Politechnika Poznańska
Wydział Informatyki
Instytut Informatyki

Praca dyplomowa inżynierska

SYSTEM KONTROLI BEZPIECZEŃSTWA – THE GUARD

Mateusz Bartos, 122437
Piotr Falkiewicz, 122537
Aleksandra Główczewska, 122494
Paweł Szudrowicz, 122445

Promotor
dr inż. Mariusz Nowak

Poznań, 2018 r.

Tutaj przychodzi karta pracy dyplomowej;
oryginał wstawiamy do wersji dla archiwum PP, w pozostałych kopiach wstawiamy ksero.

Spis treści

Rozdział 1

Wstęp

Podjęcie tematu (WHY?)

Celem pracy było zaprojektowanie i wykonanie systeu umożliwiającego wspomniane wyżej funkcje. Ponadto uzgodniono, że rozwiązanie będzie oparte na niezależnych modułach, które będzie można później, w łatwy sposób, zmodyfikować.

W ramach pracy przygotowano propozycję projektu całego systemu, od urządzeń zbierających dane, przez system moniturujący, po aplikacje klienckie. Następnie zaimplementowano zaprojektowane aplikacje, złożono zestawy urządzeń Raspberry i czujników, oraz połączono wszystkie elementy w spójny system.

Ze względu, na cel pracy oraz wykorzystane technologie i usługi, zespół oparł swoją pracę o: dokumentację usług dostępną na stronach internetowych producentów, dokumentację narzędzi dołączoną do odpowiednich repozytoriów, dokumentację sprzętu.

Struktura pracy wygląda następująco: w 1. rozdziale opisano architekturę przygotowanego systemu. Następna częsc została poswięcona opisowi działaniu komputerów jednopłytkowych Raspberry Pi. W Rozdziale 3. opisane są wykorzystywane rozwiązania chmurowe. Rozdział 4. dotyczy aplikacji przeznaczonych dla klientów, a w rozdziale 5 zawarto podsumowanie pracy.

W ramach niniejszej pracy Mateusz Bartos wykonał projekt oraz zrealizował aplikację mobilną obsługiwaną przez system Android. Ponadto przygotował maszynę wirtualną w ramach usług oferowanych przez Microsoft Azure. Zaproponował też wykorzystane usługi Google: Firebase Realtime Database, Firebase Storage oraz Firebase Authentication. Piotr Falkiewicz wykonał i zrealizował projekt serwera, obsługującego zapytania aplikacji mobilnych, w oparciu o protokół HTTP oraz jest odpowiedzialny za właciwą obsługę obrazu dostarczanego z urządzeń Raspberry do aplikacji przy wykorzystaniu modułu Nginx Rtmp. Ponadto wprowadził do projektu obsługę usługi Firebase Storage. W ramach pracy, Aleksandra Główczewska zaprojektowała i wykonała aplikację internetową, z wykorzystaniem języka programowania Python i frameworka Django. Ponadto jest odpowiedzialna za wprowadzenie uwierzytelniania użytkowników, za pomocą Firebase Authentication. Paweł Szudrowicz, w ramach pracy, przygotował urządzenia Raspberry do obsługi czujników oraz wysyłania danych do usługi Firebase Realtime Databse. Kolejnym zadaniem, które zrealizował, był projekt i wykonanie aplikacji mobilnej na urządzenia iOS.

Rozdział 2

Architektura systemu

Rozdział teoretyczny — przegląd literatury naświetlający stan wiedzy na dany temat.

Przegląd literatury naświetlający stan wiedzy na dany temat obejmuje rozdziały pisane na podstawie literatury, której wykaz zamieszczany jest w części pracy pt. *Literatura* (lub inaczej *Bibliografia*, *Piśmiennictwo*). W tekście pracy muszą wystąpić odwołania do wszystkich pozycji zamieszczonych w wykazie literatury. **Nie należy odnośników do literatury umieszczać w stopce strony.** Student jest bezwzględnie zobowiązany do wskazywania źródeł pochodzenia informacji przedstawianych w pracy, dotyczy to również rysunków, tabel, fragmentów kodu źródłowego programów itd. Należy także podać adresy stron internetowych w przypadku źródeł pochodzących z Internetu.

Rozdział 3

Zbieranie i przetwarzanie danych z czujników

Raspberry Pi

Wszystkie zestawy zbudowano w oparciu o Raspberry Pi 3 v1.2 (rys. 3.1). Zdecydowano się na to rozwiązanie, ponieważ bazuje on na dystrybucji Linuxa, posiada opowiednie interfejsy i złącza a także zintegrowany moduł WiFi. Minusem w stosunku do konkurencyjnego Arduino jest brak wejść analogowych. Problem rozwiązano dodając zewnętrzny przetwornik A/C.

Specyfikacja Raspberry Pi 3:

- Procesor 1.2 GHz
- Liczba rdzeni 4. Quad Core
- Pamięć RAM 1 GB
- Pamięć Karta microSD
- 40 GPIO

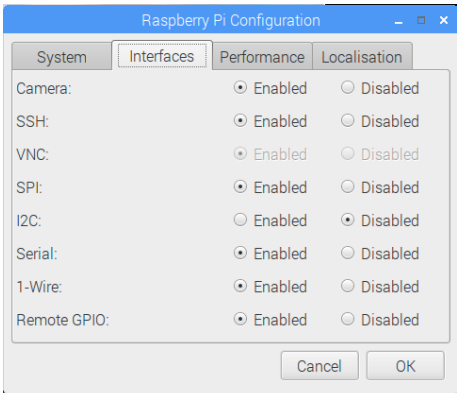
Rysunek 3.1: Raspberry Pi 3

Aby prawidłowo zainstalować oprogramowanie The Guard na dowolnym urządzeniu Raspberry Pi 3 należy wykonać poniższe czynności w terminalu:

1. sudo apt-get install libx264-dev
2. cd /usr/src
3. git clone git://source.ffmpeg.org/ffmpeg.git

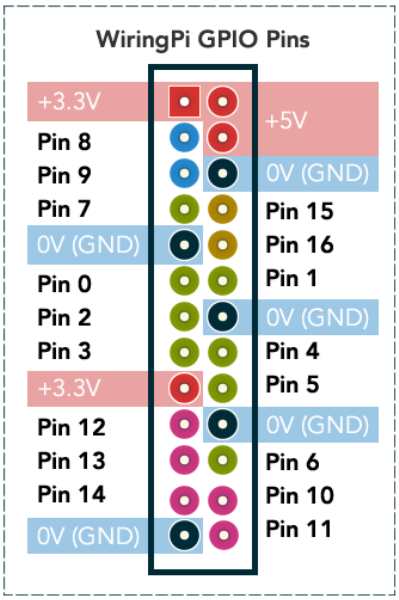
4. sudo ./configure –arch=armel –target-os=linux –enable-gpl –enable-libx264 –enable-nonfree
5. sudo make
6. sudo install
7. sudo nano /boot/config.txt
8. dopisać w pliku Dtoverlay=w1-gpio i Gpiopin=4
9. pip intall wiringpi
10. sudo pip install spidev
11. pip install pyrebase

Następnym krokiem jest włączenie odpowiednich interfejsów w panelu konfiguracyjnym. Należy zmienić ustawienia zgodnie ze schematem 3.2: W kodzie użyto biblioteki wiringpi do odczytu

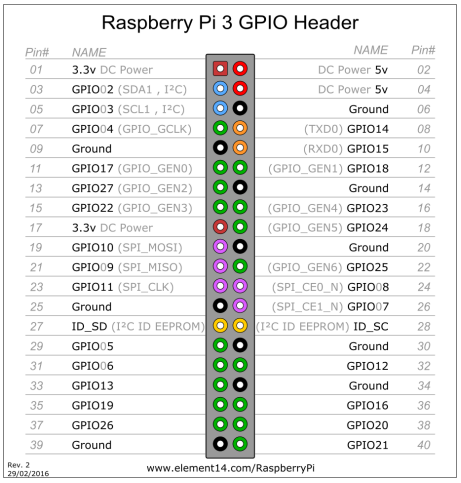


Rysunek 3.2: Ustawienia

danych z układów cyfrowych. Należy podkreślić, że numeracja fizycznych pinów(rys. 3.4) i numeracja pinów w bibliotece wiringPi(rys. 3.3) jest różna i nie zawiera wszystkich dostępnych pinów na urządzeniu. Przykładowo odczyt pinu numer 1 w wiringPi jest równoznaczny z odczytem stanu na pinie numer 12 (GPIO18). Zainstalowane oprogramowanie odpowiedzialne jest za ciągłe monito-



Rysunek 3.3: WiringPi



Rysunek 3.4: GPIO

rowanie stanów i zbieranie danych z czujników pomiarowych. Po podłączeniu układu do zasilania program jest uruchamiany automatycznie. Pierwszą czynnością jaką wykonuje Raspberry Pi jest wysłanie swojego numeru seryjnego do bazy danych Firebase. Cały proces jest w pełni zautomatyzowany. Dzięki temu użytkownicy od razu mogą dodać urządzenie i przeglądać dane z czujników na aplikacjach klienckich. Dodanie akcesorium pomiarowego następuje poprzez wprowadzenie w aplikacji jego numeru seryjnego.

Czujniki

Każdy zestaw składa się z 5 czujników analogowo cyfrowych, jednej kamery i jednego przetwornika AC.

a) Specyfikacja MQ-9 - czujnik tlenku węgla:

- Zasilanie: 5 V
- Pobór prądu: 150 mA
- Temperatura pracy: od -10 do 50 °C
- Wyjścia: analogowe oraz cyfrowe

b) Specyfikacja MQ-2 - czujnik LPG i dymu:

- Zasilanie: 5 V
- Pobór prądu: 150 mA
- Temperatura pracy: od -10 do 50 °C
- Wyjścia: analogowe oraz cyfrowe

c) Specyfikacja czujnika wykrywania płomieni:

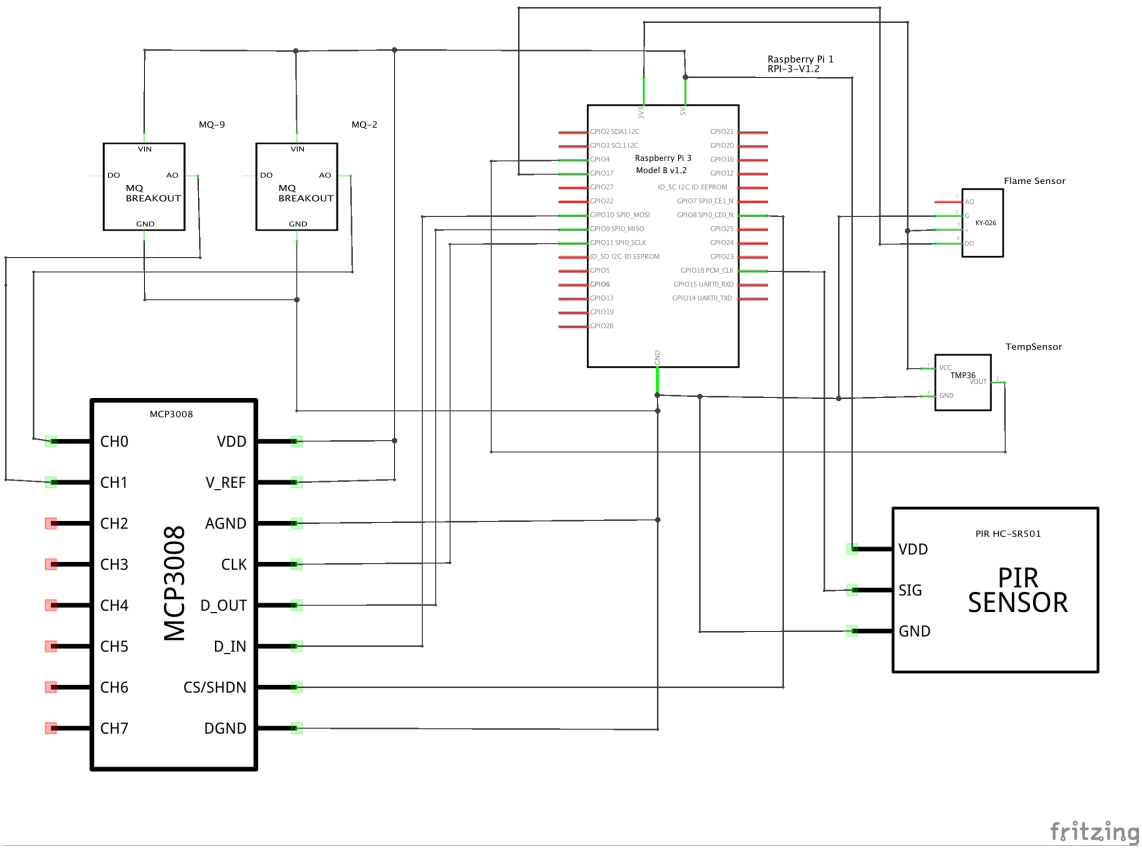
- Zasilanie: 3.3 V
- Zakres wykrywanej fali: 760 do 1100nm
- Kąt detekcji: od 0 do 60 stopni
- Temperatura pracy: od -25 do 85 °C

d) Specyfikacja DS18B20 - czujnik temperatury:

- Zasilanie: 3.3 V
- Zakres pomiarowy: od -55 do 125 °C

e) Kamera:

- Wykorzystano moduł kamery Raspberry Pi element14
- Kamera 5MP - wspierająca nagrywanie 30 klatek na sekundę w rozdzielczości Full HD



Rysunek 3.5: Schemat układu The Guard

f) Specyfikacja MCP3008 - przetwornik A/C:

- Zasilanie: od 2.7V do 5.5V
- Pobór prądu: 0.5 mA
- Interfejs komunikacyjny: SPI
- Liczba kanałów: 8
- Rozdzielczość: 10bit
- Czas konwersji: 10us

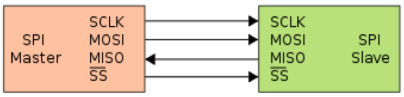
Niestety żaden model Raspberry nie posiada wbudowanego przetwornika analogowo cyfrowego dlatego konieczne było użycie układu zewnętrzenego. Wybrano przetwornik MCP3008 ze względu na jego nisko koszt i interfejs SPI, który jest wspierany przez Raspberry Pi. MCP3008 to 10-bitowy przetwornik analogowy cyfrowy. Zasilany jest napięciem 5V, napięcie VRef = 5V. Skoro jest to przetwornik 10-bitowy jest w stanie wykryć 1024 stany. Wykrywana przez niego minimalna różnica napięć na wejściu wynosi

$$1 * 5V/1024 = 4.88mV \tag{3.1}$$

Posiada 8 kanałów jednak w projekcie wykorzystano tylko 2 – dla czujników MQ-9 i MQ-2.

Interfejs SPI: SPI jest to interfejs synchroniczny. Może być do niego podłączone wiele urządzeń typu slave, jednak tylko z jednym urządzeniem Master, które generuje zegar. Master poprzez linię SS wybiera urządzenie z którym chce się komunikować.
Interfejs ten zawiera jeszcze 3 linie:

1. MOSI (ang. Master Output Slave Input):
   Poprzez tę linię wysyłane są dane z Raspberry Pi do przetwornika analogowo cyfrowego MCP3008.



Rysunek 3.6: Interfejs SPI

2. MISO (ang. Master Input Slave Output):
   Poprzez tę linię wysyłane są dane z przetwornika AC do układu Master czyli w naszym przypadku Raspberry Pi 3
3. SCLK (ang. Serial CLocK) :
   Ta linia wykorzystywana jest do przesłania zegara wygenerowanego przez Rapberry Pi 3

Do komunikacji poprzez ten interfejs wykorzystano bibliotekę spiDev.
Każdy układ monitoruje wskaźniki pomiarowe z czujników analogowych i cyfrowych. W przypadku wykrycia wskazań, które w znaczący sposób odbiegają od normy informuje właściciela o zagrożeniu. Informacja ta wysyłana jest do wszystkich urządzeń(smartfony, tablety itp), które posiada właściciel. Analizując dane z czujników analogowych w czystym powietrzu, które wynoszą wtedy odpowiednio:
Czujnik MQ-9: od 0.15 do 0.2
Czujnik MQ-2: od 0.05 do 0.15
Przyjęto, że granicą wysłania notyfikacji do użytkownika jest przekroczenie progu 0.3. Wartości te to znormalizowane dane z przetwornika AC, który jak już wcześniej wspomniano wykrywa 1024 stany. Odczytywane wartości bezpośrednio na wyjściu przetwornika MCP3008 dla czujnika MQ-9 w czystym powietrzu to około 170. Stąd $170/1024 = 0.166$. Wysłanie notyfikacji wiąże się z otrzymaniem wartości większej niż 308. Czujniki cyfrowe wykorzystane w pracy informują o wykryciu płomieni i ruchu. W przypadku detekcji takiego zagrożenia na wyjściu pojawia się stan niski i utrzymywany jest przez kilka sekund. W kodzie jednak wykonujemy instrukcje negacji, aby stan wysoki informował o niebezpieczeństwie a stan niski reprezentował jego brak. Na czujnikach znajduje się potencjometr, za pomocą którego dowolnie można ustawiać jego czułość. Odczyt danych następuje nieprzerwanie co 2 sekundy. Nie należy obawiać się, że czujnik ruchu nie wykryje zagrożenia z powodu braku odczytu we właściwym momencie, ponieważ utrzymuje on stan wysoki przez 5 sekund po wykryciu ruchu. Oprogramowanie wysyła także informacje z czujników do bazy danych Firebase. Zastosowanie takiej bazy daje możliwość monitorowania wszystkich danych w czasie rzeczywistym na aplikacjach klienckich. Dodatkowo w przypadku zagrożenia czyli przekroczeniu progu o którym mowa wyżej wysyłana jest push notyfikacja do urządzeń użytkownika a informacja o zagrożeniu zapisywana jest w bazie danych Django. Każdy jest w stanie odtworzyć całą historię wydarzeń w swoim systemie. Aby zapewnić wydajny i pewny system bezpieczeństwa przy otrzymaniu wysokich wartości na czujnikach zapisywany jest czas zdarzenia. Każda kolejna notyfikacja zostanie wysłana po upływie 10 minut od poprzedniej przy założeniu, że stan na czujniku nadal jest wysoki.

Obsługa wideo

Protokół RTMP 1. rtmp) Podstawą funkcji strumieniowania wideo, jest protokół RTMP (Real-Time Message Protocol). Jest to, oparty na protokole TCP, protokół wysyłania informacji potem
2. hls)

h264 potem

Raspberry Pi Do obsługi strumieniowania wideo, po stronie Raspberry Pi, wykorzystywany jest program FFMPEG. Pozwala on na sterowanie strumieniem, od wyboru urządzenia wejsciowego, przez statystyki strumienia, po punkt docelowy. Dostęp do unikatowego, dla każdego urządzenia Raspberry, punktu końcowego, gwarantowało wczeniejsze pobranie nr seryjnego urządzenia. Poniżej przedstawiono skrypt, wykonujący wymienione funkcje:

```
#!/bin/bash
```

```
serial_id="$(cat /proc/cpuinfo | grep Serial | cut -d ' ' -f 2)"
raspivid -o - -t 0 -fps 30 -b 1000000 | ffmpeg -re -ar 44100 -ac 2 -acodec pcm_s16le -f s16le -
```

Pierwszą czynnoscią, wykonywaną w skrypcie, jest otwarcie pliku /proc/cpuinfo, następnie znajdowana jest w nim linia, w której znajduje się wyjątkowy serial urządzenia. Na końcu, z wykorzystaniem potoku, i funkcji cut, wartość ta zostaje przypisana do zmiennej $serial_id$.

W drugiej linii skryptu wykorzystane jest narzędzie linii poleceń Raspberry - raspivid. Pozwala ono pobrać obraz z kamery.

- Pierwszym przełącznikiem jest -o z parametrem -. Oznacza to, że obraz z kamery jest wysyłany na wyjscie standardowe.
- Przełącznik -t ustawiony na 0 pozwala przekazywać obraz, z modułu kamery, przez nieokrelony czas. Aby przestać pobierać wideo, należy użyć przerwania za pomocą sygnału SIGINT (obsługiwanego w terminalu skrótem klawiszowym CTRL+C).
- Opcja -fps pozwala wskazać liczbę przechwytywanych klatek w ciągu sekundy. Tutaj wykorzystano maksymalne możliwoci wybranego modułu kamery.
- Ostatnią opcją, wykorzystaną w pobieraniu obrazu z kamery, jest bitrate, tzn wielkość pamięci, w której ma się znaleźć obraz przechwycony w ciągu 1 sekundy. Ustawienie opcji -b na 1000000 oznacza, że 1 sekunda wideo, może zajmować 125 kilobajtów pamięci. Jest to szczególnie istotna informacja, w kontekcie transmisji obrazu poza urządzenie.

Drugim poleceniem jest wywołanie narzędzia ffmpeg, połączonego za pomocą potoku, odbierającego, przechwytywany za pomocą funkcji raspivid, obraz i przekazującego go na docelowy punkt końcowy. Za jego pomocą ustala się ostatecznie opcje kodujące obraz i dźwięk w trakcie końcowej transmisji.

- Przełącznik -re pozwala odczytywać dane wejsciowe, z oryginalną częstotliwoscią. Zatem zostają przechwycone ustawienia przełącznika funkcji raspivid -fps 30.
- Opcje -ar, -ac, -acodec, -f, -strict odpowiadają kolejno za: próbkowanie dźwięku, wybór liczby kanałów, kodek audio, format dźwięku oraz wybór eksperymentalnego sposobu kodowania. Wymuszenie wykorzystania, jako wejscia strumienia dźwięku, na /dev/zero, oznacza, że strumień ten zostaje wypełniony wartosciami pustymi. Zatem opcje transmisji dźwięku są nieistotne.
- Przełącznik -vcodec ustala kodek wideo. W pracy wykorzystano standard kodowania h264.
- Następnie ustawiono wejscie obrazu. Przełącznik -i - powoduje, że narzędzie ffmpeg przechwytuje, dzięki potokowi, obraz przekazywany funkcją raspivid.
- Opcja -g 60 oznacza, że tzw klatka kluczowa (ang keyframe) pojawia się co 60 klatek. W tej sytuacji, co 2 sekundy.
- Przełącznik -f, w przypadku strumienia obrazu z kamery, wymusza format nadawanego wideo.
- Ostatnim elementem polecenia jest podanie punktu docelowego dla strumienia. Za pomocą protokołu RTMP, obługiwanego przez serwer o adresie IP 52.236.165.15 na porcie 1936, obraz wysyłany jest na aplikację o nazwie camera i punkt charakteryzowany przez serial urządzenia. Działanie tego elementu opisano w kolejnym punkcie.

Serwer Narzędziem, umożliwiającym obsługę strumieniowanie wideo z wielu źródeł, na wiele urządzeń równoczesnie, jest serwer NGINX wraz z dodatkiem o nazwie rtmp-NGINX.

Rozdział 4

Rozwiązania chmurowe

Microsoft Azure

Opis chmury oraz używanych przez nas usług

Aplikacja serwerowa

Opis Django

Baza danych

Opis technologii, schematy tabel

Rozdział 5

Aplikacje klienckie

Na podstawie statystyk rynkowych, dostępnych na stronie statcounter.com (http://gs.statcounter.com/os-market-share/mobile-tablet/worldwide/monthly-201801-201801-bar) zdecydowano się na stworzenie 3 klientów systemu The Guard, które umożliwią użytkowanie możliwie największej grupie osób:

- aplikacji mobilnej na system Android,
- aplikacji mobilnej na system iOS,
- aplikację webową.

Aplikacja Android

Wybór narzędzi

Do stworzenia aplikacji mobilnej na system Android użyto języka Kotlin - języka stworzonego przez firmę JetBrains, który 17.05.2017 r. (źródło: https://twitter.com/Android/status/864911929143197696) został uznany przez Google jako oficjalny język programowania aplikacji na platformę Android. Kotlin ściśle współegzystuje z kodem stworzonym w Javie i w przypadku Androida jest kompilowany do kodu JVM.

Skorzystano ze środowiska Android Studio w wersji 3.0.1, do automatyzacji budowy projektu został wykrozystany Gradle w wersji 4.1.

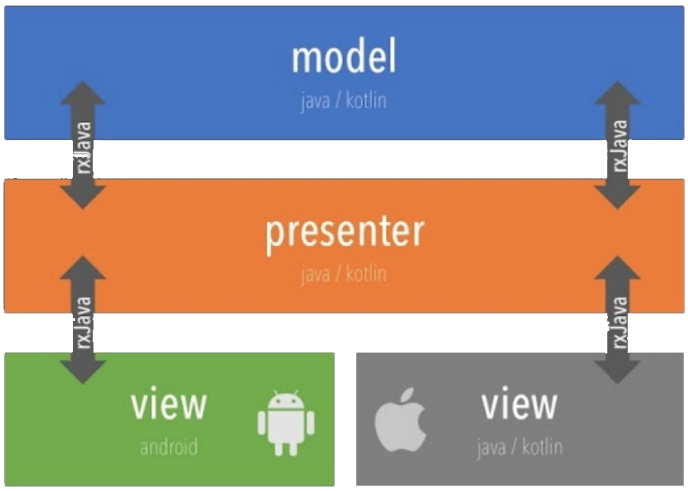
Aplikacja skierowana jest na urządzenia z systemem Android od wersji Lollipop 5.0 (o numerze SDK większym niż 20), który został wydany 12.12.2014 r. Ograniczenie wersji spowodowane jest możliwością użycia bardziej zaawansowanych komponentów, niedostępnych dla niższych wersji. W styczniu 2018 r. oficjalne statystyki informują o tym, że około 80,7 % wszystkich urządzeń z systemem Android na świecie ma wersję 5.0 lub wyższą.

5.0.1 Architektura aplikacji

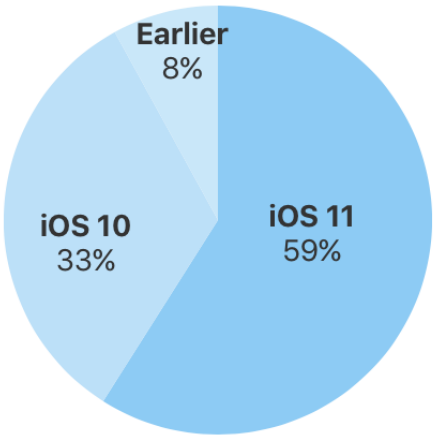
Aplikacja The Guard dla systemu Android została stworzona zgodnie z założeniami architektury Model View Presenter. Architektura MVP zakłada rozdzielenie

Aplikacja iOS

Aplikacja przeznaczona jest na urządzenia z systemem operacyjnym iOS od wersji 10.0. Nie wspiera ona wcześniejszych wersji ze względu na nowe funkcje, które Apple wprowadziło wraz z pojawieniem się iOS 10.0 (m.in. klasa UNUserNotificationCenter). Jednak 92% wszystkich obecnych użytkowników tego systemu(rys. 5.1) jest w stanie zainstalować aplikację a liczba ta stale rośnie. Aplikacja wspiera zarówno telefony komórkowe iPhone jak i tablety iPad. Napisana w stosunkowo nowym języku Swift (zaprezentowany przez Apple w 2014r na konferencji WWDC) w oparciu o architekturę MVC (Model-View-Controller) wykorzystując przy tym programowanie reaktywne i funkcjonalne. Aplikacja powstała w programie Xcode. Programowanie reaktywne zrealizowano przy pomocy biblioteki RxSwift. Ten paradygmat programowania związany jest z pojęciem obserwatora i sekwencji obserwowalnych. Każdy obserwator wywołując funkcję 'subscribe' na elemencie obserwowalnym otrzymuje informację o każdej zmianie na tym obiekcie. RxSwift wykorzystano m.in w celu wznowienia streamu obrazu z kamery

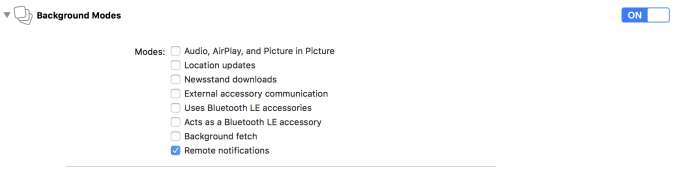


Rysunek 5.1: Architektura MVP



Rysunek 5.2: Udziały wersji systemu iOS w rynku

w momencie przejścia aplikacji z trybu background do trybu foreground. Oznacza to, że po wyjściu z aplikacji, ale pozostawiając ja działającą w tle i po ponownym jej uruchomieniu tracono obraz ze streamu. Przyczyną jest polityka Apple, która nie zaleca aby aplikacje pracowały w tle i domyślnie wyłącza każdą taką aktywność. Ma to na celu przedłużenie żywotności baterii i optymalizacji całego systemu poprzez ograniczenie ilości zajmowanych zasobów [https://developer.apple.com/library/content/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/I
Oczywiście istnieje możliwość włączenia pracy w tle, jednakże konieczne jest aktywowanie trybu "Background Modes"i zaznaczenie konkretnej aktywności, którą chcielibyśmy wykonywać. Lista dozwolonych czynności możliwych do realizacji w tle jest ograniczona (rys. 5.2). Próba oszustwa i



Rysunek 5.3: Tryby pracy w tle

wykonywania innej pracy w tle niż zaznaczona zostanie wychwycona w procesie weryfikacji przed jej publikacją na platformie Apple Store. Dzięki programowaniu reaktywnemu problem wznowienia podglądu obrazu został rozwiązany co prezentuje poniższy kod:

```
let appDelegate = UIApplication.shared.delegate as! AppDelegate
        appDelegate.inBackground.asObservable().subscribe(onNext: { (value) in
            if let streamView = self.streamView {
                if let player = self.currentPlayer {
                    if value == false {
                        self.streamVideoFrom(urlString: self.currentUrlString!)
                        print("Enter foreground")
                    } else {
                        print("Enter background")
                        streamView.layer.sublayers?.forEach({ (layer) in
                            layer.removeFromSuperlayer()
                        })
                    }
                }
            }
        }).disposed(by: disposeBag)
```

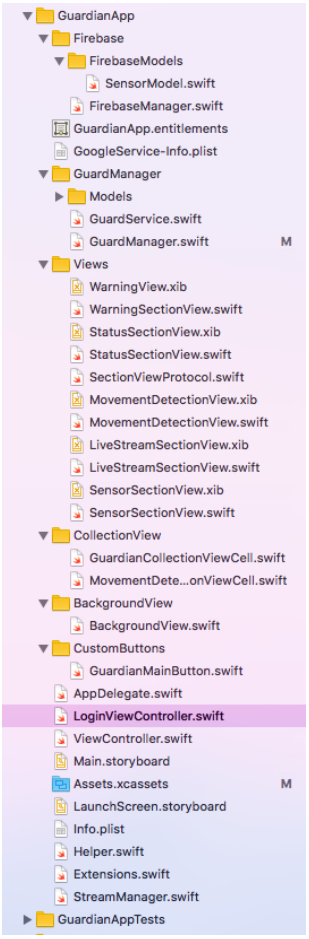
Zmienna 'inBackground', która jest zmienną obserwowalną, ustawiana jest w oddzielnej klasie AppDelegate (klasa, która zapewnie poprawną interakcję z systemem iOS) na wartość true w chwili przejścia do trybu pracy w tle i na wartość false w przeciwnym wypadku. Klasa, w której wywoływany jest funkcja 'subscribe' jest obserwatorem tej zmiennej. Kod wewnątrz funkcji subscribe uruchamiany jest przy każdej zmianie wartości 'inBackground' i wznawia ponownie stream po każdym ponownym uruchomieniu programu. "Programowanie funkcjonalne natomiast polega na traktowaniu funkcji jako obiektu. Oznacza to, że mogą być one zapisywane, kopiowane i przekazywane tak samo jak wszystkie inne obiekty. Mogą być używane jako parametry innych funkcji."[Pro Swift, Paul Hudson 04.19.2016 strona 172 link do store na iTunes : https://itunes.apple.com/us/book/pro-swift/id1111033310?mt=11]. Wykorzystane są w miejscach gdzie konieczne jest przekształcanie danych:

```
lastNotification = notifications.array.sorted(by: { (n1, n2) -> Bool in
n1.date > n2.date
}).filter({ (notif) -> Bool in return notif.type == "PIRSensor"}).first
```

Na tablicy z notyfikacjami zastosowano szereg kolejnych funkcji: posortowano je malejąco według daty, przefiltrowano w taki sposób aby wybrać tylko te o typie 'PIRSensor' czyli te pochodzące z czujnika ruchu. Na sam koniec wybrano tylko jeden pierwszy element z wybranych i wynik wpisano do zmiennej lastNotification. Ważne jest, że każda kolejna wywoływana funkcja np. filter, odbiera wynik poprzedniej.

Strukturę kodu (rys. 5.3) podzielono na kilka osobnych, logicznych części. Folder Firebase zawiera model bazy danych czujników, które zapisane są na serwerach Firebase. W folderze GuardManager znajdują się elementy odpowiedzialne za komunikację REST-ową z serwerem Django i modele bazy danych znajdującej się na naszym serwerze. Folder Views jest zbiorem widoków, które wczytywane są w zależności, w której sekcji się znajdujemy (opis sekcji niżej). ViewController.swift jest głównym kontrolerem zarządzającym widokami i modelami. Odpowiada za załadowanie odpowiedniego widoku i prezentację danych z odpowiedniej sekcji. W folderze GuardianAppTests napisane zostały testy jednostkowe, które sprawdzają poprawność przekształcania danych typu JSON (odpowiedź serwera) do obiektów zdefiniowanych w folderze GuardManager/Models. Klasy, których nazwy kończą się na Manager oznaczają obiekty typu Singleton. Celem takiego wzorca jest zapewnienie istnienia tylko jednej instancji w całej aplikacji i globalnego dostępu do tego obiektu. GuardManager, który odpowiada za pobieranie danych z bazy danych - taki obiekt nie powinien być utworzony więcej niż jeden raz, gdyż wszystkie klasy, które z niego korzystają nie potrzebują kolejnych instancji tej klasy. W ten sposób zapewniono, że zawsze odwołujemy się do tego samego obiektu. Instalacja zewnętrznych bibliotek odbywa się za pomocą CocoaPods. Jest to menadżer zależności dzięki któremu szybko możemy wyszukać i zainstalować wymagane oprogramowanie. Wszystkie użyte zależności przedstawiono poniżej:

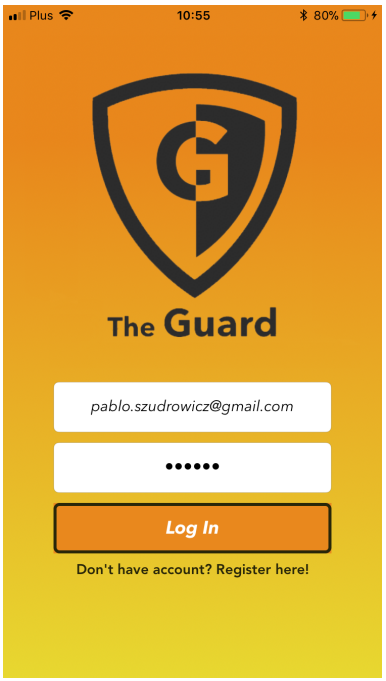
```
pod 'Moya'
pod 'MBProgressHUD', '~> 1.0'
pod 'RxSwift',    '~> 4.0'
pod 'RxCocoa',    '~> 4.0'
```



Rysunek 5.4: Struktura aplikacji

```
pod 'IHKeyboardAvoiding'
pod 'Moya-SwiftyJSONMapper'
pod 'Firebase/Core'
pod 'Firebase/Messaging'
pod 'Firebase/Auth'
pod 'Firebase/Database'
pod 'M13ProgressSuite'
```

Moya używana jest do asynchronicznej REST-owej komunikacji z serwerem Django. SwiftyJSONMapper przydatna okazuje się do przekształcenia odpowiedzi serwera w postaci JSON do wcześniej zdefiniowanego modelu. MBProgressHUD umożliwia wyświetlanie ekranu ładowania podczas pobierania informacji z serwera. RxSwift i RxCocoa to biblioteki do programowania reaktywnego. Moduły Firebas/Core itp. służą do komunikacji z serwerami Firebase. Ostatni 'pod M13ProgressSuite' służy do rysowania wykresów i animowanych elementów graficznych w systemie iOS. Po uruchomieniu aplikacji pierwszym widokiem jest ekran logowania i rejestracji użytkowników (rys 5.4). Po prawidłowym uwierzytelnieniu użytkownika uzyskiwany jest dostęp do głównego widoku aplikacji. W górnej części możliwy jest wybór 5 sekcji: sekcja czujników, sekcja historii notyfikacji, sekcja ostatnich zagrożeń przy wykryciu ruchu, sekcja streamu na żywo, sekcja ustawień. Wszystkie te sekcje dotyczą konkretnego urządzenia wybranego w pasku na dole ekranu. Przy pierwszym uruchomieniu nie istnieje żadne urządzenie przypisane do naszego konta użytkownika. Aby dodać pierwsze i kolejne stacje, od których chcemy otrzymywać notyfikacje o zagrożeniach a także śledzić i monitorować informacje z czujników należy wybrać przycisk Ńewź plusikiem w dolnej części ekranu. Pojawi się okno z prośbą o wpisanie numeru identyfikującego urządzenie. Po chwili dodany "Guard"będzie widoczny w na liście.



Rysunek 5.5: Ekran logowania

Sekcja czujników: Jest to jedna z najważniejszych sekcji aplikacji (rys 5.5). Otrzymuje ona dane z czujników w czasie rzeczywistym i prezentuje je użytkownikowi. W zależności od koloru prezentowanej wartości z czujnika użytkownik analizuje zagrożenie. Kolor zielony reprezentuje bezpieczne i prawidłowe odczyty na czujnikach, kolor pomarańczowy średnie, kolor czerwony reprezentuje bardzo wysoki poziom niebezpieczeństwa. Implementacja tej funkcjonalności zrealizowana została przy pomocy modelu HSV, który w przeciwieństwie do RGB pozwala na bardzo proste przejście z jednego koloru do kolejnego poprzez zmienę tylko jednego parametru. Zmieniając parametr Hue zmieniamy barwę przy stałym nasyceniu i jasności. Wartość tego parametru równa 120° odpowiada kolorowi zielonemu, kolor czerwony to 0°. Przekształcając wartość otrzymaną z czujników, która jest z zakresu [0-1] na wartość z przedziału [120-0] otrzymano wspomniany efekt. Poniżej przedstawiono fragment konwersji danych z czujników na kolor w modelu HSV, gdzie zmienna sensors[0] reprezentuje czujnik LPG.

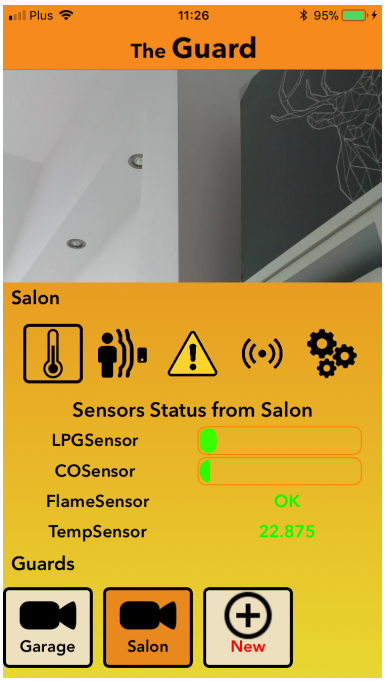
```
UIColor(hue: CGFloat(0.33 - (sensors[0].value * 0.33)),
saturation: 1, brightness: 1, alpha: 1)
```

Sekcja historii notyfikacji: W tej sekcji użytkownik ma dostęp do historii zdarzeń w systemie (rys 5.6). Po zaznaczeniu daty reprezentującej moment wystąpienia zagrożenia i wybranu przycisku 'preview' prezentowana jest informacja o miejscu niebezpieczeństwa i jego rodzaju.

Sekcja ustawień: Użytkownik ma możliwość zmiany nazwy urządzenia, które zazwyczaj reprezentuje miejsce, w którym się znajduje. Istnieje również możliwość uzbrojenia i wyłączenia każdego urządzenia. Sprowadza się to do tego, że w przypadku zaznaczenia opcji "Disarmed"użytkownik nie otrzyma kolejnych notyfikacji o zagrożeniach (rys. 5.7). Opcja ta może okazać się przydatna w momencie uszkodzenia któregoś z modułów i tym samym błędnych danych wysyłanych z czujników.

Sekcja streamu: Sekcja odpowiedzialna za prawidłowy odbiór obrazu z kamery zaznaczonej w dolnej części ekranu. Okno, w którym odbywa się stream ustawiono w taki sposób, aby bez względu na rozmiar telefonu utrzymywało proporcję 16:9. Pozbyto się dzięki temu czarnych ramek lub braku części obrazu ze streamu.

Sekcja ostatnich zagrożeń: Sekcja prezentuje ostatnie nagrane zagrożenie. Służy do szybkiego przeglądu ostatniego niebezpieczeństwa i prezentuje ostatni nagrany materiał video.



Rysunek 5.6: Sekcja czujników

Przeprowadzono kilka testów aplikacji pod pełnym obciążeniem za pomocą programu Instruments. Szczególnie interesująco przedstawia się zużycie sieci podczas streamu obrazu. Widać, że w ciągu jednej minuty pobrano 6,61MB a wysłano jedynie 24,11Kb (rys. 5.8). Obraz pobierany jest tylko wtedy kiedy aplikacja jest aktywna. W ciągu godziny działania aplikacji pobierze ona około 400MB danych. Jednak dla zapewnienia komfortu użytkowania i płynnego streamu obrazu zalecane jest posiadanie łącza umożliwiającego transfer danych na poziomie min. 200KB/s. Przeprowadzono także test na zużycie pamięci RAM i zużycie procesora. Te jednak są niewielkie i wynoszą odpowiednio 25MB pamięci RAM i średnio 1 procent zużycia procesora. Zużycie procesora wzrasta do poziomu ok. 15 procent tylko w momencie pobierania nagranego obrazu z serwera. Wtedy też zużycie pamięci RAM jest o około 5MB większe i wynosi około 30MB (rys. 5.9). Testy przeprowadzono na iPhonie 6S i iPadzie Pro.

Aplikacja internetowa

Aplikacja webowa przeznaczona jest dla użytkowników wszystkich systemów operacyjnych i została przetestowana w przeglądarce Firefox. Rozmieszczenie komponentów aplikacji różni się od tego zastosowanego w aplikacjach IOS oraz Android - spowodowane jest to inną rozdzielczością ekranu.

Panel logowania / rejestracji użytkownika:

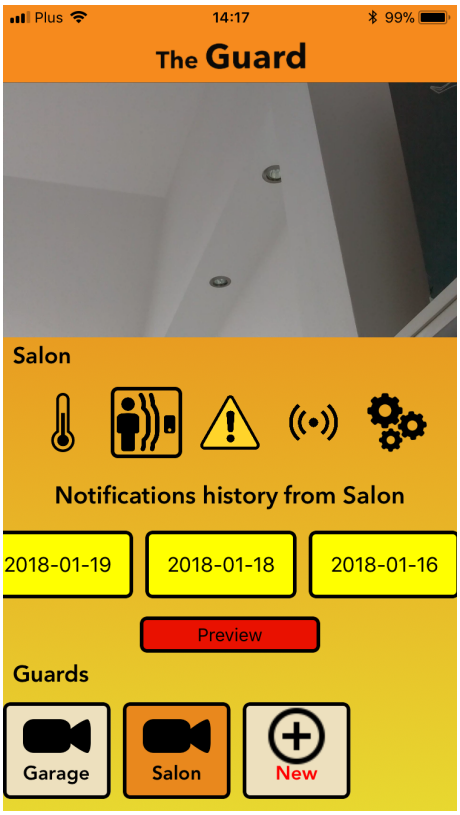
Menu wyboru urządzenia:

Główny panel aplikacji:

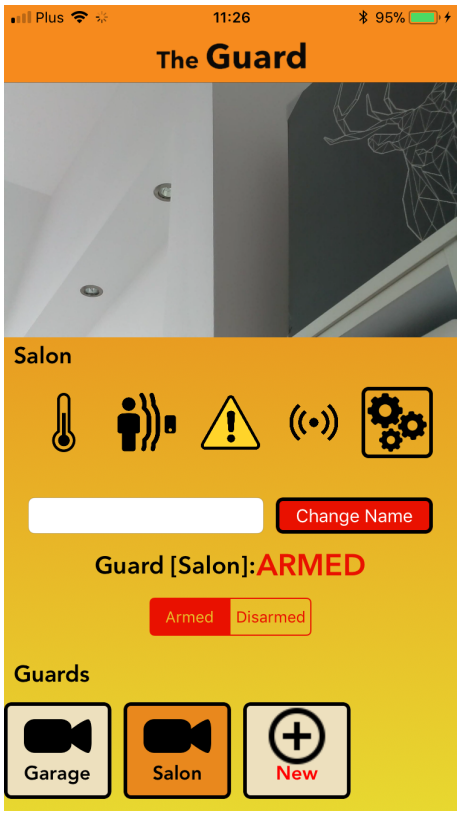
Panel rejestracji urządzenia:

Panel notyfikacji:

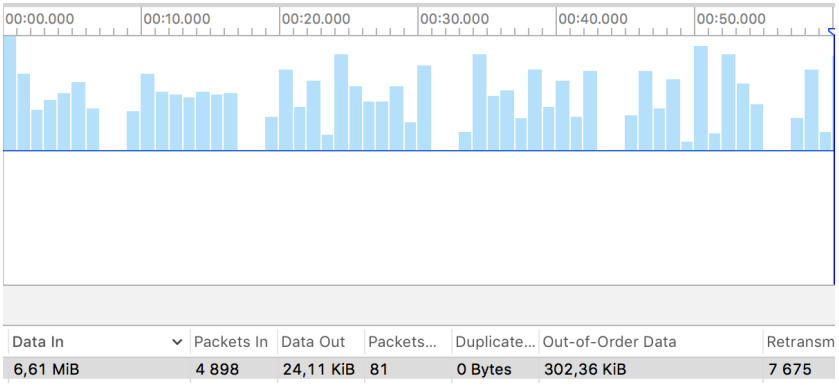
Implementacja Django - połączenie z bazą danych: Biblioteka Django posiada wbudowane rozwiązania umożliwiające pobieranie informacji z bazy danych projektu dzięki czemu aplikacja



Rysunek 5.7: Sekcja historii notyfikacji

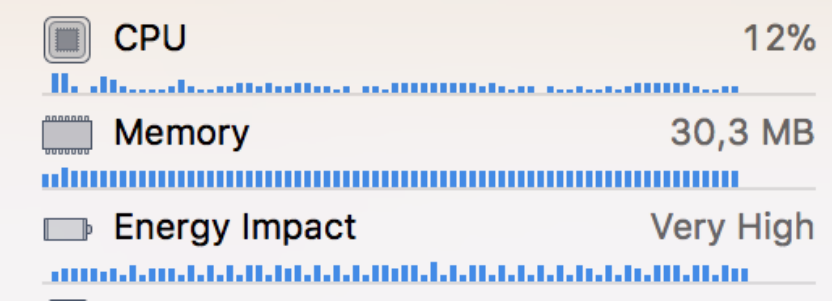


Rysunek 5.8: Sekcja ustawień



Rysunek 5.9: Zużycie sieci podczas streamu

webowa nie wysyła zapytań na określone dla aplikacji mobilnych porty, tylko komunikuje się bezpośrednio z bazą danych. Rozwiązanie to umożliwia uniezależnienie aplikacji webowej od stanu portów oraz zmniejsza liczbę potrzebnych zapytań wysyłanych do serwera.



Rysunek 5.10: Zużycie procesora i RAM podczas największego obciążenia

Rozdział 6

Zakończenie

Zakończenie pracy zwane również Uwagami końcowymi lub Podsumowaniem powinno zawierać ustosunkowanie się autora do zadań wskazanych we wstępie do pracy, a w szczególności do celu i zakresu pracy oraz porównanie ich z faktycznymi wynikami pracy. Podejście takie umożliwia jasne określenie stopnia realizacji założonych celów oraz zwrócenie uwagi na wyniki osiągnięte przez autora w ramach jego samodzielnej pracy.

Integralną częścią pracy są również dodatki, aneksy i załączniki np. płyty CDROM zawierające stworzone w ramach pracy programy, aplikacje i projekty.

Dodatek A

Parę słów o stylu `ppfcmthesis`

A.1 Różnice w stosunku do „oficjalnych" zasad składu ze stron FCMu

Autor niniejszego stylu nie zgadza się z niektórymi zasadami wprowadzonymi w oficjalnym dokumencie FCMu.[1] Poniższe elementy są składane nieco inaczej w stosunku do „oficjalnych" wytycznych.

- Promotor na stronie tytułowej jest umiejscowiony w centralnej osi pionowej strony (a nie po prawej stronie).
- Czcionka użyta do składu to nie Times New Roman.
- Spacje między tytułami akapitów oraz wcięcia zostały pozostawione takie, jak są zdefiniowane oryginalnie w pakiecie Memoir (oraz w LaTeXu). Jeśli zdefiniowano „polską" opcję składu, to będzie w użyciu wcięcie pierwszego akapitu po tytułach rozdziałów. Przy składzie „angielskim" tego wcięcia nie ma.
- Odwrócona jest kolejność rozdziałów *Literatura* i *Dodatki*.
- Na ostatniej stronie umieszczono stopkę informującą o prawach autorskich i programie użytym do składu.
- Nie do końca zgadzam się ze stwierdzeniem, iż „zamieszczanie list tabel, rysunków, wykresów w pracy dyplomowej jest nieuzasadnione". Niektóre typy publikacji zawierają tabele i rysunki, których skorowidz umożliwia łatwiejsze ich odszukanie. Ale niech będzie.
- Styl podpisów tabel jest taki sam, jak rysunków i odmienny od FCMowego. Jeśli ktoś koniecznie chce mieć zgodne z wytycznymi podpisy, to zamiast `caption` niech użyje `fcmtcaption` do podpisywania tablic oraz `fcmfcaption` do podpisywania rysunków. Podpisy pod rysunkami pozostaną pełne, a nie skrócone („Rys.").
- Styl formatowania literatury jest nieco inny niż proponowany przez FCM.

[1] `http://www.fcm.put.poznan.pl/platon/dokumenty/dlaStudentow/egzaminDyplomowy/zasadyRedakcji`