



AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA ELEKTRONIKI

PRACA DYPLOMOWA INŻYNIERSKA

Mikroprocesorowy system alarmowy sterowany smartfonem

Smartphone-controlled microprocessor based alarm system

Autor:	<i>Michał Rafałowski</i>
Kierunek studiów:	Elektronika i Telekomunikacja
Typ studiów:	<i>Stacjonarne</i>
Opiekun pracy:	<i>dr hab. inż. Ernest Jamro</i>

Kraków, 2019

Oświadczenie studenta

Uprzedzony(-a) o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2018 r. poz. 1191 z późn. zm.): „Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.”, a także uprzedzony(-a) o odpowiedzialności dyscyplinarnej na podstawie art. 307 ust. 1 ustawy z dnia 20 lipca 2018 r. Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.) „Student podlega odpowiedzialności dyscyplinarnej za naruszenie przepisów obowiązujących w uczelni oraz za czyn uchybiający godności studenta.”, oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście i samodzielnie i nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

Jednocześnie Uczelnia informuje, że zgodnie z art. 15a ww. ustawy o prawie autorskim i prawach pokrewnych Uczelni przysługuje pierwszeństwo w opublikowaniu pracy dyplomowej studenta. Jeżeli Uczelnia nie opublikowała pracy dyplomowej w terminie 6 miesięcy od dnia jej obrony, autor może ją opublikować, chyba że praca jest częścią utworu zbiorowego. Ponadto Uczelnia jako podmiot, o którym mowa w art. 7 ust. 1 pkt 1 ustawy z dnia 20 lipca 2018 r. – Prawo o szkolnictwie wyższym i nauce (Dz. U. z 2018 r. poz. 1668 z późn. zm.), może korzystać bez wynagrodzenia i bez konieczności uzyskania zgody autora z utworu stworzonego przez studenta w wyniku wykonywania obowiązków związanych z odbywaniem studiów, udostępniać utwór ministrowi właściwemu do spraw szkolnictwa wyższego i nauki oraz korzystać z utworów znajdujących się w prowadzonych przez niego bazach danych, w celu sprawdzania z wykorzystaniem systemu antyplagiatowego. Minister właściwy do spraw szkolnictwa wyższego i nauki może korzystać z prac dyplomowych znajdujących się w prowadzonych przez niego bazach danych w zakresie niezbędnym do zapewnienia prawidłowego utrzymania i rozwoju tych baz oraz współpracujących z nimi systemów informatycznych.

.....
(czytelny podpis studenta)

Spis treści

1. Wstęp	5
2. Istniejące na rynku rozwiązania	6
2.1. Satel	6
2.2. Risco	7
2.3. SimplySafe	8
3. Budowa systemu alarmowego	10
3.1. Cel oraz założenia projektu.....	10
3.2. Raspberry Pi.....	10
3.2.1. Początki oraz fenomen Raspberry Pi.....	10
3.2.2. Parametry Raspberry Pi 3B+ [18]:.....	11
3.2.3. System operacyjny	13
3.2.4. Wybór Raspberry Pi	14
3.3. Sensor ruchu.....	14
3.4. Sygnalizacja dźwiękowa.....	17
3.5. Budowa systemu alarmowego	19
4. Oprogramowanie	20
4.1. Język wysokopoziomowego programowania – Python 3.....	20
4.2. Moduł FLASK	21
4.2.1. Aplikacja webowa	21
4.2.2. „Framework” FLASK	21
4.3. HTML	22
4.4. Moduł RPi.GPIO	23
4.5. Ekran logowania.....	24
4.6. Główny algorytm.....	24
4.7. Powiadomienia i archiwizacja stanów.....	26
4.7.1. Powiadomienia.....	26

4.7.2.	Archiwizacja stanów	27
4.8.	Automatyczne uruchamianie skryptu	28
4.9.	Interfejs użytkownika	29
4.9.1.	Ekran logowania.....	30
4.9.2.	Ekran główny (Uzbrojony / Rozbrojony)	31
4.9.3.	Ekrany stanów.....	32
4.9.4.	Ekran wyłączenia	33
5.	Przykład użycia	34
6.	Wnioski oraz napotkane problemy.....	35
6.1.	Wnioski z projektu alarmu.....	35
6.2.	Napotkane problemy oraz ich rozwiązania.....	35
6.3.	Potencjalna rozbudowa systemu	36
7.	Bibliografia	38
8.	Zawartość dołączonej płyty CD	43

1. Wstęp

Żyjemy w czasach, kiedy decydujemy się na automatyzację wielu czynności ze względu na wygodę oraz oszczędność pieniędzy. Często z urządzeniami komunikujemy się zdalnie poprzez Internet, co czyni nasze urządzenia typu Internet of Things (IOT). Termin ten dosłownie oznacza „Internet rzeczy”, czyli sieć połączonych ze sobą urządzeń, które zbierają i przetwarzają informacje. Matthew Evans, kierownik programowy IOT w TechUK, w wywiadzie dla serwisu WIRED określił rolę IOT jako „dostarczanie możliwości, abyśmy byli bardziej efektywni w tym co robimy, oraz aby oszczędzić nasz czas, pieniądze oraz emisję danego procesu” [1].

Rynek inteligentnych domów (ang. Smart Home) w 2018 roku szacowany był na 31,4 miliarda dolarów amerykańskich, a wartość ta na rok 2023 jest prognozowana na 53,45 miliardów [2]. Smart Home jest oczywiście szerokim terminem określającym automatykę naszych domów, czego częścią są również systemy alarmowe. Rynek samych systemów alarmowych w 2018 roku osiągnął wartość 45,58 miliardów dolarów amerykańskich. Szacowany jest znaczny wzrost do 2023 roku, kiedy rynek ten osiągnie wartość aż 74,75 miliardów [3].

Jak widać ludzie coraz bardziej czują potrzebę bezpieczeństwa, ale też potrzebę wygody w użytkowaniu alarmów, dlatego postanowiłem stworzyć swój własny system alarmowy w oparciu o dwie popularne na rynku technologie, czyli Raspberry Pi oraz język Python 3.

2. Istniejące na rynku rozwiązania

Obecnie na rynku dostępne jest mnóstwo zdalnie sterowanych systemów alarmowych w różnych przedziałach cenowych. Można znaleźć wątpliwej jakości bezmarkowe modele za ok. 50 zł. Dostępne są również pełne systemy z kilkoma czujnikami w zestawie oraz ekranem dotykowym, pozwalającym na wygodne sterowanie w cenie od ok. 550 zł. Tańsze modele oczywiście kuszą swoją ceną, jednak idzie za nią dużo gorsza jakość wykonania oraz awaryjność. Po drugiej stronie znajdują się firmy takie jak Satel i Risco, które podchodzą profesjonalnie do ochrony i zabezpieczeń.

2.1. Satel

W ofercie Satela znajdują się 2 zestawy alarmowe: ABAX i MICRA. System ABAX został zaprojektowany z myślą o centralach Integra oraz Versa, będącymi również produktami Satela (ceny central zaczynają się od około 300 zł i rosną w zależności od ilości wejść [4]). Producent szczyci się poziomem zabezpieczeń niegdyś dostępnym tylko w systemach przewodowych dzięki dwukierunkowości, wykorzystywanej do odpytywania podłączonych urządzeń. Warty uwagi cechami są również duży zasięg systemu (do 500m w otwartej przestrzeni) oraz energooszczędność (aż do 3 lat na jednym zestawie baterii) [5].



Rys. 1. System ABAX [5]

System MICRA natomiast cechuje prostota montażu i konfiguracji. Jak podaje sam producent, system ten jest przeznaczony do ochrony małych obiektów, a dzięki jego budowie zapewnia możliwość przenoszenia całej instalacji w tymczasowe lokacje. Całość instalacji zajmuje około godziny. Sterować alarmem można przy pomocy funkcji SMS [6].



Rys 2. System MICRA [6]

2.2. Risco

Agility 3 jest dwukierunkowym systemem bezprzewodowym przeznaczonym do ochrony małych obiektów. System ten pozwala na kontrolę poprzez przeglądarkę internetową oraz aplikację na smartfon, gdzie możliwy jest podgląd na żywo z kamer, przy pomocy funkcji RISCO CLOUD opartej na chmurze. Dodatkowo system ten wykorzystuje technologię VUpoint P2P, która zapewnia podgląd z kamer w czasie rzeczywistym oraz zapisywanie krótkich fragmentów nagrań, podczas których doszło do nieuprawnionego wtargnięcia do obiektu [7].



Rys. 3. System Agility 3 [7]

LightSYS 2 przeznaczony jest do dużych obiektów. Wykorzystuje on czujki oznaczone jako GRADE 3 pozwalające na jak największą ochronę i odporność na błędne wykrycia nawet w najbardziej nieprzewidywalnych środowiskach [8]. Tak samo jak Agility 3 wykorzystuje on technologię VUpoint P2P do transmisji na żywo obrazu z kamer. Nazywany

jest alarmem hybrydowym ze względu na to, że akcesoria (na przykład czujniki) w razie potrzeby można podłączyć przewodowo, a część bezprzewodowo dwukierunkowo [9].



Rys. 4. System LightSYS 2 [9]

Trzecim modelem w ofercie RISCO jest system WiComm-Smart Interactive Wireless Security, który według producenta łączy bardzo wysoką jakość z prostą instalacją i niskimi kosztami. Tak samo jak oba powyższe modele, wykorzystuje on VUpoint P2P [10].



Rys. 5. System WiComm-Smart Interactive Wireless Security [10]

2.3. SimplySafe

Jednym z najciekawszych systemów alarmowych, z którymi się spotkałem jest rozwiązanie zaproponowane przez firmę SimplySafe. Ich system o tej samej nazwie zwraca uwagę swoim nowoczesnym projektem, wyróżniając się na tle konkurencji. Centrala przypomina swoim wyglądem popularnych ostatnio wirtualnych asystentów, jak na przykład Alexa firmy Amazon. Poza tym jest ona cały czas podłączona do sieci, w celu pobierania coraz to nowszych aktualizacji poprawiających funkcjonowanie całego systemu. W najprostszym zestawie można również znaleźć 2 czujki ruchu, 5 czujek otwarcia oraz kamerę. Alarmem można sterować poprzez dołączony panel z ekranem lub poprzez aplikację mobilną.

Prostszym rozwiązaniem jest też mały pilot w postaci breloka do kluczy pozwalający na uzbrajanie/rozbrajanie alarmu oraz na szybkie wezwanie pomocy [11].



Rys. 6. System SimplySafe [11]

3. Budowa systemu alarmowego

3.1. Cel oraz założenia projektu

Celem projektu jest wykonanie zdalnie sterowanego systemu alarmowego opartego na mikroprocesorze. Sterowanie odbywa się poprzez telefon komórkowy. Centralą systemu jest mikrokomputer Raspberry Pi 3B+, na którym uruchomiony jest serwer z aplikacją. W przypadku wykrycia ruchu zostaje zaświecona odpowiednia dioda LED i odpala się syrena (brzęczyk, ang. - buzzer). W tym samym momencie użytkownik otrzymuje zdalne powiadomienie.

3.2. Raspberry Pi

Główną częścią projektu jest mikrokomputer Raspberry Pi 3 model B+, najnowsza iteracja mikrokomputera wielkości karty kredytowej.



Rys. 7. Raspberry Pi 3B+

3.2.1. Początki oraz fenomen Raspberry Pi

Pierwszy zamysł na stworzenie Raspberry Pi zrodził się w 2006 roku w głowie Ebena Uptona, który właśnie kończył swoje studia doktorskie na uniwersytecie w Cambridge. Zafascynowany technologiami i sprzętem, zaczął zastanawiać się nad projektem urządzenia mającego pomóc w nauce programowania, tak jak to było w jego własnym przypadku, kiedy spędzał czas przed BBC Micro – komputerem, który posiadał w młodości [12].

Po licznych prototypach, pierwotna wersja ukazała się w lutym 2012 roku. Sprzęt okazał się ogromnym sukcesem, który do lutego 2015 roku sprzedał się w nakładzie 4.5 miliona sztuk. Rok później zostały ogłoszone prace nad drugą wersją sprzętu [13]. Po

licznych iteracjach sprzęt ten sprzedał się w nakładzie 19 milionów sztuk (dane z marca 2018 [14]) i doczekał się nawet budżetowej wersji Raspberry Pi Zero kosztującej 10\$.

Na przestrzeni lat Raspberry Pi urosło do rangi światowego fenomenu. Każdy, kto interesuje się technologiami, czy też studiuje kierunki techniczne spotkał się chociaż raz z nazwą tego mikrokomputera. Utworzona została działalność Raspberry Pi Foundation zrzeszająca dzieci jak i dorosłych w celu rozwijania swoich pasji informatycznych, czy też elektronicznych. Fundacja skupia się na mnóstwie inicjatyw takich jak kursy programowania dla najmłodszych, zajęcia edukacyjne dla nauczycieli, magazyny poruszające kwestie związane z urządzeniem, czy przede wszystkim liczne materiały instruktażowe [15].

Tysiące ludzi zaczęło dzielić się swoimi projektami i rozwiązaniami, tworząc w ten sposób ogromną społeczność. W serwisie internetowym Reddit (będącym swego rodzaju największym na świecie forum, które w 2018 mogło pochwalić się średnio 330 milionami aktywnych użytkowników miesięcznie [16]) istnieje grupa poświęcona Raspberry Pi, która na dzień 10.01.2019 może pochwalić się 827 tysiącami użytkowników [17].

Wybrany przeze mnie model to 3B+, będący najnowszą wersją, która pojawiła się w 2018 roku. Głównymi różnicami względem poprzedniego modelu 3B są wsparcie dla dwuzakresowego wifi 2,4 i 5 GHz, wsparcie dla bluetooth 4.2, oraz wydajniejszy procesor 1,4 GHz (200 MHz więcej niż u poprzednika).

3.2.2. Parametry Raspberry Pi 3B+ [18]:

- Główny układ scalony Broadcom BCM2837B0:
Niemal identyczny do swojego poprzednika BCM2837A0, lecz wyróżniający się lepszym rozproszaniem ciepła oraz poprawioną optymalizacją Power Integrity (integralność energii), czyli metody sprawdzającej, czy podane napięcie jest odpowiednie do potrzeb urządzenia [19]. Obie te rzeczy niosą za sobą zwiększone taktowanie procesora Cortex [20].
- Procesor Cortex-A53 z 64-bitowym rdzeniem ARMv8 (taktowanie 1.4GHz):
Następca modelu Cortex-A7. Używany typowo w smartfonach oraz w systemach wbudowanych. W porównaniu do poprzedniego modelu poprawiono wydajność [21].

- 1GB LPDDR2 SDRAM:

Rodzaj pamięci wykorzystywanej w urządzeniach mobilnych. Low Power DDR2 jak sama nazwa wskazuje cechuje się mniejszym poborem energii.

- Bezprzewodowa dwuprzepustowa sieć lokalna (dwuzakresowa sieć WiFi):

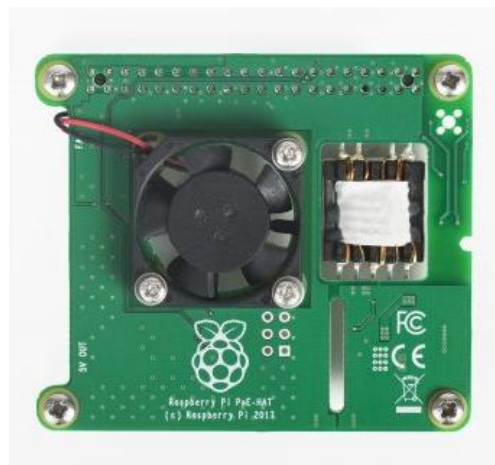
Do dyspozycji otrzymujemy dwie częstotliwości 2.4GHz oraz 5GHz oparte na standardzie IEEE 802.11.b/g/n/ac. Częstotliwość 2.4 GHz jest wykorzystywana w życiu codziennym częściej (przykładowo w kuchenkach mikrofalowych) co wpływa na stan sieci (prędkość do 450 Mb/s). Częstotliwość 5 GHz jest mniej używana, przez co oferuje większą prędkość transmisji (do 1300 Mb/s) i większą stabilność. Traci niestety na większym poborze energii oraz na zasięgu z powodu słabej penetracji obiektów takich jak ściany. Sieć 5GHz oferuje także więcej kanałów, przez co więcej użytkowników może być jednocześnie do niej podłączonych bez odczuwania obciążenia [22][23].

- Bluetooth 4.2:

Wersja 4.2 różni się od poprzednika znacząco zwiększoną szybkością transferu (z 270 kb/s aż do 800 kb/s). Dodatkowo wersja 4.2 wprowadza poprawki w kwestii protokołu Bluetooth Low Energy (BLE) zapewniając większą energooszczędność [24].

- Gigabit Ethernet po USB 2.0 oraz PoE:

Maksymalna przepustowość rzędu 300 Mb/s. Funkcja Power over Ethernet (PoE) pozwala na zasilanie urządzenia poprzez RJ45 (wymagana dodatkowa nakładka).



Rys. 8. Nakładka: Raspberry Pi PoE HAT [25]

- 40 wyprowadzeń (ang. pin) do użytku:
 2 piny zasilania 5V
 2 piny zasilania 3,3V
 8 pinów uziemienia
 2 piny EEPROM Data i EEPROM Clock używane przez dodatkowe nakładki
 26 pinów wejście-wyjście ogólnego przeznaczenia GPIO
- Port HDMI
- 4 porty USB 2.0
- Port CSI do podłączenia kamerki
- Port DSI do podłączenia ekranu dotykowego
- Wyjście stereo Jack 3.5mm
- Port na kartę Micro SD
- Wejście na zasilanie 5V/2.5A DC

3.2.3. System operacyjny

Na kupionej przeze mnie karcie pamięci dostępny był od początku NOOBS (New Out Of Box Software), czyli darmowy, uproszczony kawałek kodu pozwalający na wygodną instalację wybranego systemu operacyjnego. Na tej samej karcie pamięci włożonej do płytki znajdował się również system operacyjny Raspbian, oparty na systemie Debian i dostosowany pod urządzenia z rodziny Raspberry Pi. Jest on całkowicie darmowy i rozwijany do dziś przez użytkowników, oraz grupę twórców określanych jako „Project Debian” [26].

Debian jest darmowym systemem operacyjnym wykorzystującym jądro Linuxa [26]. Jądem systemu operacyjnego nazywamy jego część, która odpowiada za główne funkcje takie jak uruchamianie aplikacji, zarządzanie całym systemem, czy całą warstwę wizualną.

Po zainstalowaniu systemu do naszej dyspozycji dostępne są oprogramowania takie jak Mathematica, Wolfram, BlueJ, czy też Python w wersjach 2 i 3.

3.2.4. Wybór Raspberry Pi

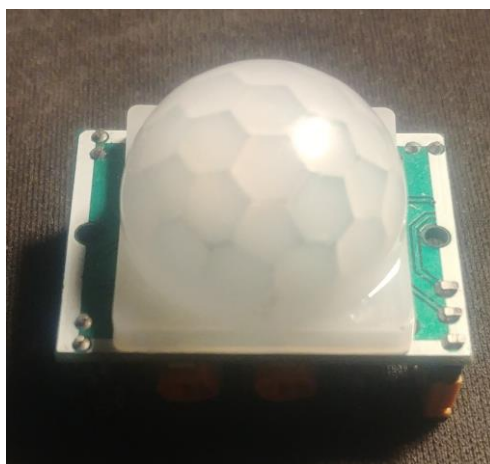
Zdecydowałem się na użycie Raspberry ze względu na szeroki wachlarz zastosowań owego urządzenia. Od kilku lat jest ono wykorzystywane do różnorodnych projektów takich jak emulacja starych gier, czy też układ do rozwiązywania kostki Rubika. Posiada on również wbudowany moduł Wi-Fi, który był konieczny do pracy mojego systemu.

Przy projekcie alarmu użyte zostały zewnętrzne wyprowadzenia:

- 2 piny zasilania 5V
- 3 piny uziemienia
- 3 piny GPIO (2 z nich ustawione jako wyjście i jedno jako wejście)

3.3. Sensor ruchu

Drugim z kolei najważniejszym element systemu alarmowego jest sensor ruchu. Wybrany przeze mnie modelem jest czujnik PIR HC-SR501.



Rys. 9. Czujnik PIR HC-SR501

Jego parametry prezentują się następująco:

- Napięcie zasilania DC od 5V do 20V
- Zasięg 7 metrów
- Kąt widzenia 100°

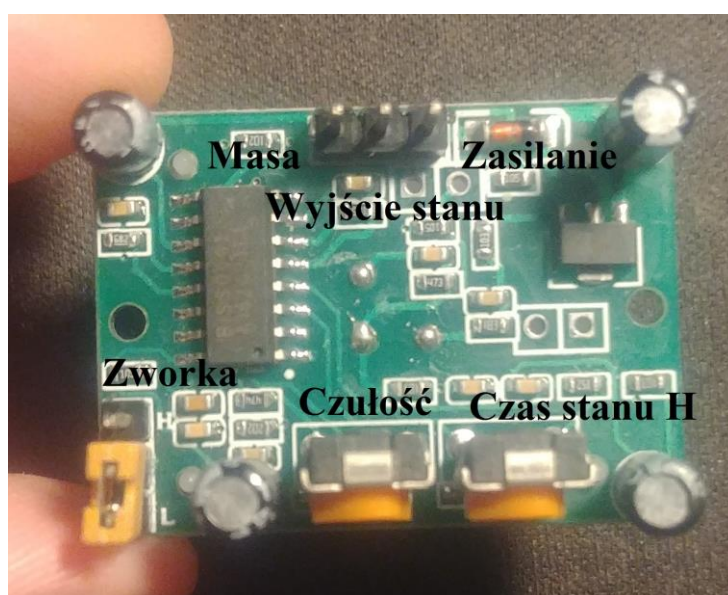
PIR oznacza „Passive Infrared”, co oznacza promieniowanie podczerwone, bo to właśnie ono jest wykorzystywane do wykrywania ruchu. „Passive”, czyli pasywność czujnika natomiast przejawia się tym, że on sam nie emituje energii aby wykrywać ruch w swoim polu widzenia. Sensor posiada wyprowadzenia na zasilanie od 5V do 20V , na masę, oraz na stan czujki (stan wysoki 3.3V w momencie wykrycia ruchu).

Dodatkowo posiada też 2 gałki [27]:

- Regulacja czułości od 3 do 7 metrów
- Regulacja czasu stanu wysokiego od 3 do 300 sekund

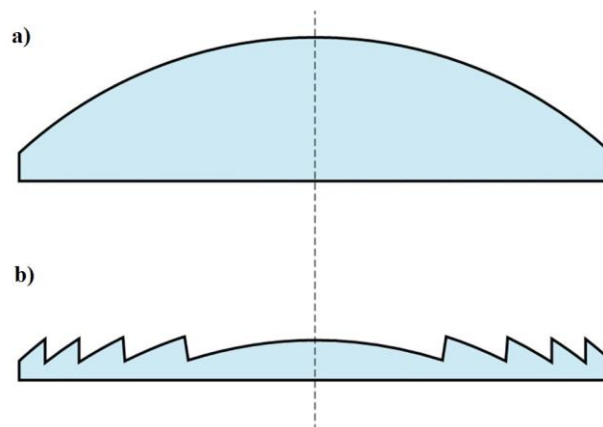
W lewym dolnym rogu znajduje się zworka pozwalająca zmienić tryb pracy czujki w zależności od tego, czy znajduje się na pozycji L czy H:

- L – Czujka wykrywa ruch raz, po czym stan wysoki utrzymuje się w zależności od ustawień gałki
- H – Czujka jest w stanie wysokim do momentu przerwania ruchu. Kiedy zabraknie ruchu czujka przestawia się na stan niski



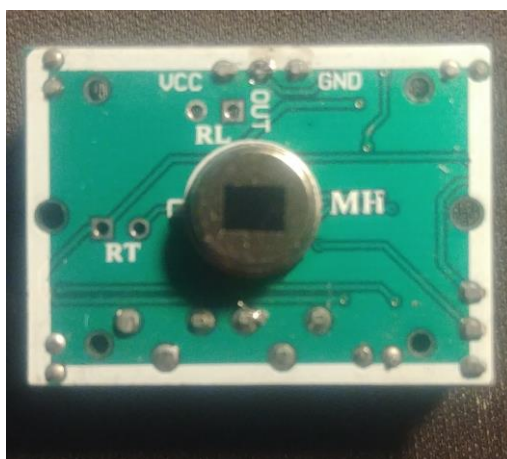
Rys. 10. Tył czujnika ruchu

Pierwszą rzeczą rzucającą się w oczy jest wykonana z plastiku kopuła czujki, na której można zauważyć sześciokątne wzory. Każdy z nich jest soczewką Fresnela, która ma specyficzną schodkową budowę. Taka budowa pozwala ograniczyć miejsce zajmowane przez soczewkę. Soczewka Fresnela podobnie jak pełna soczewka skupia światło i dodatkowo zapewnia zdecydowanie większy kąt widzenia czujnika promieniowania podczerwonego – piroelektryk jest oświetlany mniej więcej równomiernie, niezależnie od punktu wykrycia obiektu.



Rys. 11. a) Zwykła soczewka, b) Soczewka Fresnela [28]

Głównym elementem czujki jest element piroelektryczny. Jego cecha, czyli piroelektryczność polega na pojawianiu się ładunków elektrycznych w momencie zmiany temperatury elementu. Całość piroelektryka jest ukryta pod obudową chroniącą przed uszkodzeniami. Ma ona wycięte okienko wykonane z materiału przepuszczającego promieniowanie podczerwone.



Rys. 12. Piroelektryk

Zjawisko piroelektryczne po raz pierwszy zostało odkryte przez Holenderskiego fizyka Franza Aepinusa. Zaobserwował on pojawienie się ładunków elektrycznych na ogrzanym (poprzez ocieranie) kryształ z rodziny turmalin, a dokładniej to, że na jednej stronie kryształu pojawiają się dodatnie ładunki, a na drugiej ujemne [29]. Swoją pracę pokazał w 1756 roku przed Królewską Akademią Nauk w Berlinie [30].

W 1759 roku jego odkrycie potwierdził Benjamin Franklin przeprowadzając ten sam eksperyment [29]. Również w tym samym roku Angielski fizyk John Canton zauważył

odwrócenie polaryzacji kryształu pod wpływem ochłodzenia go [29]. Odkrył też, że nie tylko kamienie z rodziny turmalin są podatne na to zjawisko, ale też inne kamienie szlachetne. Dowiódł również, że kształt kryształu nie ma wpływu na jego własności. Dokonał tego poprzez rozbicie minerału na mniejsze kawałki i sprawdzeniu własności na każdym z osobna [31].

Nazwa „piroelektryk” została zaproponowana przez Szkota Davida Brewstera w 1824 roku w publikacji „Observations of Pyro-Electricity of Minerals” [30]. Wykonał on badania na kilku rodzajach kryształów przy pomocy urządzenia zaprojektowanego pod to zadanie [31].

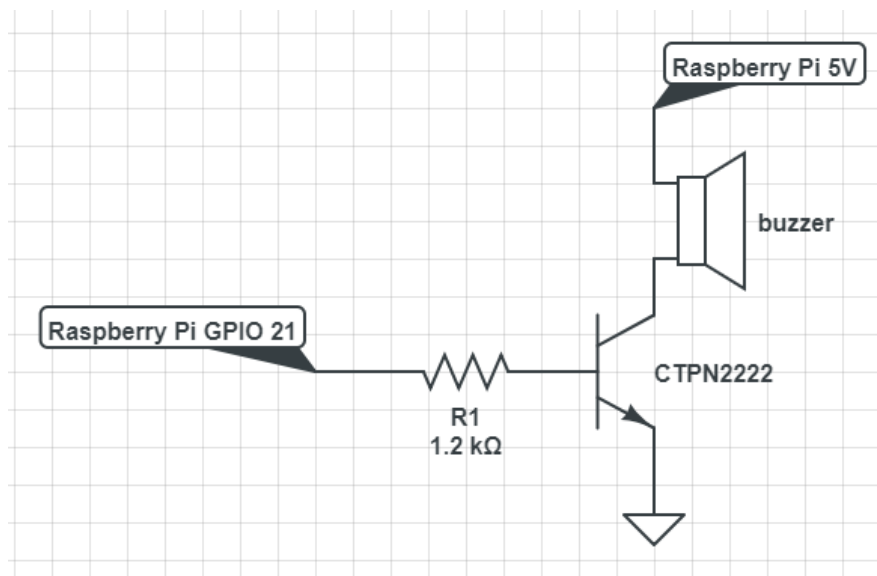
Piroelektryk zawiera przeważnie 2 elementy światłoczułe, które wykrywają taką samą ilość promieniowania podczerwonego. Kiedy obiekt pojawi się w zasięgu danego elementu, następuje zmiana wykrywanego ciepła, a co za tym idzie pojawia się różnica pomiędzy tymi elementami. Właśnie ta różnica przyczynia się do uruchomienia alarmu [32].

3.4. Sygnalizacja dźwiękowa

W momencie wykrycia ruchu, elementem wywołującym dźwięk jest 5 woltowy sygnalizator dźwiękowy - brzęczyk HCM1206X będący modelem aktywnym. Oznacza to, że niepotrzebne są urządzenia trzecie aby wygenerować dźwięk – wystarczy przyłożone do brzęczyka napięcie. Dzieje się tak, ponieważ w środku znajduje się generator przebiegu prostokątnego, przez co nie ma potrzeby generowania tego przebiegu poprzez mikrokontroler. HCM1206X jest elementem elektromagnetycznym. Jego działanie polega na wprowadzaniu w wibracje dysku pod wpływem pola magnetycznego wytworzonego przez cewkę [33].

Brzęczyk jest podpięty wraz z tranzystorem bipolarnym typu NPN o modelu CTPN2222 w konfiguracji klucza (Rys. 13). Zdecydowałem się na takie rozwiązanie z powodu przedziału zasilania brzęczyka (od około 4 lub 5V), kiedy to z pinu logicznego Raspberry Pi zapewnione jest jedynie 3,3V. Dodatkowo maksymalny prąd z takiego pinu to 16mA, a brzęczyk wymaga więcej (maksymalnie 30mA), przez co bezpośrednie podpięcie do Raspberry Pi mogłoby potencjalnie zepsuć płytkę.

Tranzystor CTPN2222 jest zamiennikiem popularnego modelu 2N2222. Jest to tranzystor małej mocy powszechnie stosowany w układach elektronicznych np. jako klucz. Za jego popularnością przemawiają mały rozmiar, cena oraz oczywiście wszechstronność [34]. Rezystor bazy został dobrany w celu ograniczenia prądu pobieranego z pinu Raspberry Pi.



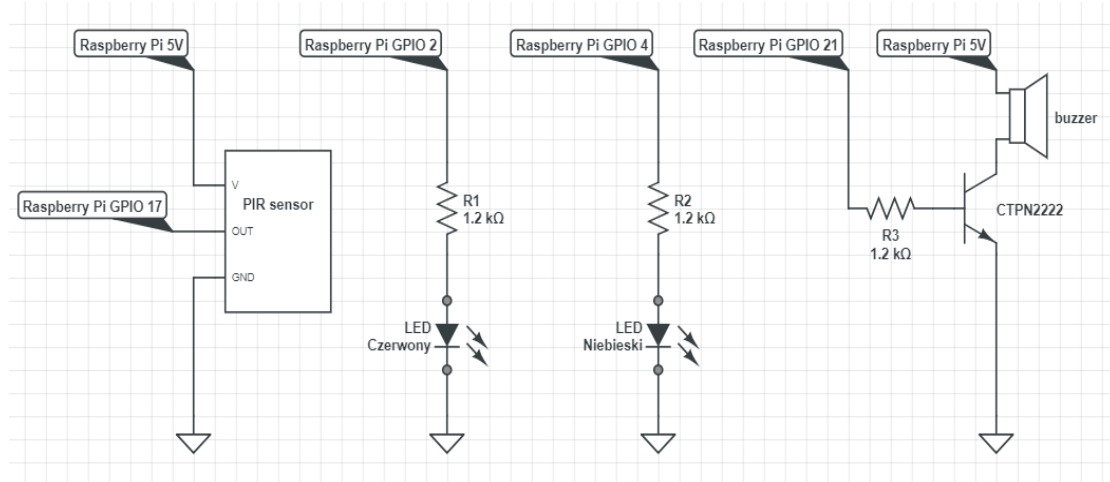
Rys. 13. Schemat podpięcia brzęczyka pod Raspberry Pi

Klucz działa w następujący sposób:

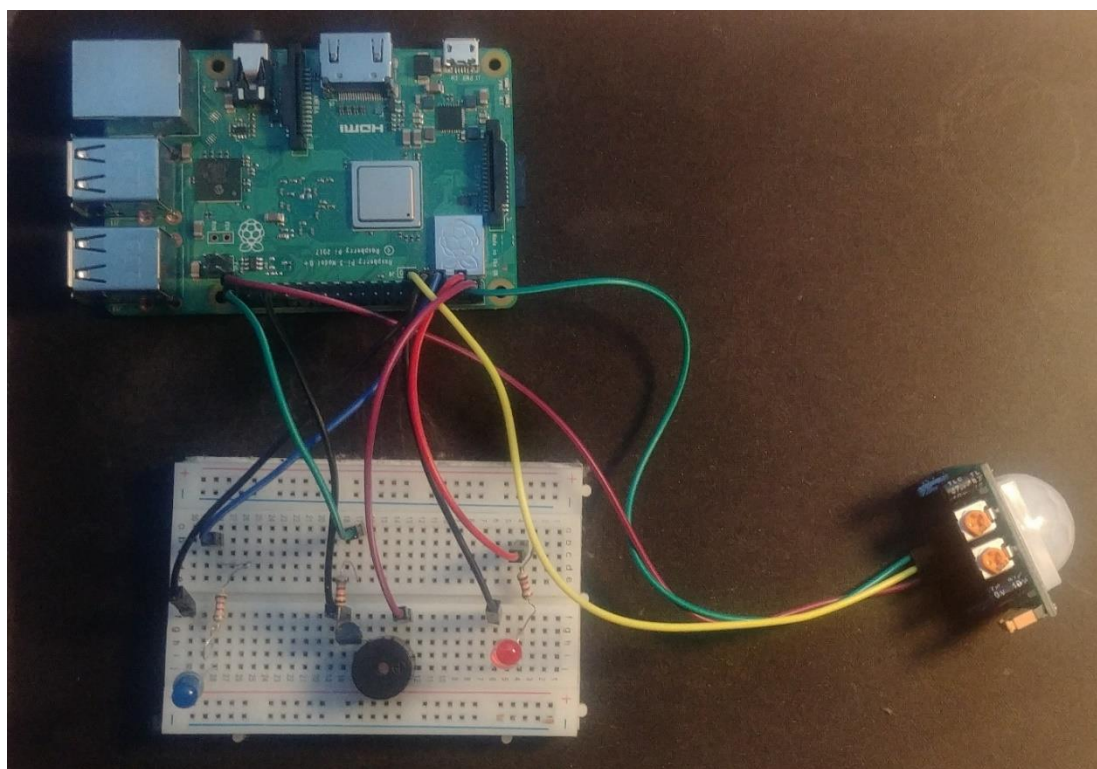
- Kiedy na wejściu tranzystora pojawia się napięcie, to przez bazę przepływa prąd. W tym momencie tranzystor NPN zaczyna przewodzić, pozwalając na przepływ większego prądu kolektora przez brzęczyk.
- W momencie, w którym napięcie na wejściu będzie w stanie logicznym niskim, a co za tym idzie, nie będzie prądu bazy, to tranzystor nie przewodzi, a prąd kolektora jest praktycznie równy zeru.

3.5. Budowa systemu alarmowego

Do budowy systemu oprócz wyżej wymienionych elementów zostały użyte również: płytki stykowa oraz czerwona i niebieska dioda LED podłączone do pinów przez rezystory 1.2kΩ.



Rys 14. Schemat elementów dołączonych do Raspberry Pi 3B+



Rys. 15. Zbudowany system alarmowy

4. Oprogramowanie

4.1. Język wysokopoziomowego programowania – Python 3

Kod realizujący system alarmowy został napisany w języku Python 3 (wersja 3.5.3), który jest wysokopoziomowym językiem. Aby przybliżyć co to oznacza, opisana zostanie hierarchia poziomów języków.

Najniżej w hierarchii jest język maszynowy. Kod binarny napisany w ten sposób nie wymaga kompilacji - jest on bezpośrednio interpretowany przez procesor. Niestety taki kod jest kompletnie nieprzejrzysty dla człowieka, dlatego też wprowadzono język Asembler [35].

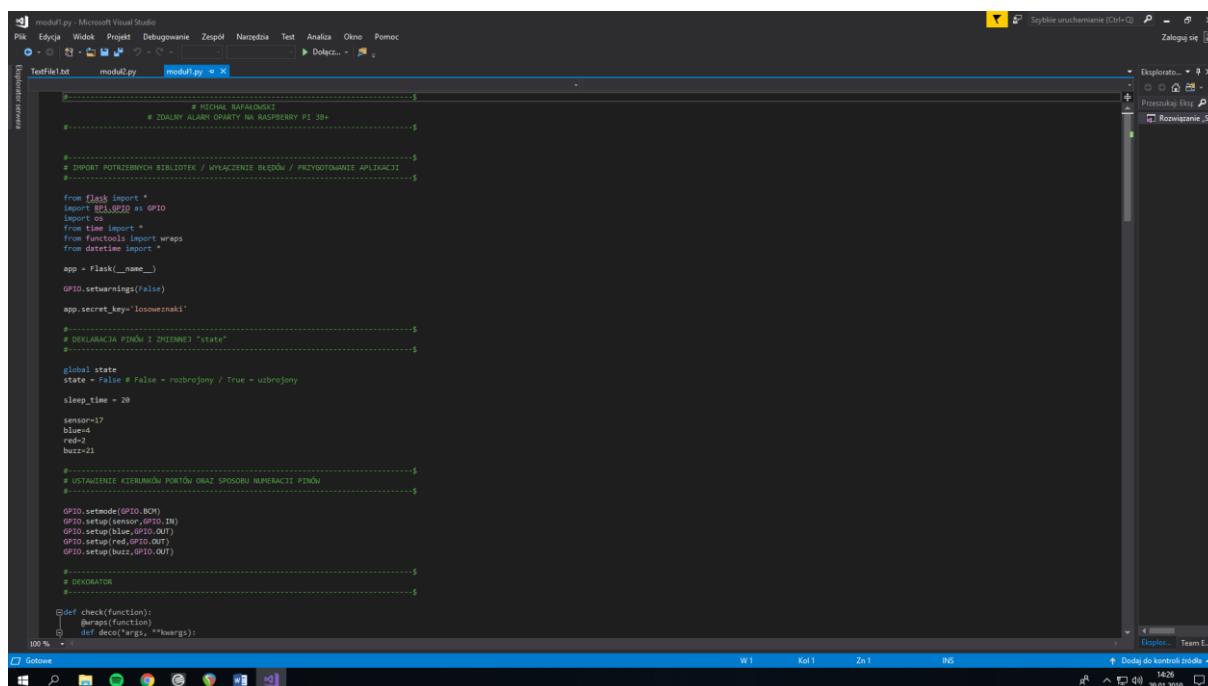
Język Asembler znajduje się wyżej w hierarchii od kodu maszynowego. Program napisany w tym języku nie może być bezpośrednio zinterpretowany przez urządzenie i wymaga kompilatora. W odróżnieniu od języka maszynowego Asembler wykorzystuje operacje i zmienne, co pozwala na zapisanie kodu nie tylko za pomocą długich ciągów zer i jedynek, ale także kluczowych zwrotów jak na przykład „ldi r16, 0x25”, co pozwala na bezpośrednie załadowanie zmiennej 25 w formacie heksadecymalnym do rejestru oznaczonego jako r16 [36].

Najwyżej w hierarchii są języki wysokiego poziomu. Ich składnia jest przystosowana w pełni do użytkownika, umożliwiając mu przejrzysty wgląd w kod. Logika kryjąca się za wysokopoziomowymi językami przypomina bardziej logikę ludzką, niż maszynę. Używane są słowa języka mówionego do określania nazw, czy też operatory arytmetyczne używane przy operacjach matematycznych. Raz napisany kod można odpalić na większości urządzeń, a nie jak w przypadku języków niskiego poziomu, gdzie programuje się pod konkretny procesor.

Zdecydowałem się na użycie Pythona ze względu na prostotę z jaką można programować układy wbudowane. Języki C/C++, będące standardem także zaliczają się do wysokopoziomowych języków, lecz są mniej intuicyjne oraz wymagają większego zaangażowania programisty niż Python, który jest bardziej przejrzysty oraz zapewnia wsparcie dla przydatnych modułów. Języki C/C++ różnią się też tym, że są językami kompilowanymi. Oznacza to, że napisany program jest tłumaczony na język maszynowy, kiedy to Python jest językiem interpretowanym, wykorzystującym interpreter do

bezpośredniego uruchomienia kodu, bez potrzeby tłumaczenia go na języki niższych poziomów.

Podczas pisania pracy został użyty Microsoft Visual Studio Community 2017 w wersji 15.9.5. z zainstalowanym dodatkowo pakietem roboczym „Opracowywanie zawartości w języku Python”.



Rys. 16. Podgląd kodu w Visual Studio 2017

4.2. Moduł FLASK

4.2.1. Aplikacja webowa

Aby użytkownik mógł wygodnie obsługiwać alarm, potrzebny jest interfejs zdalnej kontroli. W przypadku mojej pracy wybór padł na zdalną obsługę przez przeglądarkę internetową po stronie użytkownika oraz generowania stron HTML po stronie serwera (Raspberry Pi). Użytkownik uaktywnia system alarmowy poprzez aplikację internetową znajdującą się na serwerze skonfigurowanym na Raspberry Pi 3B+.

4.2.2. „Framework” FLASK

Aby stworzyć aplikację internetową potrzebny jest framework, czyli środowisko programistyczne zapewniające odpowiednie narzędzia i funkcje do tworzenia aplikacji. W przypadku niniejszej pracy szkielet aplikacji internetowej została zrealizowana przy pomocy biblioteki Flask, stworzonej pod Python 3. Jak piszą sami twórcy, jest to „microframework

dla Pythona, bazujący na Werkzeug, Jinja 2 i dobrych intencjach” [37]. Przedrostek „micro” użyty przez twórców do opisu biblioteki, odnosi się do faktu, iż nie jest to pełny framework, lecz narzędzie zapewniające minimum funkcji potrzebnych do uruchomienia i konfiguracji aplikacji [38].

Oczywiście funkcjonalności biblioteki Flask zostały rozszerzone przez twórców i społeczność na przestrzeni lat, zapewniając bardziej rozwinięte podejście do tworzenia aplikacji internetowych. W przypadku omawianej pracy wystarczył standardowy moduł Flask. Moduł ten jest intuicyjny w konfiguracji, lecz serwer dostępny jest jedynie w sieci lokalnej.

Działanie Flaska polega na wyświetlaniu podstron z użyciem HTML, a następnie przypisaniu tym stronom konkretnych funkcji, które zostaną uruchomione kiedy użytkownik wejdzie na nie. Dzieje się tak dzięki funkcji dekorującej `route()`. Po wykonaniu danych działań zostaje zwrócony arkusz HTML.

```
@app.route('/')          # dekorator
def DISARM():
    pass                  # działania funkcji
    return render_template('home.html') # zwrot danego arkusza HTML
```

Rys. 17. Przykładowa funkcja z użyciem Flask

4.3. HTML

Wszystkie podstrony aplikacji internetowej zostały napisane w języku HTML. Jako że ich struktura nie jest skomplikowana, postanowiłem zrezygnować z osobnych arkuszy stylów CSS na rzecz zawarcia wszystkich informacji dotyczących wyglądu w danym pliku HTML.

Każdy plik HTML rozpoczyna się od tych samych linii (Rys. 18). Jednej określającej kolor tła w zapisie heksadecymalnym (linia 6) oraz drugiej (linia 4), pozwalającej na skalowanie strony na urządzeniach mobilnych. Bez niej elementy byłyby zbyt małe i niewygodne w użytkowaniu.

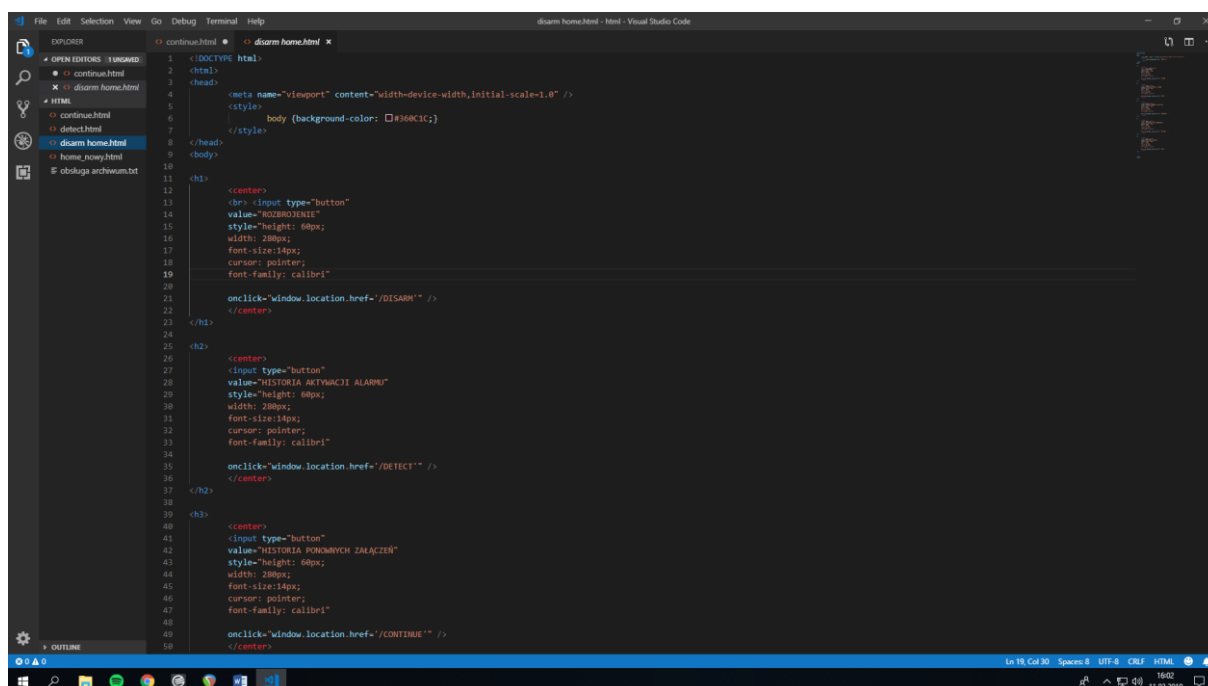
```

1  <!DOCTYPE html>
2  <html>
3  <head>
4      <meta name="viewport" content="width=device-width,initial-scale=1.0" />
5      <style>
6          body {background-color: #360C1C;}
7      </style>
8  </head>
9  <body>

```

Rys. 18. Początkowe linie arkuszy HTML

Do napisania wszystkich arkuszy posłużył program Visual Studio Code w wersji 1.31 od firmy Microsoft. Praca w takim programie jest wygodniejsza niż w zwykłym edytorze, za sprawą np. wygodnego i szybkiego podglądu poprawek i rezultatów w przeglądarce.



Rys. 19. Podgląd kodu w Visual Code Studio

4.4. Moduł RPi.GPIO

Drugim najważniejszym modulem użytym przy projekcie systemu alarmowego był moduł RPi.GPIO zapewniający wsparcie dla portów wyjścia/wejścia Raspberry Pi. Był on dostępny od samego początku, przez co nie wymagał dodatkowego pobierania. Moduł ten w prosty sposób pozwala na obsługę wyprowadzeń Raspberry Pi oraz zdarzeń wyzwalanych z boczem narastającym, opadającym lub obydwooma.

4.5. Ekran logowania

Każda podstrona jest dodatkowo przykryta dekoratorem funkcji logowania. Oznacza to, że użytkownik nie będzie mieć do niej dostępu, aż do momentu wprowadzenia hasła. Każda próba wczytania danej podstrony kończy się zwróceniem ekranu logowania. Do tego celu używany jest moduł `functools`, a dokładniej jego funkcja `wraps()`. Pozwala ona na sprawdzenie, czy użytkownik już się zalogował, a następnie przekierowuje go na stronę, na którą pierwotnie chciał się dostać (Rys. 20).

```
#-----$
# DEKORATOR LOGOWANIA
#-----$

def check(function):
    @wraps(function)
    def deco(*args, **kwargs):
        if 'logged_in' in session:           # Jeśli użytkownik jest zalogowany
            return function(*args, **kwargs) # zwróć następną funkcję
        else:
            return render_template('login.html') # albo zwróć ekran logowania
    return deco
```

Rys. 20. Dekorator logowania

4.6. Główny algorytm

Kiedy użytkownik uzbroi alarm, odpalona zostaje niebieska dioda LED symbolizująca czuwanie systemu alarmowego oraz zostaje uruchomione zdarzenie opierające się na detekcji zbocza narastającego sensora ruchu. W momencie wykrycia ruchu zostaje uruchomiona funkcja zwrotna `trigger()` podana jako argument zdarzenia (Rys. 21).

```
@app.route('/ARM')
@check
def ARM():
    GPIO.remove_event_detect(sensor)
    global state
    state = True
    GPIO.output(blue,1)

    # detektor zdarzenia - zbocze narastające na sensorze ruchu
    GPIO.add_event_detect(sensor, GPIO.RISING, callback=trigger)

    return render_template('home_disarm.html')
```

Rys 21. Ścieżka '/ARM' z funkcją uzbrajającą alarm

Funkcja zwrotna tworzy nowy wątek (dzięki wsparciu modułu threading) i realizuje w nim w tle podaną funkcję alert() (Rys. 22).

```
def trigger(sensor):  
    threading.Thread(target=alert).start()
```

Rys 22. Utworzenie wątku z główną funkcją

Główna funkcja rozpoczyna się od usunięcia detekcji zbocza sensora ruchu na czas aktywacji alarmu oraz na odpaleniu czerwonej diody oraz brzęczyka. Następnie do użytkownika zostaje wysłana wiadomość, która pojawią się na telefonie jako powiadomienie push. Data alarmu zostaje zapisana do pliku tekstowego w celu archiwizacji (Rys 23).

```
def alert():  
  
    # wyłączenie detekcji  
    GPIO.remove_event_detect(sensor)  
  
    # niebieski LED wyłączony / Czerwony LED włączony / Brzęczyk włączony  
    GPIO.output(blue,0)  
    GPIO.output(red,1)  
    GPIO.output(buzz,1)  
  
    # wysłanie wiadomości o alarmie + archiwizacja  
    os.system("curl https://notify.run/NhdTZ0gAJ1nW3IiD -d 'WYKRYTO RUCH'")  
    write("detect.txt")
```

Rys 23. Kod odpowiedzialny za czuwanie i pobudzanie alarmu

Następnym krokiem jest czas oczekiwania (domyślnie 60 sekund) na reakcję użytkownika (Rys. 24). W momencie w którym zdecyduje się na rozbrojenie alarmu, przycisk rozbrojenia przenosi na podstronę /DISARM, która wyłącza wszystkie diody i brzęczyk oraz zmienia wartość zmiennej „state” na fałsz, co doprowadza do zakończenia funkcji osobnego wątku (Rys. 25).

```
# czekanie 60 sekund na reakcje (rozbrojenie)  
for i in range(60):  
    sleep(1)  
    if state == False:  
        return
```

Rys 24. Kod czekania na reakcję użytkownika

```

@app.route('/DISARM')
@check
def DISARM():
    GPIO.remove_event_detect(sensor)
    global state
    state = False

    # niebieski LED włączony / Czerwony LED wyłączony / Brzęczyk wyłączony
    GPIO.output(blue,0)
    GPIO.output(red,0)
    GPIO.output(buzz,0)

    return render_template('home.html')

```

Rys 25. Ścieżka '/DISARM' z funkcją rozbrajającą alarm

Jeśli jednak użytkownik nie może podjąć decyzji lub jej nie chce, alarm zostaje uzbrojony ponownie po 60 sekundach, data wznowienia pracy zostaje zapisana, a użytkownik otrzymuje stosowne powiadomienie. Następnie zostaje odpalona niebieska dioda, kiedy to dioda czerwona gaśnie, a brzęczyk przestaje grać. Na końcu zostaje ponownie uruchomione zdarzenie detekcji ruchu (Rys. 26).

```

# wysłanie wiadomości o wznowieniu + archiwizacja
os.system("curl https://notify.run/NhdTZ0gAJ1nW3IiD -d 'WZNOWIONO PRACĘ'")
write("continue.txt")

# niebieski LED włączony / Czerwony LED wyłączony / Brzęczyk wyłączony
GPIO.output(blue,1)
GPIO.output(red,0)
GPIO.output(buzz,0)

# ponowne uruchomienie detekcji
GPIO.add_event_detect(sensor, GPIO.RISING, callback=trigger)

```

Rys 26. Kod odpowiedzialny za wznowienie alarmu

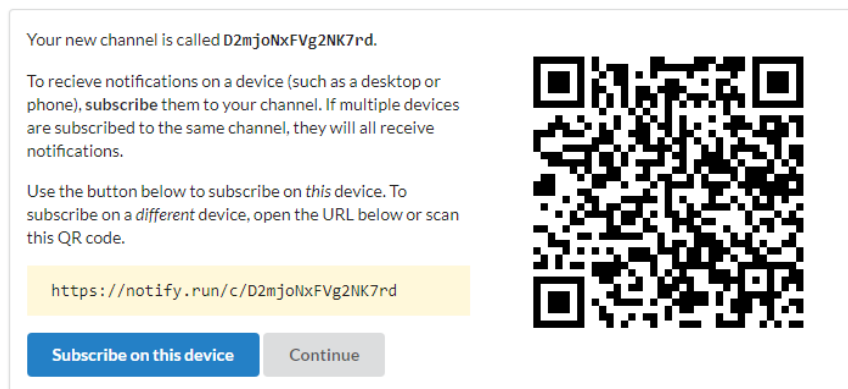
4.7. Powiadomienia i archiwizacja stanów

4.7.1. Powiadomienia

Kiedy system alarmowy wykryje ruch i następuje uruchomienie, zostaje również wysłane powiadomienie do użytkownika. Jedną z możliwości mogłoby być wysyłanie wiadomości tekstowej SMS na telefon przez odpowiednią bramkę internetową. Jednak jest to usługa płatna. Dlatego wysyłanie wiadomości odbywa się poprzez stronę *notify.run*. Cały zamysł usługi polega na utworzeniu kanału oznaczonego losowym ciągiem znaków, na który trafiają powiadomienia. Następnie należy zasubskrybować kanał przy pomocy urządzenia, na

które chcemy otrzymywać powiadomienia. Można tego dokonać np. przy pomocy wygenerowanego kodu QR unikatowego dla danego kanału (Rys. 27).

Quick Start



Rys. 27. Komunikat po utworzeniu kanału [39]

Powiadomienia na wybrany kanał należy wysłać według kodu:

```
curl https://notify.run/NAZWA_KANAŁU -d "TREŚĆ WIADOMOŚCI"
```

Powyższa linia jest realizowana przy użyciu biblioteki „os” w poniższy sposób:

```
os.system("curl https://notify.run/NhdTZ0gAJ1nW3liD -d 'WYKRYTO RUCH'")
```

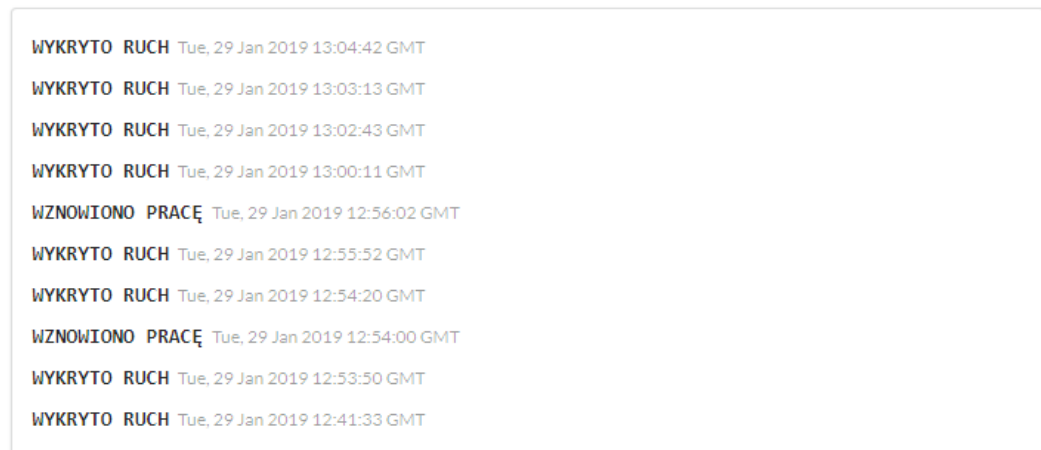
Aby powiadomienia pojawiały się na telefonie należy zeskanować nim kod QR oraz kliknąć na opcję subskrypcji. Aby poprawić jakość powiadomień można udać się do ustawień telefonu i skonfigurować opcje wybranej przeglądarki (w moim przypadku Chrome). Pod spodem do wyboru będą znajdować się ustawienia przypisane dla danej witryny – należy wybrać *notify.run*, po czym ustawić najwyższy priorytet oraz wybrać unikatowy sygnał dźwiękowy. Warto też wykluczyć przeglądarkę z trybu oszczędzania energii, który ogranicza niektóre funkcje telefonu, kiedy stan baterii jest poniżej określonego progu.

4.7.2. Archiwizacja stanów

Notify.run zapewnia na stronie danego kanału podgląd wysłanych wiadomości. Niestety opcja ta ograniczona jest jedynie do dziesięciu ostatnich komunikatów (Rys. 28), dlatego też przygotowałem własny sposób na archiwizację stanów. Od razu po otrzymaniu

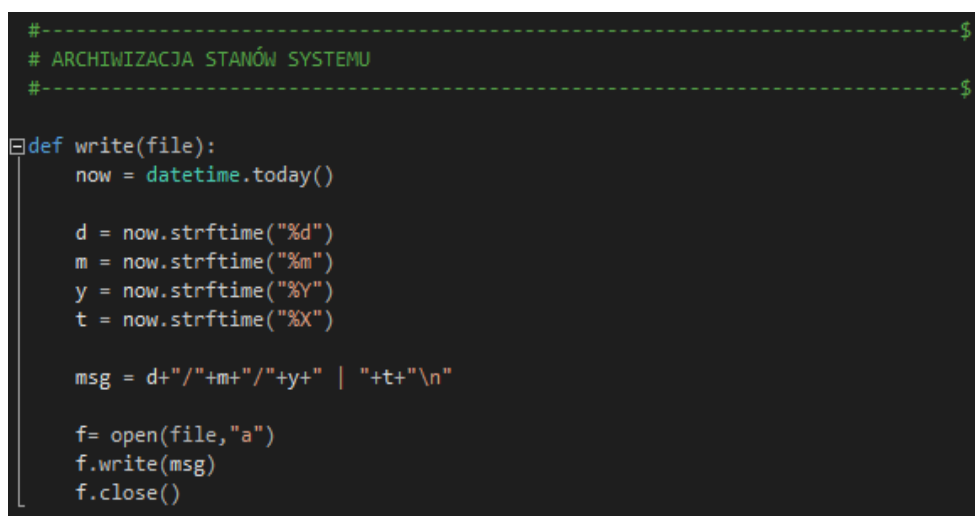
powiadomienia robiony jest wpis do wybranego pliku tekstowego (Rys. 29), którego zawartość jest następnie wpisywana do arkusza HTML na życzenie użytkownika (Rys. 32).

Recent messages



```
WYKRYTO RUCH Tue, 29 Jan 2019 13:04:42 GMT
WYKRYTO RUCH Tue, 29 Jan 2019 13:03:13 GMT
WYKRYTO RUCH Tue, 29 Jan 2019 13:02:43 GMT
WYKRYTO RUCH Tue, 29 Jan 2019 13:00:11 GMT
WZNOWIONO PRACĘ Tue, 29 Jan 2019 12:56:02 GMT
WYKRYTO RUCH Tue, 29 Jan 2019 12:55:52 GMT
WYKRYTO RUCH Tue, 29 Jan 2019 12:54:20 GMT
WZNOWIONO PRACĘ Tue, 29 Jan 2019 12:54:00 GMT
WYKRYTO RUCH Tue, 29 Jan 2019 12:53:50 GMT
WYKRYTO RUCH Tue, 29 Jan 2019 12:41:33 GMT
```

Rys. 28. Podgląd wiadomości na stronie notify.run [39]



```
#-----$
# ARCHIWIZACJA STANÓW SYSTEMU
#-----$

def write(file):
    now = datetime.today()

    d = now.strftime("%d")
    m = now.strftime("%m")
    y = now.strftime("%Y")
    t = now.strftime("%X")

    msg = d+"/"+m+"/"+y+" | "+t+"\n"

    f= open(file,"a")
    f.write(msg)
    f.close()
```

Rys. 29. Funkcja zapisu stanu alarmu do pliku

4.8. Automatyczne uruchamianie skryptu

Podczas użytkowania systemu może zaistnieć sytuacja, w której płyta zostanie wyłączona, czy to zdalnie przy pomocy aplikacji, czy też fizycznie poprzez odłączenie kabla zasilającego. Niestety w przypadku ponownego włączenia zasilania nic się nie stanie, ponieważ skrypt z alarmem nie zostanie uruchomiony. Aby to zrobić, należałoby uruchomić program własnoręcznie, co brzmi zbyt problematycznie z punktu widzenia użytkownika. Dlatego też zautomatyzowałem cały proces z użyciem programu Cron. Jest to program

zawarty w systemie operacyjnym Raspberry Pi, czyli w Raspbianie (jak również w innych odmianach Linuxa), służący do tworzenia harmonogramów. Wpisując odpowiednie komendy w zadanym formacie do pliku Crontab można zlecić wykonywanie danej pracy np. o określonej godzinie [40].

W moim przypadku skrypt alarmu musiał być wywołany z każdym uruchomieniem. Wystarczyło wpisać do Crontab poniższe polecenie:

```
@reboot python3 /home/pi/server/alarm.py
```

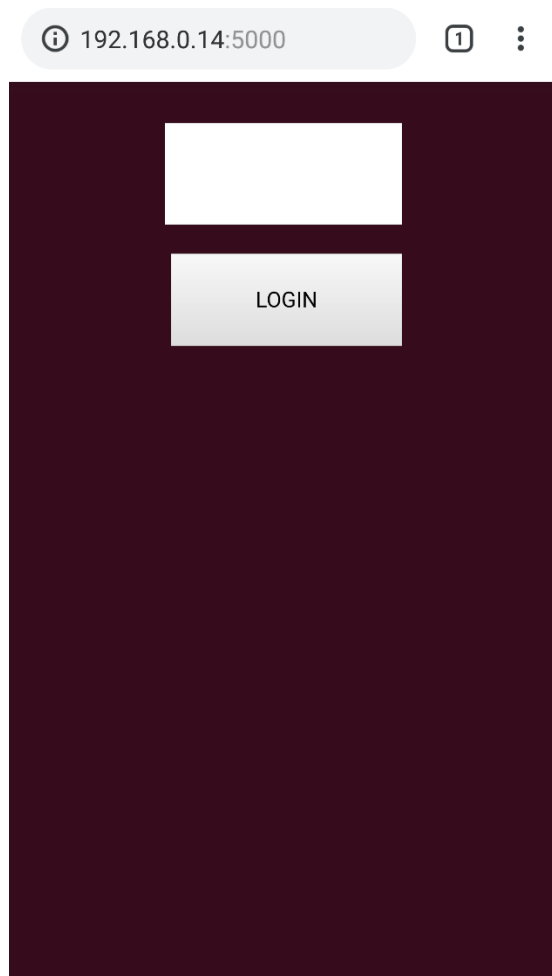
Polecenie takie informuje odpowiednio o tym, kiedy ma zostać wykonana praca (@reboot - czyli za każdym uruchomieniem systemu), przy pomocy czego ma być wykonana (python3) oraz co ma zostać wykonane (należy podać pełną lokalizację pliku). Używając w systemowym terminalu komendy „crontab -e” można wpisać dane zadanie, kiedy to komenda „crontab -l” wypisuje wszystkie aktywne zadania.

4.9. Interfejs użytkownika

Aby użytkownik mógł kontrolować alarm, należało również przygotować odpowiedni interfejs, przejrzysty i intuicyjny. W tym celu użyto arkuszy HTML generowanych przez moduł Flask.

4.9.1. Ekran logowania

Kiedy użytkownik uruchomi po raz pierwszy alarm, ukaże mu się ekran logowania proszący o podanie hasła numerycznego, które następnie zostaje porównane z tym zawartym w kodzie (Rys. 30).



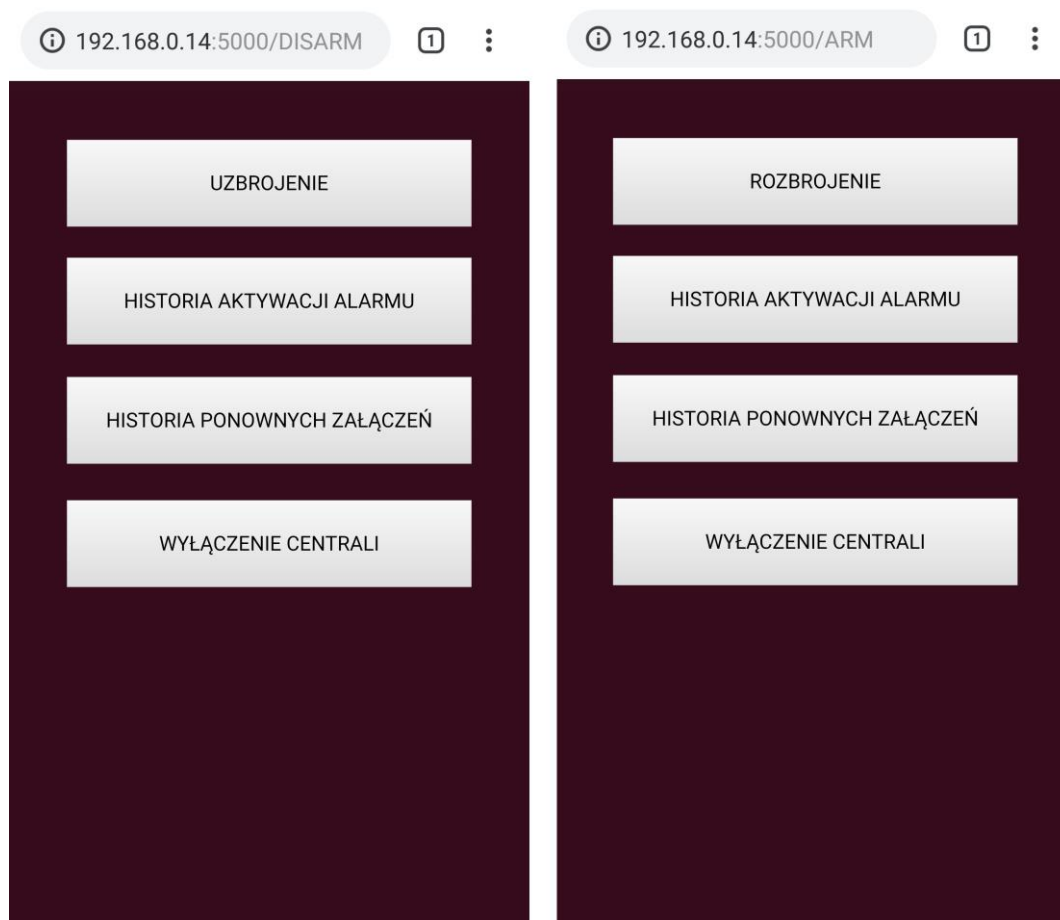
Rys. 30. Ekran logowania

Adres podstrony: /LOGIN

Powiązany plik: login.html

4.9.2. Ekran główny (Uzbrojony / Rozbrojony)

Ekran ten określony jest poprzez 2 pliki HTML różniące się funkcją pierwszego przycisku w zależności od tego czy alarm akurat jest uzbrojony, czy też rozbrojony. Następne 2 przyciski służą pokazywaniu archiwów zapisanych stanów alarmu. Ostatni przyciski służy do wyłączenia Raspberry Pi (Rys. 31).



Rys. 31. Ekrany główne aplikacji

Adresy podstron: /ARM oraz /DISARM

Powiązane pliki: home.html oraz home_disarm.html

4.9.3. Ekrany stanów

Ekrany stanów pokazują zawartości plików tekstowych detect.txt oraz continue.txt, pozwalając użytkownikowi na zaznajomienie się z historią uruchomień alarmu oraz ze wznowieniem jego pracy. Do tego celu wykorzystywany jest ten sam wspólny arkusz archive.html (Rys. 32).

192.168.0.14:5000/DETECT ⓘ ⓘ		192.168.0.14:5000/CONTINU ⓘ ⓘ	
WYKRYTO RUCH		WZNOWIONO PRACĘ:	
27/01/2019	20:57:55	27/01/2019	20:58:11
27/01/2019	20:58:18	27/01/2019	20:58:33
27/01/2019	21:00:27	27/01/2019	21:00:42
27/01/2019	21:17:49	27/01/2019	21:18:05
27/01/2019	21:21:26	27/01/2019	21:21:42
27/01/2019	21:32:39	27/01/2019	21:32:54
28/01/2019	15:12:19	28/01/2019	15:12:34
28/01/2019	15:12:38	28/01/2019	15:14:02
28/01/2019	15:13:40	28/01/2019	15:14:26
28/01/2019	15:14:11	28/01/2019	15:15:16
28/01/2019	15:15:01	28/01/2019	15:42:00
28/01/2019	15:41:45	28/01/2019	15:43:06
28/01/2019	15:42:50	28/01/2019	15:44:15
28/01/2019	15:43:59	28/01/2019	15:44:37
28/01/2019	15:44:22	28/01/2019	15:46:03
28/01/2019	15:45:47	28/01/2019	15:46:59
28/01/2019	15:46:43	28/01/2019	15:47:26
28/01/2019	15:47:11	28/01/2019	15:53:31
28/01/2019	15:53:15	28/01/2019	15:54:21

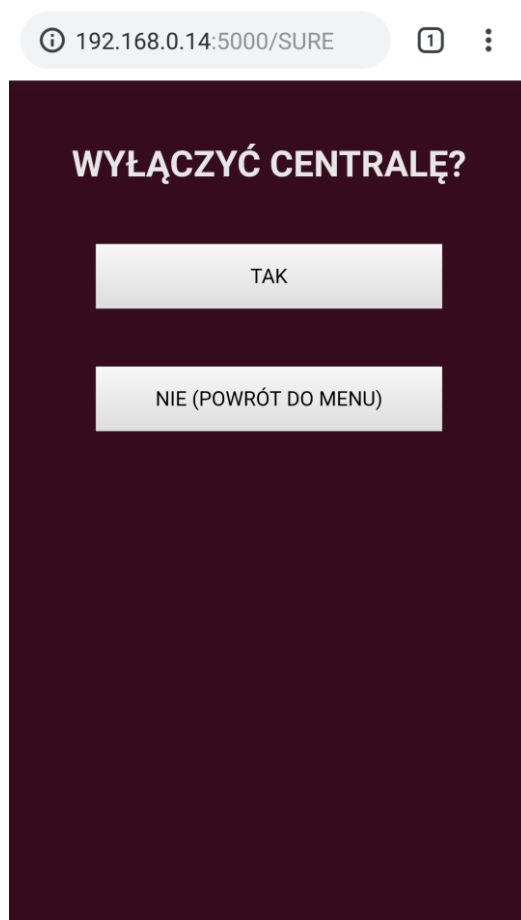
Rys. 32. Lewo: ekran z rozpiską dat wykrytych ruchów; Prawo: ekran z rozpiską dat wznowienia pracy alarmu

Adresy podstron: /DETECT oraz /CONTINUE

Powiązany plik: archive.html

4.9.4. Ekran wyłączenia

Ekran ten stawia użytkownika przed prostym wyborem: czy chce on wyłączyć centralę (Raspberry Pi), czy też wrócić z powrotem do ekranu głównego (Rys. 33).



Rys. 33. Ekran wyłączenia

Adres podstrony: /SURE

Powiązane pliki: sure.html

5. Przykład użycia

Początkowe założenie: płytką Raspberry Pi jest odłączona.

- Płytką zostaje podłączona do zasilania – po około 25 sekundach zostaje automatycznie uruchomiony serwer. Zostaje to zasygnalizowane poprzez sekwencję powitalną, czyli 2 szybkie mrugnięcia diodami i 2 krótkie dźwięki z brzęczyka.
- Po uruchomieniu aplikacji użytkownik uzbraja alarm. Zostaje zapalona dioda niebieska sygnalizująca stan czuwania, a użytkownik opuszcza mieszkanie.
- W momencie wykrycia ruchu gaśnie niebieska dioda i zostaje zapalona czerwona. Dodatkowo zostaje uruchomiony brzęczyk, generujący dźwięk na poziomie 85dB. Pomimo faktu, że obsługa aplikacji działa jedynie w sieci lokalnej, to powiadomienie o wykryciu pojawia się natychmiast niezależnie od sieci, w której aktualnie znajduje się użytkownik. W tym czasie użytkownik może podjąć kroki w sprawie włamania.
- Po pewnym czasie (aktualnie ustawionym w kodzie na 60 sekund) system alarmowy zostaje uzbrojony ponownie, a do użytkownika trafia powiadomienie o wznowionej pracy.
- Kiedy użytkownik wraca do domu i znajduje się w zasięgu swojej sieci bezprzewodowej może szybko i wygodnie rozbroić alarm z użyciem aplikacji.

6. Wnioski oraz napotkane problemy

6.1. Wnioski z projektu alarmu

Zaprojektowany alarm spełnia oczekiwania projektowe. Złożony układ oraz napisana aplikacja internetowa świetnie ze sobą współpracują, dając obraz kompetentnego systemu alarmowego, którego można używać na co dzień. Układ reaguje na akcje użytkownika szybko – bez zauważalnego opóźnienia, ustawiając błyskawicznie stany poszczególnych pinów oraz wysyłając powiadomienia.

Podczas przygotowywania pracy zaznajomiłem się z mikrokomputerem jakim jest Raspberry Pi, nauczyłem się podstaw HTML oraz poznałem w dobrym stopniu język Python, który pomimo swojej prostoty jest powszechnie używany, a czołowe firmy takie jak Google czy Facebook ciągle poszukują pracowników znających ten język[41].

6.2. Napotkane problemy oraz ich rozwiązania

Napotkane przeze mnie problemy były związane jedynie z częścią software’ową:

- Pierwszy problem występował przy użyciu pollingu, czyli odczytywania stanu sensora ruchu przy pomocy pętli. Kiedy przycisk rozbrajania był wciskany - nic się nie działo. Dopiero kiedy wykryto ruch to momentalnie nastawało rozbrajanie. Problem został rozwiązany poprzez zastąpieniem pollingu detekcją zbocza narastającego na sensorze ruchu. Funkcja detekcji jest częścią biblioteki RPi.GPIO.
- Drugim problem był związany z powiadomieniami. Pierwotnie użytkownik dostawał e-maila w razie wykrycia ruchu. W ten sposób miały zostać rozwiązane kwestie notyfikacji oraz archiwizacji. Kiedy wiadomość była wysyłana przy pomocy modułu smtplib, to nie zawsze przychodziły powiadomienia od aplikacji powiązanej z pocztą, pomimo faktu, że e-mail dochodził. Sytuacja, w której przychodziła 1/3 powiadomień była niedopuszczalna i należało ją naprawić. Aktualne rozwiązanie jest opisane w podrozdziale 4.5.
- Kolejny problem był związany z rozbrajaniem alarmu, który wykrył już ruch. Każda próba rozbrojenia w takiej okoliczności kończyła się wyłączeniem

serwera. Jediną opcją było przeczekanie, aż do automatycznego wyłączenia. Wtedy alarm znowu przechodził w stan czuwania i mógł być rozbrojony. Problemem okazało się umieszczenie całego głównego algorytmu w funkcji zwrotnej detektora zbocza. Funkcja taka była osobnym wątkiem, realizującym się w czasie trwania reszty kodu, co było potrzebne do rozbrajania poprzez zmianę podstrony w aplikacji. Rozwiązaniem okazało się wprowadzenie biblioteki obsługującej pełnoprawne wątki, oraz umieszczenie głównego algorytmu w tym wątku. Całość została opisana w podrozdziale 4.4.

- Problem związany z Crontab. Kiedy wróciłem po przerwie do dalszego opisywania pracy zauważyłem, że serwer nie uruchomił się automatycznie. Powodem tego było zniknięcie wpisów w pliku Crontab. Rozwiązaniem okazało się utworzenie dodatkowego pliku tekstowego z komendami, oraz przypisanie go do Crontab.
- Ostatni problem związany był z sekwencją powitalną. Kiedy system operacyjny Raspberry Pi uruchomił się, a skrypt z alarmem został automatycznie załączony, to kod odpowiedzialny za powitanie wykonywał się 2 razy. Rozwiązaniem tego problemu okazało się utworzenie osobnego skryptu z tym kodem, oraz przypisaniu go do Crontab.

6.3. Potencjalna rozbudowa systemu

Alarm można na przyszłość rozbudować o funkcje takie jak:

- Zmiana FLASK na pełnoprawny framework:
Zmiana frameworka na bardziej zaawansowany pozwoliłaby na więcej opcji i większe bezpieczeństwo. Serwer byłby widzialny poza siecią lokalną.
- Bezprzewodowe czujki ruchu oparte na Bluetooth:
W ten sposób ilość kabli zostanie znacznie ograniczona, a czujka nie będzie musiała znajdować się blisko Raspberry Pi. Trzeba tylko wziąć pod uwagę, że takie udoskonalenie znacznie podniesie koszt całego systemu.

- Wsparcie dla kart Sim:

Używając karty Sim można pozbyć się potrzeby łączenia Raspberry z siecią, co w połączeniu z nowym frameworkiem pozwoliłoby na dużą mobilność.

- Awaryjne zasilanie:

W razie awarii lub celowego wyłączenia prądu, Raspberry Pi zostałyby ponownie uruchomione z awaryjnego źródła, a stan alarmu zostałby przywrócony sprzed wyłączenia.

7. Bibliografia

- [1] Matt Burgess, *What is the Internet of Things? WIRED explains*,
<https://www.wired.co.uk/article/internet-of-things-what-is-explained-iot>
(Dostęp 12.02.2019)

- [2] *Forecast market size of the global smart home market from 2016 to 2022*,
<https://www.statista.com/statistics/682204/global-smart-home-market-size/>
(Dostęp 12.02.2019)

- [3] *Home Security System Market worth \$74.75 billion by 2023*,
<https://www.marketsandmarkets.com/PressReleases/home-security-system.asp>
(Dostęp 12.02.2019)

- [4] Ceny central typu Integra w sklepie alarm-sklep.pl,
<http://www.alarm-sklep.pl/13-centrale-integra-satel> (Dostęp 12.02.2019)

- [5] Systemy sygnalizacji włamania i napadu ABAX,
<https://www.satel.pl/pl/cat/3#cat3> (Dostęp: 12.01.2019)

- [6] Systemy sygnalizacji włamania i napadu MICRA,
<https://www.satel.pl/pl/cat/3#cat479> (Dostęp: 12.01.2019)

- [7] System Agility 3,
<https://www.riscogroup.com/products/solution/39> (Dostęp 12.01.2019)

- [8] Opis czujki ruchu GRADE 3,
<https://www.riscogroup.com/products/product/7803> (Dostęp 12.01.2019)

- [9] Opis systemu LightSYS 2,
<https://www.riscogroup.com/products/solution/8444> (Dostęp 12.01.2019)

- [10] Opis systemu WiComm-Smart Interactive Wireless Security,
<https://www.riscogroup.com/products/product/52708> (Dostęp 12.01.2019)

- [11] Opis najprostszego waraintu systemu SimplySafe,
<https://simplisafe.com/home-security-system-keep> (Dostęp 14.01.2019)

- [12] Brad Jones, *Meet the Brits who promised the world a \$25 PC, and delivered a revolution*,
<https://www.digitaltrends.com/computing/how-raspberry-pi-became-the-best-selling-british-computer-of-all-time/> (Dostęp 14.12.2018)

- [13] Conor Lyons, *A History Of The Raspberry Pi*,
<http://novadigitalmedia.com/history-raspberry-pi/> (Dostęp 14.12.2018)

- [14] Greg Synek, *Raspberry Pi 3 Model B+ arrives just in time for Pi Day*,
<https://www.techspot.com/news/73709-raspberry-pi-model-b-arrives-time-pi-day.html>
(Dostęp 12.02.2019)

- [15] Raspberry Pi, *Get involved with the Raspberry Pi Foundation*,
https://www.youtube.com/watch?v=QAHJVe2jq_E (Dostęp 10.01.2019)

- [16] thunderemoji, *Reddit's Year in Review: 2018*,
<https://redditblog.com/2018/12/04/reddit-year-in-review-2018/> (Dostęp 10.01.2019)

- [17] Subreddit r/raspberry_pi, https://www.reddit.com/r/raspberry_pi/ (Dostęp 10.01.2019)

- [18] Parametry Raspberry Pi 3B+,
<https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>
(Dostęp 10.01.2019)

- [19] Steve Sandler, Heidi Barnes, *Introduction to Power Integrity*,
http://www.spi2016.org/slides/SPI2016_Tutorial_Slides.pdf (Dostęp 10.01.2019)

- [20] Opis BCM2837B0,
<https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2837b0/README.md> (Dostęp 10.01.2019)

- [21] Opis procesora Cortex-A53,
<https://developer.arm.com/products/processors/cortex-a/cortex-a53>
(Dostęp 10.01.2019)
- [22] *What's the Difference Between 2.4 and 5 GHz WiFi?*,
<https://socialwifi.com/knowledge-base/wifi-technology/difference-between-24-and-5-ghz-wifi/> (Dostęp 10.01.2019)
- [23] David Nield, *Why Your Router Has Two Wifi Bands and How They Work*,
<https://gizmodo.com/why-your-router-has-two-wifi-channels-and-how-they-work-1828650288> (Dostęp 10.01.2019)
- [24] Damian Tomaszewski, *Zmiany w Bluetooth Low Energy wprowadzone przez specyfikację Bluetooth 4.2*,
<https://elektronikab2b.pl/technika/34157-zmiany-w-bluetooth-low-energy-wprowadzone-przez-specyfikacje-bluetooth-4.2> (Dostęp 10.01.2019)
- [25] Zdjęcie nakładki Raspberry Pi PoE HAT,
<https://www.raspberrypi.org/products/poe-hat/> (Dostęp 10.01.2019)
- [26] *O Debianie*, <https://www.debian.org/intro/about#what> (Dostęp 15.01.2019)
- [27] Opis czujnika ruchu HC-SR501,
<https://www.mpja.com/download/31227sc.pdf> (Dostęp 21.01.2019)
- [28] Patrick McCarthy, *Starting fires with a fresnel lens*,
<https://www.offgridweb.com/survival/starting-fires-with-a-fresnel-lens/>
(Dostęp 21.01.2019)
- [29] William T. Arkin, *Trends in Lasers and Electro-optics Research*,
Nova, Nowy Jork, 2006

- [30] Andreas Mandelis, Constantinos Christofides, *Physics, Chemistry and Technology of Solid State Gas Sensor Devices*, John Wiley & Sons, Inc., Nowy Jork, 1993
- [31] Shaul Katzir, „The Beginnings of Piezoelectricity: A Study in Mundane Physics”, Springer, Dordrecht , 2006
- [32] lady ada, *How PIRs Work*,
<https://learn.adafruit.com/pir-passive-infrared-proximity-motion-sensor/how-pirs-work>
(Dostęp 21.01.2019)
- [33] *Product spotlight: Piezo and Magnetic buzzers*,
<https://www.cui.com/product-spotlight/piezo-and-magnetic-buzzers>
(Dostęp 9.02.2019)
- [34] Syed Zain Nasir, *Introduction to 2N2222*,
<https://www.theengineeringprojects.com/2017/06/introduction-to-2n2222.html>
(Dostęp 9.02.2019)
- [35] David Hemmendinger, *Machine language*,
<https://www.britannica.com/technology/machine-language> (Dostęp 11.01.2019)
- [36] The Editors of Encyclopaedia Britannica, *Assembly language*,
<https://www.britannica.com/technology/assembly-language> (Dostęp 11.01.2019)
- [37] Strona tytułowa modułu FLASK, <http://flask.pocoo.org/> (Dostęp 14.12.2018)
- [38] Strona modułu FLASK: Foreword
<http://flask.pocoo.org/docs/1.0/foreword/#what-does-micro-mean> (Dostęp 11.01.2019)
- [39] Notify.run, <https://notify.run/c/NhdTZ0gAJ1nW3IiD> (Dostęp 29.01.2019)
- [40] Sk, *A Beginners Guide To Cron Jobs*,
<https://www.ostechnix.com/a-beginners-guide-to-cron-jobs/> (Dostęp 8.02.2019)

- [41] Vishnu, *Why Future of Python Language is Bright?*,
<https://www.probytes.net/blog/python-future/> (Dostęp 11.02.2019)

8. Zawartość dołączonej płyty CD

- Tekst pracy w formacie PDF
- Folder „Python” z plikami: welcome.py (kod sekwencji powitalnej), alarm.py (kod systemu alarmowego)
- Folder „HTML” z plikami: home.html, home_disarm.html, archive.html, login.html, sure.html