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

Wstęp[1] do pracy powinien zawierać następujące elementy:

- krótkie uzasadnienie podjęcia tematu;
- cel pracy (patrz niżej);
- zakres (przedmiotowy, podmiotowy, czasowy) wyjaśniający, w jakim rozmiarze praca będzie realizowana;
- ewentualne hipotezy, które autor zamierza sprawdzić lub udowodnić;
- krótką charakterystykę źródeł, zwłaszcza literaturowych;
- układ pracy (patrz niżej), czyli zwięzłą charakterystykę zawartości poszczególnych rozdziałów;
- ewentualne uwagi dotyczące realizacji tematu pracy np. trudności, które pojawiły się w trakcie realizacji poszczególnych zadań, uwagi dotyczące wykorzystywanego sprzętu, współpraca z firmami zewnętrznymi.

Wstęp do pracy musi się kończyć dwoma następującymi akapitami:

> Celem pracy jest opracowanie / wykonanie analizy / zaprojektowanie / .....

oraz:

> Struktura pracy jest następująca. W rozdziale 2 przedstawiono przegląd literatury na temat ..... Rozdział 3 jest poświęcony ..... (kilka zdań). Rozdział 4 zawiera ..... (kilka zdań) ..... itd. Rozdział X stanowi podsumowanie pracy.

W przypadku prac inżynierskich zespołowych lub magisterskich 2-osobowych, po tych dwóch w/w akapitach musi w pracy znaleźć się akapit, w którym będzie opisany udział w pracy poszczególnych członków zespołu. Na przykład:

> Jan Kowalski w ramach niniejszej pracy wykonał projekt tego i tego, opracował ..... Grzegorz Brzęczyszczykiewicz wykonał ....., itd.

[1] Treść przykładowych rozdziałów została skopiowana z „zasad" redakcji prac dyplomowych FCMu [**?**].

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 zbudowaliśmy w oparciu o Raspberry Pi 3 v1.2. Zdecydowaliśmy 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

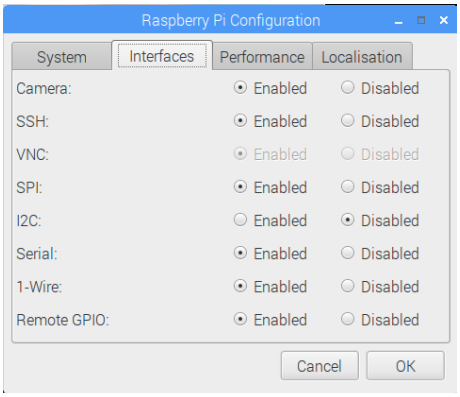
Rysunek 3.1: Raspberry Pi 3

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

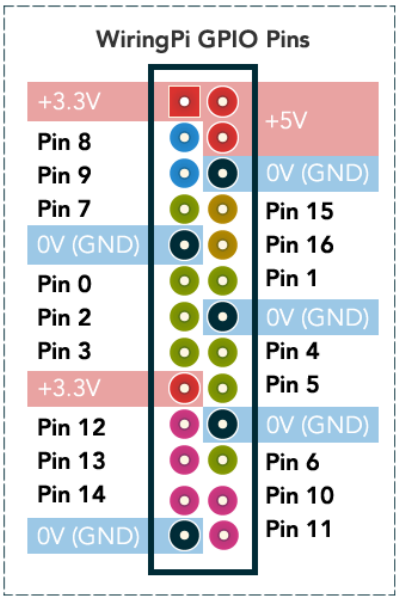
4. sudo ./configure –arch=armel –target-os=linux –enable-gpl –enable-libx264 –enable-nonfree
5. sudo make
6. sudo install
7. sudo nano /boot/config.txt
8. dopisać w pliku Dtoverlay=w1-gpio i Gpiopin=4
9. pip intall wiringpi
10. sudo pip install spidev
11. pip install pyrebase

Następnym krokiem jest włączenie odpowiednich interfejsów w panelu konfiguracyjnym. Należy zmienić ustawienia zgodnie z poniższym schematem: W kodzie używamy biblioteki wiringpi do od-

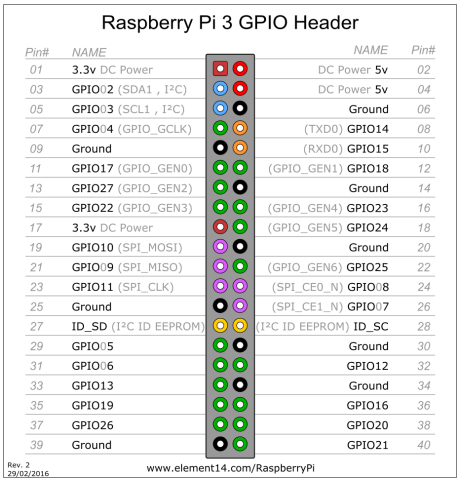


Rysunek 3.2: Ustawienia

czytu danych z układów cyfrowych. Należy zaznaczyć, że numeracja fizycznych pinów i numeracja pinów w bibliotece wiringPi jest różna i nie zawiera wszystkich dostępnych pinów na urządzeniu. Przykładowo odniesienie się do pinu numer 1 w wiringPi jest równoznaczne z odczytem stanu na pinie 12 (GPIO18).



Rysunek 3.3: WiringPi



Rysunek 3.4: GPIO

Oprogramowanie zainstalowane na Raspberry Pi odpowiedzialne jest za ciągłe monitorowanie stanów i danych z czujników pomiarowych. Po podłączeniu układu do zasilania oprogramowanie jest uruchamiane automatycznie. Pierwszą czynnością jaką wykonuje Raspberry jest wysłanie swojego numery seryjnego do bazy danych Firebase. Cały proces jest w pełni zautomatyzowany. Dzięki temu użytkownicy od razu mogą dodać urządzenie i monitorować dane z czujników na aplikacjach klienckich. Dodawanie urządzenia następuje poprzez wprowadzenie w aplikacji numery seryjnego urządzenia, które chcę dołączyć do mojego konta użytkownika.

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:

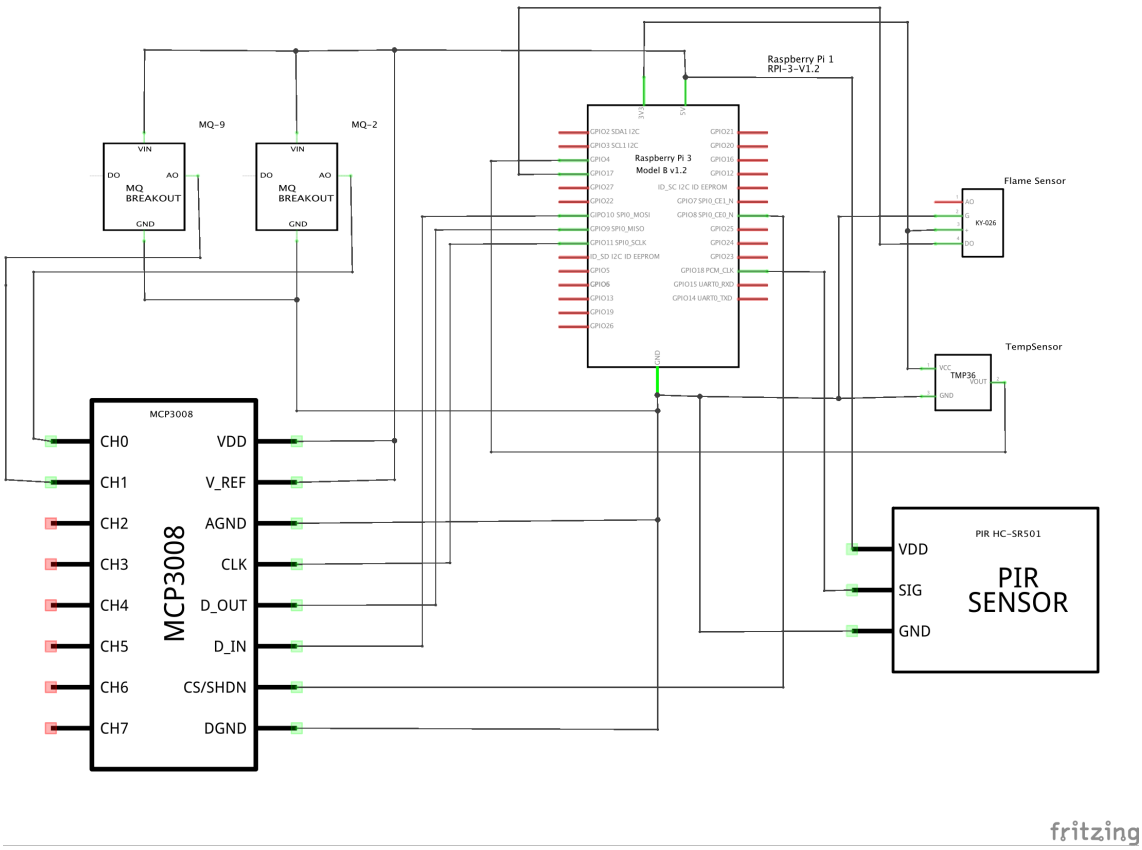
- Zasilanie: 3.3 V
- Zakres wykrywanej fali: 760 do 1100nm
- Kąt detekcji: od 0 do 60 stopni
- Temperatura pracy: od -25 do 85 °C

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:

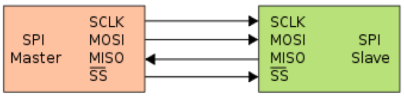
- Zasilanie: od 2.7V do 5.5V
- Pobór prądu: 0.5 mA
- Interfejs komunikacyjny: SPI
- Liczba kanałów: 8
- Rozdzielczość: 10bit
- Czas konwersji: 10us

Niestety żaden model Raspberry nie posiada wbudowanego przetwornika analogowo cyfrowego dlatego konieczne było użycie układu zewnętrzenego. Wybraliśmy 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 stanów. 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 – do podłączenia 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, który generuje zegar. Master poprzez linię SS wybiera urządzenie z którym chce się komunikować.
Interfejs ten zawiera jeszcze 3 linie:



Rysunek 3.6: Interfejs SPI

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

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 z 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, które posiada właściciel. Analizując dane z czujników analogowych w czytym powietrzu, które wynoszą wtedy odpowiednio:
Czujnik MQ-9: $0.15 - 0.2$
Czujnik MQ-2: $0.05 - 0.15$
Przyjęto, że granicą wysłania notyfikacji do urządzeń użytkownika jest przekroczenie progu 0.3. Wartości te to znormalizowane dane z układu przetwornika AC, który jak już wcześniej wspomniałem wykrywa 1024 stany. Odczytywane wartości bezpośrednio na wyjściu cyfrowym przetwornika MCP3008 dla czujnika MQ-9 w czystym powietrzu to około 170. Stąd $170/1024 = 0.166$. Wysłanie notyfikacji wiąże się z otrzymaniem wartości min. 308 bezpośrednio na wyjściu cyfrowym. Reszta to czujniki cyfrowe, które informują m.in. o wykryciu ognia i wykryciu ruchu. W przypadku detekcji zagrożenia wysyłają one na wyjście stan niski i utrzymują go przez kilka sekund. W kodzie jednak wykonujemy instrukcje negacji tak, aby stan wysoki informował o niebezpieczeństwie a stan nisko reprezentował jego brak. Na obu czujnikach znajduje się potencjometr, za pomocą którego dowolnie można ustawiać czułość czujnika. Bezpośredni odczyt danych z czujników następuje nieprzerwanie co 2 sekundy. Nie należy obawiać się, że czujnik ruchu nie wykryje zagrożenia gdyż nie zostanie odczytany w poprawnym czasie, ponieważ utrzymuje on stan wysoki przez 5 sekund po wykryciu ruchu. Oprogramowanie oprócz monitorowania danych z czujników wysyła je także do bazy danych Firebase. Dzięki temu mamy podgląd wszystkich danych z każdego z pomieszczeń w czasie rzeczywistym. Dodatkowo w przypadku wykrycia zagrożenia czyli przekroczeniu progu o którym mowa wyżej wysyłamy push notyfikacje do wszystkich urządzeń użytkownika i zapisujemy wysłaną notyfikację w bazie danych Django. Dzięki zapisywaniu danych jesteśmy w stanie odtworzyć całą historię wydarzeń w systemie. Aby zapewnić wydajny i pewny system bezpieczeństwa przy otrzymaniu wysokich wartości na czujnikach i wysłaniu notyfikacji zapisujemy timestamp zdarzenia. Dzięki temu nie ma możliwości zasypania właściciela miliardem informacji gdyż każda kolejna notyfikacja zostanie wysłana po min. 10 min od poprzedniej przy założeniu, że nadal stan na czujniku jest wysoki.

Obsługa wideo

Opis procesu, opis Nginxa, opis RTMP i HLS

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

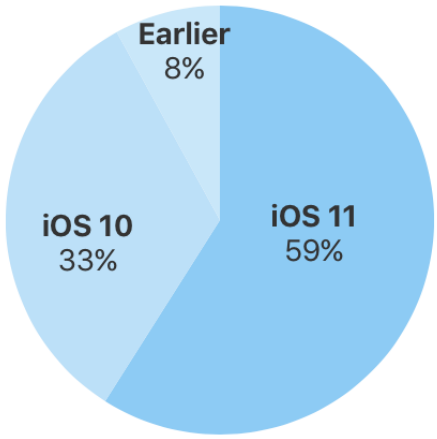
Aplikacje klienckie

Aplikacja Android

Opis Androida, screenshoty

Aplikacja iOS

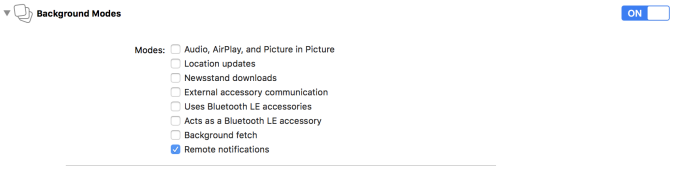
Aplikacja przeznaczona jest na urządzenia z systemem operacyjnym iOS od wersji 10.0. Nie wspiera ona wcześniejszych wersji ze względu na nowe funkcje, które Apple wprowadziło wraz z pojawieniem się iOS 10.0 (m.in. klasa UNUserNotificationCenter). Jednak jak wynika z wykresu niżej (numer rysunku) 92% wszystkich obecnych użytkowników tego systemu jest w stanie zainstalować aplikację a liczba ta z czasem stale rośnie. Aplikacja wspiera zarówno telefony komórkowe iPhone jak i tablety iPad. Napisana jest w stosunkowo nowym języku Swift (zaprezentowany przez



Rysunek 5.1: Udziały wersji systemu iOS w rynku

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. Programowanie reaktywne związane 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 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 traciliśmy 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 poprawie pracy 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 Mode"i uruchomienie konkretnej aktywności, którą chcielibyśmy wykonywać. Lista dozwolonych czynności jest ograniczona (rys x). Próba oszukania i wykonywania innej aktywności



Rysunek 5.2: Tryby pracy w tle

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)
```

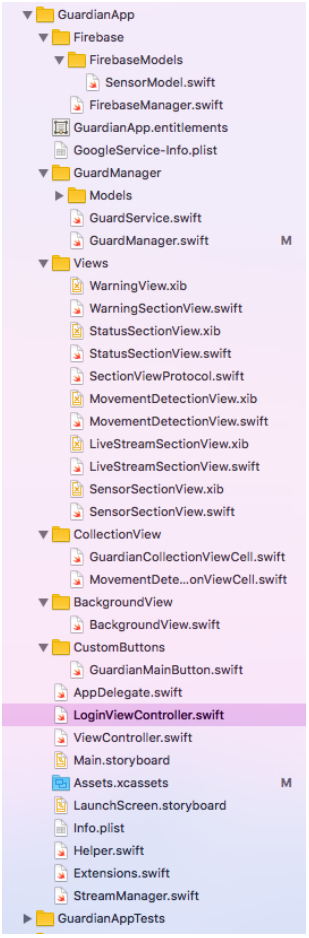
Zmienna 'inBackground', która jest zmienną obserwowalną, ustawiana jest w oddzielnej klasie AppDelegate na wartość true kiedy aplikacja przechodzi w tryb pracy w tle i na wartość false w przeciwnym wypadku. Klasa, w której wywoływany jest funkcja 'subscribe' jest obserwatorem tej zmiennej. Kod wewnątrz funkcji subscribe uruchamiany jest przy każdej zmianie wartości zmiennej 'inBackground' i wznawia ponownie stream po każdym ponownym uruchomieniu programu. Programowanie funkcjonalne natomiast wykorzystane jest w miejscach gdzie konieczne jest przekształcenie danych do innej postaci:

```
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 funkcji: posortowano je malejąco według daty, przefiltrowano w taki sposób aby wybrać tylko te o typie 'PIRSensor' czyli tylko 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. 5.2) podzielono na kilka osobnych, logicznych części. Folder Firebase zawiera model bazy danych czujników, które trzymane są na serwerach Firebase. W folderze GuardManager znajdują sie elementy odpowiedzialne za komunikację REST-ową z serwerem Django i również 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. W folderze GuardianAppTests napisane zostały testy jednostkowe, które sprawdzają poprawność przekształcania danych typu JSON 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 zapewniamy, że zawsze odwołujemy się do tego samego obiektu. Instalacja zewnętrznych bibliotek odbywa się za pomocą CocoaPods. Jest to menadżer

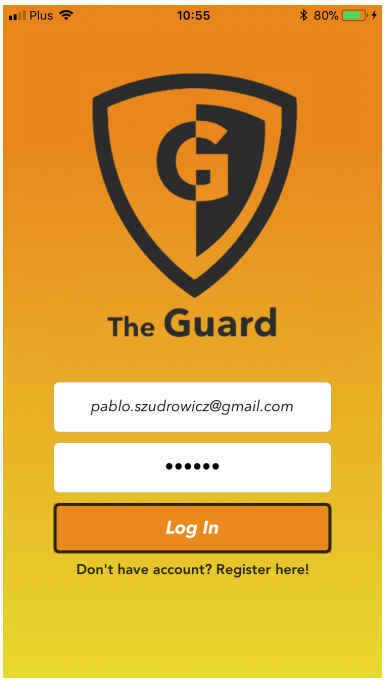


Rysunek 5.3: Struktura aplikacji

zależności dzięki któremu szybko możemy wyszukać i zainstalować wymagane oprogramowanie. Wszystkie użyte zależności przedstawia poniższy listing:

```
pod 'Moya'
pod 'MBProgressHUD', '~> 1.0'
pod 'RxSwift',    '~> 4.0'
pod 'RxCocoa',    '~> 4.0'
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'a do wcześniej zdefiniowanego modelu. MBProgressHUD umożliwia wyświetlanie ekranu ładowania danych 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 5.2). Po prawidłowej autoryzacji danych użytkownika wprowadzonych podczas logowania



Rysunek 5.4: Ekran logowania

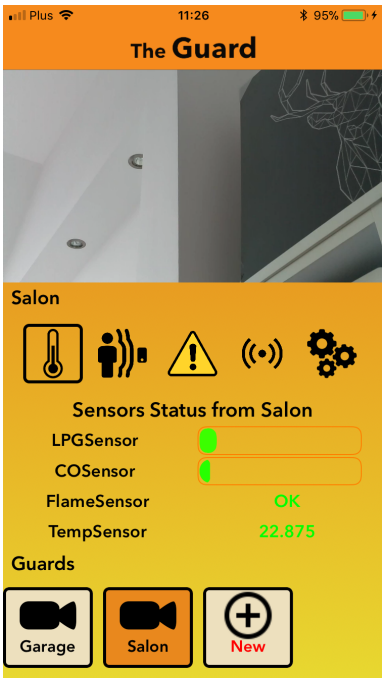
ukaże się nam główny widok aplikacji. W górnej części mamy do wyboru 5 sekcji: sekcja czujników, sekcja historii notyfikacji, sekcja zagrożeń przy wykryciu ruchu, sekcja streamu na żywo, sekcja ustawień. Wszystkie te sekcje dotyczą konkretnego urządzenia wybranego w pasku na dole ekranu. Przy pierwszym uruchomieniu nie będziemy posiadali żadnych urządzeń przypisanych 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 jednoznacznie urządzenie. Po chwili dodany Guard będzie widoczny w na liście.

Sekcja czujników: Jest to jedna z najważniejszych sekcji aplikacji (rys 5.3). 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 a nie RGB, dzięki temu 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 wyżej 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 5.4). Po zaznaczeniu interesującego nas daty reprezentującej moment wystąpienia zagrożenia i wybranu przycisku 'preview' prezentowana jest informacja o miejscu niebezpieczeństwa i jego rodzaju.

Sekcja ustawień: Ustawienia dotyczące zaznaczonego na dole ekranu urządzenia (rys 5.5). Użytkownik ma możliwość zmiany nazwy urządzenia, które zazwyczaj reprezentuje miejsce, w którym znajduje się stacja pomiarowa. Istnieje również możliwość uzbrojenia i wyłączenia konkretnego czujnika. Sprowadza się to do tego, że w przypadku zaznaczenia opcji "Disarmed"użytkownik

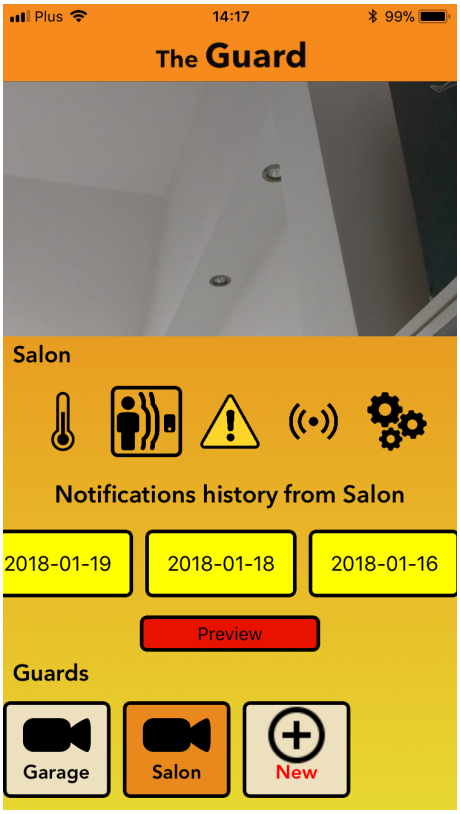


Rysunek 5.5: Sekcja czujników

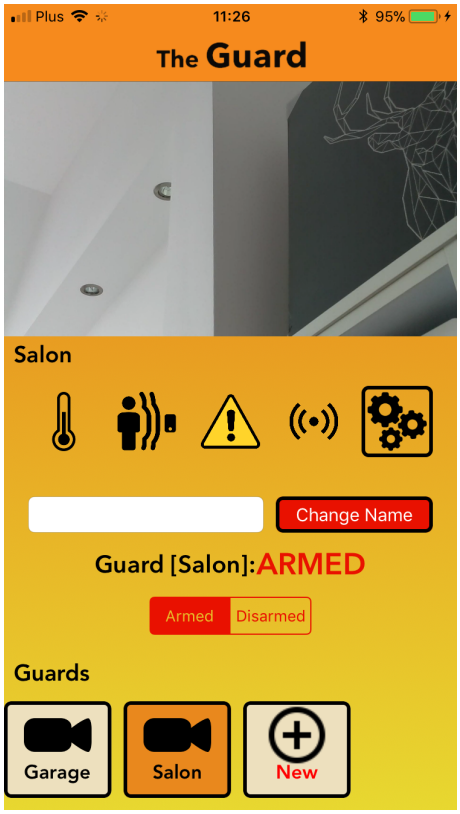
nie będzie otrzymywał kolejnych notyfikacji o zagrożeniach. Opcja ta może okazać się przydatna w momencie uszkodzenia któregoś z modułów i tym samym błednych danych wysyłanych z czujników. Przeprowadzono kilka testów aplikacji pod pełnym obciążeniem za pomocą programu Instruments. Szczególnie interesująco przedstawia się zużycie sieci podczas streamu obrazu. Widać, że w ciągu jednej minuty pobrano 6,61MB a wysłano jedynie 24,11Kb. Obraz pobierany jest tylko wtedy kiedy aplikacja jest aktywna. Nie musimy obawiać się, że stream jest aktywny, kiedy nie korzystamy z programu. 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 zużycia procesora. Te jednak są niewielkie i wynoszą odpowiednio 25MB pamięci RAM i średnio 1 procent zużycia procesora. Testy przeprowadzono na iPhonie 6S i iPadzie Pro.

Aplikacja internetowa

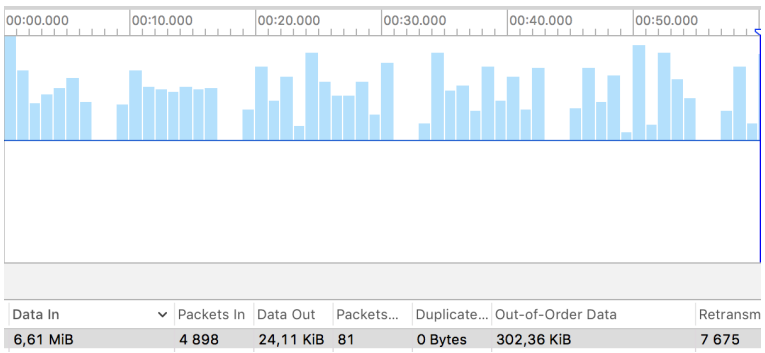
Opis weba, screenshoty



Rysunek 5.6: Sekcja historii notyfikacji



Rysunek 5.7: Sekcja ustawień



Rysunek 5.8: Zużycie sieci podczas streamu

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`