AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

**WYDZIAŁ ELEKTROTECHNIKI, AUTOMATYKI,
INFORMATYKI I INŻYNIERII BIOMEDYCZNEJ**

KATEDRA INFORMATYKI STOSOWANEJ

Praca dyplomowa inżynierska

Automatyczny detektor ruchu – aplikacja mobilna dla systemu Android
Automatic movement detector – mobile application for Android platform

Autor: *Krzysztof Misztal*
Kierunek studiów: *Informatyka*
Opiekun pracy: *dr inż. Paweł Skrzyński*

Kraków, 2015

Oświadczam, świadomy(-a) odpowiedzialności karnej za poświadczenie nieprawdy, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Serdecznie dziękuję mojej Mamie, dzięki której jestem w miejscu w którym jestem. Bez Jej pomocy niewątpliwie ta praca by nie powstała.

Spis treści

1. Wstęp

1.1. Przedmowa

Ostatnie lata były czasem gwałtownego rozpowszechnienia się smartfonów - urządzeń łączących cechy telefonu komórkowego, oraz komputera osobistego. W roku 2011 przewidywana sprzedaż smartfonów wzrosła o 175% [11]. Urządzenia te posiadają coraz więcej możliwości i są w stanie zastępować m.in. kamery, aparaty cyfrowe, odtwarzacze wideo, konsole do gier i wiele innych. Sam współtwórca wskaźnika *Smartphone Index* na amerykańskiej giełdzie NASDAQ powiedział: "Nieustanna ewolucja smartfonów i tabletów generuje nowe możliwości ich wykorzystania, zwiększając tym samym ich użyteczność i wartość, ale jednocześnie wypierając inne urządzenia z polskich domów, np. aparaty cyfrowe".

Idąc tym tropem pomysł zastąpienia kamery bezpieczeństwa smartfonem wydaje się bardzo trafiony. Rozwiązanie to umożliwia łatwe implementowanie i rozszerzanie funkcjonalności. Dodatkowo ludzie często posiadają swoje stare smartfony, przez co zakup kamery bezpieczeństwa może sprowadzać się do zainstalowania odpowiedniej aplikacji.

Praca ta będzie opisywać to zagadnienie zarówno od strony teoretycznej jak i implementacyjnej.

1.2. Opis dokumentu

Praca ta traktuje o użyciu smartfona jako kamery bezpieczeństwa z zaimplementowanym mechanizmem detekcji ruchu. Opisuje teorię wykrywania ruchu oraz technologię wykorzystaną do implementacji aplikacji na system *Android OS* jak i serwera z którą aplikacja ta będzie wymieniać dane.

W pracy zawarto rozdziały:

Algorytm detekcji ruchu rozdział ten opisuje użyty algorytm wykrywania ruchu, oraz ogarniczenia jakie wprowadza odpalanie algorytmu na smartfonie

Użyte technologie opisane zostały tu wszystkie technologie, biblioteki oraz framework-i wykorzystane w pracy. Uzasadniony został też ich wybór.

Implementacja rozdział opisuje problemy i rozwiązania części implementacyjnej systemu. Opisane zostało też API z którego korzysta aplikacja.

1.3. Cele pracy

Celem pracy jest stworzenie automatycznego detektora ruchu na system *Android OS*. Aplikacja ta będzie w stanie:

- wykrywać ruch, oraz zaznaczać na obrazie fragment sceny gdzie ten ruch został wykryty
- powiadomić użytkownika o wykrytym ruchu poprzez alarm dźwiękowy
- powiadomić serwer o tym, że jest detekcja ruchu jest włączona
- zapisać oraz przesłać na serwer klatki z wykrytym ruchem

Stworzony serwer będzie w stanie przechowywać informacje o stanie aplikacji - o tym czy włączona jest detekcja ruchu. Dodatkowo będzie przechowywał klatki zawierające wykryty ruch. Użytkownik poprzez prostą stronę www będzie w stanie zalogować się oraz oglądać logi z pracy swojego detektora.

2. Algorytm detekcji ruchu

Wybierając algorytm który miałby działać na smartfonie trzeba było uwzględnić dwa znaczące ograniczenia współczesnych urządzeń tego typu:

– mimo ciągle rosnącej mocy obliczeniowej smartfony w dalszym ciągu będą ustępować komputerom stacjonarnym jeżeli chodzi o wydajność. Ponadto założeniem tej pracy jest stworzenie taniego zamiennika dla kamer bezpieczeństwa, tak więc aplikacja powinna działać zadowalająco nawet na starszych urządzeniach

– pomijając kwestie wydajnościowe należy pamiętać o ograniczeniach jakie stawia przed nami czas pracy na baterii. Zbyt długie obciążenie procesora może bardzo szybko wyczerpywać baterię.

Dodatkowo należy zastanowić się jakie warunki powinien spełniać algorytm który będzie wykorzystany w kamerze bezpieczeństwa. Przede wszystkim:

– nie powinien reagować na drobny szum żeby nie wzbudzać niepotrzebnego alarmu, tzw. *false positive*

– nie powinien reagować na gwałtowne zmiany sceny, jak np. padający deszcz, czy zmiana oświetlenia wywołana np.: zapaleniem światła w pomieszczeniu

Biorąc pod uwagę powyższe fakty należy zastosować algorytm możliwie jak najszybszy, który spełni powyższe wymagania. Po przeanalizowaniu dostępnych opcji wybrany został prosty algorytm oparty metodę obliczania różnicy klatek [15].

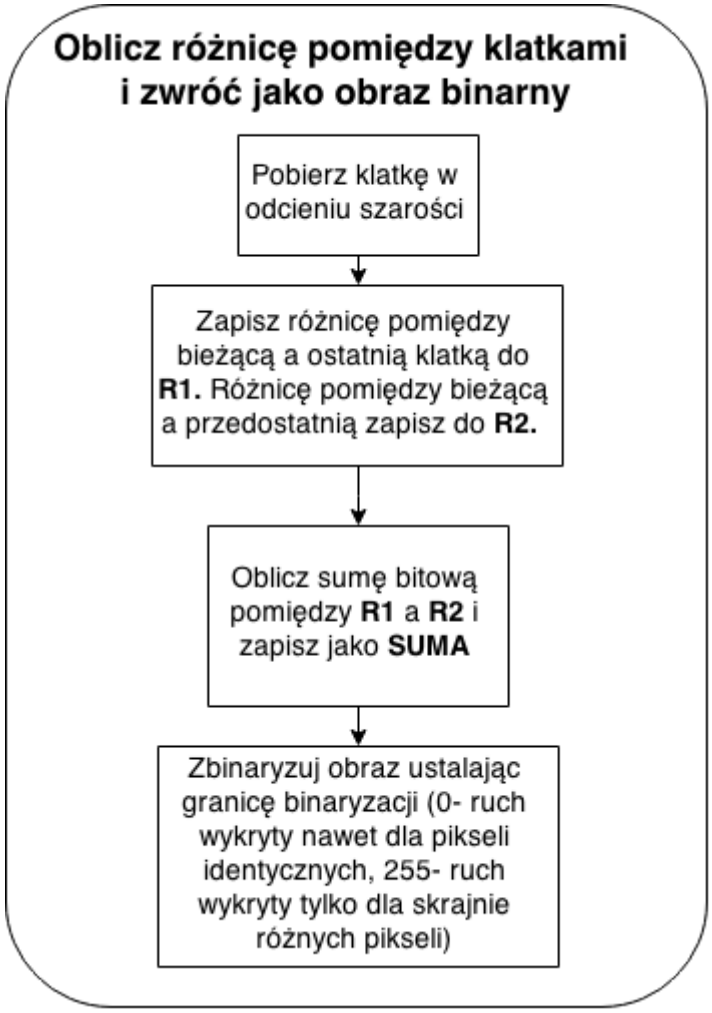
2.1. Opis użytego algorytmu

Tak jak wspomniano w poprzednim rozdziale algorytm bazuje na metodzie obliczenia różnicy klatek przedstawionej w [15]. Metoda została jednak nieznacznie zmodyfikowana w celem uzyskania lepszych rezultatów.

Należy zaznaczyć, że algorytm dostaje obraz w odcieniu szarości.

Algorytm dzieli się na dwa etapy. Pierwszy z nich polega na znalezieniu pikseli które uległy zmianie w stosunku do poprzednich klatek.

Etap pierwszy polega na wyłonieniu pikseli które się zmieniły i dzieli się na poszczególne kroki:



Rysunek 2.1: Pierwszy etap algorytmu

1. Pobranie bieżącej, **n**-tej klatki w odcieniu szarości

2. Obliczenie wartości bezwzględnej z różnicy pomiędzy bieżącą klatką **n**, a klatą ostatnią **n - 1**- zapisujemy jako **R1**. Analogicznie obliczamy różnice pomiędzy **n**-tą klatką, a klatką **n - 2**; różnicę tę zapisujemy jako **R2**

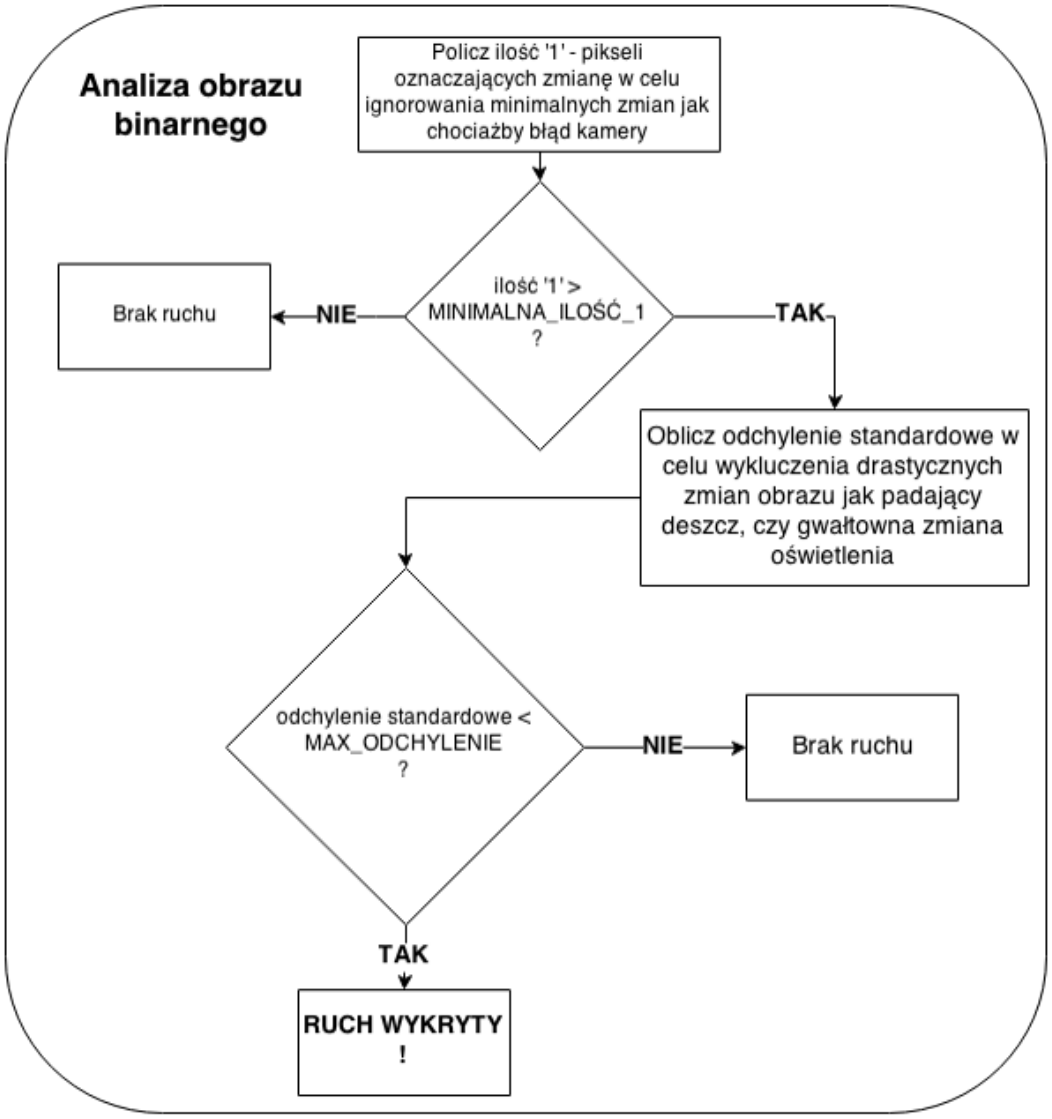
3. Obliczanie sumy bitowej pomiędzy **R1**, a **R2**. Sumę tę zapisujemy jako **S**. Zabieg ten ma na celu wyeliminowanie szumów. Podobny efekt (proponowany w [15]) można uzyskać za pomocą rozmycia Gaussa, w testach jednak rozmycie osiągało gorszą wydajność.

4. Binaryzacja osiągniętej w poprzednim kroku sumie **S**. Operacja ma na celu dostarczyć do następnego etapu jasną, zero - jedynkową informację, które piksele się zmieniły.

   Binaryzacja zastosowana tutaj jest binaryzacją o stałym progu binaryzacji. Empirycznie próg został ustalony na wartość **35**. Oznacza to, że piksele w **S** których wartość jest większa bądź równa **35** zostają uznane za zmienione i oznaczone w wynikowym obrazie binarnym jako **1**.

Im mniejszy próg binaryzacji tym algorytm potrzebuje mniejszej zmiany pikseli do uznania ich za zmienione. W skrajnej sytuacji przy progu = 0 nawet identyczne piksele zostają uznane za zmienione, a algorytm jest zbyt czuły. W sytuacji odwrotnej (próg = 255) algorytm jest nieczuły na zmiany - do uznania pikseli za zmienione potrzebuje zmiany wartości z 0 na 255.

W etapie drugim algorytm podejmuje decyzję czy otrzymany z poprzedniego etapu obraz binarny uznać za ruch czy nie. Składa się on z następujących kroków:



Rysunek 2.2: Drugi etap algorytmu

1. Liczenie zmienionych pikseli = 1. Trzeba zabezpieczyć się przed sytuacją kiedy mała ilość zmienionych pikseli sprawia, że algorytm uznaje je za ruch. Użytkownik może w opcjach aplikacji sam zmieniać próg który uznawany jest za *minimalną ilość zmienionych pikseli*. Jest to minimalny próg który musi być przekroczony, alby algorytm dalej dokonywał analizę klatki.

2. Liczenie odchylenia standardowego. Jest to krok który użytkownik może wyłączyć.

   Bazując na [2] odchylenie standardowe jest to pierwiastek kwadratowy z wariancji zmiennej losowej

   $\sigma(X) = \sqrt{D^2X} = \sqrt{E(X - EX)^2}$

   Krok ten zabezpiecza przed sytuacją kiedy gwałtowna zmiana całej sceny (jak np.: zapalenie światła w pokoju) uznawana zostaje za ruch (*false positive*). Im odchylenie standardowe większe tym większa ogólna zmiana całego obrazu. Algorytm uzna obraz za ruch w sytuacji kiedy odchylenie standardowe będzie mniejsze niż *maksymalne odchylenie standardowe*. Próg ten użytkownik może zmieniać w ustawieniach aplikacji.

Jeżeli warunki przedstawione na etapie drugim zostają spełnione algorytm uznaje że "coś się rusza". Dalsze zachowanie jest niezależne od detekcji ruchu i jest możliwe do zdefiniowanie przez użytkownika. Możliwe zachowania są opisane w rozdziale 4.2

2.2. Możliwe ulepszenia algorytmu

Algorytm zaimplementowany w pracy oczywiście może być ulepszany. [15] proponuje wiele ulepszeń i poprawek w algorytmie zwiększając poprawność działania algorytmu. Mając jednak na uwadze argumenty wymienione w 2 większą wagę ma szybkość i lekkość działania. Średnia dokładność jest wystarczająca i jej zwiększanie nie jest priorytetem.

Implementacja różnych wariantów algorytmu nie jest zawiera się w celach tej pracy. Należy jednak wymienić możliwe udoskonalenia. Można je zaimplementować jako opcjonalne zachowania (np. dla bardziej wydajnych urządzeń). Jeżeli projekt będzie rozwijany z pewnością będzie to dobry "pierwszy krok".

Przechodząc do rzeczy, bazując na [15] dobrym pomysłem będzie:

– w drugim kroku drugiego etapu opisanego w 2.1 zamiast liczyć dwie różnic można zastosować filtr uśredniający. Jest to rozwiązanie które nieznacznie lepiej eliminuje szumy przy nieznacznym zmniejszeniu prędkości

– można użyć algorytmu indeksacji w celu oddzielenia od siebie poruszających się algorytmów. Jest to zabieg zbędny jeżeli naszym celem jest dostanie binarnej informacji: jest ruch-nie ma ruchu

– osiągnięty obraz binarny można poddać operacji koniunkcji z wynikiem algorytmu Canny. Wykrywa on krawędzie. Zabieg ten bardzo dobrze radzi sobie z szumem spowodowanym przez światło.

  Zabieg ten znacząco zmniejsza prędkość działania algorytmu, dlatego jego użycie powinno być możliwe do wyłączenia przez użytkownika.

3. Użyte technologie

Tak jak opisano w sekcji 1.3 cele pracy będzie zaimplementowanie detektora ruchu na system Android, który będzie łączył się z serwerem i przekazywał tam pewne informacje o swojej pracy. Nie trudno wyciągnąć wniosek, że do pracy potrzebne będą następujące komponenty:

1. urządzenie z system Android wyposażone w kamerę plus Android SKD
2. serwer na którym będzie "postawiona" aplikacja z którą komunikować się będzie smartfon. Wybór padł na Amazon AWS
3. system do zarządzania bazami danych
4. ponieważ aplikacja webowa będzie pisana w języku Java przydatny staje się serwer aplikacji. Wybrany został Apache Tomcat.

Komunikacja pomiędzy smartfonem a aplikacją webową będzie odbywać się przy użyciu architektury REST oraz formatu danych JSON.

3.1. Android OS

Android jest system operacyjnym o jądro Linuxa. Jest to najpopularniejszy system operacyjny na urządzenia mobilne (smartfony i tablety).

W kontekście tej pracy najważniejszą rzeczą o której trzeba powiedzieć jest fakt, iż natywne aplikacje na ten system tworzy się w używając Android SDK który udostępnia API do języka Java. Dokumentacja znajduje się [1].

Jest to bardzo ważne ponieważ kod tworzony przez developerów, tak samo jak przy korzystaniu ze "zwykłej" Javy, jest kompilowany do kodu bajtowego. Kod ten nie jest rozumiany przez procesor, a do jego uruchomienia potrzebna jest maszyna wirtualna. W przypadku Androida jest to Dalvik, który został zastąpiony w najnowszej wersji systemu Android Lolipop maszyną wirtualną ART.

Kod uruchamiany w maszynie wirtualnej jest wolniejszy od kodu maszynowego. W przypadku prostych użytkowych aplikacji, które nie wykonują skomplikowanych obliczeń fakt ten nie przeszkadza istotnie, jednak kiedy piszemy program który wykonuje dużo obliczeń sprawa się komplikuje. Kod wykonuje się dużo wolniej i zużywa więcej pamięci. Z tego powodu implementowanie algorytmu detekcji

ruchu w Javie na urządzeniach mobilnych mija się z celem. Takie rozwiązanie jest całkowicie bezużyteczne. Na szczęście inżynierowie odpowiadający za Androida przewidzieli taką sytuację i udostępnili coś co nazywa się NDK - *Native Development Kit*. Jest to zestaw narzędzi umożliwiających programowanie urządzeń z Androidem w języku C/C++. NDK umożliwia (a nawet wymaga) integracje z SDK.

W praktyce NDK używa się do programowania tylko tych części aplikacji które są krytyczne pod względem wydajnościowym. W przypadku tej pracy - części aplikacji analizującej kolejne klatki z aparatu.

Trzeba też zaznaczyć że programowanie z użyciem NDK jest znacznie mniej wygodne niż SDK. Na szczęście istnieje biblioteka *OpenCV - Open source Computer Vision* [6]. Jest ona przeznaczona dla systemów przetwarzających obrazy. Udostępnia API Javo-we i jest przygotowana do pracy z system Android. Biblioteka ta została szerzej opisana w sekcji 4.2.1

3.2. Baza danych: MySQL

MySQL [12] jest serwerem oraz systemem zarządzania relacyjnymi bazami danych. Dostęp do bazy danych uzyskujemy poprzez język zapytań bazy danych SQL. MySQL jest rozwijany przez firmę Oracle.

Relacyjna baza danych jest niczym innym jak zbiorem relacji. Przedstawiając sprawę jak najprościej można powiedzieć, że relacja jest dwuwymiarową tabelą złożoną z kolumn i wierszy, gdzie liczba kolumn jest z góry określona. Każdą kolumnę określa nazwa, oraz dziedzina. Na przecięciu wiersza i kolumny znajduje się pojedyncza wartość należąca do dziedziny kolumny. Wiersze w relacji są unikalne, tj. nie nie występuje taka para wierszy że wartości dla każdej z kolumn są takie same. Kolejność wierszy i kolumn jest nieistotna.

3.3. Serwer aplikacji: Tomcat

Ponieważ aplikacja serwerowa pisana jest w języku Java EE niezbędny jest serwer aplikacji. *Apache Tomcat* [9] idealnie się do tego nadaje. Tomcat jest otwarto-źródłową implementacją technologii *Java Servlet*, oraz *JavaServer Pages*.

3.4. Forma komunikacji: REST i Jersey

REST - Representational State Transfer [7] - jest to abstrakcyny wzorzec architektury oprogramowania opisujący sposób komunikacji aplikacji rozproszonych. Jest bezstanowy, to znaczy, że klient musi dostarczyć wszelkich niezbędnych informacji do realizacji zapytania.

JAX-RS: Java API for RESTful Web Services jest API Javowym zapewniającym wsparcie w tworzeniu usług internetowych (*web service*) zgodnych z wzrocem REST. *JAX-RS* jest częścią Javy EE 6.

Rest Jersey [4] jest otwarto-źródłowym framework'iem wspomagającym tworzenie webservice'ów RESTowych. *Jersey* w pełni wspiera *JAX-RS*.

W pracy aplikacja kliencka komunikuje się z serwerem przy użyciu formatu *JSON* [5]. Jest to lekki, tekstowy i nadający się do czytania przez człowieka format wymiany danych. Przykładowe dane przekazywane w tym formacie:

```
{"obiad": {
  "kalorie": 800,
  "godzina": "14:00",
  "składniki": [
    {
        "nazwa": "schabowy",
        "kalorie": 600
    },
    {
        "nazwa": "ziemniaki",
        "kalorie": 200
  ]
}}
```

4. Implementacja

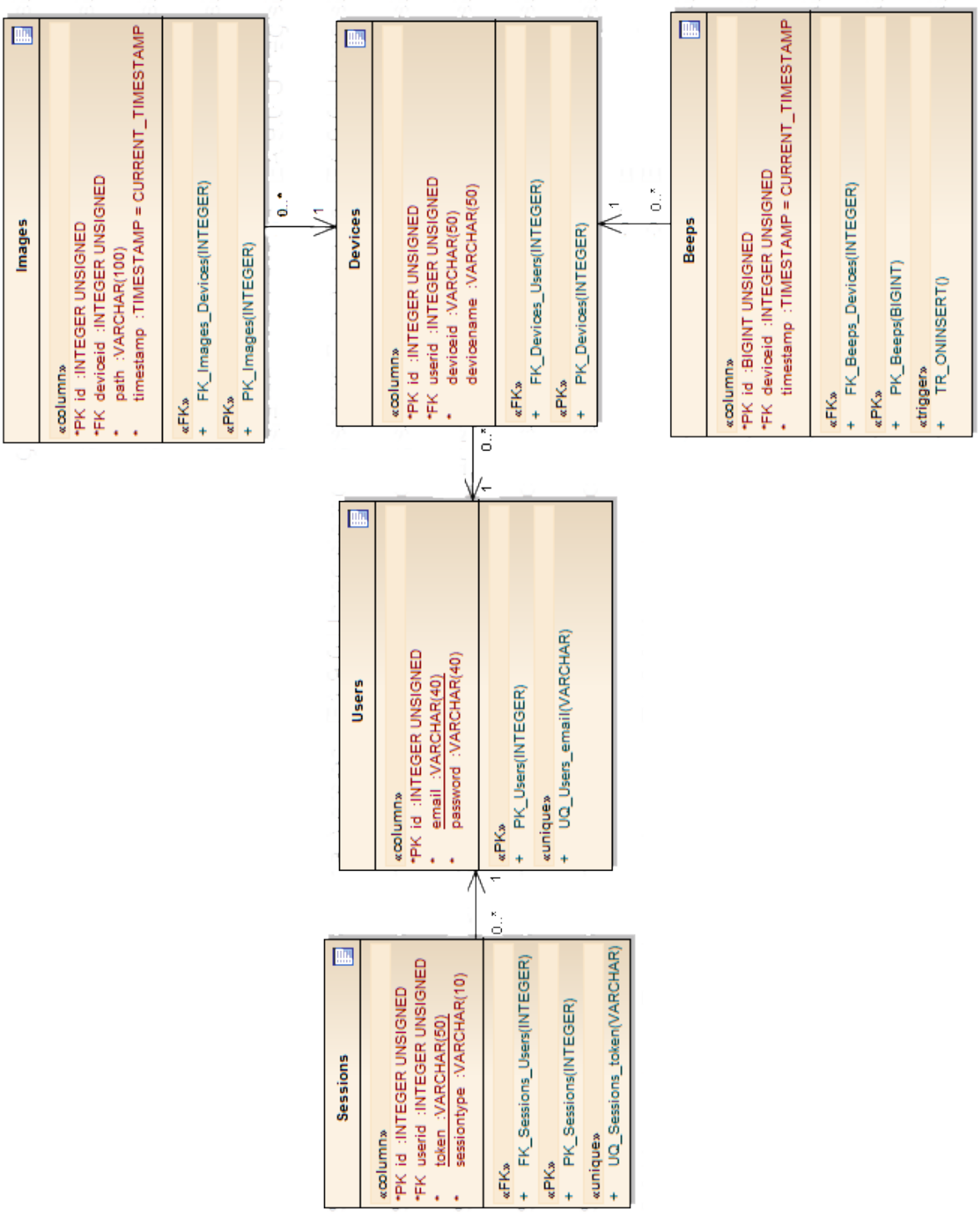
Rozdział ten traktuje o tym w jaki sposób praktycznie zaimplementowano wszystkie funkcjonalności systemu - zostały przedstawione tu kluczowe elementy systemu niezbędne do jego działania.

4.1. Serwer

Sekcja ta dotyczy aplikacji serwerowej, a więc elementu systemu, który pobiera dane od detektorów, odpowiednio je przechowuje oraz umożliwia ich pobieranie upoważnionym użytkownikom.

4.1.1. Baza danych

W pracy użyto relacyjnej bazy danych MySQL [12]. W sekcji 5.3 załączono skrypt którego użyto do założenia bazy danych. Poniżej przedstawiony jest schemat użytej w systemie bazy danych:



Rysunek 4.1: Baza danych użyta na serwerze

4.1.2. RESTful API

Aplikacja serwerowa wystawia REST-owe API, które umożliwia pełną obsługę systemu. Cechuje się możliwością:

- rejestracji nowego użytkownika
- logowania z rozpoznaniem typu aplikacji która wykonuje logowanie (detektor, czy np. klient www)
- wylogowania (kończenia sesji)
- powiadomienia serwera przez detektor o fakcie wykrywania ruchu
- ustawienia nazwy dla detektora ruchu celem łatwiejszego rozpoznawania urządzenia przez użytkownika
- wysyłania zdjęć przez detektor na serwer
- pobrania wszystkich urządzeń podłączonych pod użytkownika
- sprawdzenia czy dane urządzenie aktualnie pracuje, jest online i wykrywa ruch
- pobrania zdjęć wysłanych przez dane urządzenie

W sekcji 5.2 umieszczony jest fragment kodu bezpośrednio odpowiadający za udostępnienie usług webowych. Aplikacja została napisana w języku Java z wykorzystaniem biblioteki [4]. Do podłączenia się do bazy danych frameworka Hibernate [3]

Poniżej znajduje się tabela przedstawiająca typy zapytań, zwracane wartości, oraz potrzebne argumenty do ich wywołania.

Użytkownicy

	POST
URL	/users/register
Zwracany typ	application/json
Nagłówek	**api_key**: string
Parametry	**email**: string, **password**: string [SHA-256]
Opis	Rejestruje w systemie nowego użytkownia
SUKCES	**200**
PORAŻKA	**401**: zły API key, **406** użytkownik o podanym adresie email istnieje, lub podano nieprawidłowe dane, **ErrorJSON**

	POST
URL	/users/login
Zwracany typ	application/json
Nagłówek	**api_key**: string
Parametry	**email**: string, **password**: string [SHA-256]
Opis	Otwiera nową sesję dla aplikacji klienckiej (wyświetlającej dane)
SUKCES	**200**: {"Token": string}
PORAŻKA	**401**: zły API key, **406** niepoprawne dane logowania, **ErrorJSON**

	DELETE
URL	/users/logout
Zwracany typ	Kod odpowiedzi HTTP
Nagłówek	**api_key**: string, **token**: string
Parametry	
Opis	Zamyka sesje
SUKCES	**200**
PORAŻKA	**401**: zły API key lub niewłaściwy token

API dla detektora ruchu

	POST
URL	/detector/login
Zwracany typ	application/json
Nagłówek	**api_key**: string
Parametry	**email**: string, **password**: string [SHA-256], **device_id**: string
Opis	Otwiera sesję dla detektora ruchu. Jeżeli datektor o podanym device_id nie istnieje tworzony jest odpowiedni wpis w bazie danych
SUKCES	**200**: {"Token": string}
PORAŻKA	**401**: zły API key, **406** niepoprawne dane logowania, **ErrorJSON**

	POST
URL	/detector/beep
Zwracany typ	Kod odpowiedzi HTTP
Nagłówek	**api_key**: string, **token**: string
Parametry	**device_id**: string
Opis	Dodaje wpis o tym, że detektor działa
SUKCES	**200**
PORAŻKA	**401**: zły API key, lub niewłaściwy token, **404** urządzenie nie znalezione

	POST
URL	/detector/devicename
Zwracany typ	application/json
Nagłówek	**api_key**: string, **token**: string
Parametry	**device_id**: string, **name**: string
Opis	Ustawia nazwę dla urządzenia
SUKCES	**200**
PORAŻKA	**401**: zły API key, lub niewłaściwy token, **404** urządzenie nie znalezione

	POST
URL	/detector/photo
Konsumuje	mediapart/form-data
Zwracany typ	Kod odpowiedzi HTTP
Nagłówek	**api_key**: string, **token**: string
Parametry	**device_id**: string, **photo**: file
Opis	Wysyła zdjęcie oznaczające ruch
SUKCES	**200**
PORAŻKA	**401**: zły API key, lub niewłaściwy token, **404** urządzenie nie znalezione

API dla klienta

	GET
URL	/client/devices
Zwracany typ	application/json
Nagłówek	**api_key**: string, **token**: string
Parametry	
Opis	Zwraca listę wszystkich detektorów ruchu użytkownika
SUKCES	**200**: JSON {"Devices": [{"DeviceId": string, "DeviceName": string}, ...]}
PORAŻKA	**401**: zły API key, lub niewłaściwy token, **404** użytkownik nie znaleziony

	GET
URL	/client/isonline
Zwracany typ	application/json
Nagłówek	**api_key**: string, **token**: string
Parametry	**device_id**: string
Opis	Sprawdza czy detektor jest OnLine
SUKCES	**200**: JSON {“Online”: boolean}
PORAŻKA	**401**: zły API key, lub niewłaściwy token, **404** urządzenie nie znalezione

	GET
URL	/client/iamges
Zwracany typ	application/json
Nagłówek	**api_key**: string, **token**: string
Parametry	**device_id**: string
Opis	Zwraca listę zdjęć wysłanych przez urządzenie
SUKCES	**200**: JSON {"Images": [{"Id": integer, "Date": string}, ...]}, gdzie Id oznacza Id zdjęcia w strumieniu zdjęć
PORAŻKA	**401**: zły API key, lub niewłaściwy token, **404** urządzenie nie znalezione, ErrorJSON

	GET
URL	/client/fetch/{id}
Zwracany typ	image/jpg
Nagłówek	**api_key**: string, **token**: string
Parametry	**device_id**: string, **{id}**: integer
Opis	Zwraca zdjęcie o {id} wysłane przez urządzenie o danym device_id
SUKCES	**200**: image/jpg
PORAŻKA	**401**: zły API key, lub niewłaściwy token, **404** urządzenie nie znalezione, lub zdjęcie nie znalezione, ErrorJSON

Poniżej przedstawiona jest forma wiadomości ErrorJSON:

```
ErrorJSON:
    {
        "OperationType": string,
        "ErrorMessage": string
    }
```

4.2. Aplikacja

Do implementacji aplikacji na system Android użyto IDE Android Studio w wersji 1.0.0, oraz systemu do automatycznego budowania aplikacji Gradle. Do działania wymaga wersji 4.0 lub wyższej systemu Android. Najważniejsze fragmenty kodu załączono w sekcji 5.1.

Aplikacja cechuje się następującymi funkcjonalnościami:

– Wykrywania ruchu oraz prezentacji jego wyników w trybie: kolorowym, odcieniu szarości, oraz binarnym

– Łączenia się ze zdalnym serwerem, oraz przesyłania mu informacji o tym czy wykrywanie jest włączone

– Wysyłania na serwer zdjęć zawierających klatki z wykrytym ruchem

– Zaznaczania, oraz wycinania obszaru ruchu z klatki, oraz jego zapisu na kartę pamięci

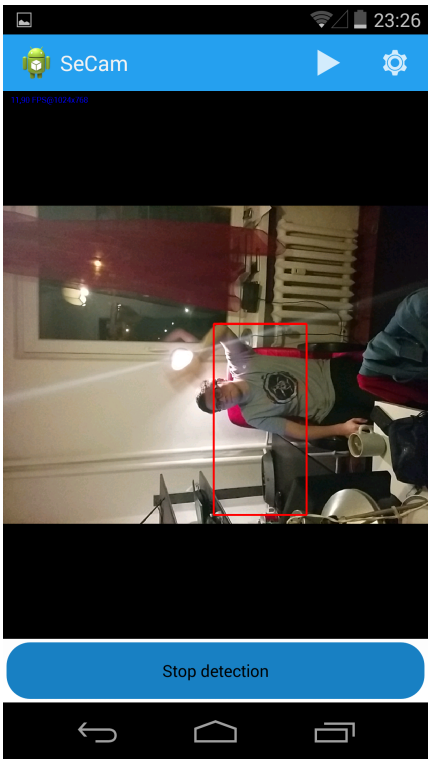
– Włączenia alarmu głosowego w razie wykrytego ruchu

– Ustalenia parametrów algorytmu, które zostały opisane w sekcji 2.1

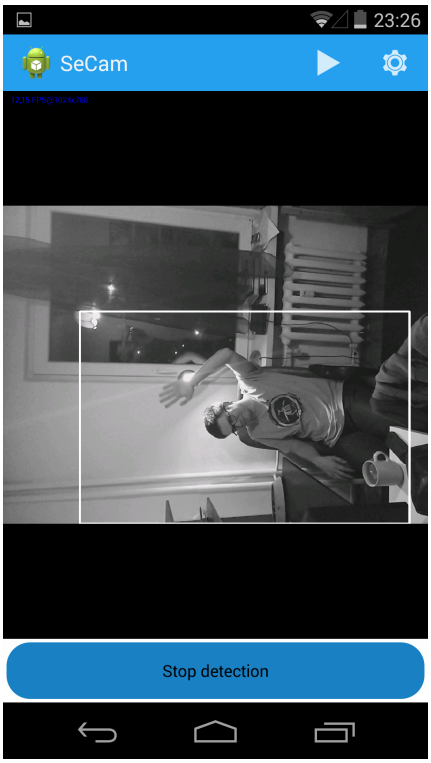
Aplikacja do wykonania obliczeń na obrazach wykorzystuje bibliotekę OpenCV, napisaną w C++. Dla zwiększenia OpenCV potrzebuje bibliotek napisanych pod konkretny hardware. Z tego względu wymagane jest pobranie dodatkowej aplikacji. Proces ten wykonywany jest przy pierwszym uruchomieniu detektora ruchu.

Po naciśnięciu przycisku detektor rozpoczyna analizę kolejnych klatek w czasie rzeczywistym, zgodnie z algorytmem opisanym w sekcji 2.1. Proces ten jest jednowątkowy, mimo to nawet na słabszych urządzeniach otrzymano zadowalające rezultaty 13fps.

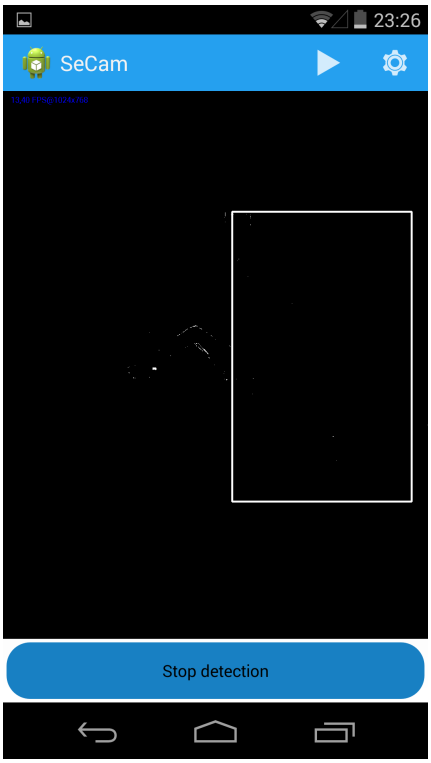
Poniżej przedstawiono zrzuty ekranu:



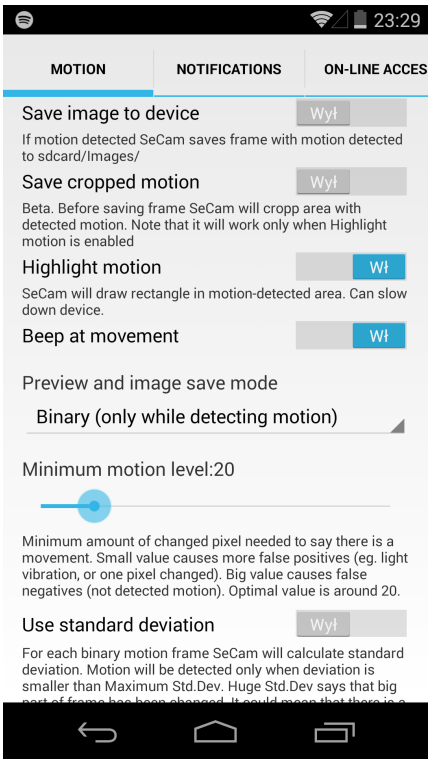
(a) Wykrywanie ruchu w trybie RGB



(b) Wykrywanie ruchu w trybie GRAY



(c) Wykrywanie ruchu w trybie binarnym



(d) Zrzut ekranu części opcji programu

Rysunek 4.2: Zrzuty ekrany działającej aplikacji

4.2.1. Użyte biblioteki zewnętrzne

Przy tworzeniu aplikacji wykrywającej ruch na system Android użyto kilku bibliotek, jeżeli nie napisano inaczej udostępnionych na licencji Apache 2.0 [8]. Oto one:

OPEN CV [6], wersja 2.4.9

Open Source Computer Vision jako jedyna z wykorzystanych bibliotek udostępniona na licencji BSD. OpenCV udostępnia wiele narzędzi z zakresu przetwarzania obrazów. Została napisana przy użyciu Android NDK w języku C++ dzięki czemu działa ona znacznie szybciej. OpenCV wymaga zainstalowania na Smartfonie binarnych plików dla konkretnego hardware'u, jednak aplikacja stworzona w ramach tej pracy przeprowadza użytkownika przez cały proces.

Retrofit [14], wersja 1.8.0

Retrofit jest lekkim klientem REST'a dla Androida. Umożliwia zarówno synchroniczne jak asynchroniczne wywołania. Retrofit zamienia RESTful API w interfejs, oraz używa biblioteki GSON do automatycznego parsowania odpowiedzi serwera. Zastosowanie biblioteki załączono w sekcji **??**

GSON [10], wersja 2.3.1

GSON jest biblioteką która automatycznie przekształca String w formie JSON [5] na obiekt Javowy.

ButterKnife [13]

ButterKnife jest małą bibioteką która za pomocą *Java annotations* ułatwia odnajdowanie i programowanie widoków

5. Dodatek A - fragmenty kodów źródłowych

5.1. Android

5.1.1. Klient RESTowy

Kod klienta

```
package pl.edu.agh.misztal.secam.rest;

import pl.edu.agh.misztal.secam.rest.callback.LoginCallback;
import pl.edu.agh.misztal.secam.rest.model.EmptyEntity;
import pl.edu.agh.misztal.secam.rest.model.Login;
import retrofit.Callback;
import retrofit.http.DELETE;
import retrofit.http.Field;
import retrofit.http.FormUrlEncoded;
import retrofit.http.Header;
import retrofit.http.Multipart;
import retrofit.http.POST;
import retrofit.http.Part;
import retrofit.mime.TypedFile;

public interface SecamApi {

    @FormUrlEncoded
    @POST("/detector/login")
    void login(@Header("api_key") String apiKey, @Field("email") String
        email, @Field("password") String password, @Field("device_id")
        String deviceId, Callback<Login> callback);

    @DELETE("/users/logout")
    void logout(@Header("api_key") String apiKey, @Header("token") String
        token, Callback<EmptyEntity> callback);

    @FormUrlEncoded
```

```
    @POST("/detector/devicename")
    void setDevicename(@Header("api_key") String apiKey, @Header("token")
        String token, @Field("device_id") String deviceId, @Field("name")
        String name, Callback<EmptyEntity> callback);

    @FormUrlEncoded
    @POST("/detector/beep")
    void beep(@Header("api_key") String apiKey, @Header("token") String
        token, @Field("device_id") String deviceId, Callback<EmptyEntity>
        callback);

    @Multipart
    @POST("/detector/photo")
    void postPhoto(@Header("api_key") String apiKey, @Header("token")
        String token, @Part("device_id") String deviceId,
        @Part("photo")TypedFile photo, Callback<EmptyEntity> callback);

}
```

Tworzenie obiektu będącego klientem:

```
// Tworzenie GSON'a, tak aby poprawnie automatycznie konwertował wiadomość
    JSON na obiekt POJO ({\it Plain Old Java Object})
Gson gson = new GsonBuilder()
                .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
                .create();

              // Ustawienie konwertera i adresu serwera
          RestAdapter adapter = new RestAdapter.Builder()
                .setConverter(new GsonConverter(gson))
                .setEndpoint(ENDPOINT)
                .build();

          SecamApi api = adapter.create(SecamApi.class);
```

5.1.2. Wykrywanie ruchu

```
curr.copyTo(prev);
next.copyTo(curr);
inputFrame.gray().copyTo(next);
```

```
binarizeMovement();

// (...)

if(isMovement())
        reactMovement(in);
```

Po wykoananiu *binarizeMovement()* w zmiennej *result* mamy binarną macierz z zanzaczonymi pikselami które się zmieniły.

Oto kod funkcji *isMovement()* stwierdzającej czy ruch wystąpił:

```
private boolean isMovement() {
    //sprawdź czy jest minimalna ilość zmienionych pikseli
    if(result != null && Core.countNonZero(result) >
       preferences.getMinimumMovement()) {
        if(preferences.isUseStdDev()) { //czy bierzemy pod uwagę odchylenie
           standardowe ?
            MatOfDouble mean = new MatOfDouble();
            MatOfDouble stddev = new MatOfDouble();
            Core.meanStdDev(result, mean, stddev);

            if(stddev.get(0, 0)[0] < preferences.getMaxStdDev())
                return true;
            else
                return false;
        }
        return true;
    }

    return false;
}
```

5.2. Serwer API

Implementacja klasy odpowiadającej za udostępnienie API

```
@Path("v1.0")
public class Api {

    private static final String apiKey = "asd";
```

```
private static final int MINUTES_ONLINE = 5; //minutes in the past that
    beeps is considered to be online

private DBController db = new DBController();
private String baseImagePath = "/var/www/images/secam/";
//private String baseImagePath = "F:/SeCam/";
private Gson gson = new GsonBuilder()
    .setFieldNamingPolicy(FieldNamingPolicy.UPPER_CAMEL_CASE)
    .excludeFieldsWithoutExposeAnnotation()
    .create();

// USERS ============================================================

@POST
@Path("users/register")
public Response register(@HeaderParam("api_key") String apiKey,
    @FormParam("email") String email, @FormParam("password") String
    password) {
    Response ret;
    if(!apiKey.equals(Api.apiKey)) {
        ret = Response.status(Response.Status.UNAUTHORIZED).build();
    } else {
        boolean ok = db.newUser(email, password);
        if(ok) {
            ret = Response.ok().build();
        } else {
            Error error = new Error("User register", "User exists");
            ret =
                Response.status(Response.Status.NOT_ACCEPTABLE).entity(gson.toJson(error
        }
    }
    return ret;
}

@POST
@Produces(MediaType.APPLICATION_JSON)
@Path("/users/login")
public Response login(@HeaderParam("api_key") String apiKey,
    @FormParam("email") String email, @FormParam("password") String
    password) {
    Response ret;
    if(!apiKey.equals(Api.apiKey)) {
        ret = Response.status(Response.Status.UNAUTHORIZED).build();
```

```
        } else {
            String token = db.newSession(email, password, "client");
            if(token != null) {
                ret = Response.ok(gson.toJson(new Token(token))).build();
            } else {
                Error error = new Error("User login", "Incorrect
                    credentials");
                ret =
                    Response.status(Response.Status.NOT_ACCEPTABLE).entity(gson.toJson(
            }
        }

        return ret;
    }

    @DELETE
    @Path("users/logout")
    public Response logout(@HeaderParam("api_key") String apiKey,
        @HeaderParam("token") String token) {
        Response ret;

        if(!apiKey.equals(Api.apiKey)) {
            ret = Response.status(Response.Status.UNAUTHORIZED).build();
        } else {
            boolean ok = db.deleteSession(token);

            if(ok)
                ret = Response.ok().build();
            else
                ret = Response.status(Response.Status.UNAUTHORIZED).build();
        }



        return ret;
    }

    // DETECTOR ============================================================


    @POST
    @Produces(MediaType.APPLICATION_JSON)
    @Path("/detector/login")
```

```
public Response detectorLogin(@HeaderParam("api_key") String apiKey,
    @FormParam("email") String email, @FormParam("password") String
    password,
        @FormParam("device_id") String deviceId) {
    Response ret;
    if(!apiKey.equals(Api.apiKey)) {
        ret = Response.status(Response.Status.UNAUTHORIZED).build();
    } else {
        String token = db.newSession(email, password, "detector");
        if(token != null) {
            int uid = db.getUserIdByToken(token);
            if(db.getDeviceIdByImei(deviceId, uid) < 0) { //dodaj
                urządzenie jeżeli nie istnieje
                boolean ok = db.newDevice(uid, deviceId);
                if(ok)
                    ret = Response.ok(gson.toJson(new
                        Token(token))).build();
                else
                    ret =
                        Response.status(Response.Status.INTERNAL_SERVER_ERROR).build();
            }
            else
                ret = Response.ok(gson.toJson(new Token(token))).build();
        } else {
            Error error = new Error("User login", "Incorrect
                credentials");
            ret =
                Response.status(Response.Status.NOT_ACCEPTABLE).entity(gson.toJson(error
        }
    }

    return ret;
}

@POST
@Path("detector/beep")
public Response beep(@HeaderParam("api_key") String apiKey,
    @HeaderParam("token") String token, @FormParam("device_id") String
    deviceId) {
    Response ret;
    int uid = db.getUserIdByToken(token);

    if(!apiKey.equals(Api.apiKey) || uid < 0) {
```

```
            ret = Response.status(Response.Status.UNAUTHORIZED).build();
        } else {
            int _deviceid = db.getDeviceIdByImei(deviceId, uid);
            if(_deviceid >= 0) {
                boolean ok = db.beep(_deviceid);
                if(ok)
                    ret = Response.ok().build();
                else
                    ret =
                        Response.status(Response.Status.INTERNAL_SERVER_ERROR).build();
            } else
                ret = Response.status(Response.Status.NOT_FOUND).build();
        }

        return ret;
    }

    @POST
    @Path("detector/devicename")
    public Response setDeviceName(@HeaderParam("api_key") String apiKey,
        @HeaderParam("token") String token, @FormParam("device_id") String
        deviceId,
            @FormParam("name") String name) {
        Response ret;

        int uid = db.getUserIdByToken(token);

        if(!apiKey.equals(Api.apiKey) || uid < 0) {
            ret = Response.status(Response.Status.UNAUTHORIZED).build();
        } else {
            int _deviceid = db.getDeviceIdByImei(deviceId, uid);
            if(_deviceid < 0)
                ret = Response.status(Status.NOT_FOUND).build();
            else {
                boolean ok = db.setDeviceName(_deviceid, name);
                if(ok)
                    ret = Response.ok().build();
                else
                    ret =
                        Response.status(Status.INTERNAL_SERVER_ERROR).build();
            }
        }
```

```
        return ret;
    }

    @POST
    @Path("detector/photo")
    @Consumes(MediaType.MULTIPART_FORM_DATA)
    public Response postPhoto(@HeaderParam("api_key") String apiKey,
        @HeaderParam("token") String token, @FormDataParam("device_id")
        String deviceId,
            @FormDataParam("photo") InputStream fileInputStream) {

        Response ret;

        int uid = db.getUserIdByToken(token);

        if(!apiKey.equals(Api.apiKey) || uid < 0) {
            ret = Response.status(Response.Status.UNAUTHORIZED).build();
        } else {
            int _deviceid = db.getDeviceIdByImei(deviceId, uid);
            if(_deviceid < 0)
                ret = Response.status(Status.NOT_FOUND).build();
            else {
                File dir = new File(baseImagePath + Integer.toString(uid) +
                    "/" + Integer.toString(_deviceid));
                dir.mkdirs();
                int imageCount = db.getImages(_deviceid).size();
                String filepath = dir.getAbsolutePath() + "/" +
                    Integer.toString(imageCount) + ".jpg";

                if(writeToFile(fileInputStream, filepath)){
                    boolean ok = db.addImage(_deviceid, filepath);
                    if(ok)
                        Response.ok().build();
                    else
                        Response.status(Status.INTERNAL_SERVER_ERROR);
                }
                ret = Response.ok().build();
            }
        }

        return ret;

    }
```

```
// CLIENT ============================================================

@GET
@Path("client/devices")
@Produces(MediaType.APPLICATION_JSON)
public Response getDevices(@HeaderParam("api_key") String apiKey,
    @HeaderParam("token") String token) {
    Response ret;

    int uid = db.getUserIdByToken(token);

    if(!apiKey.equals(Api.apiKey) || uid < 0) {
        ret = Response.status(Response.Status.UNAUTHORIZED).build();
    } else {
        List<Devices> list = db.getDevices(uid);
        if(list != null) {
            DevicesList devicesList = new DevicesList(list);
            ret = Response.ok(gson.toJson(devicesList)).build();

        } else
            ret = Response.status(Status.NOT_FOUND).build();
    }

    return ret;

}

@GET
@Path("client/isonline")
@Produces(MediaType.APPLICATION_JSON)
public Response isOnline(@HeaderParam("api_key") String apiKey,
    @HeaderParam("token") String token, @QueryParam("device_id") String
    deviceId) {
    Response ret;

    int uid = db.getUserIdByToken(token);

    if(!apiKey.equals(Api.apiKey) || uid < 0) {
        ret = Response.status(Response.Status.UNAUTHORIZED).build();
    } else {
        Beeps b = db.getBeeps(deviceId, uid);
        IsOnline online;
```

```
            if(b != null) {
                if(isOnline(b.getTimestamp()))
                    online = new IsOnline(true);
                else
                    online = new IsOnline(false);

                ret = Response.ok(gson.toJson(online)).build();
            } else
                ret = Response.status(Status.NOT_FOUND).build();
        }

        return ret;

    }


    @GET
    @Path("client/images")
    @Produces(MediaType.APPLICATION_JSON)
    public Response getImages(@HeaderParam("api_key") String apiKey,
        @HeaderParam("token") String token, @QueryParam("device_id") String
        deviceId) {
        Response ret;

        int uid = db.getUserIdByToken(token);

        if(!apiKey.equals(Api.apiKey) || uid < 0) {
            ret = Response.status(Response.Status.UNAUTHORIZED).build();
        } else {
            int _deviceid = db.getDeviceIdByImei(deviceId, uid);
            if(_deviceid < 0)
                ret= Response.status(Status.NOT_FOUND).build();
            else {
                List<Images> list = db.getImages(_deviceid);
                ImagesList images = new ImagesList(list);

                ret = Response.ok(gson.toJson(images)).build();
            }
        }


        return ret;
```

```
}

@GET
@Path("client/images/fetch/{id}")
@Produces("image/jpeg")
public Response getImage(@HeaderParam("api_key") String apiKey,
    @HeaderParam("token") String token, @PathParam("id") String _id,
    @QueryParam("device_id") String deviceId) {
    Response ret;

    int uid = db.getUserIdByToken(token);

    if(!apiKey.equals(Api.apiKey) || uid < 0) {
        ret = Response.status(Response.Status.UNAUTHORIZED).build();
    } else {
        try {
            int id = Integer.parseInt(_id);
            int _deviceid = db.getDeviceIdByImei(deviceId, uid);
            if(_deviceid < 0)
                ret = Response.status(Status.NOT_FOUND).build();
            else {
                String filepath = baseImagePath + "/" +
                    Integer.toString(uid) + "/" +
                    Integer.toString(_deviceid) + "/"
                        + Integer.toString(id) + ".jpg";

                File file = new File(filepath);

                if(file.exists()) {
                    ret = Response.ok(file, "image/jpeg").build();
                } else {
                    ret =
                        Response.status(Response.Status.NOT_FOUND).build();
                }
            }
        } catch(NumberFormatException ex) {
            ret = Response.status(Response.Status.BAD_REQUEST).build();
        }
    }

    return ret;
}
```

```
    // PRIVATE ================================================================

    private boolean writeToFile(InputStream uploadedInputStream,
            String uploadedFileLocation) {

            try {
                OutputStream out = new FileOutputStream(new File(
                        uploadedFileLocation));
                int read = 0;
                byte[] bytes = new byte[1024];

                out = new FileOutputStream(new File(uploadedFileLocation));
                while ((read = uploadedInputStream.read(bytes)) != -1) {
                    out.write(bytes, 0, read);
                }
                out.flush();
                out.close();
            } catch (IOException e) {
                e.printStackTrace();
                return false;
            }
            return true;

        }

    private boolean isOnline(Timestamp t) {
        long milis = (new Date()).getTime();
        Date border = new Date(milis - MINUTES_ONLINE * 60 * 1000);
        if(t.before(border))
            return false;

        return true;
    }

}
```

5.3. Baza danych - kod SQL

```
USE secam;
```

```
SET FOREIGN_KEY_CHECKS=0;


-- Drop Tables, Stored Procedures and Views

DROP TABLE IF EXISTS Beeps CASCADE;
DROP TABLE IF EXISTS Devices CASCADE;
DROP TABLE IF EXISTS Images CASCADE;
DROP TABLE IF EXISTS Sessions CASCADE;
DROP TABLE IF EXISTS Users CASCADE;

-- Create Tables
CREATE TABLE Beeps
(
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    deviceid INTEGER UNSIGNED NOT NULL,
    timestamp TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (id),
    UNIQUE UQ_Beeps_deviceid(deviceid),
    UNIQUE UQ_Beeps_id(id),
    KEY (deviceid)

) ;


CREATE TABLE Devices
(
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    userid INTEGER UNSIGNED NOT NULL,
    deviceid VARCHAR(50) NOT NULL,
    devicename VARCHAR(50),
    PRIMARY KEY (id),
    UNIQUE UQ_Devices_id(id),
    UNIQUE UQ_PAIR_Deviceid_Userid(deviceid, userid),
    KEY (userid)

) ;


CREATE TABLE Images
(
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
```

```
    deviceid INTEGER UNSIGNED NOT NULL,
    path VARCHAR(100) NOT NULL,
    timestamp TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (id),
    UNIQUE UQ_Images_id(id),
    KEY (deviceid)

) ;


CREATE TABLE Sessions
(
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    userid INTEGER UNSIGNED NOT NULL,
    token VARCHAR(50) NOT NULL,
    sessiontype VARCHAR(10) NOT NULL,
    expires DATETIME,
    PRIMARY KEY (id),
    UNIQUE UQ_Sessions_id(id),
    UNIQUE UQ_Sessions_token(token),
    KEY (userid)

) ;


CREATE TABLE Users
(
    id INTEGER UNSIGNED NOT NULL AUTO_INCREMENT,
    email VARCHAR(40) NOT NULL,
    password VARCHAR(40) NOT NULL,
    PRIMARY KEY (id),
    UNIQUE UQ_Users_email(email),
    UNIQUE UQ_Users_id(id)

) ;



SET FOREIGN_KEY_CHECKS=1;


-- Create Foreign Key Constraints
ALTER TABLE Beeps ADD CONSTRAINT FK_Beeps_Devices
```

```
    FOREIGN KEY (deviceid) REFERENCES Devices (id);

ALTER TABLE Devices ADD CONSTRAINT FK_Devices_Users
    FOREIGN KEY (userid) REFERENCES Users (id);

ALTER TABLE Images ADD CONSTRAINT FK_Images_Devices
    FOREIGN KEY (deviceid) REFERENCES Devices (id);

ALTER TABLE Sessions ADD CONSTRAINT FK_Sessions_Users
    FOREIGN KEY (userid) REFERENCES Users (id);




-- Create Triggers
DROP TRIGGER IF EXISTS tr_setexpiredate;

DELIMITER $$
CREATE TRIGGER tr_setexpiredate BEFORE INSERT ON Sessions
FOR EACH ROW
BEGIN
    SET NEW.expires = DATE_ADD(NOW(), INTERVAL 280 DAY);
END $$
;
```

Bibliografia

[1] Android sdk reference. http://developer.android.com/reference/.

[2] Biblioteka pwn. http://encyklopedia.pwn.pl/.

[3] Hibernate. http://hibernate.org/.

[4] Jersey - restful web services in java. https://jersey.java.net/.

[5] Json. http://www.json.org/.

[6] Opencv - open source computer vision. http://opencv.org/.

[7] Representational state transfer (rest). http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm.

[8] T. A. S. Foundation. Apache license 2.0. http://www.apache.org/licenses/LICENSE-2.0.

[9] T. A. S. Foundation. Apache tomcat. http://tomcat.apache.org/.

[10] Google. Gson. https://code.google.com/p/google-gson/.

[11] P. Kusiciel. Użytkowników smartfonów będzie czterokrotnie więcej. http://di.com.pl/news/38010,0,U%C5%BCytkownik%C3%B3w_smartfon%C3%B3w_jest_czterokrotnie_wi%C4%99cej.html, 2011.

[12] Oracle. Mysql. http://www.mysql.com/.

[13] J. Wharton. Butterknife. http://jakewharton.github.io/butterknife/.

[14] J. Wharton. Retrofit. http://square.github.io/retrofit/.

[15] D. Xiaohui. An improved moving object detection algorithm based on frame difference and edge detection. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4297140, 2007.