POLITECHNIKA POZNAŃSKA
WYDZIAŁ ELEKTRYCZNY
INFORMATYKA SEMESTR 6

Podstawy Teleinformatyki

Dokumentacja projektu
Rozpoznawanie twarzy i śledzenie ruchu

Łukasz Żołądkiewicz - 126 916
Jonasz Świtała - 121 267
Marek Rybicki - 126 904

Prowadzący:
mgr inż. Przemysław Walkowiak

1. Charakterystyka ogólna projektu

Głównym tematem naszego projektu jest “Rozpoznawanie twarzy i śledzenie ruchu”. Postanowiliśmy nieco rozwinąć naszą charakterystykę i rozłożyliśmy ją na następujące podcele :

1.1. Analiza obrazu z kamery.

Jednym z naszych celów jest analiza obrazu z kamery. Każda przesłana klatka zostanie przeanalizowania w celu wykrycia obiektów według zaimplementowanych modeli. W naszym przypadku głównym celem będzie wykrywanie ludzkiej twarzy.

1.2. Wykorzystanie algorytmu uczenia maszynowego.

Aby w ogóle możliwe było wykrycie twarzy musimy do tego wykorzystać algorytm maszynowy. Przed uruchomieniem wykrywania należy najpierw przeprowadzić trening algorytmu i nauczyć go twarzy zaimplementowanych w bazie danych.

1.3. Wykrywanie twarzy osób.

Ten aspekt wydaje się być naszym głównym celem projektu. Wykrywanie twarzy na podstawie kaskad zawartych w bibliotece OpenCV.

1.4. Identyfikacja.

Po wykryciu twarzy kolejnym krokiem jest identyfikacja. W tym celu właśnie wykorzystany zostanie algorytm maszynowy, który po nauczeniu będzie w stanie dopasować odpowiednie próbki do użytkownika w bazie.

1.5. Stworzenie listy obecności użytkowników

To drugi z naszych głównych celów projektowych. Każdy wykryty użytkownik zostanie dodany do listy, a następnie taką listę będzie można podejrzeć, zapisać czy edytować.

2. Wybór tematu

Nad naszym projektowym tematem zastanawialiśmy się dość długo i wybraliśmy go po przedstawieniu naszej krótkiej listy “za i przeciw”.

Temat został wybrany na podstawie następujących argumentów:

a) **Wstępna znajomość zagadnienia przetwarzania obrazów.**

Z podstawami przetwarzania obrazów zapoznaliśmy się podczas zajęć z przedmiotu “Przetwarzanie Obrazów i Systemy Wizyjne” na 4 semestrze.

b) **Znajomość biblioteki OpenCV.**

Już wcześniej mieliśmy styczność z tą biblioteką OpenCV. Między innymi była ona prezentowana na wspomnianym już przedmiocie “Przetwarzanie Obrazów i Systemy Wizyjne”, jak także i późniejszym “Języki i paradygmaty programowania”.

c) **Ciekawość zagadania.**

Tematyka przetwarzania obrazu jest interesująca, oraz wykorzystywana w wielu gałęziach informatyki, dlatego kusząca była dla nas możliwość poszerzenia swojej wiedzy i umiejętności praktycznych, w związku z tym zdecydowaliśmy się na wybranie tego tematu projektu.

d) **Uczenie maszynowe.**

Kolejnym dość ważną gałęzią w rozwoju programisty jest sztuczna inteligencja. Zorientowaliśmy się wstępnie, że w projekcie będziemy mogli wykorzystać aspekty uczenia maszynowego co jeszcze bardziej przekonało nas do wyboru tego tematu.

e) **Bezpieczeństwo.**

Wykrywanie i rozpoznawanie twarzy jest jednym z aspektów bezpieczeństwa. Ponieważ naszym zainteresowaniem jak i specjalizacją jest tematyka “Bezpieczeństwo Systemów Informatycznych” również postanowiliśmy się temu przyjrzeć pod tym kątem.

3. Funkcjonalność aplikacji.

Zakres naszych prac określiliśmy jako wymagania funkcjonalne i niefunkcjonalne. Wśród wymagań funkcjonalnych został określony tylko jeden aktor jakim jest użytkownik.

3.1. Wymagania funkcjonalne.

a) **Dodanie, usunięcie lub edycja użytkownika w bazie danych.**

Chodzi o dodanie użytkownika i jego zdjęcia do struktury danych w celu późniejszego rozpoznania w algorytmie uczenia maszynowego.

b) **Wykrycie twarzy człowieka.**

Główny cel naszego projektu czyli wykrycie twarzy na podstawie kaskadowego modelu zawartego w bibliotece OpenCV.

c) **Rozpoznanie twarzy zawartych w bazie danych.**

Funkcjonalność, która zawarta jest w temacie i ma umożliwić sporządzenie listy obecności użytkowników będących w sali wykładowej.

d) **Tworzenie bazy danych rozpoznanych twarzy.**

Każda rozpoznana twarz na polecenie użytkownika może zostać zapamiętana za pomocą kilku obrazów odcieniach szarości i opisana za pomocą krótkiej etykiety zawierającej np. imię i nazwisko. Etykieta zostanie wyświetlona na wyświetlanym z kamery obrazie, jeżeli program rozpozna twarz osoby stojącej w obiektywie.

e) **Generowanie listy rozpoznanych osób.**

Aplikacja umożliwia wygenerowanie listy wszystkich osób, które zostały rozpoznane przez aplikację, pod warunkiem, że wizerunek każdej osoby, która ma znaleźć się na liście został wcześniej zapamiętany wraz z opisem.

f) **Autoryzacja.**

Rozpoznawanie twarzy ma bardzo szerokie praktyczne zastosowanie w Informatyce jest stosowane m.in. w ochronie, systemach bezpieczeństwa, systemach kontroli dostępu, policyjnych bazach danych

3.2. Wymagania niefunkcjonalne.

a) **Języki programowania C#.**

Językiem przez nas wybranym było C#. Wybraliśmy go z powodu największej znajomości tego środowiska.

b) **Wykorzystanie EmguCV.**

EmguCV nie jest biblioteką lecz .Netowym wrapperem do OpenCV. Biblioteki, która w sobie ma zawarte m.in. mechanizmy wykrywania i identyfikacji twarzy.

c) **Interfejs aplikacji w języku polskim.**

Docelową grupą dotarcia są osoby posługujące się językiem polskim.

d) **Dane przechowywane w relacyjnej bazie.**

Schemat bazy został przygotowany jako schemat relacyjnej bazy danych.

4. Architektura aplikacji

Określają nam planowane narzędzia oraz metody w jakich implementowane będą poszczególne części projektu. Określone zostały również narzędzia wspomagające projektowanie.

a) **Aplikacja desktopowa.**

Aplikacja zaimplementowana w środowisku C#. Całość będziemy realizować w Visual Studio 2017.

b) **Baza danych w plikach tekstowych.**

Z powodu ograniczeń czasowych nie zdecydowaliśmy się na implementację bazy SQL.

c) **Przechowywanie danych lokalnie.**

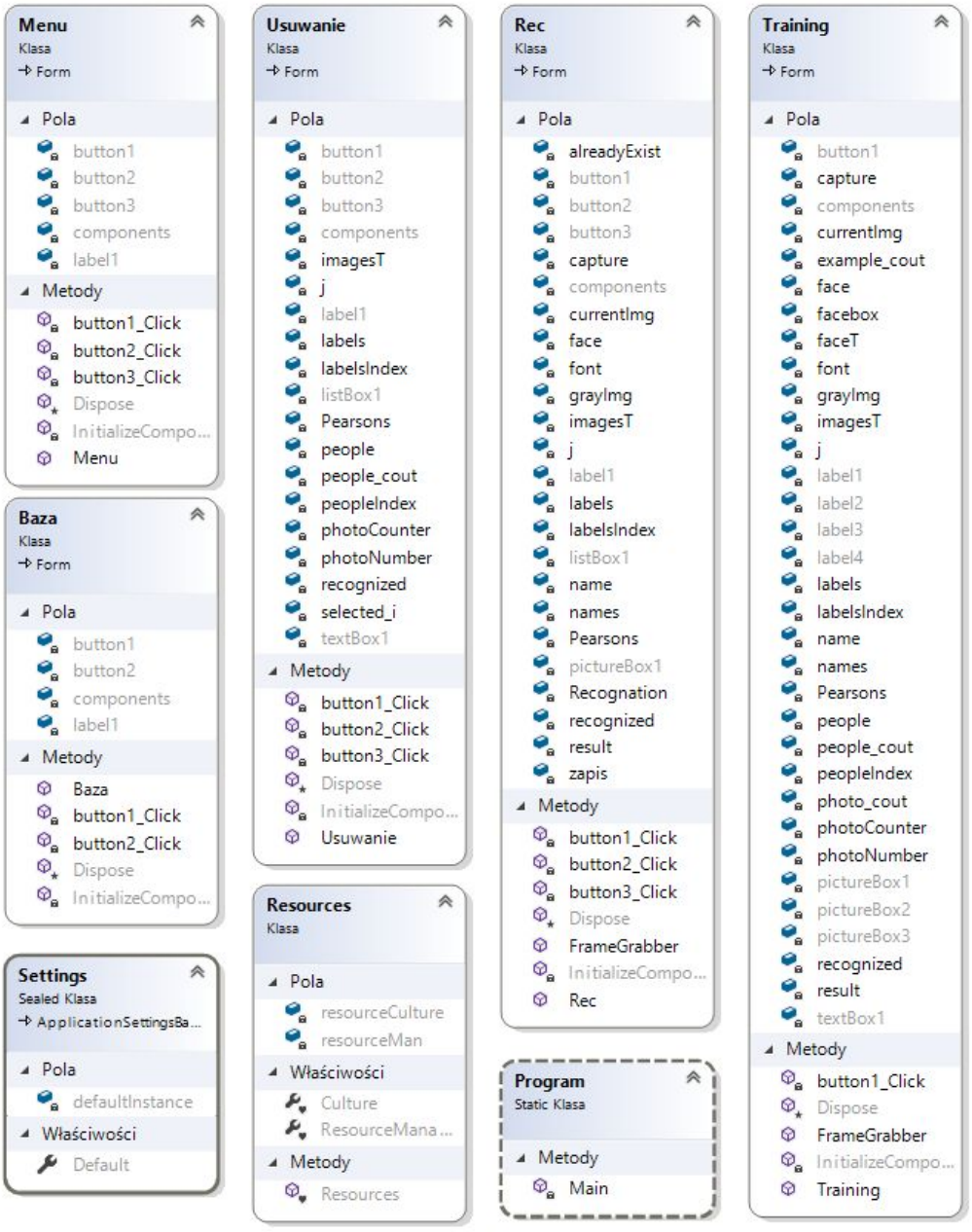
Krótkie opisy rozpoznanych twarzy wprowadzane przez użytkowników oraz obrazy potrzebne do prawidłowego działania aplikacji przechowywane są lokalnie na urządzeniu na, którym uruchomiona jest aplikacja. Nie zdecydowaliśmy się na przechowywanie tych danych na zewnętrznym serwerze, ponieważ zapamiętywane obrazy zawierające rozpoznane twarze są w odcieniach szarości i mają niewielkie wymiary.

4.1. Schematy UML.

Notacja UML jest graficznym przedstawieniem systemu, w postaci diagramu. Modele zapisane w języku UML przedstawiają system od ogółu do szczegółu. Daje możliwość obejrzenia systemu z wybraną w danym momencie szczegółowością.

a) Diagram klas.

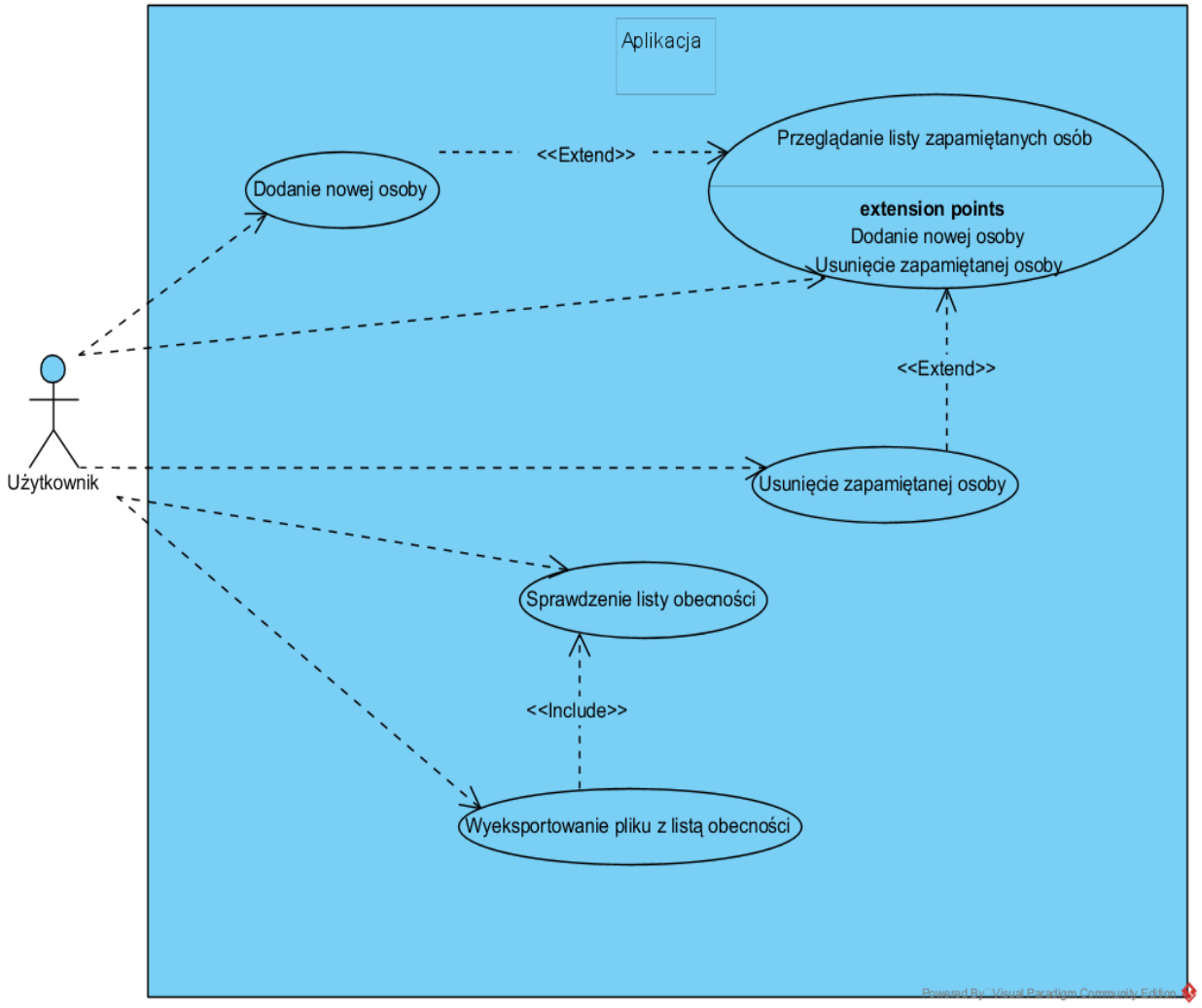
Diagram klas obrazuje pewien zbiór klas, interfejsów i kooperacji oraz związki między nimi. Jest on grafem złożonym z wierzchołków i łuków. Diagram klas stanowi opis statyki systemu, który uwypukla związki między klasami, pomijając pozostałe charakterystyki.



Obraz 1: Diagram klas projektu.

b) Schemat przypadków użycia.

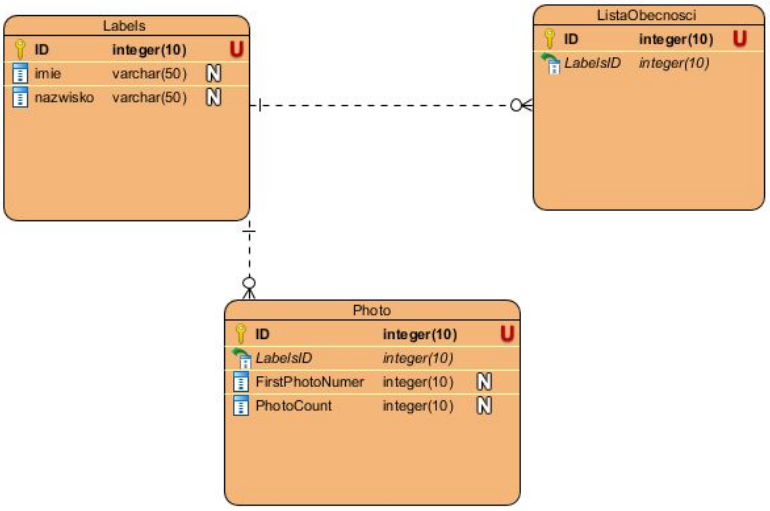
Diagram przypadków użycia jest diagramem, który przedstawia funkcjonalność systemu wraz z jego otoczeniem. Diagramy przypadków użycia pozwalają na graficzne zaprezentowanie własności systemu tak, jak są one widziane po stronie użytkownika.



Obraz 2: Schemat przypadków użycia.

c) Schemat ERD bazy danych SQL

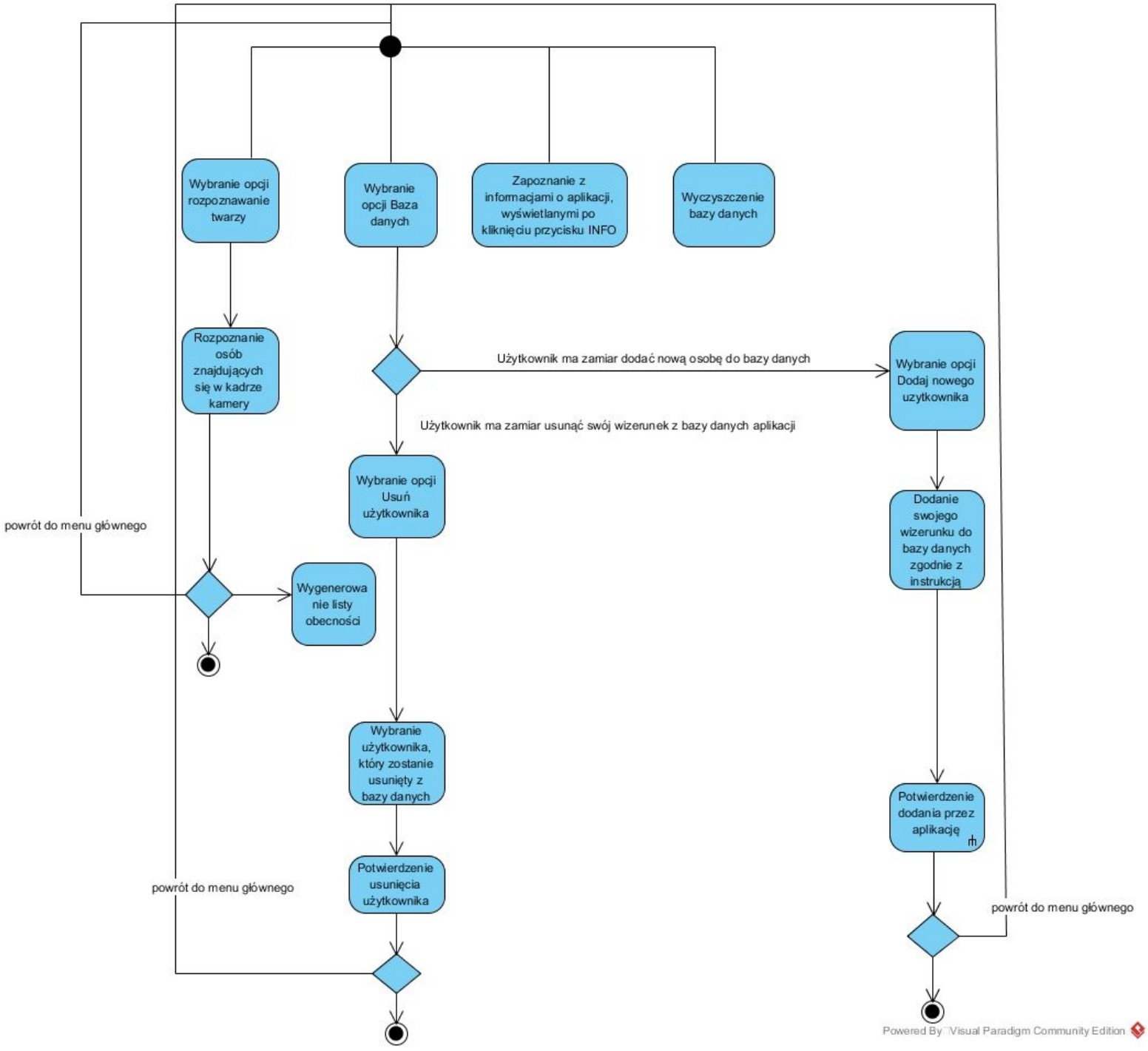
Diagram ERD powinien zawierać związki pomiędzy danymi pamiętanymi, tzn. takimi, które nie mogą być wyprowadzone z innych danych. Obecnie coraz częściej używane tylko do modelowania baz danych podczas projektowania fizycznego.



Obraz 3: Schemat relacyjnej bazy danych.

d) Diagram aktywności

Diagram aktywności pokazuje przepływ kontroli od aktywności do aktywności. Pokazuje współbieżność, rozgałęzienie, przepływ sterowania i przepływ obiektów.



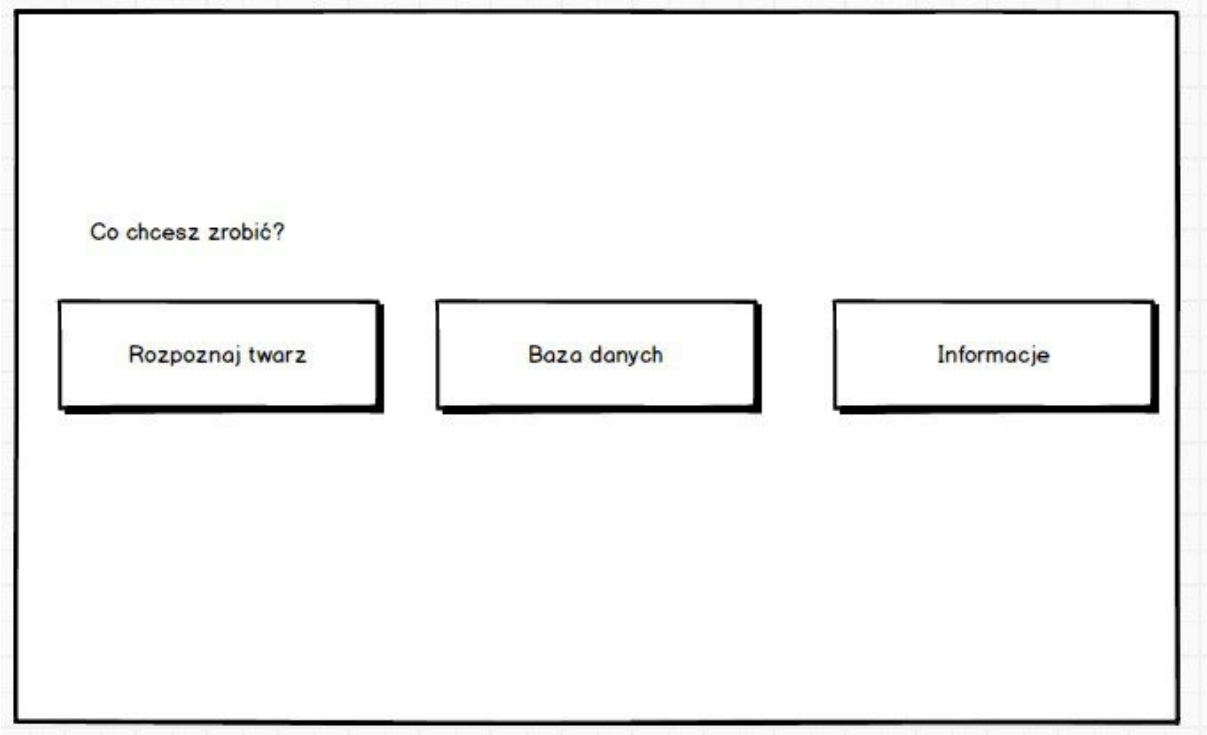
Obraz 4: Schemat diagramu aktywności.

4.2. Interfejs użytkownika.

Interfejs użytkownika to część aplikacji odpowiedzialna za interakcję z użytkownikiem. Aby możliwa była bezpośrednia komunikacja między człowiekiem, a komputerem aplikację są wyposażone w odpowiednie grafiki oraz przyciski tworzące razem interfejs użytkownika.

a) Menu główne.

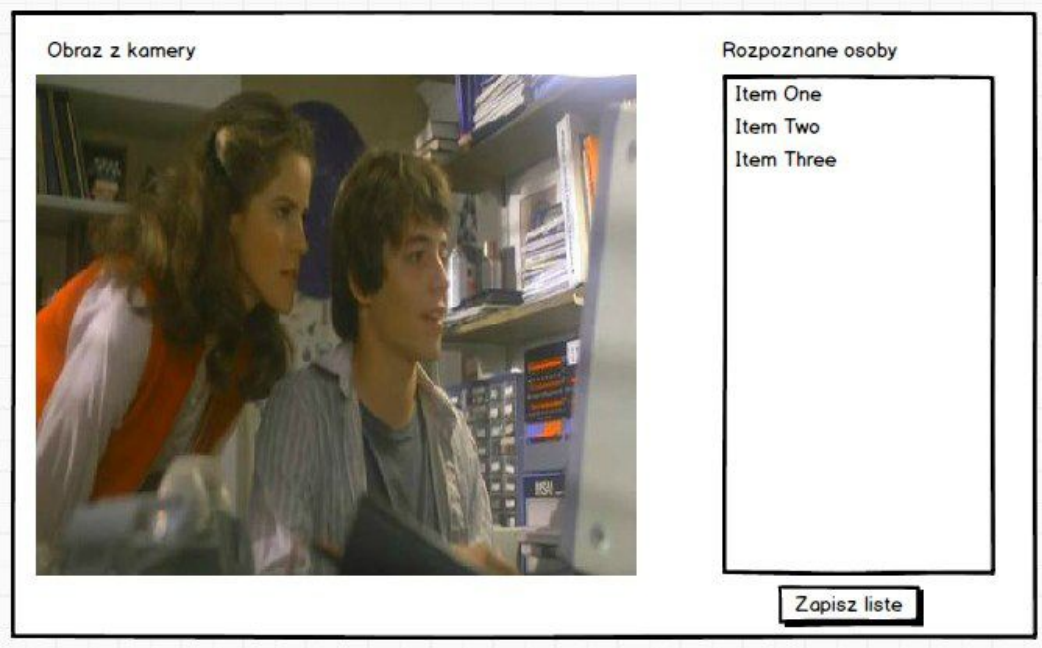
W projekcie menu główne ma zawierać trzy opcje : “Rozpoznawanie twarzy”, “Baza Danych” oraz “informacje”



Obraz 5: Projekt menu głównego.

b) Interfejs rozpoznawania twarzy.

Okno rozpoznawania twarzy będzie składać się z obrazu kamery, listy wykrytych twarzy oraz przycisku umożliwiającego zgranie listy do pliku.



Obraz 6: Projekt interfejsu rozpoznawania twarzy.

c) **Interfejs dodawania użytkownika do bazy.**

Okno to będzie zawierać pola z imieniem i nazwiskiem użytkownika oraz możliwością dodania jego zdjęcia.

Obraz 7: Projekt interfejsu dodawania twarzy.

5. Wybrane technologie

Technologie wybrane przez nas odpowiadają naszym upodobaniom i doświadczeniu zdobytemu przy poprzednich projektach oraz innych zajęciach w trakcie studiowania. Przy większości z nich używaliśmy rozwiązań firmy Microsoft.

5.1. NET Framework.

.NET wybraliśmy ze względu na nasze wcześniejsze doświadczenie i znajomość języka C\#. Podjęcie takiego wyboru umożliwiło istnienie wrappera EmguCV, który umożliwia wykorzystanie biblioteki OpenCV w aplikacji rozwijanej w środowisku .NET.

5.2 IDE.

Kwestia wyboru IDE była dosyć oczywista - korzystając z rozwiązań Microsoftu najwygodniejszym rozwiązaniem było wybranie kolejnego produktu firmy - Visual Studio 2017 oraz edytora kodu Visual Studio Code.

5.3 EmguCV.

Wieloplatformowy wrapper do biblioteki OpenCV służącej do przetwarzania obrazów. Zdecydowaliśmy się na wybranie EmguCV ze względu na podstawową znajomość biblioteki OpenCV, oraz faktu, że jest ona powszechnie wykorzystywana do przetwarzaniu obrazu w czasie rzeczywistym.

5.4. Windows Form.

Interfejs pozwalający na łatwe tworzenie aplikacji zawierających graficzny interfejs użytkownika w ramach .NET Framework.

5.5. System kontroli wersji - GitHub.

Całość naszej pracy była odnotowywana w systemie kontroli wersji GitHub.

5.6. Metoda wykrywania twarzy.

Wykorzystaliśmy dostępną w OpenCV metodę wykrywania twarzy korzystającą z uczenia maszynowego oraz klasyfikatora kaskadowego opartego na cechach.

5.7. Wybór metody rozpoznawania twarzy.

Wybraliśmy metodę rozpoznawania twarzy PCA (Principal Components Analysis Eigenfaces) w skrócie Eigenfaces ze względu na największą dokładność rozpoznawania twarzy, dostępność w EmguCV oraz wymagane zasoby m.in.niewielki rozmiar przechowywanych obrazów.

5.8. Opis wykorzystanej metody rozpoznawania twarzy.

Metoda Eigenfaces wywodzi się z teorii informacji. Polega na projekcji wielowymiarowego wektora (obrazu) na nowy układ współrzędnych, który tworzą pewne charakterystyczne obrazy czyli Eigenfaces. Główną ideą jest przedstawienie obrazu jako liniowej kombinacja bazowych obrazów (Eigenfaces). Współczynniki liniowe kombinacji określają dany obraz. Wymiar wektora współczynników jest dużo mniejszy niż wymiar obrazu np. dla obrazu 50x50 wymiar wektora wynosi 2500. Może on zostać przedstawiony za pomocą wektora współczynników liniowej kombinacji np. o wymiarze 20. Obraz może być widoczny jako punkt w wielowymiarowej przestrzeni. Jeżeli mamy dostępny pewien zbiór obrazów możliwe jest wyznaczenie macierzy korelacji między tymi wektorami i obliczenie wektorów własnych tej macierzy. Wektory własne to właśnie bazowe obrazy tzw. Eigenfaces.

6. Interesujące problemy

W trakcie prac natrafiliśmy na kilka problemów związanych z rozwijaniem aplikacji, które opisaliśmy w poniżej.

6.1. Integracja Emgu CV z projektem aplikacji

Niewielka ilość dokumentacji, oraz częste i duże zmiany pomiędzy kolejnymi wersjami biblioteki początkowo uniemożliwiły nam integrację wrappera z tworzoną aplikacją ze względu na to, że często migrowaliśmy między poszczególnymi wydaniami biblioteki. Ostatecznie znaleźliśmy wersję, z którą nie mieliśmy większych problemów i byliśmy w stanie rozpocząć prace nad docelową funkcjonalnością aplikacji.

6.2. Błędne rozpoznawanie zapamiętanych twarzy.

Przyczyną tego problemu była niewielka ilość zapamiętanych obrazów odpowiadających za reprezentację konkretnej twarzy oraz zbyt duże podobieństwo tych obrazów (zdjęcia danej twarzy wykorzystywane przez metodę Eigenfaces muszą być wykonane z kilku różnych profili). Problem rozwiązaliśmy dodając funkcję podpowiadającą w jaki sposób ustawić twarz przy dodawania kolejnego zdjęcia danej osoby, dzięki czemu rozwiązanie stało się bardziej dokładne.

6.3. Błędne rozpoznawanie zapamiętanych twarzy po zmianie warunków oświetleniowych.

Przyczyną tego problemu było wykonywanie zdjęć na potrzeby procesu uczenia przy tym samym oświetleniu, najlepszym rozwiązaniem tego problemu jest powtórzenie wykonania serii zdjęć z takich profilii jak podpowiada aplikacja w różnych warunkach oświetleniowych np. światło ciepłe, zimne, naturalne, półmrok itp.

6.4. Optymalizacja.

Kolejnym istotnym problemem była optymalizacja aplikacji, tak by aplikacja była w stanie analizować przetwarzany w czasie rzeczywistym obraz w rozsądnym przedziale czasowym. Sprzęt, na którym uruchamiana była aplikacja nie należał do najwydajniejszych, a analizowanie ok. 26 klatek na sekundę przy rozdzielczości

720p sprawiał, że wykrycie twarzy i przypisanie jej konkretnej etykiety zajmował zdecydowanie za dużo czasu. Problem rozwiązaliśmy kompresując obraz oraz ustawiając koligację procesu aplikacji, tak by wykorzystywał dostępne zasoby komputera.

6.5. Zbyt niska rozdzielczość próbek.

Dużym problemem przy wykrywaniu twarzy była zbyt niska rozdzielczość próbek. Z początku próbowaliśmy pobierać próbki o rozmiarach 50x50. Kolejnym krokiem były próbki 100x100. Optymalnie zdecydowaliśmy się jednak na próbki 200x200 które poprawiły trafność rozpoznawania.

Obraz 8: Przykładowe próbki.

6.6. Blokowanie działania aplikacji i edycji plików.

Po pierwszym uruchomieniu aplikacji niektóre programy antywirusowe blokują jej działanie ze względu na dostęp do kamery internetowej. Rozwiązanie tego problemu jest bardzo proste wystarczy dodać naszą aplikację do wyjątków lub ewentualnie jeżeli nie ma takiej możliwości po każdym uruchomieniu zezwalać jej na dostęp do kamery.

6.7. Nieprawidłowe dodanie zdjęcia.

Częstym problemem było również nieprawidłowe dodanie zdjęcia. Czasem osoba nie została wykryta, a formularz przeszedł dalej przez co w bazie brakowało pliku. Rozwiązaniem tego problemu była implementacja wyjątków, które zabezpieczyły program przed tego typu błędem.

6.8. Zbyt mała ilość próbek.

Problemem z wykrywaniem twarzy była również zbyt mała ilość próbek. Na początku rozpoczęliśmy nasze testy z użyciem tylko dwóch zdjęć dla danej osoby. Okazało się jednak, że algorytm często mylił wtedy wykrywane osoby i praktycznie identyfikacja odbywała się z pewną losowością. Poprawiliśmy to używając pięciu próbek co wydaję się być optymalną liczbą.

7. Opis implementacji.

Poniżej opisane zostały najciekawsze oraz najważniejsze fragmenty kodu prezentowanej aplikacji.

7.1. Wykrywanie twarzy.

Wykrywanie twarzy zostało oparte na klasyfikatorze kaskadowym „Haar Feature-based Cascade Classifier".

```
face = new
HaarCascade("haarcascade_frontalface_default.xml");

//…

MCvTermCriteria termCrit = new MCvTermCriteria(recognized,
0.001);

EigenObjectRecognizer recognizer = new
EigenObjectRecognizer(imagesT.ToArray(), labels.ToArray(),
3000, ref termCrit);

currentImg.Draw(name, ref font, new Point(f.rect.X - 2,
f.rect.Y - 2), new   Bgr(Color.GreenYellow));
```

a) **face** - obiekt klasy HaarCascade przechowujący klasyfikator wczytany z pliku .xml

b) **termCrit** - obiekt klasy MCvTermCriteria reprezentujący kryteria rozpoznawania twarzy na podstawie obrazów szkoleniowych zawierający m.in maksymalną liczbę iteracji, która może zostać wykonana w celu rozpoznania

konkretnej twarzy.

c) **recognizer** - obiekt klasy EigenObjectRecognizer implementującej opisana wyżej metodę rozpoznawania twarzy Eigenfaces.

d) Za pomocą standardowej metody Draw dostępnej w klasie Image rysowane jest czerwone obramowanie wykrytej twarzy na obrazie pochodzącym z kamery.

7.2. Klasyfikator kaskadowy.

Bardzo krótki fragment pliku haarcascade_frontalface_default.xml zawierającego klasyfikator przygotowany do wykrywania twarzy.

```
<maxWeakCount>26</maxWeakCount>

<stageThreshold>-1.4103703498840332e+00</stageThreshold>
      <weakClassifiers>
        <_>
          <internalNodes>
            0 -1 532 -1.0988018475472927e-02</internalNodes>
          <leafValues>
            6.4358645677566528e-01
-2.3149165511131287e-01</leafValues></_>
```

7.3. Rozpoznawanie zapamiętanych twarzy.

Aby móc wykrywać twarzę należy wytrenować klasyfikator. Nie musimy jednak tego robić sami. Trenowanie klasyfikatora odbywa się poprzez dostarczenie kilkuróżnych próbek, na których znajduję się dany obiekt.

```
   MCvAvgComp[][] facesDetected =
grayImg.DetectHaarCascade(face, 1.2, 10,
Emgu.CV.CvEnum.HAAR_DETECTION_TYPE.DO_CANNY_PRUNING, new
Size(20, 20));

   foreach (MCvAvgComp f in facesDetected[0])
        {
            j++;
  result = currentImg.Copy(f.rect).Convert<Gray,
byte>().Resize(200, 200,
Emgu.CV.CvEnum.INTER.CV_INTER_CUBIC);
  currentImg.Draw(f.rect, new Bgr(Color.OrangeRed), 2);

                if (imagesT.ToArray().Length != 0)
                {
                    MCvTermCriteria termCrit = new
MCvTermCriteria(recognized, 0.001);
                    EigenObjectRecognizer recognizer = new
EigenObjectRecognizer(imagesT.ToArray(), labels.ToArray(),
3000, ref termCrit);
                    name = recognizer.Recognize(result);
                    currentImg.Draw(name, ref font, new
Point(f.rect.X - 2, f.rect.Y - 2), new
Bgr(Color.GreenYellow));
                    }
                }
}
```

a) **imagesT** - lista wszystkich obrazów na, których aplikacja uczy się rozpoznawać daną twarz.

b) **Labels** - lista przechowująca informację o każdej zapamiętanej twarzy jest ona tworzona ze zmiennej Labelsinfo za pomocą metody Split należącej do typu String, separatorem jest znak #.

c) **termCrit -** obiekt odpowiadający za parametry wykrywania twarzy.

d) **recognizer** - obiekt uczenia maszynowego z informacjami jako parametry.

e) **name -** zmienna, któremu zostanie zwrócona nazwa poprawnie wykrytej twarzy.

7.4. Zapamiętywanie nowych twarzy.

Po kliknięciu na przycisk zapisz następuje przeniesienie obrazów z tablicy bajtów do plików. Nazwa każdego zdjęcia wykorzystanego w procesie uczenia zaczyna się od słowa person, każde zdjęcie zapisywane jest tylko w skali szarości w formacie .bmp.

```
for (int i = 1; i < imagesT.ToArray().Length + 1; i++)
  {
   imagesT.ToArray()[i - 1].Save(Application.StartupPath +
"pearson" + i + ".bmp");
File.AppendAllText(Application.StartupPath + "labels.txt",
labels.ToArray()[i - 1] + "#");           }
```

7.5. Dodawanie twarzy do listy.

Dodawanie twarzy do listy odbywa się na podstawie sprawdzenia czy wykryta twarz jest unikalna. Jeśli tak, a nazwa istnieje wtedy kolejne nazwiska zostają dodane do listy.

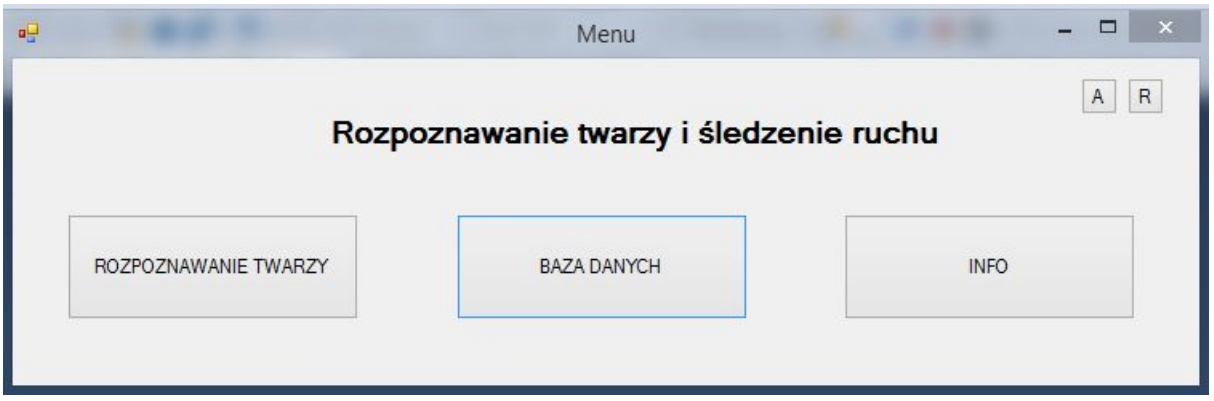
```
alreadyExist = Recognation.Contains(name);
            if (!alreadyExist && name != "") {
               Recognation.Add(name); }
```

8. Instrukcja użytkowania aplikacji

8.1. Początkowy interfejs aplikacji

Początkowy interfejs aplikacji zawiera menu, w którym do wyboru mamy trzy opcje : “Rozpoznawanie twarzy”, “Baza danych” oraz “Info”. Pierwsza opcja dotyczy samego mechanizmu rozpoznawania twarzy, w tym takich opcji jak automatyczne dodawanie użytkowników do listy obecności. Druga z opcji to baza danych, w której to możemy dodać

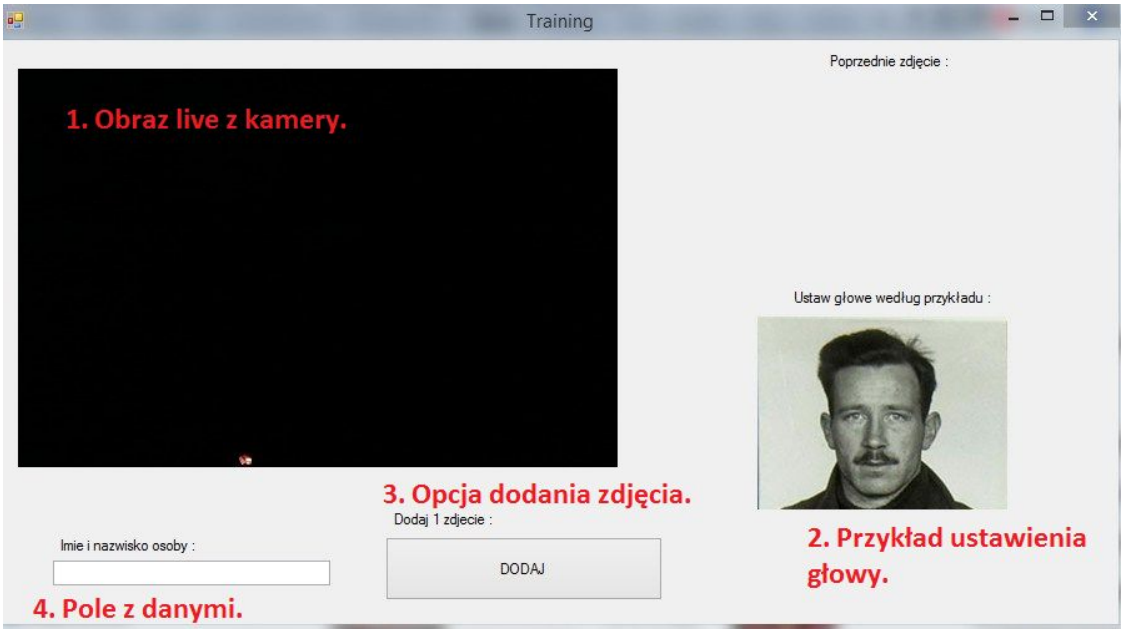
usunąć lub modyfikować istniejącego użytkownika wraz z twarzą. Trzecia opcja to informacja na temat aplikacji.



Obraz 8: Początkowy interfejs aplikacji.

8.2. Dodawanie użytkownika do bazy danych aplikacji.

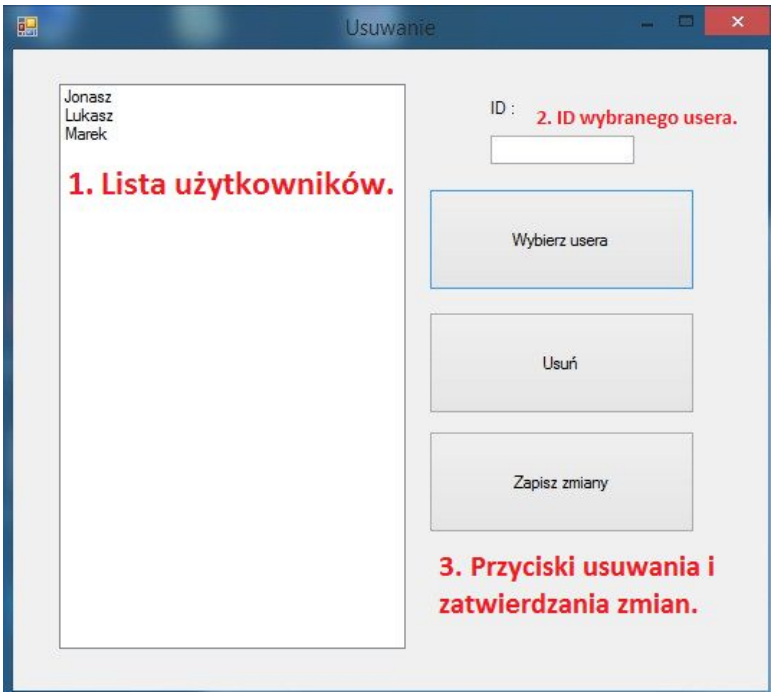
Pierwszą rzeczą od której powinniśmy rozpocząć jest dodawanie użytkownika do bazy. Aby to zrobić musimy przejść w menu głównym do opcji “Baza danych”. Następnie kliknąć w opcję “Dodaj nowego użytkownika”. Po uruchomieniu ukaże nam się okno zarządzania dodawaniem. W nim powinniśmy zacząć od wpisania imienia i nazwiska osoby, której chcemy dodać. Po uzupełnieniu pola powinniśmy ustawić się według schematu i kliknąć na “dodaj”. Kolejne punkty to opcje dodania kolejnych zdjęć w celu poprawienia rozpoznawanie twarzy. Aby dodać kolejne zdjęcia ustawiamy główne według przykładu i klikamy na “dodaj”.



Obraz 9: Okienko dodawania nowego użytkownika.

8.3. Usuwanie użytkownika z bazy danych.

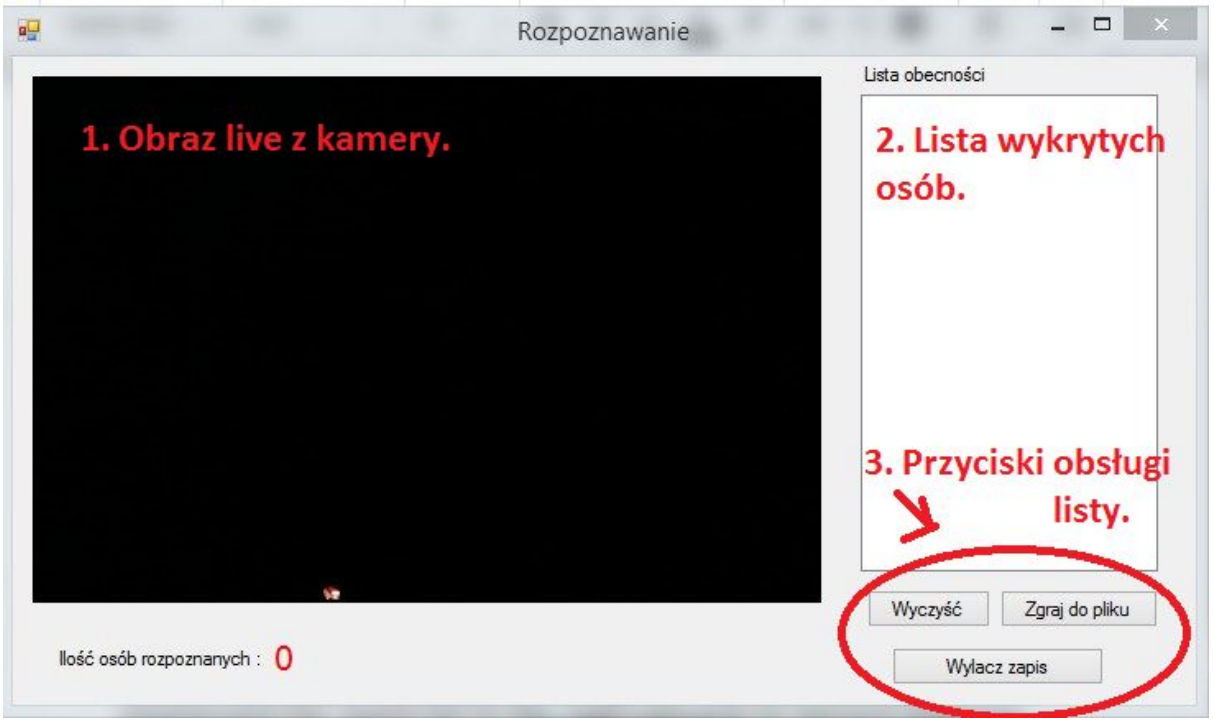
Aby rozpocząć operacja usuwania użytkowników z bazy musimy wybrać opcje “Baza Danych” w menu głównym. Kolejno powinniśmy kliknąć w opcję “Usuń użytkownika”. Zostaniemy przeniesieni do kreatora usuwania, w którym to dostępne mamy opcję “wybierz usera”, “usuń” oraz “zapisz zmiany”. Dodatkowo dostępną mamy listę z użytkownikami obecnie dodanymi. Aby usunąć użytkownika musimy wybrać go z listy i kliknąć na “wybierz usera”. Następnie klikamy “usuń” oraz zatwierdzamy naciskając na przycisk “zapisz zmiany”.



Obraz 10: Okienko usuwania użytkowników z bazy.

8.4. Wykrywanie twarzy.

Aby przejść do okienka wykrywania twarzy musimy wejść w menu głównym w opcje “Wykrywanie twarzy”. Ukaże nam się formularz z obrazem na żywo z kamerki. Listą wykrytych uczestników oraz przyciskami obsługi listy. Przyciski te pozwalają na wyczyszczenie listy, zgranie listy do pliku, bądź wyłączenie lub włączenie dodawania wykrytych użytkowników na listę.



Obraz 11: Okienko rozpoznawania twarzy.

9. Testowanie

Testowanie naszej aplikacji odbyło się na zasadzie weryfikacji działania wszystkich funkcji zgodnie z pierwotnymi założeniami.

9.1. Funkcjonalność bazy danych.

Testy rozpoczęliśmy od sprawdzenia funkcjonalności naszej bazy danych. Na wstępie każdy z autorów programu na autonomicznym sprzęcie dodał do bazy kilku unikalnych użytkowników. Po każdym dodaniu został stworzony przegląd listy użytkowników oraz zdjęć i relacji między nimi. Kolejno sprawdzone zostało wczytywanie bazy, do którego został przygotowany test jednostkowy. Test ten sprawdzał czy wczytana do tablicy z pliku baza jest zgodna i nie zawiera błędów. Przetestowane zostało również usuwanie zawodników oraz sprawdzanie czy poprzednie obiekty nie pozostawiały po sobie jakiś śladów. Tutaj również nie wykryliśmy żadnych niepokojących anomalii.

9.2. Rozpoznawanie twarzy.

Samo testowanie rozpoznawania twarzy rozpoczęliśmy od sprawdzenia algorytmu rozpoznawania twarzy. Na początku zweryfikowaliśmy czy algorytm poprawnie odczytuje i interpretuje zawartość naszej bazy danych. Kolejno przygotowaliśmy kilka zdjęć testowych osób oraz wideo w różnych pozycjach. Algorytm z dużą dokładnością wykrywał osoby na przykładowych wideo.

9.3. Rozpoznawanie twarzy w różnych warunkach oświetleniowych

Zdecydowaliśmy się również sprawdzić wpływ jakości oświetlenia na działanie naszej aplikacji. Pomimo, że nie udało nam się wykorzystać profesjonalnej kamery test wypadł dosyć dobrze. Aplikacja trochę gorzej radziła sobie z wykrywaniem i rozpoznawaniem twarzy jedynie przy słabym oświetleniu pomieszczenia lub oświetleniu zimnym światłem.

9.4. Przetestowanie wpływu parametrów sprzętowych na jakość działania aplikacji.

Naszym celem było takie zoptymalizowanie aplikacji aby można było z niej również swobodnie korzystać na mniej wydajnym sprzęcie. Test przeprowadziliśmy na kilku komputerach o zróżnicowanych parametrach sprzętowych. Na urządzeniu o najniższych parametrach działanie aplikacji było tylko trochę wolniejsze niż na znacznie bardziej wydajnym sprzęcie, nadal było możliwe normalne korzystanie z aplikacji.

9.5. Przetestowanie wpływu ilości zdjęć wykorzystywanych w procesie uczenia na jakość działania aplikacji.

Test wykazał, że minimalna liczbą zdjęć wykonanych na potrzeby procesu uczenia rozpoznawania danej twarzy powinna wynosić 5, zdjęcia powinny być wykonywane w takich profilach jak zostało to opisane w punkcie 6.2. Najlepszym rozwiązaniem jest powtórzenie tej serii w różnych warunkach oświetleniowych, tak aby algorytm nie popełniał ewentualnych błędów po ich zmianie.

9.6. Testy całościowe.

Końcowym etapem naszego projektu były testy całościowe. Przetestowaliśmy w nich całokształt działania aplikacji. Kilkukrotnie dodaliśmy, usunęliśmy i przeprowadzaliśmy modyfikacje użytkowników w bazie. Za każdym razem sprawdzaliśmy też skuteczność wykrywania przez algorytm kolejnych osób. Algorytm pracował bez zarzutów i nie wykryliśmy żadnych nieprawidłowości.

10. Podział prac

Przedstawienie podziału prac rozdzieliliśmy na podpunkty z konkretnymi autorami oraz ich wkładem w wykonanie projektu.

10.1. Marek Rybicki.

a) **Początkowa faza projektu.**

Jedna najważniejszych części całego projektu czyli dokumentacja. Marek opanował początkowe założenia projektu wraz z metodami funkcjonalnymi oraz niefunkcjonalnymi. Kolejnym krokiem było przygotowanie wykresów oraz schematu bazy danych.

b) **Przetestowanie i wybór jednego z trzech algorytmów.**

Biblioteka OpenCV domyślnie do wyboru oferuje trzy algorytmy wykrywania twarzy są to : eigenfaces, fisherfaces oraz lbph. Marek zaimplementował wszystkie te trzy algorytmy oraz poddał je testom wydajności. Z uwagi na stosunek jakości do wydajności zdecydowaliśmy się na algorytm eigenfaces.

c) **Dokładna implementacja algorytmu eigenfaces.**

Po wyborze konkretnego algorytmu rozpoczęła się faza implementacyjna. Marek zajął się implementacją samego rozpoznawania twarzy. Opanował bibliotekę EmguCV oraz dołożył elementy graficzne jak zaznaczanie wykrytej twarzy czy pojawienie się przy użytkowniku imienia i nazwiska.

d) **Testy jednostkowe bazy danych oraz wykrywania twarzy.**

Po zaimplementowaniu głównej funkcjonalności należało wykonać testy jednostkowe. W tym punkcie dotyczyły one sprawdzenia bazy danych oraz

poprawności wykrywania twarzy. Marek przygotował kilkadziesiąt próbek po czym poddał sprawdzeniu obie implementacje.

e) Poprawna błędów związana z głównym algorytmem.

Po wstępnej implementacji nasz projekt nie był idealny. Wykorzystując swoją wiedzę na temat algorytmu eigenfaces, Marek poprawił główne błędy oraz dopracował pewne niedoskonałości, które zostały wykazane podczas testów naszej aplikacji.

10.2. Łukasz Żołądkiewicz.

a) Initial research.

W głównej mierze punkt ten polegał na wyborze środowiska oraz biblioteki, w której realizowany był nasz projekt. Łukasz rozpoczął od przetestowania oraz sprawdzenia dostępnych narzędzi w językach C++, C#, Java oraz Python. Po krótkim reseachru zdecydował, że najbardziej odpowiedni będzie C# z użyciem wrappera EmguCV.

b) Optymalizacja wydajności.

Od początku założyliśmy wydajność jako jeden z głównym punktów naszego projektu. Łukasz skupił się na jak największym podniesieniu wydajności przy umiarkowanym wpływie na jakość. Po implementacji algorytmu wykrywania twarzy proces analizy był bardzo kosztowny i aplikacja była bardzo “ciężka w użyciu”. Zaimplementowana została wielowątkowość oraz obniżone zostały pewne parametry jak rozdzielczość czy ilość analizowanych klatek na sekundę.

c) Opanowanie problemów z EmguCV.

Od samego początku biblioteka EmguCV sprawiała nam dość dużo błędów. Chodziło między innymi o problem z niezgodnościami wersji co do dokumentacji czy problemami z wersją systemu operacyjnego. Łukasz cierpliwe debugował błędy i znajdował ich rozwiązania.

d) Testy interfejsu oraz user experience.

Łukasz zajął się testami interfejsu oraz mocno postawił na user experience. Miało to na celu poprawę jakości naszej aplikacji oraz ułatwienia jej obsługi. Po tym zabiegu część funkcjonalności została połączona, a część zmieniała swoje położenie.

e) Debugowanie i poprawa błędów.

W tym punkcie mówimy o “mikrobłędach”, które choć nie wpływały bezpośrednio na działanie aplikacji to mogły powodować spadki wydajności czy zbyt duże wykorzystywanie pamięci. Łukasz przeanalizował kod linijka i linijce i wyeliminował odnalezione zapachy kodu i poddał je refaktoryzacji.

10.3. Jonasz Świtała.

a) **Projekt interfejsu i schemat bazy.**

Początkową fazą było przygotowanie projektu interfejsu oraz zbioru danych, w którym miały być przechowywane próbki i użytkownicy. Jonasz przygotował mockupy wstępne projektu. Dodał również od siebie schemat bazy danych, w której dokładnie opisał relacje między poszczególnymi tabelami/plikami.

b) **Pobieranie próbek.**

Głównym celem tego punktu było znalezienie źródłowych formatów oraz optymalnych próbek celem wykrywania twarzy. Początkowo testowaliśmy wykrywanie twarzy na różnych wielkościach obrazu. Ostatecznie wybraliśmy wielkość 200x200 oraz po pięć próbek dla każdej osoby.

c) **Implementacja bazy danych.**

Po przygotowaniu schematu bazy, kolejnym krokiem była jej implementacja. Baza została oparta na plikach, które połączone są relacją między sobą. Całą implementacje tej funkcjonalności stworzył Jonasz.

d) **Poprawna interpretacja użytkowników.**

Dużym problemem była również interpretacja wykrytych użytkowników. Algorytm w początkowej fazie często się mylił i należało przetestować dość dokładnie różne warianty jego działania. Zwiększenie jego wrażliwości oraz powiększenie rozmiarów analizowanych zdjęć pomogło w lepszym i wydajniejszym rozpoznawaniu użytkowników.

e) **Lista użytkowników.**

Kolejnym założeniem funkcjonalnym naszego projektu było przygotowanie listy z obecnymi w sali użytkownikami. Użytkowników należało wykryć, następnie sprawdzić i dodać ich na listę obecności. Należało również zaimplementować funkcjonalności i wyświetlanie listy. Wśród funkcjonalności znalazło się m.in. wyczyszczenie czy zapisanie listy do pliku.

f) **Poprawa błędów związanych z bazą danych.**

I w tym aspekcie nie uniknęliśmy błędów implementacji. Poprawa tego punktu głównie polegała na usprawnieniu dodawania do bazy oraz dodaniu unifkacji użytkowników. Zostały wprowadzone również wyjątki, które mają chronić przed niewłaściwym lub niepełnym dodaniem do bazy.

Ponadto każdy z autorów w równomierny sposób pracował nad dokumentacją projektu.

11. Bibliografia.

a) Michał Bereta: Rozpoznawanie twarzy za pomocą sieci neuronowych. Politechnika Krakowska.

b) Adam Kaehler: Komputerowe rozpoznawanie obrazu w C++ przy użyciu biblioteki OpenCV.

c) http://www.emgu.com/wiki/files/2.4.10/document/index.html

d) https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html

e) https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html

f) https://www.researchgate.net/publication/261703721_REAL_TIME_MULTIPLE_FACE_RECOGNITION_SECURITY_SYSTEM_RTM-FS