Multi Client Multi Security Camera Master Control Server with Motion Detection

Bichilie Tudor Ovidiu

UTCN AC – CTI

30234

Cuprins

1. Introducere

1.1 Scop:

Acest proiect reprezintă o aplicație de supraveghere video client – server cu sisteme avansate de recunoașterea mișcării bazată pe cadre consecutive.

Este permis în a fii accesat serverul de către mai mulți clienți, aceștia specificând către ce dispozitiv doresc să fie conectați.

Serverul dispune de master control asupra clienților și le poate limita accesul la informație și totodată, îi poate deconecta.

Aplicația poate fii utilizată de către personae fizice cât și juridice pentru a asigura securitatea unor încăperi în momente în care se dorește acestea să fie lipside de mișcare / activitate, beneficiand de sistem de alarmă către client în momentul în care mișcarea este detectată.

1.2 Specificatii tehnologice:

Mediul de lucru folosit a fost IntelliJIdeea pentru programare java, mediu ajustat cerințelor programului spre a menține mai multe procese deschise simultan și a le manageria conform așteptărilor.

S-a introdus additional în mediul de lucru librăria externă OpenCV pentru a facilita proiectul cu interacțiunea între program și camera web sau de supraveghere (în funcție de ce este disponibil și conectat la server).

Pentru bună funcționare aplicația lucrează pe threaduri sincronizate în scop de a furniza cea mai bună funcționare.

Relația server – clienți este realizată cu ajutorul librăriilor java ServerSocket.

1.3 Obiective:

Obiectivele acestui proiect sunt de a face această tehnologie mai accesibilă în cazul în care nu se dorește performanță maximă, dar permisiuni de modelare și de familiarizare cu relația server-clienți.

2. Studiu

În cadrul studiului meu asupra lucrului cu VideoCapture în OpenCV, am constatat că acesta reprezintă o componentă esențială pentru prelucrarea și analiza de fluxuri video în aplicațiile de viziune artificială. Folosind biblioteca OpenCV în limbajul java, am învățat să inițializez VideoCapture, să deschid și să capturez fluxuri video de la diferite surse, precum camere web sau fișiere video. De asemenea, am explorat modul în care putem procesa fiecare cadru al acestor fluxuri pentru analiza de date.

Punctul culminant în cadrul studiului a fost înțelegerea de a folosi clasa Mat pusă la dispoziție de OpenCV. Toate funcțiile se învârt în jurul acestui format deoarece este felul în care datele sunt stocate, în cazul de față, cadrele luate de camera, urmând apoi că imaginea să fie prelucrată din formaltul Mat în BufferedImage pentru a fi urcată într-un label de Swing:

```
private BufferedImage matToBufferedImage(Mat mat) {
   int type = BufferedImage.TYPE_BYTE_GRAY;
   if (mat.channels() > 1) {
      type = BufferedImage.TYPE_3BYTE_BGR;
   }
   int bufferSize = mat.channels() * mat.cols() * mat.rows();
   byte[] b = new byte[bufferSize];
   mat.get(0, 0, b);
   BufferedImage image = new BufferedImage(mat.cols(), mat.rows(), type);
   final byte[] targetPixels = ((DataBufferByte) image.getRaster().getDataBuffer()).get Data();
   System.arraycopy(b, 0, targetPixels, 0, b.length);
   return image;
}
```

3. Analiza

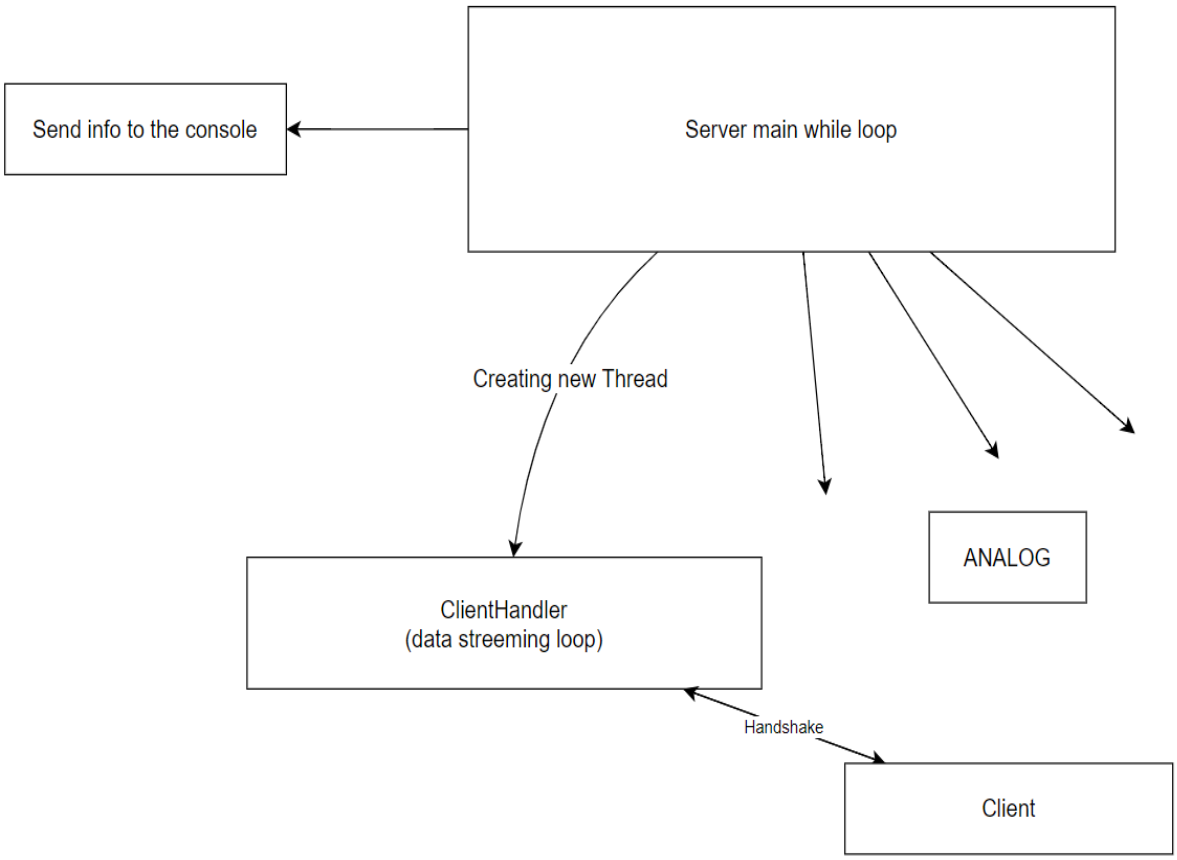
3.1 Server:

În cadrul serverului a trebuit adoptată o metodă de a acceptă mai mulți clienți și de a transmite simultan imagine fiecarora.

Serverul când se deschide își pregătește resursele necesare pentru rulare, cum ar fii inițierea librăriei OpenCV și deschierea camereleor de Vedere conectate la unitate pentru a putea putea transmite în timp optim clientului.

Am mers pe mulți threading, astfel mereu când un client este conectat serverul în buclă principala stabilește niște parametrii în care deschide un thread de rulare pentru program prin clasa ClientHandler.

ClientHandler trebuie să aibă anumite flaguri în federe pentru a transmite un flux de date continu și armonios drept rezultat ajungând la controlul informației trimise mai departe.



3.2 Client:

În ceea ce privește clientul, treaba principala a acestuia e să pună în aplicare informația trimisă de server, primește imaginea că o succesiune de bytes, transpusă în BufferedImage și o încarcă în laber.

Primește de la server un flag important pentru alarmă, cadrele sunt analizate în server pentru detectarea mișcării și trimise sub formă de Boolean pentru a semnaliza evenimentul.

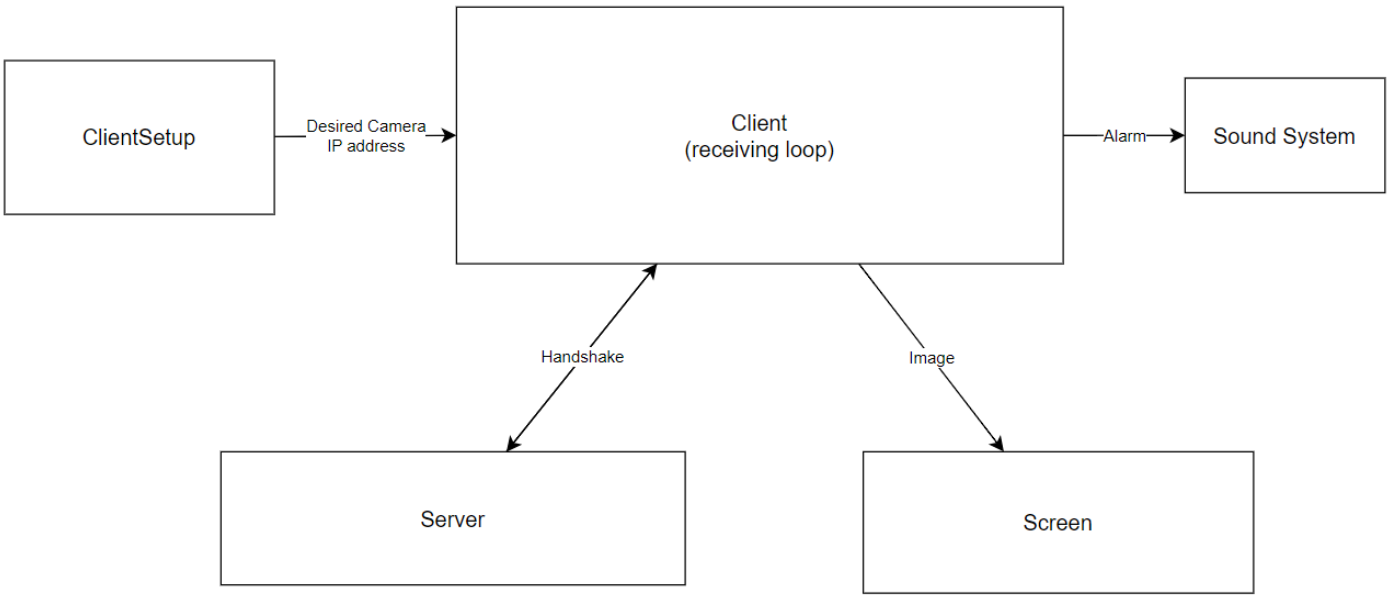
Clientul pornește o alarmă pentru cazul în care flagul respective este True și deschide un timer pentru a nu porni alarmă de prea multe ori simultan în caz de evenimente continue.

Prezintă clasa pentru urcarea, prelucrarea și pornirea alarmei în cazul în care este necesar.

3.3 Handshake:

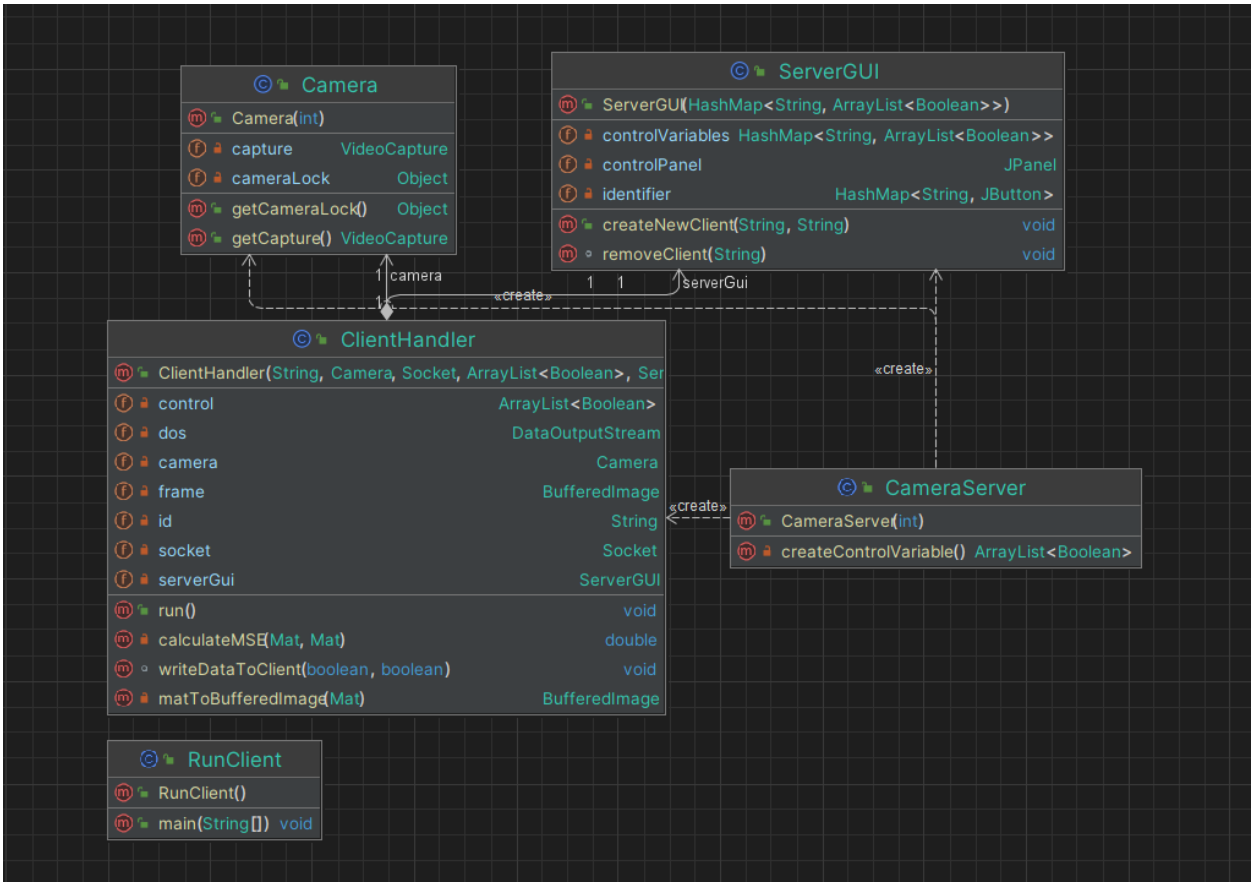
Intre client si server se realizeaza permanent un Handshake:

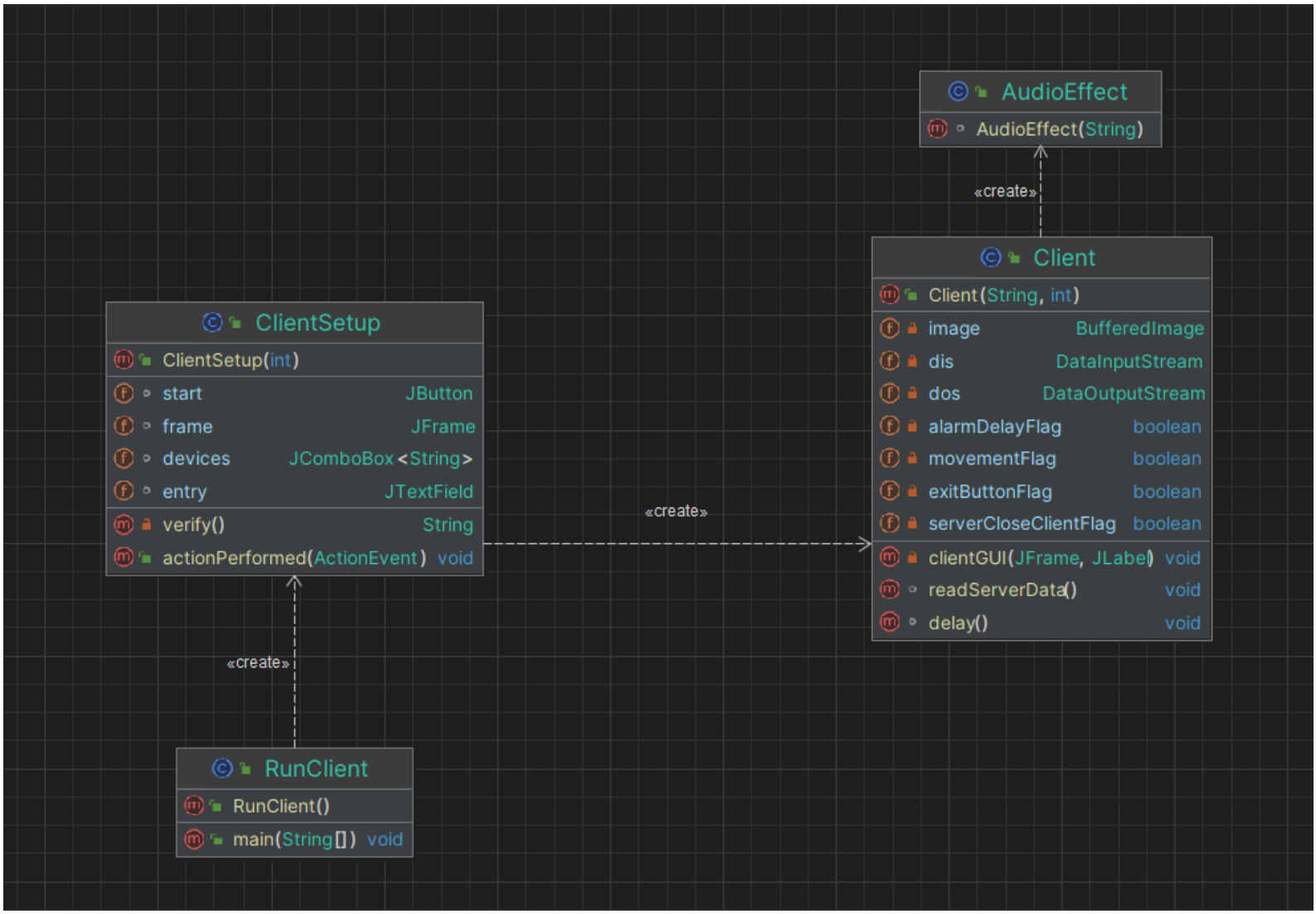
- La inceputul executiei clientul ii transmite serverului la ce dispozitiv doreste sa fie conectat



- In fiecare iteratie a buclei serverul si clientul isi comunica unul altuia daca mai doresc sa fie conectati, astfel serverul eliberand resursele clientului cand acesta inchide sau cand este inchis de server.

4. Design





În cadrul designului las diagramele UML ale proiectului din care reiese structurarea și logică de funcționare.

Am mers pe model classic, fără metode statice sau moșteniri pentru a nu oferii așa mare access pentru obiecte cu instantieri globale pe clasa.

5. Implementare

Pentru implementare voi prezemta clasele principale si cateva cuvinde despre metodele cheie folosite:

```
public class CameraServer {

  public CameraServer(int connectedDevicesCount) {
    System.loadLibrary(Core.NATIVE_LIBRARY_NAME);

    HashMap<Integer, Camera> cameras = new HashMap<>();
    HashMap<String, ArrayList<Boolean>> controlVariables = new HashMap<>();
    ServerGUI serverGUI = new ServerGUI(controlVariables);
    AtomicInteger clientIDs = new AtomicInteger(1);

    try (ServerSocket serverSocket = new ServerSocket(2505)) {
      System.out.println("Server is starting, please wait...");
      Thread.sleep(4000);

      System.out.println("Console log >>\n");
      System.out.println("> Server is running. Waiting for a client to connect...");

      for (int i = 0; i < connectedDevicesCount; i++) {
        cameras.put(i, new Camera(i));
      }
      while (true) {
        Socket clientSocket = serverSocket.accept();
        System.out.println("> Client connected: " + clientSocket.getInetAddress());

        DataInputStream dis = new DataInputStream(clientSocket.getInputStream());
        int device = dis.readInt();

        String currID = "Client " + clientIDs.getAndIncrement();

        controlVariables.put(currID, createControlVariable());
        serverGUI.createNewClient(currID, clientSocket.toString());
        new Thread(new ClientHandler(currID, cameras.get(device), clientSocket,
            controlVariables.get(currID), serverGUI)).start();
      }
    } catch (IOException e) {
      System.err.println("> Server closed");
    } catch (InterruptedException e) {
      throw new RuntimeException(e);
    }
  }

  private ArrayList<Boolean> createControlVariable() {
    ArrayList<Boolean> controlInit = new ArrayList<>();
    controlInit.add(true);
```

```
        controlInit.add(false);
        controlInit.add(true);
        return controlInit;
    }

}
```

```
public class ClientHandler implements Runnable {

    private final String id;
    private final Camera camera;
    private final Socket socket;
    private final ArrayList<Boolean> control;
    private final ServerGUI serverGui;

    private BufferedImage frame;
    private DataOutputStream dos = null;

    public ClientHandler(String id, Camera camera, Socket socket, ArrayList<Boolean> control,
ServerGUI serverGUI) {
        this.id = id;
        this.camera = camera;
        this.socket = socket;
        this.control = control;
        this.serverGui = serverGUI;
    }

    @Override
    public void run() {
        double threshold = 50.0;
        boolean isFrameDiff;
        try {
            dos = new DataOutputStream(socket.getOutputStream());
            DataInputStream dis = new DataInputStream(socket.getInputStream());

            Mat previousFrame = new Mat();
            Mat currentFrame = new Mat();

            while (!(!control.get(0) | dis.readBoolean())) {
                if (control.get(1)) {
                    writeDataToClient(false, false);
                    Thread.sleep(100);
                    continue;
                }

                synchronized (camera.getCameraLock()) {
                    camera.getCapture().read(currentFrame);
                }
                if (currentFrame.empty()) {
```

```
                continue;
            }

            Imgproc.cvtColor(currentFrame, currentFrame, Imgproc.COLOR_BGR2GRAY);

            if (previousFrame.empty()) {
                previousFrame = currentFrame.clone();
            }

            if (control.get(2)) {
                double mse = calculateMSE(previousFrame, currentFrame);
                isFrameDiff = mse > threshold;
            } else {
                isFrameDiff = false;
            }

            frame = matToBufferedImage(currentFrame);

            writeDataToClient(isFrameDiff, false);

            previousFrame = currentFrame.clone();
        }
    } catch (IOException e) {
        e.printStackTrace();
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    } finally {
        System.err.println("> Client disconnected: " + socket);

        serverGui.removeClient(id);
        try {
            if (dos != null) {
                writeDataToClient(false, true);
            }
            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

void writeDataToClient(boolean isFrameDiff, boolean shouldClose) throws IOException {
    ByteArrayOutputStream byteArrayOutputStream = new ByteArrayOutputStream();
    ImageIO.write(frame, "jpg", byteArrayOutputStream);
    byte[] imageBytes = byteArrayOutputStream.toByteArray();

    dos.writeBoolean(shouldClose);
    dos.writeBoolean(isFrameDiff);
    dos.writeInt(imageBytes.length);
    dos.write(imageBytes, 0, imageBytes.length);
    dos.flush();
```

```
        byteArrayOutputStream.close();
    }

    private double calculateMSE(Mat frame1, Mat frame2) {
        Mat diff = new Mat();
        Core.absdiff(frame1, frame2, diff);
        Mat squaredDiff = new Mat();
        Core.multiply(diff, diff, squaredDiff);
        Scalar mse = Core.mean(squaredDiff);

        return mse.val[0];
    }

    private BufferedImage matToBufferedImage(Mat mat) {
        int type = BufferedImage.TYPE_BYTE_GRAY;
        if (mat.channels() > 1) {
            type = BufferedImage.TYPE_3BYTE_BGR;
        }
        int bufferSize = mat.channels() * mat.cols() * mat.rows();
        byte[] b = new byte[bufferSize];
        mat.get(0, 0, b);
        BufferedImage image = new BufferedImage(mat.cols(), mat.rows(), type);
        final byte[] targetPixels = ((DataBufferByte) image.getRaster().getDataBuffer()).getData();
        System.arraycopy(b, 0, targetPixels, 0, b.length);
        return image;
    }
}
```

```
public class Client {
    private boolean
            alarmDelayFlag = true,
            movementFlag = false,
            serverCloseClientFlag = false,
            exitButtonFlag = false;

    private DataInputStream dis;
    private DataOutputStream dos;
    private BufferedImage image;

    public Client(String ip_addr, int device) {
        try (Socket socket = new Socket(ip_addr, 2505)) {
            JFrame frame = new JFrame("Camera index " + device);
            JLabel label = new JLabel();

            clientGUI(frame, label);

            dos = new DataOutputStream(socket.getOutputStream());
            dos.writeInt(device);
```

```
            dis = new DataInputStream(socket.getInputStream());

            while (true) {
                dos.writeBoolean(exitButtonFlag);
                readServerData();

                if (serverCloseClientFlag) {
                    frame.dispose();
                    break;
                }

                if (movementFlag & alarmDelayFlag) {
                    new AudioEffect("sounds/sound_smecher.wav");
                    delay();
                }

                label.setIcon(new ImageIcon(image));
                frame.pack();
            }
        } catch (IOException e) {
            System.out.println("Client disconnected");
        } finally {
            try {
                if (dos != null & !serverCloseClientFlag) {
                    dos.writeBoolean(exitButtonFlag);
                    System.exit(0);
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    void readServerData() throws IOException {
        serverCloseClientFlag = dis.readBoolean();
        movementFlag = dis.readBoolean();
        int imageSize = dis.readInt();
        byte[] imageBytes = new byte[imageSize];
        dis.readFully(imageBytes, 0, imageSize);
        image = ImageIO.read(new java.io.ByteArrayInputStream(imageBytes));
    }

    void delay() {
        alarmDelayFlag = false;
        Thread thread = new Thread(() -> {
            try {
                Thread.sleep(10000);
                alarmDelayFlag = true;
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        });
```

```
        thread.start();
    }

    private void clientGUI(JFrame frame, JLabel label) {
        frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        frame.setSize(640, 480);

        frame.add(label);
        frame.setVisible(true);
        frame.setIconImage(new ImageIcon("images/camera.jpg").getImage());

        WindowListener windowListener = new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent e) {
                exitButtonFlag = true;
                frame.dispose();
            }
        };

        frame.addWindowListener(windowListener);
    }
}
```

Clasa ClientHandler:

Este parte a serverului care se ocupă de gestionarea conexiunilor client din aplicație. Această clasă implementează interfața Runnable, ceea ce înseamnă că este concepută să ruleze într-un fir de execuție separat pentru fiecare client care se conectează.

Scopul său principal este de a comunica cu clientul prin intermediul unui socket și de a trimite imagini și informații legate de acestea. În timpul execuției, clasa primește cadre video de la o cameră, le procesează pentru a detecta diferențe semnificative între cadre, și le transmite către client.

De asemenea, poate să trimită un semnal de închidere a conexiunii la client și să gestioneze în mod corespunzător închiderea conexiunii. Această clasă utilizează biblioteca OpenCV pentru manipularea imaginilor și are capacitatea de a detecta diferențe între cadrele video pentru a notifica clientul atunci când există schimbări semnificative.

Totodată, accesarea camerei se face sincronizat cu ceilalți clienți, un singur thread având access la citirea imaginii pentru a prevedea sărirea în excepții de către program.

Clasa Client:

Reprezintă componenta client a aplicației care se conectează la server pentru a primi și afișa fluxuri video de la camere de supraveghere.

Această clasă are rolul de a gestiona interfața grafică a clientului, primirea datelor de la server și tratarea evenimentelor, cum ar fi închiderea ferestrei clientului sau detectarea mișcării.

Aplicația permite utilizatorului să vizualizeze fluxurile video de la diferite camere, să primească alerte în caz de mișcare detectată și să închidă conexiunea la server atunci când este necesar. De asemenea, clasa conține funcționalitate pentru a reda sunete de avertisment în caz de mișcare detectată.

Reprezinta implementarea a unui client pentru monitorizarea camere de supraveghere într-un mediu de rețea.

6. Testare

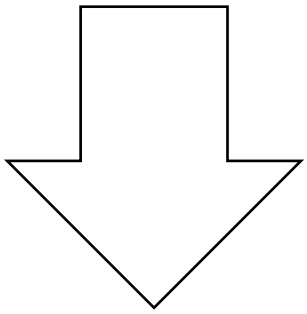
Notă: Testarea a fost realizată și prin conexiune server și legare cu Hamachi pentru a verifică funtionalitatea în parametrii optimi și în cazul în care clientul nu este conectat la aceeași rețea de internet.

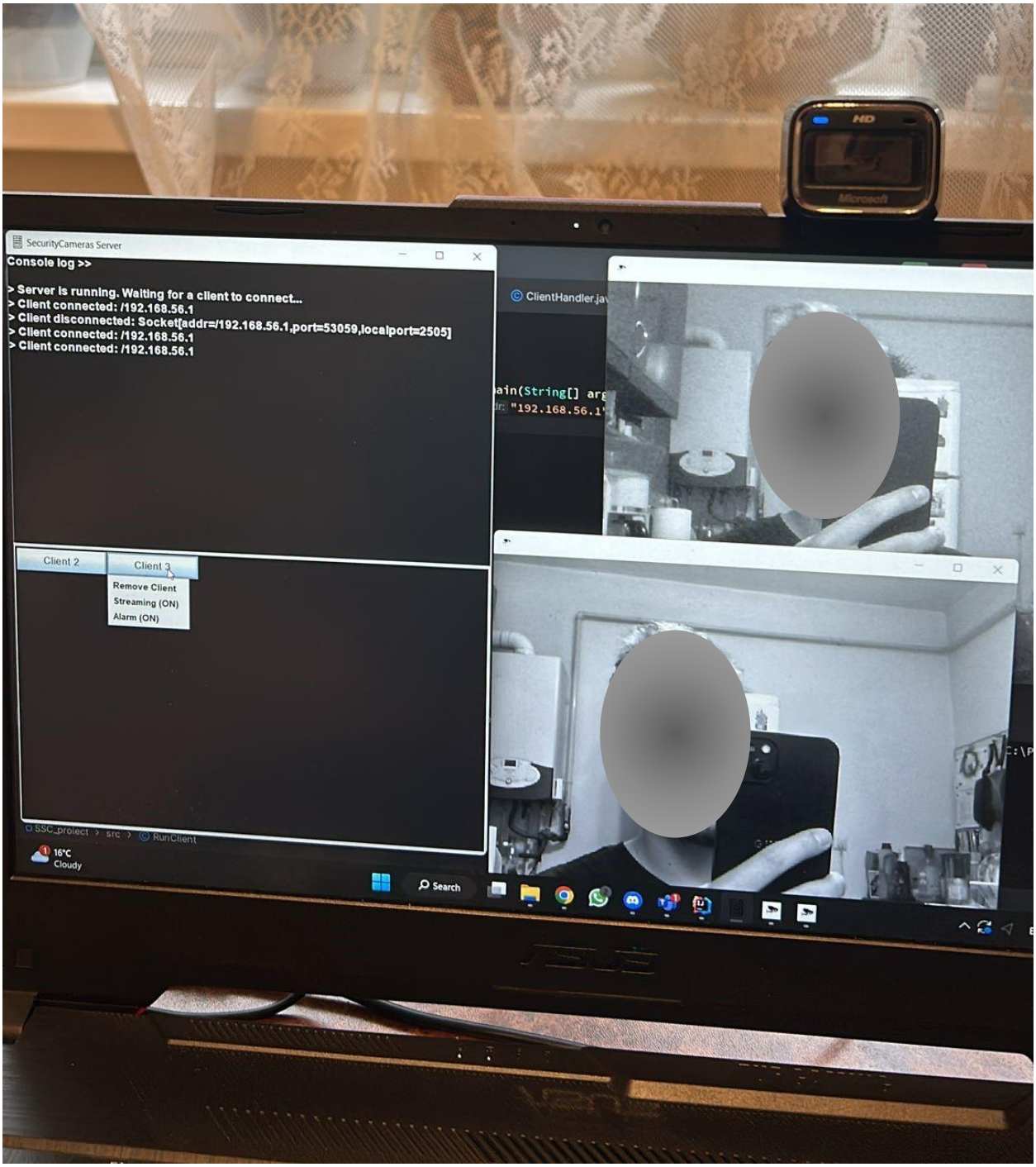
A fost testat pe același dispozitiv cât și serverul și clienții rulați din dispositive diferite pentru a vedea dacă întradevar este folosită conexiunea cu internetul.

Alarmă prezintă o constanța Treshold pentru verificarea cadrelor, când diferența dintre cadre depășește această constanța se pornește sunetul. Ea poate fii reglată după preferințe de sensibilitate și trebuie ajustată după camera folosită și lumina din încăpere.

Mai departe voi lasă o poză pentru evidențierea funcționalității, a se observă că amandoua camerele din poză sunt pornite și rulează fiecare într-un client separate, legate la același server.

Totodată se va vedea în fereastră de control a serverului că există un ConsoleLog specific însoțit de câte un meniu pentru fiecare client conectat pentru a oferii master control la conexiune, streaming și pornirea sau oprirea alarmei.





7.Concluzii

În concluzie, avem o aplicație complet funcțională care funcționează fără eroare pentru supraveghere și Securitate cu sisteme de alarmă pentru cazul evenimentelor neașteptate.

Această conferă imagine continuă alarmă Sonoră și posibilitatea clientului de a alege camera de supraveghere din cele conectate la server.

În urmă realizării proiectului m-am perfecționat în lucrul client-server, familiarizat cu funtiile de interacțiune între dispozitiv și camera web din cadrul librăriei OpenCV și am învățat cum să folosesc și să pun în aplicare sunete în cadrul unei aplicații java.

8.Bibleografie

https://stackoverflow.com/questions/27086120/convert-opencv-mat-object-to-bufferedimage

- Mat to Buffered image transformation

https://docs.opencv.org/4.x/d3/d63/classcv_1_1Mat.html

- Mat class contents and functionality

https://docs.opencv.org/4.x/

- Overall functionality

https://www.digitalocean.com/community/tutorials/java-socket-programming-server-client

- Java socket, serverSocket, client – server working

https://www.geeksforgeeks.org/multithreaded-servers-in-java/

- ClientHandler working and multithreading server

https://www.baeldung.com/java-play-sound

- Play sounds in java