

PODSTAWY TELEINFORMATYKI

SFR – Student Face Recognition

AUTORZY:

Mikołaj Drygas 116216

Patryk Dzwoniarski 121997

Krzysztof Figiel 123652

SPIS TREŚCI

1. Wstęp	3
1.1. Dlaczego wybraliśmy ten temat?	3
1.2. Podobne aplikacje	3
2. Cel i zakres pracy	4
2.1. Zadania SFR – Student Face Recognition	4
2.2. Podział prac	4
2.3. Zadania SFR – Student Face Recognition	4
3. konstruowanie systemu	5
3.1. środki implementacji	5
3.2. Środowisko	5
3.3. Narzędzia	5
3.4. Metodyka	5
3.5. Metody modelowania	5
3.6. Opis technologii i algorytmów wykorzystanych do implementacji projektu	5
4. Wymagania funkcjonalne projektu	10
5. Wymagania POZAFUNKCJONALNe projektu	11
6. Diagramy UML	12
7. Interfejs oraz obsługa aplikacji	16
8. Najważniejsze metody wykorzystane do implementacji systemu sfr	20
9. Plany dalszego rozwoju aplikacji	29
10. Podsumowanie	30

1. WSTĘP

Celem naszego projektu jest stworzenie aplikacji do rozpoznawania twarzy osób wchodzących do pomieszczenia, którym może być np. laboratorium. Aplikacja przyda się do selekcji osób wchodzących na salę przed ważnym egzaminem, bądź zwyczajnie, do rozpoznawania studentów przed wejściem do sali laboratoryjnej i wykrywania osób spoza roku.

1.1. DLACZEGO WYBRALIŚMY TEN TEMAT?

Głównym powodem wybrania tego tematu była chęć zapoznania się z biblioteką *OpenCV* i bibliotekami podobnymi, które są dziś powszechnie używane w wielu przydatnych aplikacjach i systemach informatycznych.

Dodatkowo chcieliśmy podnieść swoje umiejętności w pisaniu aplikacji w technologii *C# WPF*, z którą mieliśmy okazję zapoznać się w trakcie pisania różnych projektów w czasie studiów. Uważamy, że jest ona dobrym rozwiązaniem, głównie ze względu na prostotę tworzenia interfejsu graficznego programu i jego szybkiej edycji.

1.2. PODOBNE APLIKACJE

Aktualnie na rynku istnieje wiele aplikacji służących do rozpoznawania twarzy i korzystających z silników *OpenCV*. Większość mobilnych aplikacji takich jak *Snapchat*, *Instagram*, czy *Facebook Messenger* oferuje dziś bardziej zaawansowane funkcje takie jak zamiana twarzy na inną, czy nakładanie filtrów na twarz w czasie rzeczywisty. Nie spotkaliśmy się jednak z aplikacją, która miałaby rozpoznawać twarze studentów wchodzących do sali. Może się ona okazać przydatnym narzędziem, gdyż wyszukiwanie studentów na liście przy dużej liczbie osób w grupie często bywa problematyczne i czasochłonne.

2. CEL I ZAKRES PRACY

2.1. ZADANIA SFR – STUDENT FACE RECOGNITION

- Wykrywanie twarzy osoby, bądź wielu osób wchodzących do sali, stojących naprzeciwko kamery IP
- Możliwość zapisania rozpoznanej twarzy osoby do lokalnej bazy danych poprzez podanie jej imienia, nazwiska i numeru indeksu
- Wyświetlenie imienia, nazwiska i numeru indeksu osoby w przypadku rozpoznania jej twarzy
- W przypadku nierozpoznania osoby wchodzącej do sali – wyświetlenie alertu ostrzegawczego

2.2. PODZIAŁ PRAC

Aplikacja tworzona będzie w trakcie wspólnych spotkań grupowych, a podział zadań będzie ustalany w momencie tworzenia projektu. Harmonogram prac ukazany jest w tabeli *Tabela 2.2.1.*

2.3. ZADANIA SFR – STUDENT FACE RECOGNITION

- Głównym zadaniem aplikacji jest rozpoznawanie osób na podstawie zdjęć zawartych w lokalnej bazie danych.
- Osoba obsługująca aplikację, chcąc zyskać jej pełną funkcjonalność może dodać przechwycone przez kamerę zdjęcia wraz z opisem zidentyfikowanej twarzy do lokalnej bazy danych.
- SFR ma ułatwić procedury związane z czasochłonnym sprawdzaniem tożsamości studentów, na przykład przy organizowaniu egzaminu, bądź kolokwium.

Tabela 2.2.1. Tabela podziału prac.

Osoba	Zadania
Patryk Dzwoniarski	Funkcjonalność aplikacji i dokumentacja
Mikołaj Drygas	Funkcjonalność aplikacji i dokumentacja
Krzysztof Figiel	Funkcjonalność aplikacji i dokumentacja

3. KONSTRUOWANIE SYSTEMU

3.1. ŚRODKI IMPLEMENTACJI

- *C#, WPF* – technologia ta umożliwia wygodną pracę nad aplikacją z interfejsem graficznym
- *Emgu CV* – jest to *.Net*’owy wrapper do *OpenCV*, służący do przetwarzania i analizy obrazów (statycznych i ruchomych). Biblioteka ta zawiera ogrom funkcji służących do pracy z CPU oraz GPU. Wrapper ten może być kompilowany do takich systemów jak *Linux*, czy *Mac OS X*. Największą zaletą tej biblioteki jest to, że stara się ona jak najlepiej wykorzystać zasoby, które oferuje nam nasza maszyna, a głównie jej karta graficzna.

3.2. ŚRODOWISKO

- *Microsoft Visual Studio 2015 Enterprise* – rozbudowane środowisko stworzone przez firmę *Microsoft* idealne do stworzenia aplikacji typu *WPF*.

3.3. NARZĘDZIA

- *GitHub* – system kontroli wersji, który umożliwi nam dostęp do kodu aplikacji z każdego komputera, na którym akurat pracujemy. Odnotowanie wszystkich zmian z notatką przy zapisywaniu postępów pozwala na łatwe analizowanie historii kodu.
- Konsola *Git*.
- *Visual Paradigm 13.2*.

3.4. METODYKA

- Tworzenie systemu odbywało się na zasadzie cotygodniowych spotkań.
- W trakcie spotkań stawialiśmy sobie określone zadania, które staraliśmy się wykonywać na bieżąco.
- Na każdym zajęciach projektowych słownie oznajmialiśmy kolejną funkcjonalność, którą mieliśmy w planie dodać do projektu, a następnie przez następne dwa tygodnie tworzyliśmy kolejne moduły projektu, ukierunkowując się w danym temacie.
- Wspólnie publikowaliśmy utworzony kod na repozytorium *GitHub*.

3.5. METODY MODELOWANIA

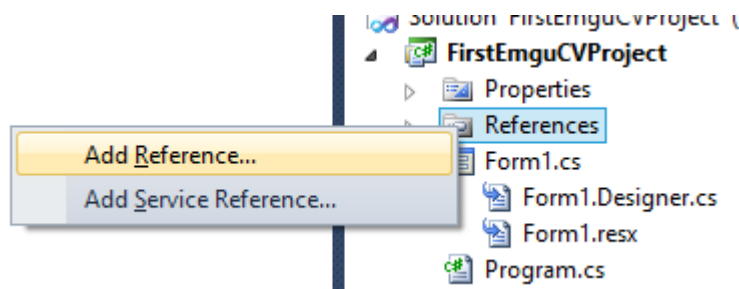
Modelowanie naszego projektu oparliśmy na projektowaniu diagramów UML w środowisku *Visual Paradigm 13.2*. Tworzenie diagramów odbywało się na samym początku etapu implementowania aplikacji, przez co mogliśmy postępować zgodnie z przyjętym schematem.

3.6. OPIS TECHNOLOGII I ALGORYTMÓW WYKORZYSTANYCH DO IMPLEMENTACJI PROJEKTU

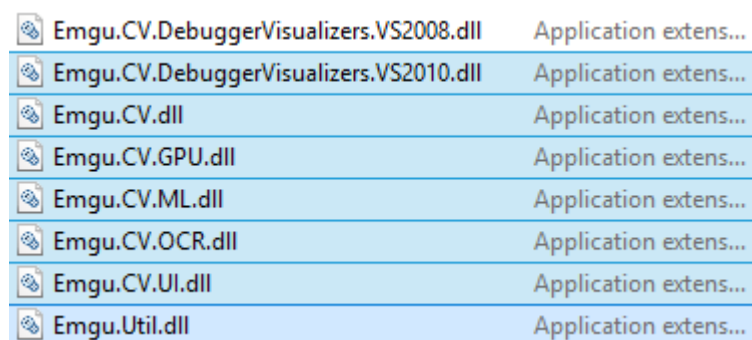
- *Emgu CV 3.0* – wrapper *.Net* do biblioteki przetwarzania obrazu *OpenCV*. Pakiet może być kompilowany z poziomu *Visual Studio*, *Xamarin Studio*, czy *Unity* i może działać na systemach *Windows*, *Linux*, *Mac OS X*, *iOS*, *Android*, czy *Windows Phone*. Zawiera

funkcje wykorzystywane podczas obróbki obrazu, oparte na otwartym kodzie. Zapoczątkowana została przez firmę *Intel*. Autorzy tej biblioteki skupili się szczególnie na przetwarzaniu obrazu w czasie rzeczywistym. Istotną cechą biblioteki *OpenCV*, na której oparty jest *EmguCV* jest fakt, iż stara się ona wykorzystać jak najlepiej zasoby dostępne na maszynie, na której jest ona uruchomiona, ze szczególnym uwzględnieniem karty grafiki. Oprócz tego *OpenCV* wspiera *machine learning*.

Tworzenie projektu z użyciem *EmguCV* wymaga przeprowadzenia kilku istotnych czynności. Po utworzeniu pustego projektu *WPF* należy dodać referencje do bibliotek *EmguCV*.

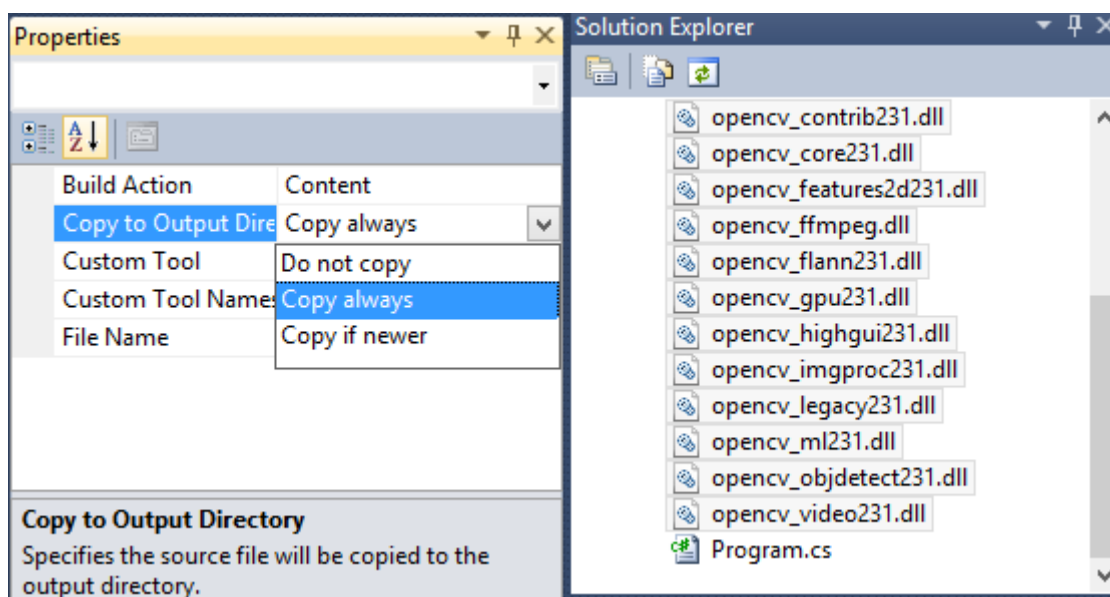


Rys. 3.1. Dodawanie referencji *EmguCV* do projektu.



Rys. 3.2. Pliki *EmguCV*, które należy dodać jako referencje do projektu.

Następnie wszystkie pliki należy dołączyć do folderu *Debug* projektu. Robi się to podobnie jak na rysunku (Rys. 3.3.).



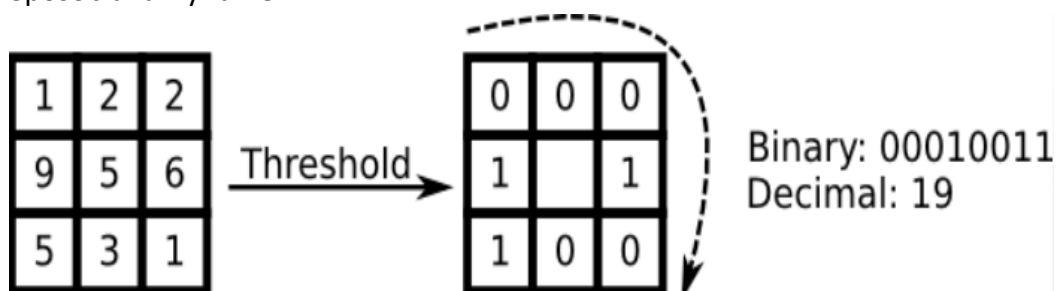
Rys. 3.3. Dodawanie plików EmguCV do folderu Debug.

Teraz dopiero możliwe jest korzystanie z biblioteki *EmguCV*.

- Algorytm *LBPHFaceRecognizer* (*Local Binary Patterns Histogram*) – algorytm ten używa rozszerzonych lokalnych wzorców binarnych i ma następujące wartości domyślne:
 - *radius* = 1
 - *neighbors* = 8
 - *grid_x* = 8
 - *grid_y* = 8

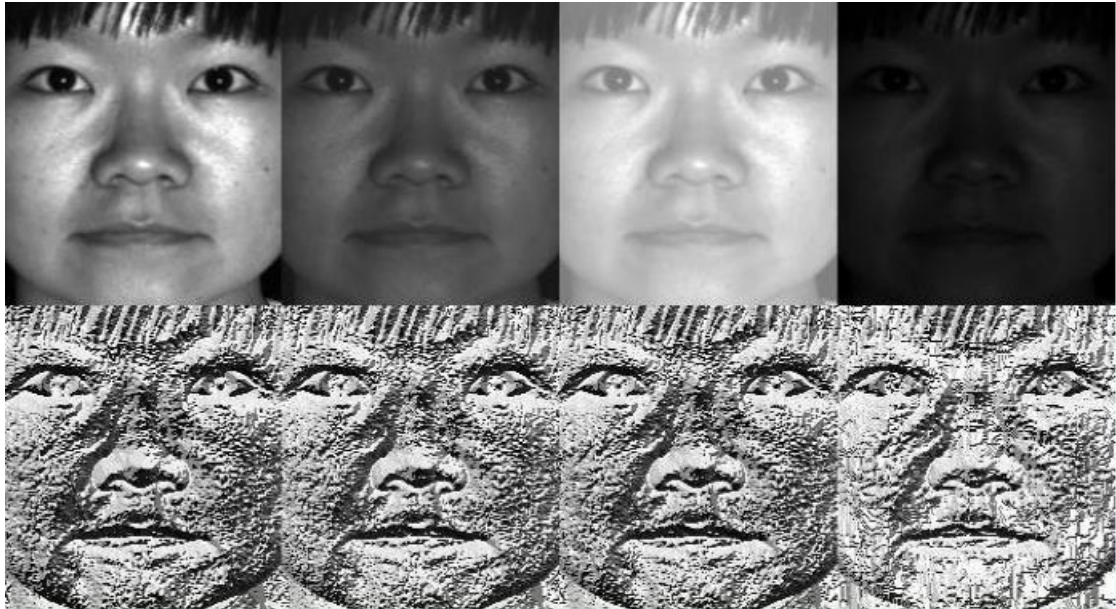
Główną ideą algorytmu jest sumowanie lokalnej struktury zdjęcia poprzez porównywanie każdego piksela z jego sąsiedztwem. Wybierany jest środkowy piksel kwadratowego obszaru (w wersji podstawowej 3x3 pikseli), a jego sąsiedztwo jest poddawane progowaniu. W zależności od tego, czy dany sąsiadujący piksel jest większy od progu, czy nie, przyjmuje wartość 1 lub 0. Następnie odczytuje się liczbę binarną zapisaną dookoła środkowego piksela. Dla ośmiu pikseli sąsiadujących istnieje 256 kombinacji, zwanych Local Binary Patterns.

Sposób analizy ramek:



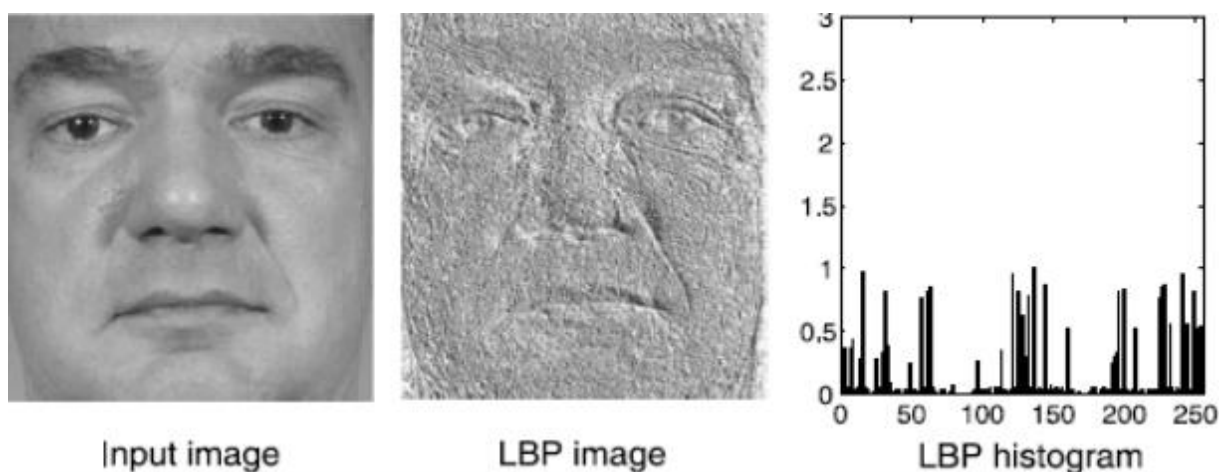
Rys. 3.4. Sposób analizy ramek w algorytmie LBPH.

Przykład transformacji twarzy metodą Local Pattern Binary Histogram w różnych warunkach oświetleniowych:



Rys. 3.5. Przykład transformacji metodą Local Pattern Binary Histogram w różnych warunkach oświetleniowych.

Jako klasyfikator do analizy zdjęć w dziedzinie LBPH, dzieli się obraz na pewną liczbę obszarów i wykorzystuje się porównywanie histogramów. Histogramy zapisane w postaci wektora, o kolejnych elementach będących liczbami pikseli w danym zakresie intensywności, mogą być porównywane w prosty sposób, za pomocą metody Najbliższego Sąsiada.



Rys. 3.6. Przykład histogramu LBPH dla obrazów.

W *LBPH* każdy obraz analizowany jest niezależnie, podczas gdy np. metoda *Eigenfaces* traktuje zestaw danych jako całość. Metoda *LBPH* jest nieco prostsza, w tym sensie, że charakteryzujemy każdy obraz w zbiorze danych lokalnie, a gdy pojawia się nowy, nieznany obraz wykonujemy tą samą analizę i porównujemy wynik z każdym z obrazów w zestawie danych. Sposób w jaki analizujemy obrazy charakteryzuje lokalne wzorce w każdej lokalizacji obrazu. Metoda *LBPH* może działać lepiej od pozostałych metod takich jak *Eigenface*, czy *Fisherface* w różnych środowiskach i przy różnych warunkach świetlnych, ale zależy to od zestawu danych szkoleniowych i testowych. Z reguły potrzebne jest około 10 różnych wizerunków danej osoby, aby móc ją rozpoznać.

```
FaceRecognizer _faceRecognizer = new LBPHFaceRecognizer(1, 8, 8, 8, 100);
```

Rys. 3.7. Przykład inicjalizacji instancji obiektu dla algorytmu *LBPH*.

4. WYMAGANIA FUNKCJONALNE PROJEKTU

- Projekt będzie zapewniał możliwość monitorowania pracy systemu i administrowania zasobami systemu z jednego, centralnego miejsca.
- System posiadać będzie graficzny interfejs użytkownika.
- Prowadzący zajęcia w sali laboratoryjnej będzie miał możliwość zweryfikowania tożsamości studenta poprzez rozpoznanie go na podstawie zestawienia zdjęcia twarzy studenta z obrazem pochodzącym z kamery internetowej.
- Użytkownik będzie miał możliwość dodania zdjęć twarzy nowych osób do lokalnej bazy danych w celu ich późniejszej weryfikacji.
- W celu rozpoczęcia przechwytywania obrazu z kamery internetowej i rozpoznawania twarzy należy kliknąć przycisk *Start Capturing*, natomiast w celu ich zakończenia przycisk *Stop Capturing*.
- Użytkownik będzie mógł przechwycić obraz z kamery oraz zapisać go na dysku w wybranej lokalizacji.
- Użytkownik będzie miał możliwość zmiany parametrów wyświetlanego obrazu pochodzącego z kamery, takich jak: jasność, kontrast i ostrość.
- Użytkownik będzie miał możliwość zresetowania ustawionych przez siebie preferencji wyświetlanego obrazu.

5. WYMAGANIA POZAFUNKCJONALNE PROJEKTU

- Program automatycznie rozpoznawać będzie twarze osób, które zbliżą się do kamery internetowej.
- Rozpoznana twarz zostanie automatycznie zweryfikowana na podstawie zdjęć zawartych w folderze aplikacji.
- Twarz nierozpoznana przez program zostanie zaznaczona czerwonym prostokątem. Dodatkowo pojawi się stosowny komunikat o braku osoby w bazie danych.
- Twarz rozpoznana przez program zostanie zaznaczona zielonym prostokątem. Dodatkowo, w odpowiednim polu pojawią się dane zweryfikowanej osoby.
- Istnieje możliwość wykrycia twarzy kilku osób naraz i zweryfikowania ich tożsamości.
- Program automatycznie zliczać będzie liczbę wykrytych twarzy i wyświetlać ją w oknie aplikacji.
- Program automatycznie rozpoznaje nazwę uruchomionej kamery.

6. DIAGRAMY UML

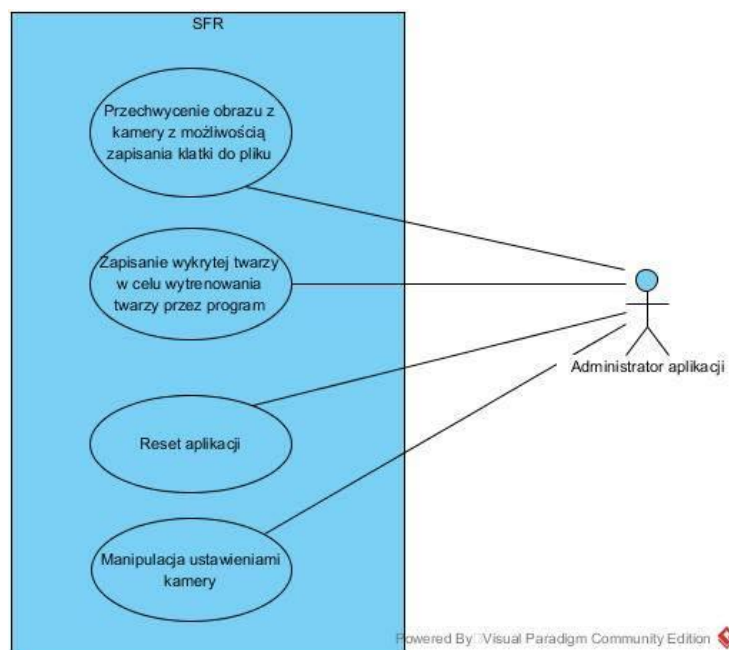
W poniższym rozdziale przedstawione zostaną diagramy UML wraz z aktorami systemu *SFR*.

Aktorzy systemu:

Tabela 6.1. Tabela aktorów aplikacji.

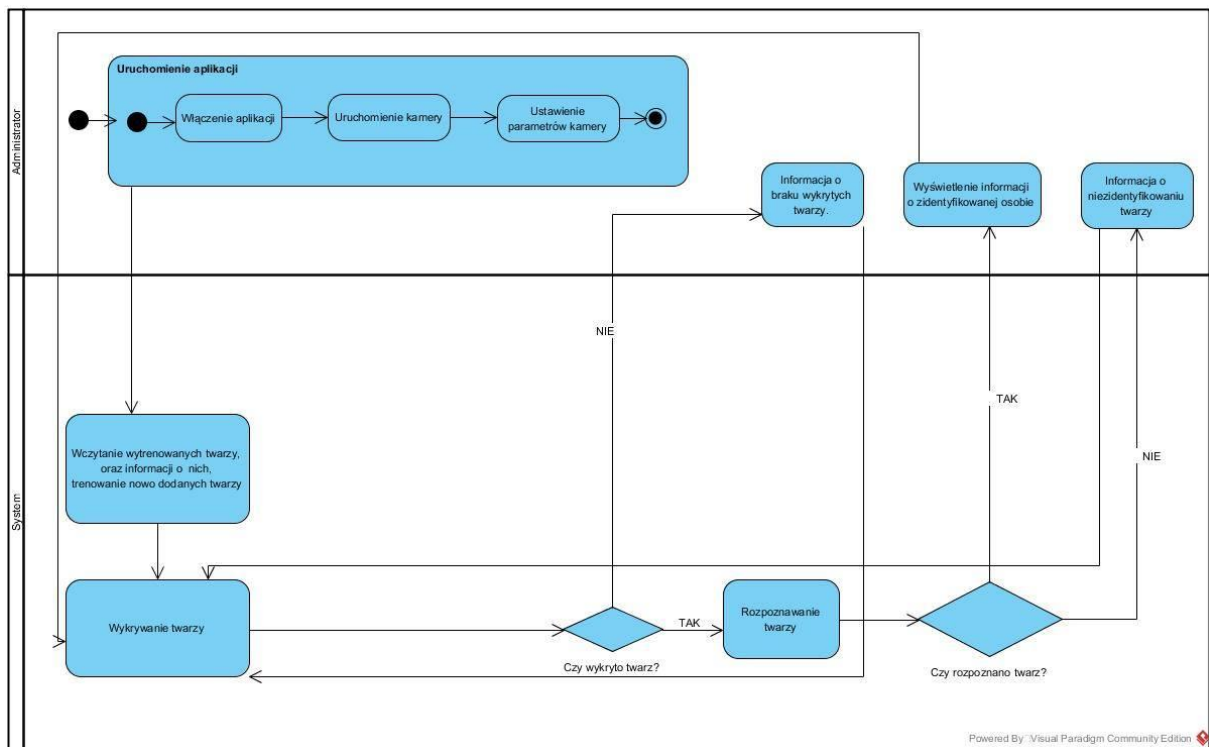
Nazwa aktora	Funkcja
Administrator	<ul style="list-style-type: none"> • Dodawanie osób do lokalnej bazy danych aplikacji. • Uruchomienie funkcji przechwytywania obrazu z kamery i zatrzymywanie jej. • Manipulacja ustawieniami wyświetlanego programu. • Inicjowanie procesu rozpoznawania twarzy studentów.

- Diagram przypadków użycia:



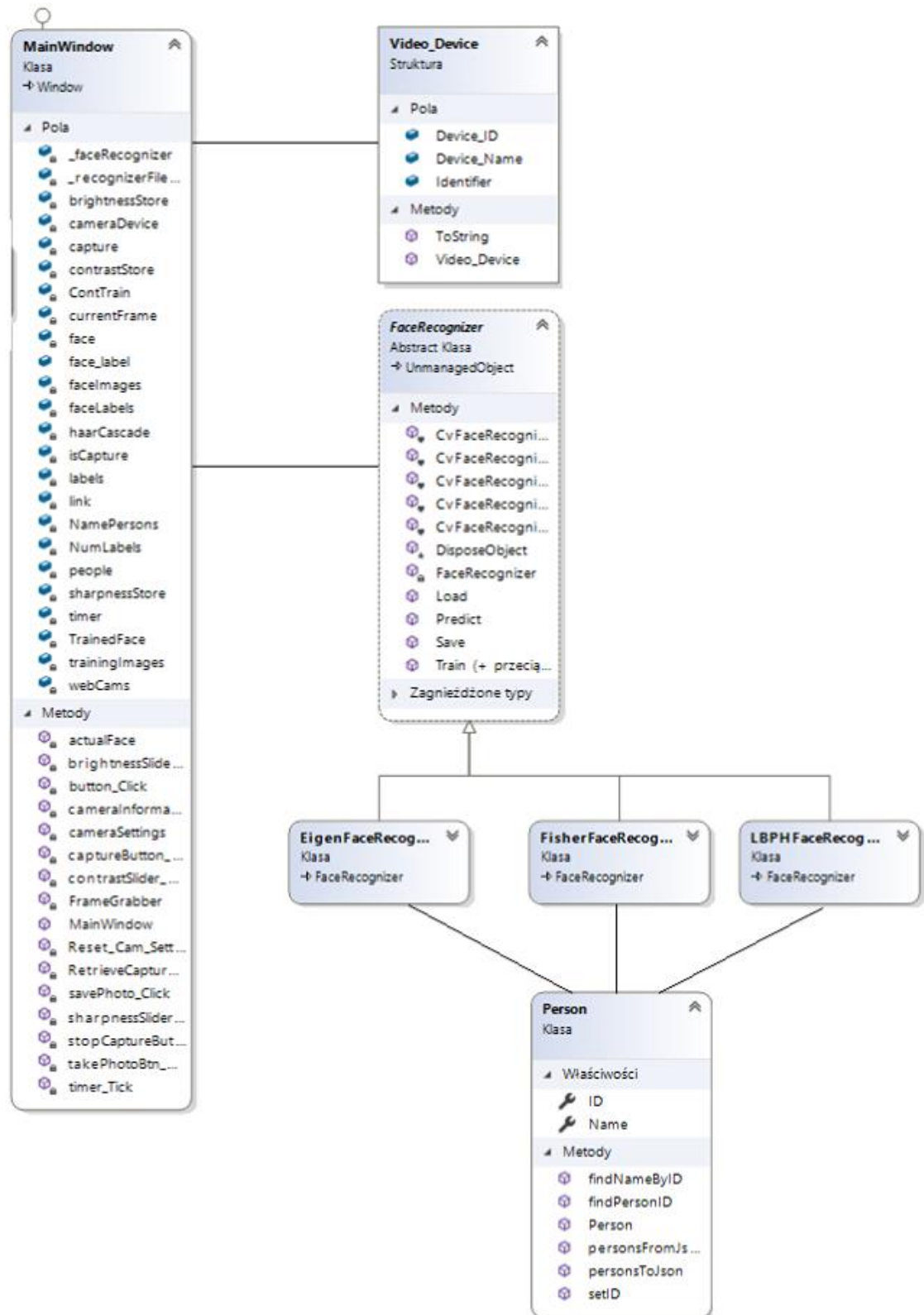
Rys. 6.1. Diagram przypadków użycia dla aplikacji SFR.

- Diagram aktywności:



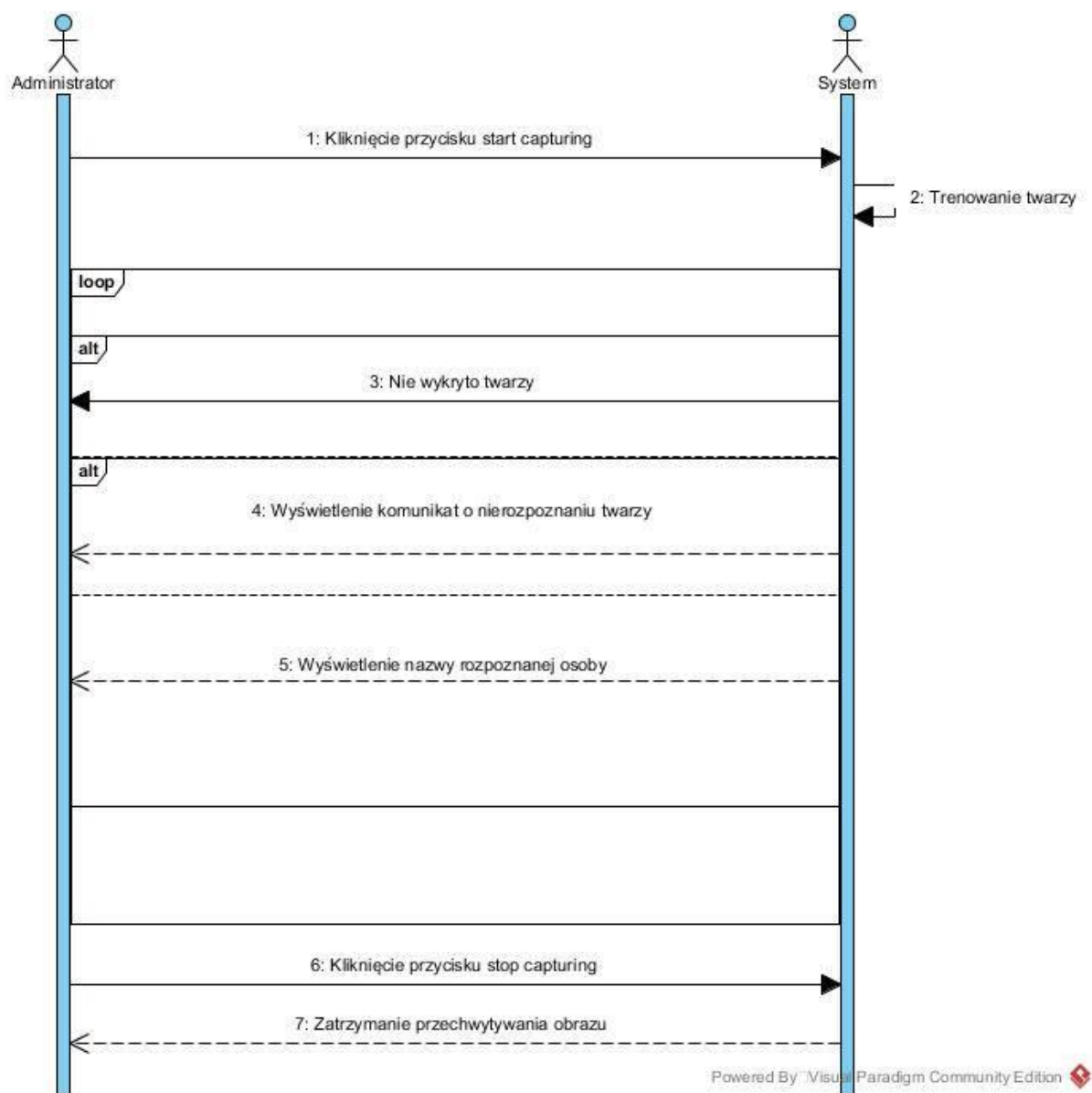
Rys. 6.2. Diagram aktywności dla aplikacji SFR.

- Diagram klas:



Rys. 6.3. Diagram klas dla aplikacji SFR.

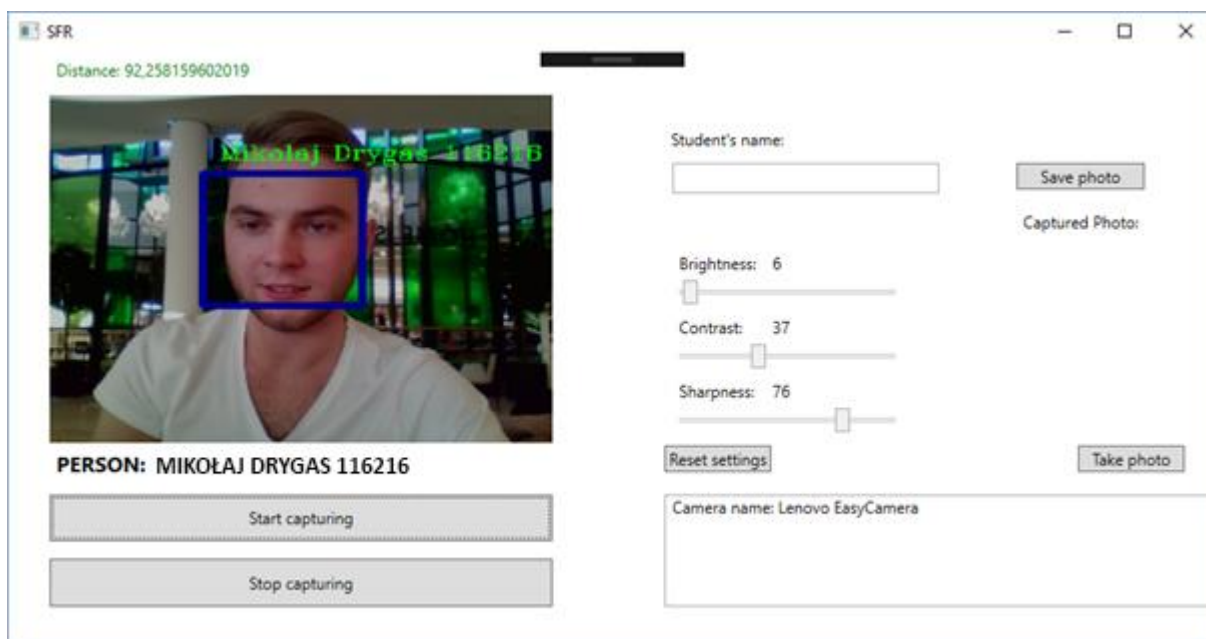
- Diagram sekwencji



Rys. 6.4. Diagram sekwencji dla aplikacji SFR.

7. INTERFEJS ORAZ OBSŁUGA APLIKACJI

W poniższym rozdziale przedstawiony zostanie interfejs aplikacji programu *SFR*.

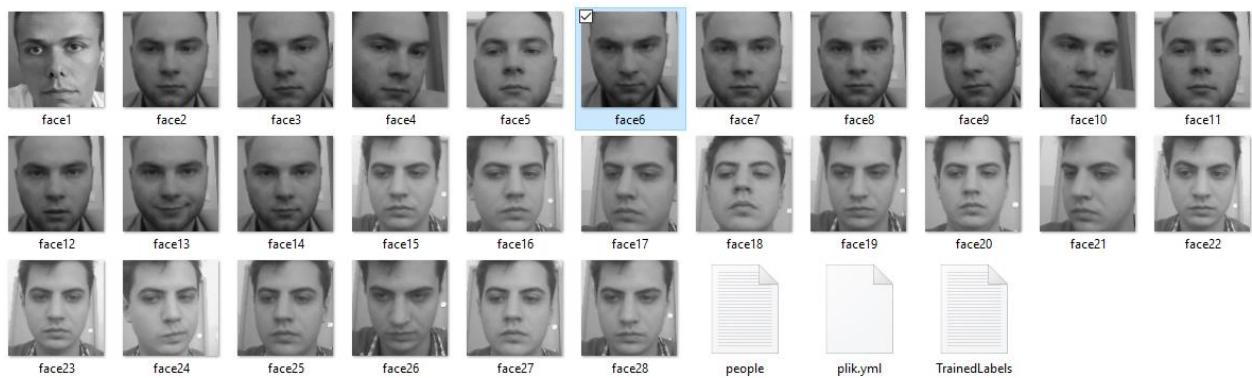


Rys. 7.1. Interfejs aplikacji SFR.

- **Start capturing**
 - Po kliknięciu tego przycisku zostaje uruchomiona kamera,
 - Następuje trenowanie twarzy ,
 - Wykrywanie twarzy z kamery,
 - Porównywanie twarzy z wytrenowanymi twarzami,
 - Wyświetlenie jednej z trzech informacji:
 - Nie wykryto twarzy
 - Nie rozpoznano twarzy
 - Informacje o rozpoznanej osobie
- **Stop capturing**
 - Wyłączenie kamery
 - Wyłączenie wykrywania twarzy
- **Save photo**
 - Trzeba uzupełnić textbox „Student’s Name”|
 - Gdy wykryto twarz, można zrobić zdjęcie aby zapisać je do bazy
- **Reset Settings:**
 - Po kliknięciu tego przycisku następuje reset ustawień kamery do domyślnych
- **Captured Photo**
 - Poniżej tego textboxa wyświetli się przechwycone zdjęcie, które zostało zapisane do bazy

- **Take photo**
 - Po kliknięciu tego przycisku następuje zrobienie zdjęcia z kamery
- **Brightness**
 - Regulacja jasności
- **Contrast**
 - Regulacja kontrastu
- **Sharpnes**
 - Regulacja ostrości
- Ponizej przycisku *Reset Settings*, zostają wyświetlone informacje odnośnie kamery z której aktualnie korzysta program

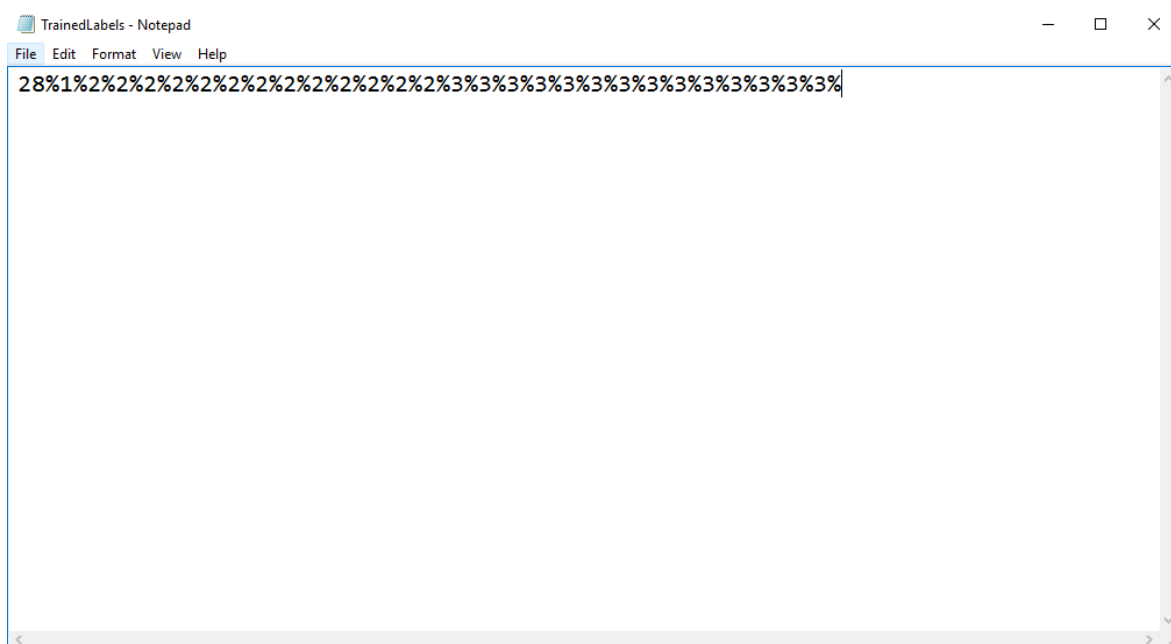
Folder z zapisanymi twarzami



Rys. 7.2. Folder Trained Faces

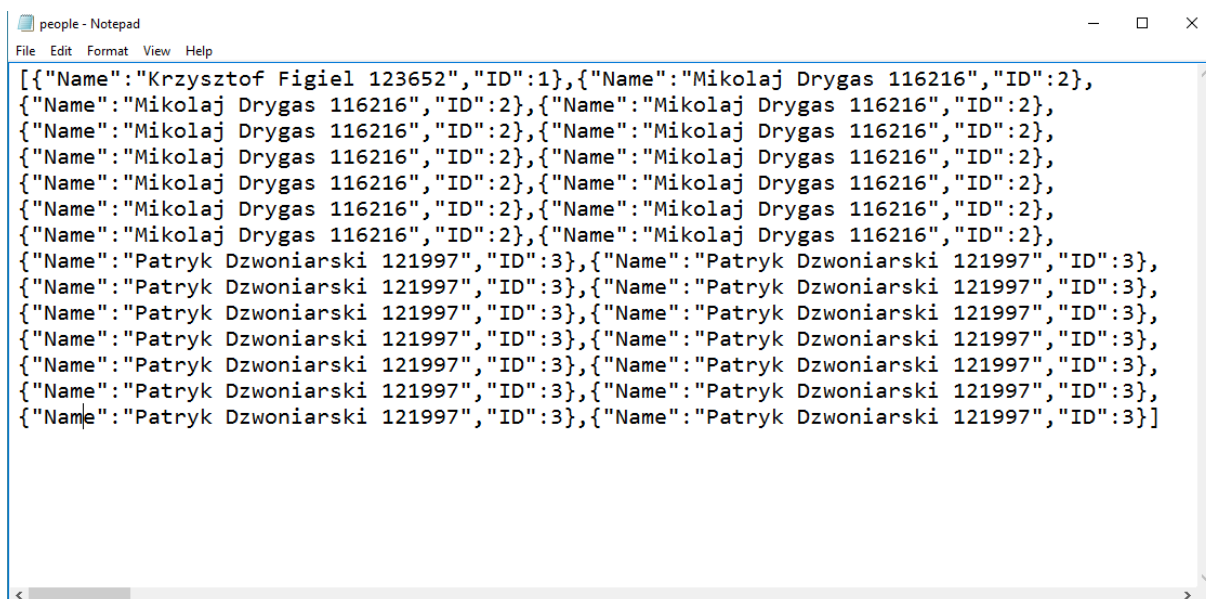
Twarze są przechowywane w folderze „Trained Faces” w którym dodatkowo znajdują się trzy pliki:

- **Trained Labels** - jest to plik w którym pierwsza liczba oznacza ilość twarzy w folderze, natomiast kolejne liczby oznaczają ID każdej twarzy. Każda liczba oddzielona jest znakiem „%”




Rys. 7.3. Plik *TraineLabels.txt*

- people.txt – jest to plik, w którym przechowywane są w postaci JSON wszystkie nazwy przechowywanych twarzy, wraz z ich odpowiednimi identyfikatorami (ID).



Rys. 7.4. plik people.txt

- Plik.yml – jest to plik w którym znajdują się odległości wytrenowanych twarzy.



```

1  #YAML:1.0
2  radius: 1
3  neighbors: 8
4  grid_x: 8
5  grid_y: 8
6  histograms:
7    - !!opencv-matrix
8      rows: 1
9      cols: 16384
10     dt: f
11     data: [ 3.4722239e-002, 6.94444450e-003, 0., 0., 2.08333340e-002,
12             0., 0., 6.94444450e-003, 0., 0., 0., 0., 0., 0., 0.,
13             6.94444450e-003, 1.38888890e-002, 0., 0., 0., 0., 0., 0.,
14             6.94444450e-003, 0., 0., 0., 0., 6.94444450e-003, 0., 0.,
15             6.94444450e-003, 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
16             0., 0., 0., 0., 0., 6.94444450e-003, 0., 0., 0., 0., 0.,
17             0., 2.08333340e-002, 0., 0., 0., 5.55555560e-002, 0., 0., 0.,
18             1.38888890e-002, 0., 0., 0., 2.77777780e-002, 0., 0., 0.,
19             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
20             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 6.94444450e-003,
21             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 2.77777780e-002,
22             0., 0., 0., 0., 0., 0., 3.4722239e-002, 6.94444450e-003,
23             0., 0., 7.63888881e-002, 0., 0., 1.38888890e-002, 0.,
24             6.94444450e-003, 0., 4.16666679e-002, 0., 0., 0.,
25             6.94444450e-003, 0., 0., 0., 0., 0., 6.94444450e-003, 0.,
26             6.94444450e-003, 0., 0., 0., 0., 6.94444450e-003, 0., 0., 0.,
27             0., 0., 0., 0., 0., 0., 0., 0., 6.94444450e-003, 0., 0., 0.,
28             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
29             0., 0., 0., 0., 6.94444450e-003, 0., 0., 0., 0., 0.,
30             6.94444450e-003, 0., 2.77777780e-002, 0., 1.25000000e-001, 0.,
31             0., 0., 4.16666679e-002, 0., 0., 0., 0., 0., 0., 0., 0.,
32             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
33             6.94444450e-003, 0., 0., 6.94444450e-003, 0., 2.77777780e-002,
34             0., 0., 0., 6.94444450e-003, 0., 0., 0., 0., 0., 0.,
35             6.94444450e-003, 2.08333340e-002, 0., 2.08333340e-002, 0., 0.,
36             0., 6.94444450e-003, 6.94444450e-003, 6.94444450e-003, 0.,
37             2.77777780e-002, 1.38888890e-002, 3.4722239e-002, 0.,
38             1.11111112e-001, 1.38888890e-002, 1.38888890e-002, 0.,

```

YAML Ain't Markup Language length: 3,384,431 lines: 48,264 Ln: 1 Col: 1 Sel: 0 | 0

Rys. 7.5. plik.yml początek

```

plik.yml
48228 6.94444450e-003, 6.94444450e-003, 0., 0., 0., 0., 0., 0., 0., 0.,
48229 0., 0., 0., 0., 0., 0., 0., 0., 6.94444450e-003,
48230 6.94444450e-003, 0., 0., 6.94444450e-003, 0., 0., 0.,
48231 6.94444450e-003, 0., 6.94444450e-003, 0., 3.4722239e-002,
48232 6.94444450e-003, 1.38888890e-002, 1.38888890e-002,
48233 3.4722239e-002, 1.38888890e-002, 0., 0., 0., 0., 0., 0.,
48234 6.94444450e-003, 0., 0., 0., 0., 0., 0., 6.94444450e-003,
48235 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
48236 1.38888890e-002, 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
48237 0., 0., 0., 0., 6.94444450e-003, 0., 0., 0., 0., 0., 0.,
48238 2.77777780e-002, 0., 0., 0., 2.77777780e-002, 0.,
48239 1.38888890e-002, 2.08333340e-002, 6.94444450e-003,
48240 1.38888890e-002, 0., 6.94444450e-003, 0., 0., 6.94444450e-003,
48241 3.4722239e-002, 0., 0., 0., 0., 0., 6.94444450e-003, 0.,
48242 2.08333340e-002, 0., 0., 0., 0., 0., 0., 6.94444450e-003,
48243 0., 0., 0., 0., 0., 0., 2.08333340e-002, 0., 0., 0., 0.,
48244 0., 0., 0., 6.94444450e-003, 0., 0., 0., 0., 0., 0., 0.,
48245 0., 0., 0., 6.94444450e-003, 0., 0., 0., 6.94444450e-003, 0.,
48246 0., 0., 6.94444450e-003, 0., 0., 0., 4.16666679e-002, 0.,
48247 6.94444450e-003, 0., 2.08333340e-002, 1.38888890e-002,
48248 6.94444450e-003, 0., 3.4722239e-002, 0., 0., 0., 0.,
48249 6.94444450e-003, 0., 0., 0., 6.94444450e-003, 0.,
48250 6.94444450e-003, 0., 0., 0., 6.94444450e-003, 0., 0., 0.,
48251 6.94444450e-003, 0., 0., 0., 0., 0., 1.38888890e-002, 0.,
48252 0., 6.94444450e-003, 6.94444450e-003, 0., 0., 0., 0., 0.,
48253 0., 6.94444450e-003, 1.38888890e-002, 0., 0., 0., 0., 0.,
48254 0., 0., 6.94444450e-003, 0., 1.38888890e-002, 0., 0., 0.,
48255 1.11111112e-001 ]
48256 labels: !!opencv-matrix
48257   rows: 28
48258   cols: 1
48259   dt: i
48260   data: [ 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3,
48261           3, 3, 3, 3, 3, 3, 3 ]
48262 labelsInfo:
48263   []
48264
YAML Ain't Markup Language          length: 3,384,431   lines: 48,264   Ln: 1   Col: 1   Sel: 0|0

```

Rys. 7.6. plik.yml koniec

8. NAJWAŻNIEJSZE METODY WYKORZYSTANE DO IMPLEMENTACJI SYSTEMU SFR

- Inicjalizacja głównego okna:

```

public MainWindow()
{
    InitializeComponent();
    timer = new DispatcherTimer();

    timer.Interval = TimeSpan.FromMilliseconds(1);

    people = new List<Person>();
    face = new CascadeClassifier("haarcascade_frontalface_default.xml");
    try
    {
        people =
        Person.personsFromJson(File.ReadAllText("TrainedFaces/people.txt"));

        string Labelsinfo = File.ReadAllText(link +
        "/TrainedFaces/TrainedLabels.txt");
        string[] Labels = Labelsinfo.Split('%');
        NumLabels = Convert.ToInt16(Labels[0]);
        ContTrain = NumLabels;
        string LoadFaces;
        faceImages = new Image<Gray, byte>[NumLabels];
        faceLabels = new int[NumLabels];
        for (int tf = 1; tf < NumLabels + 1; tf++)
        {
            LoadFaces = "face" + tf + ".bmp";

```

```

        trainingImages.Add(new Image<Gray, byte>(link + "/TrainedFaces/" +
LoadFaces));
        labels.Add(Labels[tf]);
    }
}
catch (Exception e)
{
    MessageBox.Show("Exception " + e, "Trained faces load",
MessageBoxButton.OK, MessageBoxImage.Exclamation);
}
}

```

Jest to konstruktor głównego okna, zostaje on wywołany podczas uruchomienia programu. W nim powstaje inicjalizacja wszystkich komponentów. Następuje też wczytanie wszystkich zapisanych twarzy z pliku do tablicy.

Następnie zostaje wczytany format tekstowy w którym znajdują się wszystkie imiona wraz z ID osób których twarze są zapisane. Obiekty w tym pliku są zakodowane w formacie JSON, w tej metodzie następuje wczytanie tego pliku i zamiana go na listę obiektów typu „Person”

- Implementacja przycisku *Capture*:

```

private void captureButton_Click(object sender, RoutedEventArgs e)
{
    timer.Start();
    isCapture = true;
    try
    {
        capture = new Capture();
    }
    catch (NullReferenceException exception)
    {
        MessageBox.Show(exception.Message);
    }
    timer.Tick += new EventHandler(timer_Tick);
    timer.Tick += new EventHandler(FrameGrabber);
    haarCascade = new CascadeClassifier(link +
"/haarcascade_frontalface_default.xml");

    cameraInformation();
    capture.FlipHorizontal = !capture.FlipHorizontal;
    for (int i = 0; i < NumLabels; i++)
    {
        faceImages[i] = trainingImages[i];
        faceLabels[i] = Convert.ToInt32(labels[i]);
    }
    _faceRecognizer.Train(faceImages, faceLabels);
    _faceRecognizer.Save(_recognizerFilePath);
}

```

Po kliknięciu przycisku start capturing, następuje wywołanie powyższego kodu. W nim zostaje uruchomiony timer oraz zostaje uruchomiona kamera. Następnie dzięki timerowi zostaje wywoływane zdarzenie Frame Grabber, które będzie opisane pod deklaracją tej metody.

Następnie zostaje wywołana metoda „*cameraInformation()*” która zostanie opisana pod deklaracją tej metody oraz następuje obrót kamery na orientację poziomą

Na samym końcu następuje jedna z ważniejszych rzeczy dla działania tego programu, mianowicie trenowanie twarzy, które odbywa się za pomocą metody „*Train()*”, która jest dostępna dzięki bibliotece EMGU CV 3.0. Po wytrenowaniu twarzy następuje zapisanie ich do pliku.

- Wyświetlenie informacji o kamerze:

```
private void cameraInformation()
{
    DsDevice[] systemCameras =
DsDevice.GetDevicesOfCat(FilterCategory.VideoInputDevice);
    webCams = new Video_Device[systemCameras.Length];
    for (int i = 0; i < systemCameras.Length; i++)
    {
        webCams[i] = new Video_Device(i, systemCameras[i].Name,
systemCameras[i].ClassID);
    }
    richTextBox.AppendText("Camera name: " + webCams[cameraDevice].Device_Name
+ "\n\n");
}
```

Zadaniem tej procedury jest wyświetlenie informacji o kamerze w oknie TextBox, która jest używana przez program, w celu przechwytywania obrazu oraz rozpoznawania twarzy. Procedura jest uruchamiana po kliknięciu przycisku, który uruchamia kamerkę.

- Ustawianie wartości ustawień kamery:

```
private void cameraSettings()
{
    brightnessLabel.Content = ((int)brightnessSlider.Value).ToString();
    contrastLabel.Content = ((int)contrastSlider.Value).ToString();
    sharpnessLabel.Content = ((int)sharpnessSlider.Value).ToString();

    brightnessSlider.Value =
(int)capture.GetCaptureProperty(Emgu.CV.CvEnum.CapProp.Brightness);
    contrastSlider.Value =
(int)capture.GetCaptureProperty(Emgu.CV.CvEnum.CapProp.Contrast);
    sharpnessSlider.Value =
(int)capture.GetCaptureProperty(Emgu.CV.CvEnum.CapProp.Sharpness);
}
```

Procedura ta odpowiada za inicjalizację wartości ustawień kamery takich jak jasność, kontrast czy ostrość. Oczywiście wartości te administrator może dowolnie zmieniać, co jest przydatne w słabszych warunkach oświetleniowych

- Funkcja zapisu zdjęcia przechwyconej twarzy

```
private void savePhoto_Click(object sender, RoutedEventArgs e)
{
    try
    {
        //Trained face counter
        ContTrain = ContTrain + 1;

        //Get a gray frame from capture device

        UMat grayFrame = new UMat();
        currentFrame = capture.QueryFrame().ToImage<Bgr, Byte>();
        CvInvoke.CvtColor(currentFrame, grayFrame, ColorConversion.Bgr2Gray);
        imageBox.Source =
        Emgu.CV.WPF.BitmapSourceConvert.ToBitmapSource(grayFrame);

        //Face Detector
        System.Drawing.Rectangle[] facesDetected = face.DetectMultiScale(
            grayFrame,
            1.2,
            10,
            new System.Drawing.Size(20, 20));

        //Action for each element detected
        foreach (System.Drawing.Rectangle f in facesDetected)
        {
            TrainedFace = currentFrame.Copy(f).Convert<Gray, byte>();
            break;
        }
        imageBox.Source =
        Emgu.CV.WPF.BitmapSourceConvert.ToBitmapSource(TrainedFace);
        //resize face detected image for force to compare the same size with
        the
        //test image with cubic interpolation type method
        TrainedFace = TrainedFace.Resize(100, 100,
        Emgu.CV.CvEnum.Inter.Cubic);

        trainingImages.Add(TrainedFace);

        int _id = Person.findPersonID(people, nameTextBox.Text);
        if (_id != -1)
        {
            people.Add(new Person(nameTextBox.Text, _id));
            labels.Add(_id.ToString());
        }
        else
        {
            int __id = Person.setID(people);
            people.Add(new Person(nameTextBox.Text, __id));
            labels.Add(__id.ToString());
        }

        File.WriteAllText("TrainedFaces/people.txt",
        Person.personsToJson(people));
        //Show face added in gray scale
    }
}
```

```

string exePath = Environment.GetCommandLineArgs()[0];
string startupPath = System.IO.Path.GetDirectoryName(exePath);
//Write the number of triained faces in a file text for further load
File.WriteAllText(startupPath + "/TrainedFaces/TrainedLabels.txt",
trainingImages.ToArray().Length.ToString() + "%");

```

```

//Write the labels of triained faces in a file text for further load
for (int i = 1; i < trainingImages.ToArray().Length + 1; i++)
{
    trainingImages.ToArray()[i - 1].Save(startupPath +
"/TrainedFaces/face" + i + ".bmp");
    File.AppendAllText(startupPath +
"/TrainedFaces/TrainedLabels.txt", labels.ToArray()[i - 1] + "%");
}
MessageBox.Show(nameTextBox.Text + "'s face detected and added!",
"Training OK", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
catch (Exception a)
{
    MessageBox.Show(a.ToString());
    MessageBox.Show("Enable the face detection first", "Training Fail",
MessageBoxButtons.OK, MessageBoxIcon.Exclamation);
}
}

```

Metoda ta wywoływana jest po kliknięciu przycisku zapisującego twarz. Dana klatka zostaje przechwycona, jej kolor zostaje zamieniony na odcienie szarości oraz następuje skalowanie obrazu. Ponadto wraz ze zdjęciem zapisane zostają również dane o osobie która jest na zdjęciu, i które są następnie wyświetlane, jeśli zostanie rozpoznana dana twarz. Próba kliknięcia przycisku gdy nie jest włączona kamera, oczywiście skutkuje wyświetleniem stosownego komunikatu. Gdy zdjęcie zostało przechwycone poprawnie, po kliknięciu przycisku wyświetli się komunikat z informacją o poprawnym dodaniu zdjęcia.

- *Frame Grabber* dla aplikacji

```

void FrameGrabber(object sender, EventArgs e)
{
    string name;
    if (actualFace() != null)
    {
        var result = _faceRecognizer.Predict(actualFace());
        name = Person.findNameByID(people, result.Label);
        face_label = name;
        LabelName.Content = "PERSON: " + name;
        labelDistance.Foreground = new SolidColorBrush(Colors.Green);
        labelDistance.Content = "Distance: " + result.Distance;
    }
    else
    {
        face_label = "Person undetected";
        LabelName.Content = "Person undetected";
    }
}

```


Jest to procedura, która działa w tle programu i jest uruchamiana co 1ms przy użyciu timera. Jej zadaniem za pośrednictwem funkcji *actualFace()* jest wyświetlanie przechwyconego z kamery obrazu w czasie rzeczywistym wraz z zaznaczeniem wykrytej twarzy oraz podpisem, jeśli twarz została rozpoznana.

- Porównywanie wykrytej twarzy ze zdjęciami zawartymi w bazie danych:

```
private Image<Gray, byte> actualFace()
{
    currentFrame = capture.QueryFrame().ToImage<Bgr, Byte>().Resize(300,250,
    Emgu.CV.CvEnum.Inter.Cubic);
    Image<Gray, Byte> grayFrame = currentFrame.Convert<Gray, byte>();
    //konwertowanie do klatki w odcieniach szarości
    //var faces = haarCascade.DetectMultiScale(grayFrame, 1.1, 10,
    System.Drawing.Size.Empty); //aktualna detekcja twarzy

    //Face Detector
    System.Drawing.Rectangle[] facesDetected = face.DetectMultiScale(
    grayFrame,
    1.2,
    10,
    new System.Drawing.Size(20, 20));

    //TrainedFace = null;
    //Action for each element detected
    foreach (System.Drawing.Rectangle f in facesDetected)
    {
        TrainedFace = currentFrame.Copy(f).Convert<Gray, byte>();
        currentFrame.Draw(f, new Bgr(System.Drawing.Color.DarkBlue), 3);
    //podświetlenie twarzy za pomocą box'a rysowanego dookoła niej
        CvInvoke.PutText(currentFrame, face_label, new
        System.Drawing.Point(f.Location.X + 10, f.Location.Y - 10),
        Emgu.CV.CvEnum.FontFace.HersheyComplex, 0.5, new Bgr(0, 255, 0).MCvScalar);
        break;
    }

    image.Source =
    Emgu.CV.WPF.BitmapSourceConvert.ToBitmapSource(currentFrame); //przekazanie obrazu na
    komponent Image
    System.Drawing.Rectangle[] facesTab =
    haarCascade.DetectMultiScale(grayFrame, 1.1, 10, System.Drawing.Size.Empty); //tablica
    z wykrytymi twarzami

    if (TrainedFace != null)
        return TrainedFace.Resize(100, 100, Emgu.CV.CvEnum.Inter.Cubic);
    else
        return null;
}
```

Metoda ta wywoływana jest cały czas podczas gdy kamera jest włączona. Dana klatka zostaje przechwycona, jej kolor zostaje zamieniony na odcienie szarości oraz następuje skalowanie obrazu.

Następnie zostaje wykonywane wykrywanie twarzy i wyświetlenie wokół niej kwadratu w kolorze zielonym. Kolejnym krokiem jest przekazanie obrazu na komponent Image oraz zwrócenie obrazu przeskalowanego do rozmiaru 100x100.

- Klasa *Person*

```
public class Person
{
    public string Name { get; set; }
    public int ID { get; set; }

    public Person(string name, int id)
    {
        this.Name = name;
        this.ID = id;
    }

    public static string personsToJson(List<Person> persons)
    {
        return JsonConvert.SerializeObject(persons);
    }

    public static List<Person> personsFromJson(string json)
    {
        return JsonConvert.DeserializeObject<List<Person>>(json);
    }

    public static int findPersonID(List<Person> people, string name)
    {
        foreach (var item in people)
        {
            if (item.Name == name)
                return item.ID;
        }
        return -1;
    }

    public static int setID(List<Person> people )
    {
        if (people.Count == 0)
            return 1;

        int newID = 1;
        foreach (var item in people)
        {
            if (item.ID > newID)
                newID = item.ID;
        }

        return (newID + 1);
    }

    public static string findNameByID(List<Person> people, int ID)
    {
        foreach (var item in people)
        {
            if (ID == item.ID)
                return item.Name;
        }
        return "";
    }
}
```

Klasa Person jest klasą służącą do identyfikacji osoby, zawiera ona pola „name” oraz „ID”, w polu name znajdują się dane osoby a pole ID służy do identyfikacji danej osoby z twarzami zapisanymi w pliku.

Klasa ta posiada również statyczne metody takie jak:

- personToJSON
 - w tej metodzie następuje serializacja listy obiektów typu Person do formatu JSON
- personFromJSON
 - w tej metodzie następuje deserializacja formatu JSON do listy obiektów typu Person
- setID
 - Jeżeli dodajemy nową osobę, która nie znajduje się w bazie, metoda ta wyszukuje ostatnie ID i nadaje nowej osobie kolejne wolne ID.
- findNameByID
 - Metoda ta służy do znalezienia danej osoby po ID.
- Okno *MainWindow*

```
<Window x:Class="SFR.MainWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    xmlns:local="clr-namespace:SFR"
    mc:Ignorable="d"
    Title="SFR" Height="484.877"
    Width="922">
    <Grid HorizontalAlignment="Left" Width="912">
        <Image x:Name="image" HorizontalAlignment="Left" Height="261"
Margin="31,32,0,0" VerticalAlignment="Top" Width="377" Stretch="Fill"/>
        <Button x:Name="startCaptureButton" Content="Start capturing"
HorizontalAlignment="Left" Margin="31,332,0,0" VerticalAlignment="Top" Width="377"
Height="35" Click="captureButton_Click"/>
        <Button x:Name="stopCaptureButton" Content="Stop capturing"
HorizontalAlignment="Left" Margin="31,380,0,0" VerticalAlignment="Top" Width="377"
Height="36" Click="stopCaptureButton_Click"/>
        <RichTextBox x:Name="richTextBox" Height="84" VerticalAlignment="Top"
Margin="492,332,10,0">
            <FlowDocument>
                <Paragraph>
                    <Run Text=""/>
                </Paragraph>
            </FlowDocument>
        </RichTextBox>
        <Slider x:Name="brightnessSlider" HorizontalAlignment="Left"
Margin="498,171,0,0" VerticalAlignment="Top" Height="24" Width="172" Minimum="1"
Maximum="100" ToolTip="Set brightness value" SmallChange="1"
ValueChanged="brightnessSlider_ValueChanged"/>
        <Label x:Name="label" Content="Brightness:" HorizontalAlignment="Left"
VerticalAlignment="Top" Margin="498,145,0,0" ToolTip="Set brightness value"/>
        <Label x:Name="brightnessLabel" Content="Waiting for capturing..."
HorizontalAlignment="Left" VerticalAlignment="Top" Margin="568,145,0,0"/>
    </Grid>
</Window>
```

```

        <Button x:Name="button" Content="Reset settings" HorizontalAlignment="Left"
Margin="492,295,0,0" VerticalAlignment="Top" Width="80" Click="button_Click"/>
        <Slider x:Name="contrastSlider" HorizontalAlignment="Left"
Margin="498,219,0,0" VerticalAlignment="Top" Height="24" Width="172" Minimum="1"
Maximum="100" ToolTip="Set contrast value" SmallChange="1"
ValueChanged="contrastSlider_ValueChanged"/>
        <Label x:Name="label1" Content="Contrast:" HorizontalAlignment="Left"
Margin="498,193,0,0" VerticalAlignment="Top"/>
        <Label x:Name="contrastLabel" Content="Waiting for capturing..."
HorizontalAlignment="Left" Margin="568,193,0,0" VerticalAlignment="Top"/>
        <Slider x:Name="sharpnessSlider" HorizontalAlignment="Left"
Margin="498,267,0,0" VerticalAlignment="Top" Height="24" Width="172" Minimum="1"
Maximum="100" ToolTip="Set sharpness value" SmallChange="1"
ValueChanged="sharpnessSlider_ValueChanged"/>
        <Label x:Name="label3" Content="Sharpness:" HorizontalAlignment="Left"
Margin="498,241,0,0" VerticalAlignment="Top"/>
        <Label x:Name="sharpnessLabel" Content="Waiting for capturing..."
HorizontalAlignment="Left" Margin="568,241,0,0" VerticalAlignment="Top"/>
        <Button x:Name="takePhotoBtn" Content="Take photo" HorizontalAlignment="Left"
Margin="802,295,0,0" VerticalAlignment="Top" Width="80" Click="takePhotoBtn_Click"/>
        <Image x:Name="imageBox" HorizontalAlignment="Left" Height="124"
Margin="762,145,0,0" VerticalAlignment="Top" Width="130"/>
        <Button x:Name="savePhoto" Content="Save photo" HorizontalAlignment="Left"
Margin="756,83,0,0" VerticalAlignment="Top" Width="96" Click="savePhoto_Click"
RenderTransformOrigin="1.425,-6.25"/>
        <Label x:Name="capturedPhoto" Content="Captured Photo:"
HorizontalAlignment="Left" Margin="756,114,0,0" VerticalAlignment="Top"/>
        <TextBox x:Name="nameTextBox" HorizontalAlignment="Left" Height="23"
Margin="498,83,0,0" TextWrapping="Wrap" Text="" VerticalAlignment="Top" Width="200"
RenderTransformOrigin="0.133,0.565"/>
        <Label x:Name="nameLabel" Content="Student's name:" HorizontalAlignment="Left"
Margin="492,52,0,0" VerticalAlignment="Top" Width="100"/>
        <Label x:Name="labelDistance" HorizontalAlignment="Left" Margin="31,0,0,0"
VerticalAlignment="Top" Width="207" Height="27"/>
        <Label x:Name="LabelName" FontWeight="Bold" FontSize="16"
HorizontalAlignment="Left" Margin="31,293,0,0" VerticalAlignment="Top" Width="377"
Height="34"/>
    </Grid>
</Window>

```

Jest to kod XAML definiujący aspekty wyglądu programu takie jak kolory poszczególnych elementów, ułożenie czy rozmiary.

9. PLANY DALSZEGO ROZWOJU APLIKACJI

W przypadku podjęcia się dalszego rozwoju aplikacji planowalibyśmy zaimplementować przede wszystkim funkcje związane z trzema, najważniejszymi aspektami bezpieczeństwa jakimi są:

- Poufność,
- Integralność ,
- Dostępność.

Ponadto chcielibyśmy dołożyć wszelkich starań, aby samo rozpoznawanie twarzy działało jeszcze płynniej, a algorytmy były skalibrowane w taki sposób, aby program mylił się znacznie rzadziej niż dotychczas. Jest to kwestia manipulowania wartościami progów dystansów miar euklidesowych i wymaga nieco dłuższego czasu w celu znacznego pogłębienia wiedzy na ten temat. Mimo to, uważamy, że aplikacja działała zadowalająco i sama procedura rozpoznawania studentów wypadła w trakcie prezentacji bardzo dobrze na tle konkurencji.

Najważniejsze zmiany jakie chcielibyśmy w przyszłości wprowadzić to między innymi:

- Dodanie możliwości rejestracji i logowania dla czynnego użytkownika aplikacji.
- Gromadzenie danych zarejestrowanego użytkownika w lokalnej bazie danych.
 - Hasła przechowywane w postaci jednokierunkowej funkcji skrótu.
 - Hasła składające się z małych i dużych liter oraz cyfr, minimum osiem znaków.
- Bezpieczniejsza forma przechowywania zdjęć studentów. Jest to kwestia do przemyślenia, gdyż warto by było ukryć te dane przed nienadzorowanym dostępem osób trzecich.
- Usprawnienie interfejsu aplikacji. Przeniesienie suwaków z ustawieniami kamery do osobnej zakładki w menu górnym.
- Powiększenie okna podglądu obrazu z kamery.
- Czytelniejsze zaznaczanie twarzy nierozpoznanych studentów.
- Możliwość identyfikowania twarzy kilku studentów naraz w celu przyspieszenia procesu weryfikacji.
- Dodanie zakładki *Help* w celu łatwiejszego i szybszego dostępu do instrukcji użytkowania aplikacji.

10. PODSUMOWANIE

Udało nam się zrealizować założenia początkowe w 100%, mimo wielu problemów jakie napotkaliśmy, chociażby z brakiem rzetelnej dokumentacji do *EmguCV 3.0*. Ponadto w warunkach akademickich trzeba pracować nad wieloma projektami w jednym czasie co powoduje drastyczne zmniejszenie możliwości przy wykorzystaniu najlepszych metodologii pracy zespołowej. Jednakże zastosowanie takich narzędzi jak *Slack* czy *GitHub* powoduje, że praca zespołowa staje się wygodna i przyjemna dla każdego członka zespołu. Przepływ informacji nie jest ograniczany czasem a jednocześnie każdy z uczestników projektu ma dostęp do aktualnej wersji aplikacji.