcționalității metodelor importante:

- *onCreate*
  - apelează metoda ce inițialiează conexiunea către RabbitMQ
  - apelează metoda ce atașează listenerii corespunzători butoanelor (de alegerea calității și butoanelor de încuiat/descuiat ușa si pornire/oprire alarmă)
  - apelează metoda ce se conectează prin socket către serverul de Live Stream și preia fluxul de imagini și îl afișează
- *sendCommand*
  - trimite o comandă formatată ca json către RabbitMQ. Această metodă este folosită atunci când utilizatorul apasă pe butoanele de încuiere/descuiere ușă, pornire/oprire alarmă

- *receiveStreamFromSocketAndSendToScreen*
  - deschide un nou fir de execuție cu ajutorul *executorService*, și în acesta se conectează la nivel de socket la serverul de Live Stream
  - într-o buclă infinită citește fluxul de imagini și îl afișează pe ecran

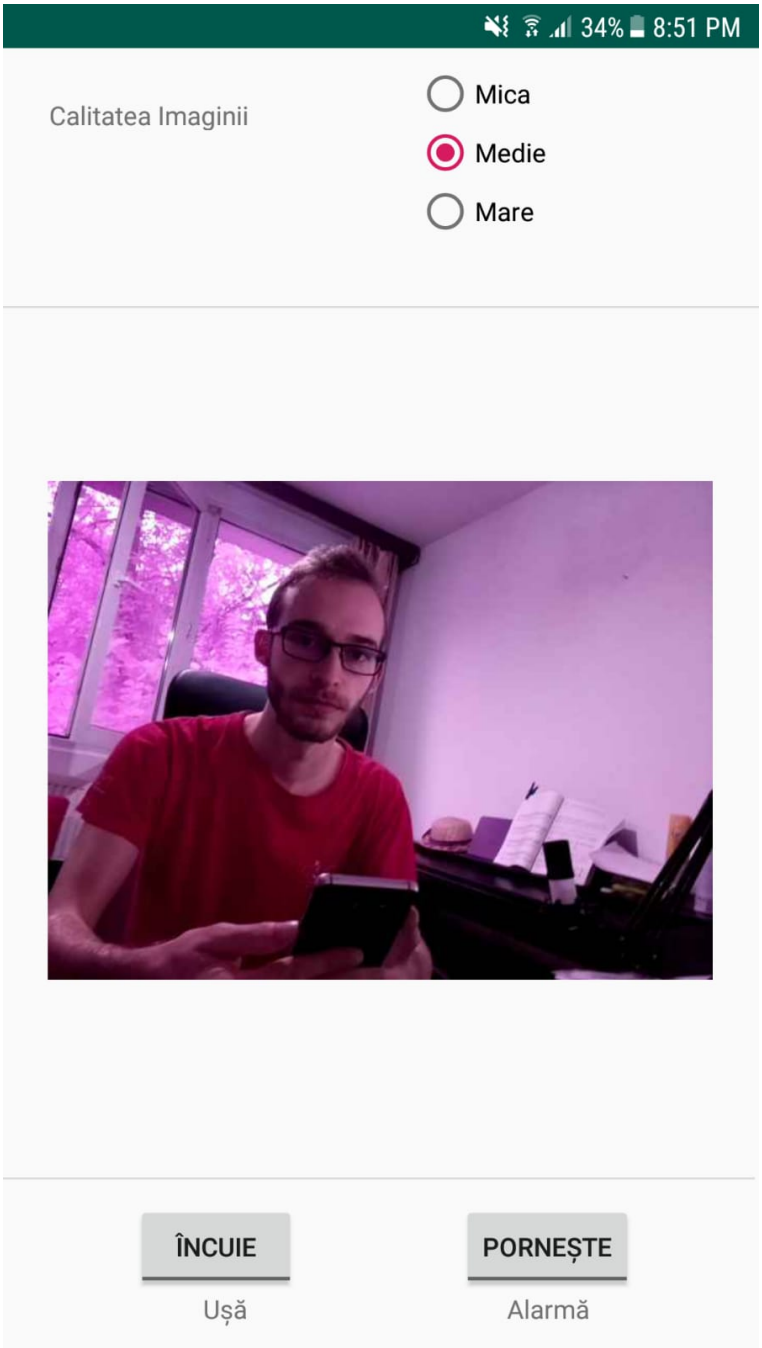

Figura 4.28: *Ecranul de Live Stream*

4.4.5 Ecranul de setări

În cadrul acestui ecran, utilizatorul poate selecta frecvența cu care dorește să fie notificat în legatură cu persoanele detectate. Valorile pe care le poate selecta sunt următoarele:

- mică - 1 notifcare / minut
- medie - 1 notificare / 30 secunde
- mare - 1 notificare / secundă

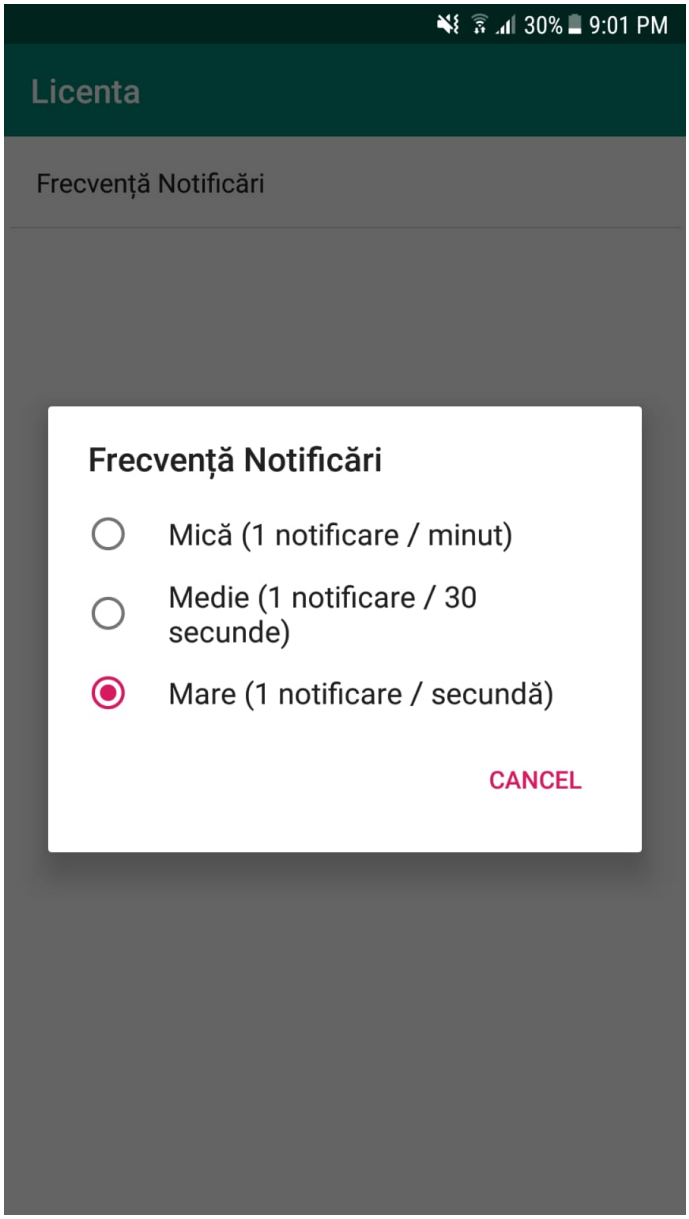

Figura 4.29: *Ecranul de setări*

Capitolul 5

Concluzii

Dezvoltarea acestei aplicații a fost în concordanță cu pașii de dezvoltare a oricărui produs: planificare, proiectare, implementare, testare și punere în funcțiune, fiecare pas fiind parcurs cu succes. Cea mai importantă caracteristică a acestui proiect este detectarea facială automată, care îl deosebește de sistemele de supraveghere clasice, unde un utilizator trebuia să piardă timp în timp ce vizualiza imaginile live. În acest sistem, un utilizator trebuie doar să instaleze aplicația de mobil și va fi automat notificat atunci când o persoană este detectată, și va putea apoi să intre să verifice imaginile live.

O altă caracteristică importantă este pornirea doar la nevoie a funcționalității de Live Stream, rezultând eficiență. Acest lucru este datorat gestionării inteligente a conexiunilor la nivelul serverului de Live Stream, în sensul că atunci când niciun dispozitiv nu este conectat, va trimite un mesaj către microcontroler și acesta din urma va opri streamingul. Acest lucru crește eficiența de procesare cu aproximativ **30%**.

De asemenea, de notat este faptul că deși nu este inovator (dar este foarte util), utilizatorul poate trimite comenzi direct către microcontroler pentru încuierea/descuierea ușii, dar și pentru pornirea/oprirea alarmei. Acest lucru nu se face prin conexiune directă pentru a maximiza siguranța, deoarece conexiunea directă ar implica deschiderea spre internet (deci spre oricare atacator) a rețelei locale.

5.1 Contribuții personale

În dezvoltarea serverului de Live Stream am folosit conceptul de *Multithreading* pentru a gestiona conexiunile paralele venite de la oricâte aplicații Android, dar și pentru a implementa un algoritm pentru comunicarea eficientă la nivel de socket cu Raspberry Pi și cu aplicațiile Android pentru livrarea de imagini live. Tot pe acest server am gestionat în mod eficient conexunile, oprind sau pornind funcționalitatea în funcție de numărul de clienți.

În dezvoltarea procedurilor de pe microcontrolerul Raspberry Pi, am implementat un sistem de gestionare a mesajelor primite de la serverul de mesagerie și acționare a serviciilor în concordanță; inclusiv aici a fost folosit foarte mult conceptulde paralelizare și *Multithreading*, rezultând folosirea la maxim a capacității de procesare a microcontrolerului.

Pe parcursul dezvoltării acestei aplicații, am aprofundat cunoștintele de Java, și am învățat și am aprofundat limbajul de programare Python, care este în prezent unul dintre cele mai folosite limbaje de programare din lume. Am folosit și conceptul de inteligență artificială și detecție facială, deschizându-mi astfel drumul către acest nou orizont al tehnologiei, încotro se îndreaptă acum fiecare ramură a fiecărei industrii ce poate fi automatizată și optimizată.

5.2 Perspective de viitor

Pentru îmbunătățirea aplicației pe viitor vreau să adaug un senzor de mișcare pe microcontroler astfel încât detecția de fețe să nu ruleze mereu, ci doar atunci când este declanșat senzoru. Acest lucru ar crește eficiența procesorului microcontrolerului foarte mult. Tot la nivelul microcontrolerului se poate adăuga un script care clasifică o persoană după ce a fost detectată, astfel putând să dăm acces persoanelor în locuință doar bazându-ne pe fața acestora.

O altă funcționalitate pe care aș vrea să o dezvoltat pe viitor este crearea unei interfețe web care să complementeze aplicația de Android, astfel încât utilizatorului să-i fie foarte ușor să vizualizeze și să comande sistemul de supraveghere inteligent, de pe orice dispozitiv cu acces la internet.

[1] *Viola–Jones object detection framework*, . URL https://en.wikipedia.org/wiki/Viola-Jones_object_detection_framework.

[2] *Client-server*, . URL https://ro.wikipedia.org/wiki/Client-server.

[3] *Advanced Message Queueing Protocol*, . URL https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol.

[4] *Eclipse (software)*, . URL https://en.wikipedia.org/wiki/Eclipse_(software).

[5] *Notepad++*, . URL https://ro.wikipedia.org/wiki/Notepad\%2B\%2B.

[6] *Android Studio*, . URL https://ro.wikipedia.org/wiki/Android_Studio.

[7] *Java (limbaj de programare)*, . URL https://ro.wikipedia.org/wiki/Java_(limbaj_de_programare).

[8] *Python*, . URL https://ro.wikipedia.org/wiki/Python.

[9] *OpenCV*, . URL https://opencv.org/about/.

[10] *Spring Framework*, . URL https://en.wikipedia.org/wiki/Spring_Framework.

[11] *Face Detection using Haar Cascades*, . URL https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_objdetect/py_face_detection/py_face_detection.html.

[12] *Apache Maven*, . URL https://ro.wikipedia.org/wiki/Apache_Maven.

[13] *RabbitMQ*, . URL https://www.rabbitmq.com/features.html.

[14] *VirtualBox*, . URL https://en.wikipedia.org/wiki/VirtualBox.

[15] *Virtual Network Computing*, . URL https://en.wikipedia.org/wiki/Virtual_Network_Computing.

[16] *Git*, . URL https://ro.wikipedia.org/wiki/Git.

[17] *SSH*, . URL https://ro.wikipedia.org/wiki/SSH.

[18] *WinSCP*, . URL https://ro.wikipedia.org/wiki/WinSCP.

[19] *Integral image*, . URL https://en.wikipedia.org/wiki/Summed-area_table.

[20] *Thread (computing)*, . URL https://en.wikipedia.org/wiki/Thread_(computing).

[21] *Multithreading*, . URL https://ro.wikipedia.org/wiki/Multithreading.

[22] *Aplicații de retea. Socluri. URL.*, . URL https://ms.sapientia.ro/~manyi/teaching/oop/oop_romanian/curs9/curs9.html.

[23] Yoav Freund and Robert E.Schapire. *A Short Introduction to Boosting*, Septembrie 1999. URL https://cseweb.ucsd.edu/~yfreund/papers/IntroToBoosting.pdf.

[24] Virginia Săndulescu. *Note de curs "Rețele de calculatoare"*, 2020.

[25] Paul Viola and Michael Jones. *Rapid Object Detection Using a Boosted Cascade of Simple Features*, Mai 2004. URL https://www.merl.com/publications/docs/TR2004-043.pdf.