Politechnika Warszawska

WYDZIAŁ ELEKTRONIKI
I TECHNIK INFORMACYJNYCH

Instytut Systemów Elektronicznych

Praca dyplomowa inżynierska

na kierunku Elektronika
w specjalności Elektronika i inżynieria komputerowa

System bezpieczeństwa z dostępem sieciowym

Piotr Antosiuk
Numer albumu 268960

promotor
dr inż. Gustaw Mazurek

WARSZAWA 2018

Spis treści

1. Wstęp teoretyczny

1.1. Współczesne systemy monitoringu i bezpieczeństwa

1.2. Koncepcja "inteligentnego domu"

1.3. Motywacje i cel pracy

2. Architektura systemu

2.1. Wymagania techniczne i funkcjonalne

System bezpieczeństwa powinien łączyć funkcję monitoringu z możliwością kontroli otwarcia drzwi i okien oraz pomiaru warunków środowiskowych. Monitoring powinien być zrealizowany poprzez kamery USB ze względu na ich cenę i dostępność. System powinien być skalowalny i umożliwiać rozbudowę o dodatkowe kamery, co odpowiada np. rozszerzeniu monitoringu o dodatkowe pomieszczenia. Dane zebrane przez system należy udostępnić przez graficzny interfejs użytkownika w postaci strony WWW. Podłączenie systemu do sieci domowej LAN (ang. *Local Area Network*) i wykonanie odpowiednich ustawień w panelu administracyjnym rutera umożliwi dostęp do strony WWW z dowolnego miejsca przez Internet.

System powinien być zrealizowany w postaci komputera jednopłytkowego z systemem Linux, na którym będzie działał serwer WWW. Odpowiada on za dostęp do strony WWW i danych zebranych przez system. Kontrolę otwarcia drzwi i okien można zrealizować poprzez czujniki stykowe, które należy podłączyć do wyprowadzeń GPIO. Dostępne na rynku zintegrowane czujniki warunków środowiskowych (np. temperatury i wilgotności) wymagają obsługi magistral (np. I2C czy OneWire). Komputer powinien być wyposażony w gniazda USB, do których będą podłączone kamery. Monitoring przy pomocy kamer nie musi być prowadzony w postaci ciągłego nagrania, które wymaga sporej ilości miejsca w pamięci. Wystarczające będzie okresowe wykonywanie zdjęć oraz dodatkowo wyzwalanie zdjęcia w przypadku otwarcia drzwi lub okna – otwarcia czujnika stykowego przypisanego do kamery.

Interfejs użytkownika musi prezentować aktualny stan systemu w postaci dostępu do ostatnich wykonanych zdjęć, odczytów czujników środowiskowych oraz stanu czujników stykowych (otwarty/zamknięty). Ponadto użytkownik powinien mieć dostęp do przeszłych zdjęć i pomiarów w postaci archiwum. Historia odczytów systemu powinna być również dostępna w formie eksportowanej listy np. w formacie *csv*. Użytkownik powinien mieć również możliwość dokonania zmian ustawień systemu przez panel administracyjny. Może on umożliwiać np. włączenie wysyłania powiadomień e-mail w przypadku otwarcia czujnika.

Dane zebrane przez system z kamer i czujników powinny być więc zapisane w systemie tak, by umożliwić do nich dostęp serwerowi WWW, na którym będzie działać strona WWW. Pomiary wykonywane przez system powinny być zapisywane wraz z datą i godziną ich wykonania. Dostęp do tych danych powinien być niezależny od programu zapisującego wyniki pomiarów i prowadzącego monitoring.

Podsumowując, wymagania techniczne stojące przed systemem to:

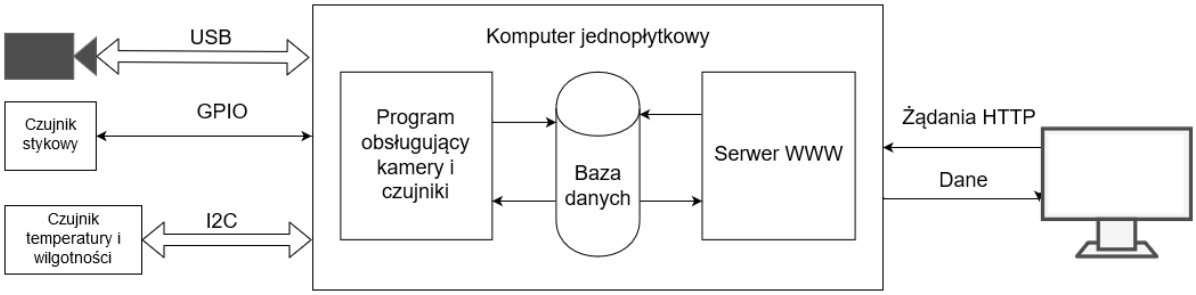
- wykorzystanie komputera jednopłytkowego,
- użycie systemu Linux,
- użycie serwera WWW,
- obsługa wielu kamer USB,
- obsługa wielu dwustanowych czujników stykowych poprzez wyprowadzenia GPIO,
- obsługa wielu czujników środowiskowych korzystających np. z I2C lub OneWire,

- możliwość połączenia z Internetem poprzez Ethernet lub Wi-Fi.

Wymagania funkcjonalne systemu to:

- dostęp do stanu systemu poprzez stronę WWW,
- dostęp do archiwalnych stanów systemu,
- możliwość eksportu listy pomiarów i zdarzeń do pliku (np. w formacie *csv*),
- dostęp do ustawień systemu poprzez interfejs użytkownika,
- wysyłanie powiadomień e-mail w przypadku otwarcia czujnika,
- wywoływanie zdjęcia po otwarciu czujnika.

Na podstawie powyższych wymagań opracowana została architektura systemu, którą jest przedstawiona na rysunku **??** i zostanie omówiona w poniższych podrozdziałach.



Rysunek 1: Architektura systemu bezpieczeństwa.

2.2. Przegląd dostępnych platform

Jednymi z najbardziej popularnych komputerów jednopłytkowych jest Raspberry Pi rozwijany przez Raspberry Pi Foundation jako narzędzie edukacyjne do nauczania programowania i elektroniki. Wokół tego projektu powstała duża grupa amatorów i entuzjastów dzielących się swoimi projektami. Jednym z konkurentów Raspberry Pi jest BeagleBone rozwijany przez Texas Instruments również zgodnie z filozofią *open-hardware* – sprzętu o otwartym dostępie do źródła i projektów. Oba komputery oparte są o procesory ARM Cortex w formie *System On Chip* – układu scalonego zawierającego procesor wraz z układami pamięci, peryferiami i przetwornikami analogowo-cyfrowymi i cyfrowo-analogowymi. Oba komputery umożliwiają zainstalowanie na nich dystrybucji systemu Linux. Najnowszymi modelami w momencie rozpoczęcia projektu (w marcu 2017 roku) były Raspberry Pi Model 3B oraz BeagleBone Black. Porównanie obu komputerów przedstawiono w tabeli **??**.

Przewagą komputera Raspberry Pi 3B jest szybszy procesor o większej liczbie rdzeni, co pozwala na równoległe wykonywanie większej liczby procesów, oraz większa pojemność pamięci RAM. Ponadto ma on możliwość podłączenia zarówno do sieci

	Raspberry Pi 3B	BeagleBone Black
Rok wydania	2016	2013
CPU	ARM Cortex-A53	ARM Cortex-A8
Liczba rdzeni	4	1
Częstotliwość taktowania	1,2 GHz	1 GHz
RAM	1 GB LPDDR2	512 MB DDR3L
Liczb gniazd USB	4	1
WLAN	b/g/n	brak
Ethernet	10/100	10/100
I2C	Tak	Tak
Liczba wyprowadzeń GPIO	40	66
Maksymalny pobór mocy	3 W	2,3 W

Tabela 1: Porównanie specyfikacji technicznej Raspberry Pi 3B i BeagleBone Black [**?**] [**?**] [**?**]

przewodowej (Ethernet) jak i bezprzewodowej (WLAN). Może być to przewaga w przypadku, gdyby system miał być zainstalowany z daleka od rutera. Komputer BeagleBone Black ma niższą wartość poboru mocy oraz więcej wyprowadzeń GPIO. Na każdy czujnik stykowy będzie jednak potrzebne jedno wyprowadzenie GPIO oraz podłączenie do masy, zatem 40 wyprowadzeń Raspberry Pi 3B z pewnością będzie wystarczające. Z tych powodów wybrałem platformę Raspberry Pi 3B do projektowanego systemu.

Wśród dostępnych dystrybucji systemu Linux na Raspberry Pi są m.in. Raspbian (domyślna dystrybucja oparta o Debian) i Ubuntu Mate. W projekcie zostanie użyty system Raspbian, ponieważ jest on uniwersalny i spełnia wszystkie stawiane wymogi – obsługę wyprowadzeń GPIO, magistrali I2C oraz możliwość działania jako serwer WWW (np. przy użyciu Apache). System Raspbian posiada również interfejs graficzny, co może być przydatną cechą w przypadku, gdyby konieczna była bezpośrednia interwencja administratora systemu z pominięciem stworzonego interfejsu użytkownika.

2.3. Czujniki warunków środowiskowych

Wymogiem technicznym systemu jest również obsługa czujników środowiskowych. Najważniejszymi parametrami, które należy zmierzyć w warunkach domowych są temperatura i wilgotność względna powietrza. Wpływają one na jakość powietrza w pomieszczeniach oraz na poziom zachorowań wśród przebywających w nich. Badania wykazały związek między wilgotnością względną w domu i pracy oraz liczbą dni spędzonych na zwolnieniu lekarskim. Zbyt niska lub zbyt wysoka wilgotność powietrza w pomieszczeniach prowadzi do zwiększenia przypadków zachorowań na choroby układu oddechowego. [**?**]

Przykładowymi czujnikami temperatury i wilgotności dostępnymi na rynku są Adafruit SHT31, Adafruit Si7021 oraz Grove TH02. Są to czujniki umieszczone na płytkach *breakout-board*, które posiadają wyprowadzenia linii I2C, masy oraz zasilania. Porównanie specyfikacji czujników znajduje się w tabeli **??**.

Na podstawie danych przedstawionych w tabeli **??** wybrałem czujnik Si7021 jako użyty w projekcie czujnik środowiskowy. Zapewnia on pomiar temperatury i wilgotności względnej w zakresie wartości panujących w warunkach pokojowych oraz ma naj-

	Temperatura		Wilgotność		
	Zakres pomiarowy	Dokładność	Zakres pomiarowy	Dokładność	Cena
SHT31	-40 – 125 °C	$\pm$ 0,3 °C	0 – 100% RH	$\pm$ 2 %RH	79,80 zł
Si7021	-10 – 85 °C	$\pm$ 0,4 °C	0 – 80% RH	$\pm$ 3 %RH	39,70 zł
TH02	-40 – 85 °C	$\pm$ 0,5 °C	0 – 80% RH	$\pm$ 4,5 %RH	54,00 zł

Tabela 2: Porównanie specyfikacji czujników temperatury i wilgotności [**?**] [**?**] [**?**] (ceny za sztukę: `www.botland.com.pl`).

niższą cenę. Przewagą czujnika SHT31 była możliwość zmiany adresu I2C poprzez podłączenie wyprowadzenia do stanu wysokiego lub niskiego. Byłoby to jednak rozwiązanie możliwe do zastosowania przy użyciu tylko dwóch czujnikach w systemie.

2.4. Czujniki stykowe

Magnetyczne czujniki zbliżeniowe mogą być użyte do określenia pozycji drzwi lub okien i tym samym realizować nadzór nad ich stanem. Czujniki te posiadają dwa stany – zamknięty i otwarty. W systemie zostały użyte czujniki zbliżeniowe MC-38 składające się z kontaktronu i magnesu. Obwód kontaktronu jest domyślnie rozwarty, a po zbliżeniu magnesu następuje jego zamknięcie i przepływ prądu. Wybrałem ten czujnik ze względu na łatwość ewentualnego montażu – czujnik posiada taśmę samoprzylepną i otwory na śruby montażowe.

2.5. Kamery cyfrowe

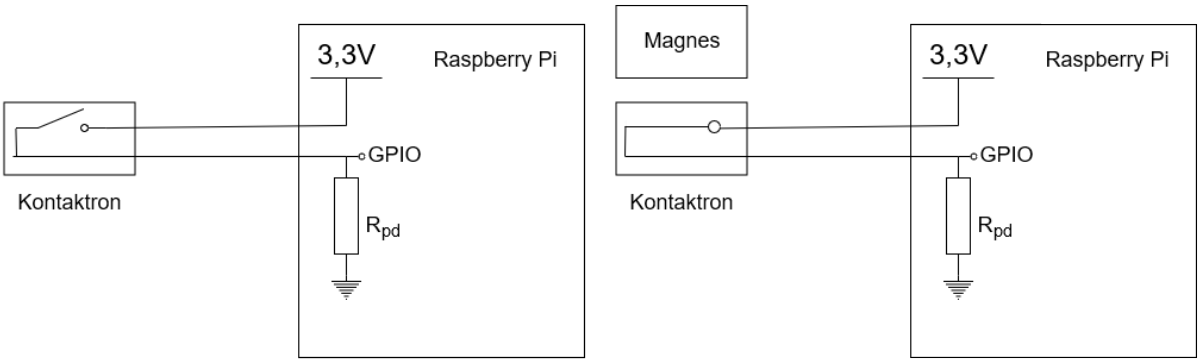
Kamery USB zastosowane w systemie mogą pozwolić na realizację domowego monitoringu niedużym kosztem. Ich uniwersalność pozwala również wykorzystać je w innych domowych zastosowaniach np. jako kamerę do rozmów przez komunikatory internetowe takie jak Skype czy Google Hangouts. W projektowanym systemie zadaniem kamer będzie cykliczne wykonywanie zdjęć. Celem powinno być znalezienie złotego środka między jakością zdjęć wykonywanych przez kamery i ich ceną. Jakość powinna być na tyle dobra, by zdjęcie umożliwiło identyfikację ewentualnego intruza. Koszt jest kategorią bardziej subiektywną, ale kamera USB powinna być konkurencyjna cenowo wobec kamer IP, których koszt zaczyna się od ok. 150 zł.[**?**]

W systemie zastosowałem dwie kamery z różnego przedziału cenowego. Kamera Titanum Onyx posiada matrycę CMOS, która umożliwia wykonywanie zdjęć o rozdzielczości do 5 megapikseli przy zastosowaniu interpolacji. Co ważne kamera jest kompatybilna z UVC (*USB Video Class*) – sterownikiem wbudowanym w system Linux. Kamera posiada również przełącznik włączający 3 diody LED, które mogą być przydatne w przypadku zdjęć nocnych. Koszt kamery to ok. 50 zł. Drugą kamerą jest Creative VFO 790, która posiada matrycę wykonującą zdjęcia w rozdzielczości HD 720p (1280x720 pikseli). Jest ona również kompatybilna z UVC. Optyka kamery jest stałoogniskowa w przeciwieństwie do kamery Titanum Onyx. Koszt kamery to ok. 110 zł.

2.6. Projekt części sprzętowej systemu

Połączenie między komputerem Raspberry Pi 3B a dołączonymi czujnika zrealizowałem na prototypowej płytce stykowej. Umożliwia ona tworzenie prototypów bez konieczności lutowania połączeń. Połączenie między płytką prototypową a wyprowadzeniami GPIO Raspberry Pi jest wykonane przez ekspander wyprowadzeń ProtoPi Plus.

W systemie zostały zastosowane czujniki zbliżeniowe magnetyczne MC-38. Kontaktron został podłączony do wyprowadzenia GPIO poprzez wbudowany rezystor ściągający do masy (ang. *pull-down*) oraz napięcia zasilania równego 3,3 V. Gdy kontaktron i magnes są zbliżone, obwód jest zamknięty i na wyprowadzeniu pojawia się wysoki stan. W przypadku oddalenia czujników i otwarcia obwodu na wyprowadzeniu pojawia się stan niski poprzez rezystor ściągający do masy. Na rysunku **??** został przedstawiony schemat podłączenia czujnika stykowego do platformy Raspberry Pi (z pominięciem ekspandera ProtoPi Plus).



Rysunek 2: Schemat podłączenia czujnika MC-38 w przypadku oddalonego (lewo) i zbliżonego(prawo) magnesu.

Wymogiem systemu jest również obsługa wielu kamer i czujników, w tym czujników temperatury i wilgotności poprzez magistralę I2C. Adres I2C wybranych do projektu czujników Si 7021 to 0x40. Nie mają one możliwości programistycznej lub sprzętowej zmiany adresu. Oznacza to, że do jednej magistrali mógłby być podłączony bezpośrednio tylko jeden czujnik. Podłączenie większej liczby czujników o tym samym adresie doprowadziłoby do sytuacji, w której nie można jednoznacznie określić, z którego czujnika został odczytany wynik pomiaru. Raspberry Pi3 B posiada możliwości obsługi dwóch magistral I2C. Oznaczałoby to jednak, że liczba czujników temperatury w systemie byłaby mniejsza niż kamer i czujników stykowych. Innym rozwiązaniem tego problemu jest zastosowanie multipleksera I2C. Multiplekser zadziała jak przełącznik, który pozwoli na komunikację z jednym czujnikiem naraz. Przykładem takiego urządzenia jest TCA 9548A firmy Adafruit. Pozwala on na podłączenie do 8 urządzeń korzystających z magistrali I2C.

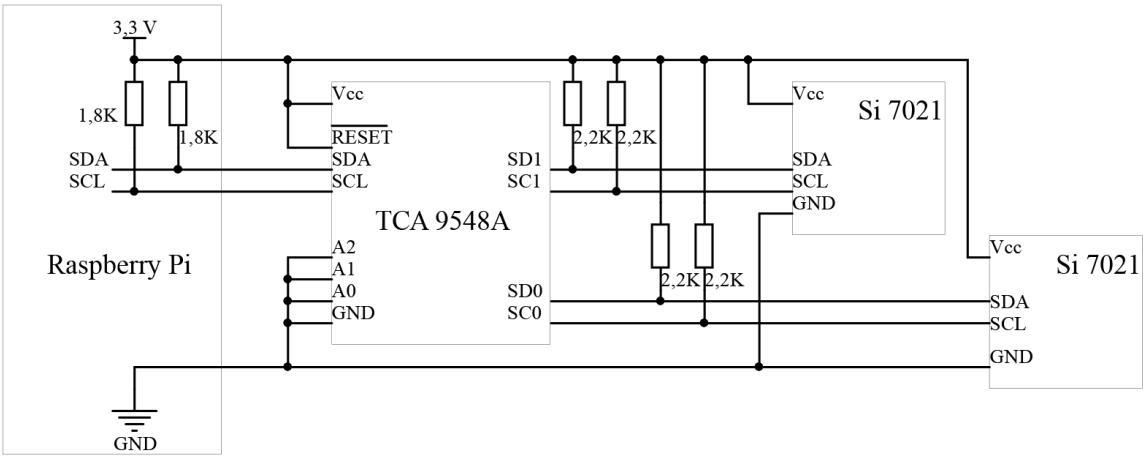
Multiplekser TCA 9548A jest podłączony do wyprowadzeń GPIO2 i GPIO 3, które mogą być aktywowane jako odpowiednio linia SDA i SCL magistrali I2C. Ponadto wyprowadzenia GPIO2 i GPIO3 są podłączone przez wbudowane rezystory podciągające do napięcia zasilania 3,3 V[**?**], co jest zgodne z zaleceniami podanymi w karcie katalogowej.[**?**] Multiplekser jest zasilany z wyprowadzenia 3,3 V platformy Raspberry Pi i podłączony do jej masy. Wyprowadzenia adresowe A2, A1 i A0 są podłączone do

masy, co oznacza, że adres multipleksera to 0x70. Nieużywane wyprowadzenie $\overline{\text{RESET}}$ jest z kolei podłączone do zasilania poprzez rezystor podciągajacy, który ogranicza pobór prądu.

Czujniki Si7021 są przyłączone do kanałów SDx i SCx, gdzie x jest numerem kanału od 0 do 7. W karcie katalogowej czujnika podany jest typowy schemat przyłączenia, w którym użyte są rezystory podciągające 10 kΩ linie SDA i SCL do napięcia zasilania. Producent multipleksera TCA 9548A podaje z kolei w karcie katalogowej wzór **??** pozwalający na wyznaczenie minimalnej wartości rezystorów podciągających.[**?**] Napięcie V_{DPU} jest napięciem linii sygnałowych, które jest równe napięciu zasilania V_{CC}. Napięcie $V_{OL(max)}$ to maksymalne napięcie wyjściowe, a prąd I_{OL} to prąd wyjściowy czujnika. Obie wartości są podane w karcie katalogowej czujnika Si7021.[**?**]

$$R_{p(min)} = \frac{V_{DPU} - V_{OL(max)}}{I_{OL}} = \frac{3,3V - 0,6V}{2,5mA} = 1,08k\Omega \tag{1}$$

Im większa rezystancja rezystora podciągającego tym mniejszy pobór prądu. Rezystor podciągający tworzy wraz z pojemnością magistrali C_b filtr dolnoprzepustowy RC. Jego stała czasowa nie może być na tyle duża, że zbocza zostaną rozciągnięte w czasie tak, że ich czas narastania przekroczy wymagania czasowe magistrali. Pojemność magistrali jest trudna do zmierzenia, więc rezystancja powinna być zbliżona do minimalnej wymaganej. W ramach projektowanego systemu użyte zostały rezystory podciągające 2,2 kΩ. Rysunek **??** przedstawia schemat połączeń między komputerem Raspberry Pi a multiplekserem TCA 9548A i dwoma czujnikami Si7021. Przedstawiono dwa czujniki poglądowo – system może być powiększony o dodatkowe czujniki.



Rysunek 3: Schemat podłączenia multipleksera TCA9548A i czujników Si7021.

Podłączenie większej liczby kamer USB do Raspberry Pi może spowodować chwilowe przekroczenie dopuszczalnego poboru mocy z gniazd USB. Rozwiązaniem tego problemu jest użycie rozgałęziacza (ang. *hub*) USB z zewnętrznym zasilaniem. Zwiększa on jednocześnie zasięg kamer podłączonych przewodowo kamer. Należy uważać jednak, by rozgałęziacz nie zasilał wstecznie (ang. *backfeed*) komputera Raspberry Pi, ponieważ grozi to uszkodzeniem systemu. Z tego powodu w projekcie użyłem dedykowanego rozgałęziacza firmy The Pi Hut.

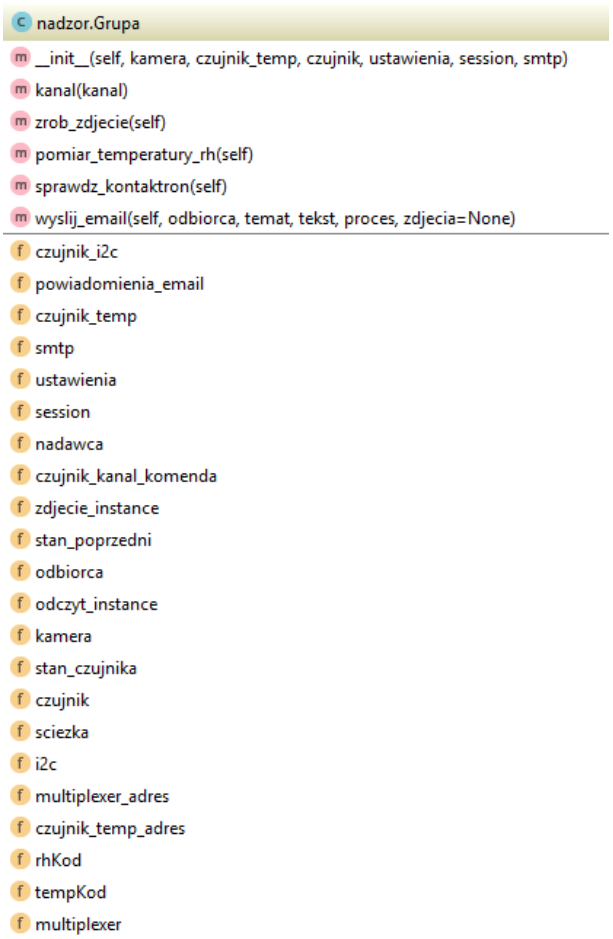
3. Obsługa kamer i czujników

Do napisania programu obsługującego czujniki i kamery wybrałem język Python, ponieważ umożliwia on poprzez wiele dostępnych modułów wysokopoziomową obsługę wyprowadzeń GPIO, magistrali I2C oraz kamer USB. Ponadto istnieje wiele pakietów pozwalających na łatwą komunikację z bazą danych, która będzie miejscem przechowywania wyników pomiaru. Program w języku Python jest też odpowiedzialny za wysłanie powiadomień email poprzez serwer SMTP Google. Na potrzeby projektu zostało stworzone konto w usłudze Gmail, przez które będą wysyłanie powiadomienia.

Zadaniem programu nadzor.py jest cyklicznie wykonywanie pomiarów, odczytów i zdjęć oraz zapis wyników do bazy danych. Potrzebne jest narzędzie, które pozwoli okresowo wykonywać funkcje w ramach jednego programu. Wewnątrz języka Python istnieją mechanizmy (moduł `sched`), które umożliwiają wykonywanie zadań po minięciu pewnego czasu lub zaplanowanie ich do wykonania o konkretnej porze. Wykorzystanie tego modułu wymagałoby jednak ponownego zaplanowania zadania po każdym jego wykonaniu. Modułem, który umożliwia dokonanie tego w prostszy sposób, jest biblioteka `schedule`. Jest ona przeznaczona do planowania cyklicznego wykonywania zadań. Przykładowe wywołanie szeregowania przy użyciu biblioteki `schedule` przedstawiono poniżej.

```
schedule.every(kamera.czestotliwosc_zdjecia).seconds.do(grupa.zrob_zdjecie)
```

Jako argument `every()` podawana jest częstotliwość wykonywania zadania, następnie podawane są jednostki oraz nazwa uruchamianej funkcji jako argument `do()`.



Rysunek 4: Diagram UML klasy Grupa

Rysunek **??** przedstawia strukturę klasy Grupa. Skrótem *m* oznaczone są metody klasy, a *f* - atrybuty. Działanie poszczególnych metod jest opisane w poniższych podrozdziałach.

Program jest napisany przy użyciu metodyki obiektowej. Takie podejście pozwala powiązać czujnik z kamerą, która będzie wykonywała zdjęcia w przypadku zmiany stanu. Klasa Grupa reprezentuje czujnik i kamerę oraz dodatkowo przypisany do nich czujnik temperatury. Zawiera ona metody, które wykonują zdjęcie, pomiar temperatury i wilgotności oraz odczyt stanu wyprowadzenia GPIO, do którego jest dołączony czujnik stykowy. Zarządza ona również wysłaniem wiadomości email z powiadomieniem w przypadku zmiany stanu czujnika.

3.1. Obsługa kamer USB

Kamery USB są obsługiwane przez metodę `wyslij_email`. Do samego wykonywania zdjęć użyta jest aplikacja `fswebcam`. Jest to samodzielny program pozwalający na pobieranie zdjęć z kamer USB podłączonych do komputera z systemem typu Linux. Program ten pozwala na wykonanie zdjęcia z określonej kamery, zdefiniowane rozdzielczości zdjęcia, umieszczenie na zdjęciu podpisu z bieżącą datą i godziną.

Wywołanie programu `fswebcam` z poziomu programu w języku Python wymaga użycia modułu `subprocessing`. Daje on możliwość otwierania programów w osob-

nych procesach. W tym przypadku użyta została klasa `Popen` z tego modułu, otwierająca nowy podproces. Klasa jest dostępna zarówno w wersji języka Python 2 jak i 3. Tworząc obiekt klasy, należy przekazać do niego listę składającą się z nazwy programu, który chcemy uruchomić i jego argumentów. Dodatkowo można określić, czy i gdzie kierować informacje ze standardowych strumieni wejścia/wyjścia procesu:

```
proces = Popen(["fswebcam", "-q", "-d/dev/video0", "-r 640x480", "/var/www/
    html/img/2018-05-24 22:32:10.jpg"], stdout=PIPE, stderr=PIPE)
```

Użytymi argumentami programu `fswebcam` w powyższym przykładowym wywołaniu są:

- `-r` – rozdzielczość ,
- `-d` – nazwa wirtualnego węzła kamery,
- `-q` – tryb cichy,
- ścieżka, gdzie ma być zapisany plik z wykonanym zdjęciem.

Standardowe strumienie wyjścia i błędów są przekierowane do obiektów `pipe`, dzięki czemu ich wartość będzie można następnie odczytać. Program `fswebcam` domyślnie przekierowuje wszystkie komunikaty do strumienia błędów. Użycie trybu cichego zapewnia, że w strumieniu błędów znajdzie się jedynie informacja o błędach.

Zdjęcie jest zapisywane w formacie JPG w rozdzielczości 640x480 pikseli. Jest to maksymalna możliwa rozdzielczość zastosowanej kamery Titanum Onyx. Kolejną komendą jest nazwa wirtualnego węzła kamery (ang. *virtual device node*). Jest to plik, który system Linux tworzy podczas uruchomienia i który jest przypisany do konkretnego urządzenia. Plik ten jest przechowywany w folderze `/dev`. Dzięki użyciu wirtualnego węzła możliwe jest wskazanie konkretnej kamery do wykonania zdjęcia. Zdjęcie jest zapisywane w miejscu wskazanym przez atrybut klasy Grupa o nazwie sciezka. Do przekazywanej do programu ścieżki dołączana jest nazwa zdjęcia w postaci daty zaplanowania jego wykonania. Format daty to "'rok-miesiąc-dzień godzina:minuta:sekunda"' i stanowi unikalny identyfikator zdjęcia. Jest to możliwe, ponieważ w jednej chwili czasu działa wyłącznie jedno wywołanie funkcji `wyslij_zdjecie()`.

Po wywołaniu procesu funkcja oczekuje na jego wykonanie i odbiera informacje ze strumienia wyjścia i błędów przy pomocy metody `communicate()`. Jeśli zawartość strumienia błędów nie jest pusta, podniesiony jest wyjątek `IOError`. Obsługa wyjątku polega na zapisie wartości pustej (`None`) do zmiennej `nazwa`, która zostanie użyta do stworzenia nowego wpisu w bazie danych. Poprawnie wykonane zdjęcie jest więc oznaczane nazwą pliku ze zdjęciem, a niepoprawne – wartością pustą.

3.2. Obsługa czujników podłączonych do magistrali I2C

3.2.1. Komunikacja z multiplekserem TCA9548A

Multiplekser TCA9548A umożliwia przyłączenie do 8 urządzeń korzystających z magistrali I2C. Użyte czujniki temperatury i wilgotności Si 7021 posiadają taki sam adres, więc multiplekser powinien być skonfigurowany do zestawienia komunikacji z każdym z nich osobno. Aktywacja pojedynczego kanału odbywa się poprzez przesłanie do multipleksera 8 bitowego kodu odpowiadającego numerowi kanału. Kod ten tworzony jest

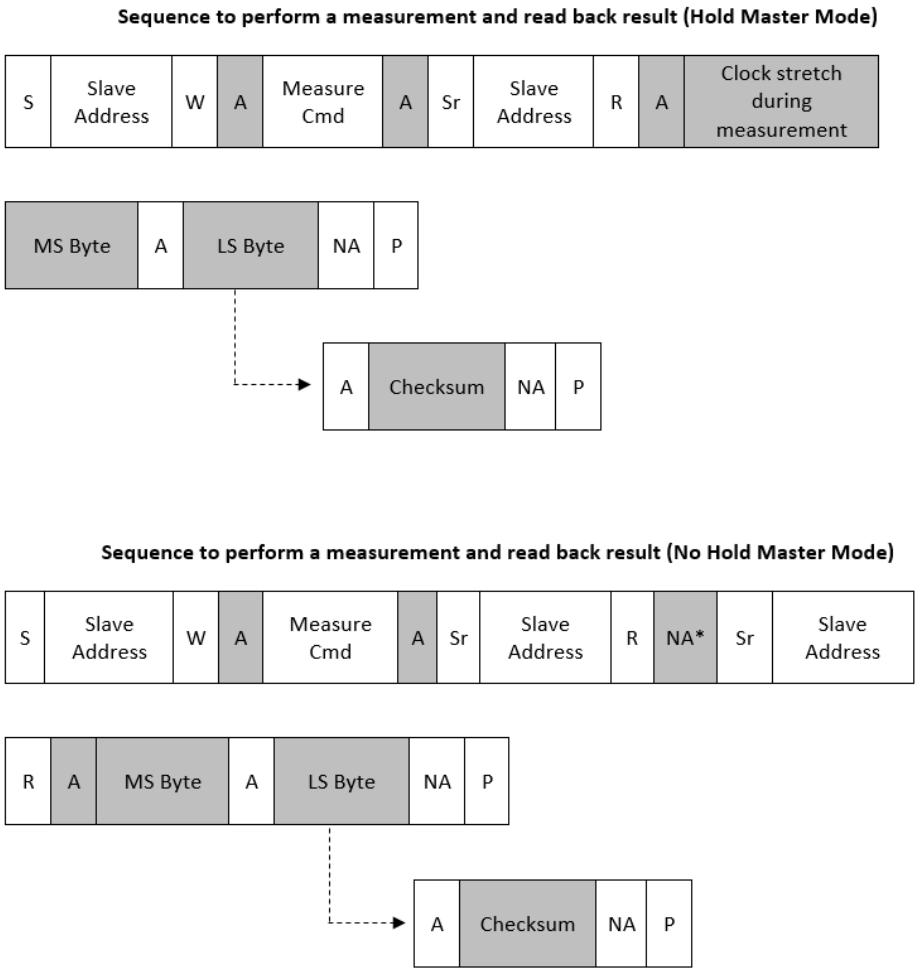
przez ustawienie bitu o pozycji równej numerowi kanału jako 1. Pozostałe pola powinny być ustawione jako 0. Operację tą wykonuje pomocnicza funkcja `kanal`. Zwraca ona 8-bitową komendę odpowiadającą numerowi kanału. Definicja kodu została pokazana w tabeli **??**. Po wysłaniu przez I2C komendy i znaku STOP kanał jest aktywowany. Dalsze komendy I2C można już adresować, używając adresu czujnika temperatury.

Bity rejestru sterującego								Działanie
B7	B6	B5	B4	B3	B2	B1	B0	
X	X	X	X	X	X	X	0 1	Kanał 0 nieaktywny Kanał 0 aktywny
X	X	X	X	X	X	0 1	X	Kanał 1 nieaktywny Kanał 1 aktywny
X	X	X	X	X	0 1	X	X	Kanał 2 nieaktywny Kanał 2 aktywny
X	X	X	X	0 1	X	X	X	Kanał 3 nieaktywny Kanał 3 aktywny
X	X	X	0 1	X	X	X	X	Kanał 4 nieaktywny Kanał 4 aktywny
X	X	0 1	X	X	X	X	X	Kanał 5 nieaktywny Kanał 5 aktywny
X	0 1	X	X	X	X	X	X	Kanał 6 nieaktywny Kanał 6 aktywny
0 1	X	X	X	X	X	X	X	Kanał 7 nieaktywny Kanał 7 aktywny

Tabela 3: Definicja bajtu sterującego. Źródło: [**?**]

3.2.2. Komunikacja z czujnikiem Si7021

Komunikacja z czujnikiem Si7021 odbywa się przez magistralę I2C. Pierwszą komendą, którą należy wysłać do czujnika, jest komenda wykonania pomiaru wilgotności względnej. W dokumentacji czujnika Si7021 opisane są dwie metody pomiaru - Hold Master Mode oraz No Hold Master Mode. Ich porównanie zostało przedstawione na rysunku **??**. Kolorem białym są oznaczone komendy i dane wysyłane przez urządzenie nadrzędne (ang. *master*), a szarym przez urządzenie podrzędne (ang. *slave*). W pierwszej z nich urządzenie nadrzędne wysyła żądanie pomiaru (*Measure Cmd*). Po potwierdzeniu odbioru wysyłane jest żądanie odczytu (*R*). Urządzenie podrzędne potwierdza otrzymanie żądania i dokonuje pomiaru. Wymaga to zastosowania rozciągania zegara (ang. *clock stretching*), które polega na utrzymywaniu przez urządzenie podrzędne linii zegarowej SCK w stanie niskim. Dzieje się to aż do momentu zakończenia pomiaru przez urządzenie i wpisaniu jego wyniku do rejestru. Wynik pomiaru składa się z dwóch bajtów, które należy odebrać w jednej transakcji. Ostatecznie komunikacja kończy się poprzez wystawienie znaku STOP (*P*). Drugi tryb pomiaru (No Hold Master Mode) różni się tym, że po wysłaniu kodu pomiaru (0xF5) oraz żądania odczytu urządzenie nie potwierdza odbioru aż do momentu zakończenia pomiaru. Następnie należy odczytać dwubajtowy wynik pomiaru z rejestru czujnika.



Rysunek 5: Porównanie sekwencji komend I2C do wykonania pomiaru czujnikiem Si7021. Źródło: [**?**]

Do programowej komunikacji przez I2C potrzebna była odpowiednia biblioteka. Pierwszym zastosowanym modułem był moduł `python-smbus`. Korzysta on ze sterownika wbudowanego w jądro systemu Linux. Do przeprowadzenia pomiaru w trybie Hold Master Mode można użyć funkcji `i2c_smbus_read_word_data()` lub `i2c_smbus_read_i2c_block_data()`. Obie funkcje są zgodne z przedstawioną powyżej sekwencją pomiaru, lecz różnią się liczbą odebranych bajtów. Pierwsza funkcja odbiera dokładnie dwa bajty, a druga odbiera bajty z urządzenia do momentu zakończenia komunikacji przez urządzenie podrzędne[**?**]. Próba przeprowadzenia pomiaru przy ich użyciu zakończyła się błędem o kodzie `io errno5`. Wynikał on z tego, że pakiet `python-smbus` obsługuje magistralę I2C zgodnie ze standardem SMBus – rozszerzeniem I2C. Posiada on bardziej ścisłe reguły dotyczące czasu trwania transakcji na magistrali. Obie wymienione funkcje odczytu pakietu `python-smbus` są poprzedzone wysłaniem żądania do urządzenia. Oznacza to, że nie można przy ich pomocy odczytać samego wyniku pomiaru po wysłaniu kodu pomiaru w trybie No Hold Master Mode (0xE5). Inna dostępna funkcja `i2c_smbus_read_byte()` pozwala na odczyt wyłącznie jednego bajtu. Dwukrotne wysłanie żądania przy pomocy tej funkcji prowadzi do dwukrotnego odczytania pierwszego bajtu pomiaru. Z tych powodów moduł `python-smbus` nie może zostać wykorzystany do obsługi czujnika Si7021.

Innym pakietem umożliwiającym komunikację poprzez magistralę I2C jest moduł `pigpio`. Umożliwia on obsługę wyprowadzeń GPIO na platformi Raspberry Pi w tym tych, które są skonfigurowane jako magistrala I2C. Opiera on swoje działanie na bibliotece napisanej w języku C. Przed rozpoczęciem działania pakietu konieczne jest uruchomienie programu `pidpiod`. Jest to demon – program działający w tle bez interakcji z użytkownikiem. Musi on działać, zanim wywołany zostanie program `nadzor.py`. Można to zapewnić, korzystając z narzędzia `cron`. Wpis do jego tabeli z wywołaniem programu `pigpiod` poprzedzony atrybutem `@reboot` umieszczony przed podobnym wpisem programu `nadzor.py` zapewnia, że program zostanie uruchomiony za każdym razem, gdy uruchamiany będzie system operacyjny.

Pierwszym krokiem jest stworzenie obiektu klasy `pigpio.pi`. Obiekt ten jest przechowywany jako atrybut *i2c* klasy Grupa. Następnie konieczne jest otwarcie komunikacji z urządzeniem i zwrócenie identyfikatora, który będzie używany do wysyłania żądań do urządzeń. Funkcje tego modułu umożliwiają przeprowadzenie pomiaru wilgotności względnej w trybie *No Hold Master Mode*. W pierwszej kolejności wysyłana jest 8-bitowa komenda pomiaru przy pomocy funkcji `i2c_write_byte()`. Realizuje ona sekwencję pomiaru do momentu potwierdzenia odbioru komendy przez urządzenie podrzędne. Następnie urządzenie oczekuje znaku powtórzonego startu (*Sr*). Dopuszczalne jest jednak również przesłanie znaku STOP i zakończenie komunikacji.

Następnie program jest usypiany na czas 0.05s przy pomocy komendy `time.sleep()`. Jest to wartość większa niż najdłuższy czas konwersji pomiaru wilgotności względnej podany w karcie katalogowej czujnika Si 7021.[**?**] W tym czasie wykonywany jest pomiar i można przejść do tej części procedury, która składa się z wysłaniu bitu odczytu (*R*) i odebrania dwóch bajtów wyniku pomiaru. Można to zrealizować poprzez funkcję `i2c_read_device()`[**?**]. Funkcja zwraca liczbę odczytanych bajtów oraz odczytane bajty w formie tablicy. Bajty te można następnie wykorzystać do obliczenia wilgotności względnej na podstawie wzoru **??** podanego w karcie katalogowej.[**?**] Kod_{RH} oznacza tablicę bajtów, do której został zapisany wynik pomiaru.

$$\%RH = \frac{(Kod_{RH}[0] * 256 + Kod_{RH}[1]) * 125}{65536} - 6 \tag{2}$$

Czujnik Si 7021 wykonuje pomiar temperatury w ramach procedury wyznaczenia wilgotności względnej. Wynik tego pomiaru jest przechowywany w urządzeniu. Można go odczytać, wysyłając komendę o kodzie 0xE0. Całą transakcję można zrealizować przy użyciu funkcji `i2c_read_i2c_block_data` bez obawy o rozciąganie zegara, ponieważ nie ma potrzeby oczekiwania na wykonanie pomiaru temperatury. Dwa bajty wyniku pomiaru są zapisane do tablicy i użyte do obliczenia wartości temperatury na podstawie wzoru **??**.[**?**]

$$Temperatura(^{\circ}C) = \frac{(Kod_{temp}[0] * 256 + Kod_{temp}[1]) * 175,72}{65536} - 46,85 \tag{3}$$

Obsługa wyjątków w komunikacji przy użyciu magistrali I2C jest przeprowadzona poprzez wypisanie do konsoli komunikatu o typie urządzenia, dla którego nastąpił błąd (multiplekser albo czujnik). W takim wypadku wartościom temperatury i wilgotności są przypisywane wartości `None`. Są one następnie zapisane w bazie danych. Jeżeli natomiast nie wystąpiły błędy, w bazie danych zapisywane są obliczone wartości temperatury i wilgotności.

3.3. Obsługa czujników stykowych

Korzystanie z wyprowadzeń GPIO w języku Python na komputerze Raspberry Pi jest możliwe przy użyciu pakietu `RPi.GPIO`. Umożliwia on odczyt stanu wyprowadzeń. Konfiguracja obsługi GPIO rozpoczyna się od określenia sposobu numeracji wyprowadzeń. Dostępne są dwie możliwości: GPIO.BOARD oraz GPIO.BCM. Pierwszy odnosi się do fizycznej lokalizacji wyprowadzeń na płytce drukowanej. Drugi sposób oznacza numerację kanałów zgodną z układem system-on-chip (SOC) firmy Broadcom, który jest użyty w komputerze Raspberry Pi 3B. Ze względu na korzystanie z płytki prototypowej oraz modułu Proto Pi Plus, który korzysta z oznaczeń odpowiadających kanałom SOC, w projekcie użyto tego właśnie sposobu numeracji.

Inicjalizacja obsługi GPIO wywoływana jest w funkcji:

```
def init_gpio():
    GPIO.setmode(GPIO.BCM)
```

Wykonywana jest ona na początku działania programu `nadzor.py` w funkcji `main()`. Obsługa poszczególnych czujników odbywa się w ramach obiektów klasy Grupa. Aktywacja rezystora ściągającego (ang. *pull-down*) do masy jest wywoływana w programie `nadzor.py` w konstruktorze obiektu klasy Grupa w następujący sposób:

```
GPIO.setup(self.czujnik.gpio, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

Pierwszym argumentem tej funkcji jest numer wyprowadzenia GPIO, drugi określa, czy wyprowadzenie ma być skonfigurowane jako wejście (`GPIO.IN`) czy wyjście (`GPIO.OUT`). Trzeci argument to znacznik określający, czy ma zostać użyty rezystor ściągający do masy czy podciągający do napięcia zasilania (ang. *pull-up*).

Funkcja realizująca obsługę czujników stykowych zaczyna od sprawdzenia wyprowadzenia o numerze przechowywanym jako atrybut klasy. Zwrócenie stanu wysokiego (oznaczanego jako `1/GPIO.HIGH/True`) oznacza przypisanie do zmiennej `stan_czujnika` wartości 1. Jeśli natomiast stan jest niski, to zmiennej przypisywana jest wartość 0. Następnie sprawdzany jest poprzedni stan wyprowadzenia przechowywany jako atrybut obiektu klasy Grupa. Jeśli poprzednio stan był wysoki (styki czujnika był zamknięte), to znaczy, że nastąpiło otwarcie czujników. Zgodnie z wymogami projektowanego systemu następuję wtedy wykonanie zdjęcia - funkcji obsługującej kamery. Na koniec program przechodzi do wysłania powiadomienia e-mail.

3.4. Wysyłanie powiadomień e-mail

Funkcjonalnością systemu jest również wysyłanie powiadomienia mailowego po otwarciu czujnika. Jest ono wysyłane przez serwer SMTP Google. Ta decyzja projektowa wynika z tego, że konfiguracja serwera SMTP działającego na Raspberry Pi wymagałaby posiadania stałego adresu IP dostarczanego przez dostawcę usług internetowych oraz domeny przypisanej do tego adresu. Prostszym rozwiązaniem jest utworzenie konta e-mail w zewnętrznej usłudze (np. Gmail) i korzystanie z jej serwera SMTP.

Połączenie z serwerem SMTP Google jest nawiązywane przy użyciu modułu `smtplib` na początku działania programu `nadzor.py`:

```
def init_smtp(sender, password):
```

```
    smtp_server = smtplib.SMTP_SSL("smtp.gmail.com", 465)
    smtp_server.login(sender, password)
    return smtp_server
```

Wymogiem usługi Gmail jest stosowanie szyfrowanego połączenia SSL, które jest nawiązywane przez funkcję `SMTP_SSL`. Komunikacja odbywa się na standardowym dla tego połączenia porcie TCP 465. Następnie dokonuje się uwierzytelnienie użytkownika na serwerze poprzez przesłanie adresu email konta, z którego mają być wysyłane wiadomości, oraz hasła do niego. Funkcja zwraca referencję obiektu reprezentującego połączenie z serwerem SMTP.

W przypadku, gdy zostanie otwarty czujnik stykowy, oprócz wykonania zdjęcia, sprawdzany jest stan znacznika powiadomień e-mail. Jeśli jest on równy "on", a adres e-mail odbiorcy powiadomień nie jest pusty, rozpoczyna się procedura wysłania powiadomienia. Przygotowany jest tekst wiadomości zawierający informację o otwarciu czujnika i przypisanej mu nazwie. Na temat wiadomości składa się informacja o otwarciu czujnika oraz dacie i godzinie otwarcia. Następnie zostaje utworzony nowy wątek, w którym jest wywoływana funkcja `wyslij_email()`. Argumentami tej funkcji są: adres e-mail odbiorcy, temat wiadomości, jej treść oraz nazwa zdjęcia, która będzie dołączona jako załącznik. Do otwarcia wątku użyta jest pomocnicza funkcja `run_threaded`. Użycie wątku zapewnia, że funkcja wysyłająca powiadomienie jest wykonywana współbieżnie z pozostałymi funkcjami sprawdzającymi stany czujników.

Do przygotowania wiadomości e-mail jest wykorzystany obiekt klasy `MIMEMultipart`. Reprezentuje on wieloczęściową wiadomość e-mail zgodną ze standardem MIME (ang. *Multipurpose Internet Mail Extensions*[**?**][**?**]. W polach: `Subject`, `To` i `From` zostają wpisane odpowiednio temat, odbiorca i nadawca wiadomości e-mail. Następnie funkcja otwiera plik ze zdjęciem, które zostało wywołane przez otwarcie czujnika i tworzony jest obiekt `MIMEImage`, który reprezentuje część wiadomości będącej obrazem. Nie jest tu podawany format zdjęcia, ponieważ obiekt sam dokona sprawdzenia typu zdjęcia przy pomocy modułu `imghdr`[**?**]. Dzięki temu zmiana formatu zdjęcia np. z JPEG na PNG nie wymaga zmiany tej części kodu. Przed dołączeniem do wiadomości pliku ze zdjęciem konieczne jest również ustawienie nagłówka `Content-Disposition` z informacją o tym, że jest to załącznik (*attachment*) oraz o jego nazwie. Dzięki temu obraz zostanie dodany jako załącznik do pobrania w kliencie poczty elektronicznej[**?**]. Po dołączeniu załączników następuje wysłanie wiadomości e-mail do serwera SMTP.

4. Baza danych

4.1. Założenia wstępne

Relacyjna baza danych składa się relacji (tabel), które są połączone związkami. W takim modelu organizacji bazy danych łatwo przedstawić rzeczywiste obiekty. Tabela składa się z nagłówka i zawartości. Nagłówek to zbiór atrybutów opisujących zawartość składającą się ze zbioru wierszy. Każda tabela posiada klucz główny, który pozwala jednoznacznie zidentyfikować każdy wiersz. Związki między relacjami są realizowane przez obecność w jednej z tabel uczestniczących w związku klucza obcego, który pozwala jednoznacznie zidentyfikować wiersz z drugiej tabeli.

Programistyczny dostęp do baz danych jest możliwy przez programistyczny interfejs silnika bazy danych opartego na transakcjach. Transakcje to zestaw operacji, które powinny być wykonane w całości albo wcale. Dzięki nim możliwe jest zachowanie integralności danych w sytuacji, gdy wiele klientów (programów) zapisuje dane do bazy danych lub odczytuje z niej.

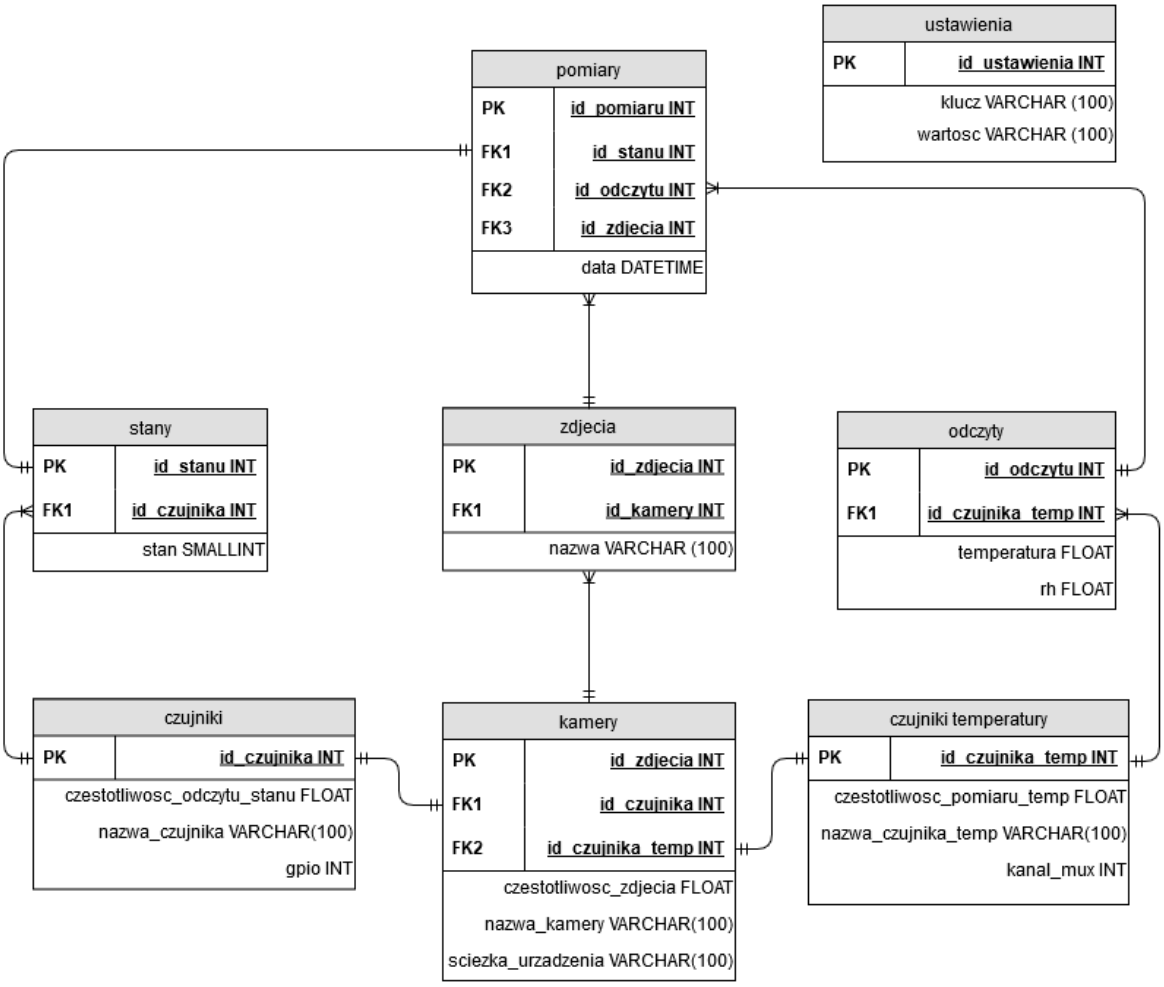
Projekt bazy danych należy rozpocząć od zdefiniowania, jakie obiekty mają być przechowywane w bazie. W projektowanym systemie potrzebne są informacje o czujnikach i kamerach podłączonych do Raspberry Pi oraz pomiarach wykonanych przez nie. Wszystkie urządzenia mają nazwę identyfikującą ich położenie lub przeznaczenie np. "Kamera w kuchni" lub "Czujnik - drzwi wejściowe". Każdy typ urządzenia różni się jednak specyficznymi dla niego informacjami. Czujnik temperatury dodatkowo potrzebuje informacji o numerze kanału multipleksera, do którego jest podłączona, a czujnik stykowy o numerze swojego wyprowadzenia GPIO. Ponadto każdy typ urządzenia wykonuje inny typ pomiaru i odczytu i z tego powodu informacje o urządzeniach będą przechowywane w różnych tabelach – osobno kamery, czujniki stykowe i czujniki temperatury.

Baza danych powinna przechowywać informację o stanie czujnika zbliżeniowego, wartości zmierzonej temperatury i wilgotności oraz wykonanym zdjęciu. W bazie danych nie muszą być przechowywane pliki ze zdjęciami, może być to unikalny identyfikator zdjęcia, który pozwoli zlokalizować je w pamięci urządzenia. Zdjęcia, pomiary temperatury i odczyty czujnika stykowego będą wykonywane z różną częstotliwością. Użytkownik potrzebuje natomiast powiązania, jaki w danej chwili jest stan czujnika, odczyt temperatury i wilgotności oraz zdjęcie nadzorowanego miejsca. Konieczna jest więc tabela, która będzie zbierała dla każdej grupy urządzeń składającej się z czujnika, czujnika stykowego i kamery informacje o ostatnich wykonanych pomiarach. Wpisy do tej tabeli muszą być wykonywane tak często, jak najczęściej wykonywany pomiar. Pozostałe wielkości będą ostatnio zmierzonymi. Pozwala to na jasne dopasowanie zdjęcia oraz zmierzonej temperatury i wilgotności do odczytu stanu.

4.2. Projekt bazy danych

Na podstawie powyższych założeń stworzyłem projekt bazy danych, który jest przedstawiony na rysunku **??**. Przedstawiony schemat bazy danych na poziomie logicznym to zbiór relacji i związków między nimi, które mogą zostać stworzone w systemie zarządzania bazą danych.

Baza zawiera tabele `czujniki`, `kamery` i `czujniki_temperatury` z informacjami o poszczególnych typach urządzeń dołączonych do systemu. Wszystkie tabele



Rysunek 6: Schemat bazy danych na poziomie logicznym

mają podobne pole przechowujące nazwę urządzenia typu VARCHAR(100). Każde urządzenie przechowuje również częstotliwość wywoływania zdjęcia, pomiaru lub odczytu w polu typu FLOAT. Wartości te są wyrażone w sekundach. Ponadto każde urządzenie przechowuje informację pozwalającą zlokalizować je fizycznie w systemie. W przypadku czujników stykowych jest to numer wyprowadzenia GPIO, do którego podłączony jest czujnik. Czujniki temperatury i wilgotności są identyfikowane przez numer kanału multipleksera, do którego są dołączone. Kamery można rozróżnić dzięki polu `sciezka_urzadzenia` zawierającym ścieżkę pliku wirtualnego węzła urządzenia. Jest to plik, który system Linux używa do identyfikowania dołączonych urządzeń.

Tabele `stany`, `zdjęcia` i `odczyty` zawierają informacje o realizacjach pomiarów wykonanych przez te urządzenia. Relacja `stany` odpowiada odczytom stanów czujników stykowych. Każdy wpis w tabeli jest jednym odczytem stanu jednego czujnika. Odczyt stanu jest przechowywany w polu `stan` typu SMALLINT, ponieważ jest zaprojektowany do przechowywania tylko wartości 0 albo 1. W tabeli `odczyty` przechowywane są pomiary temperatury i wilgotności względnej z jednego czujnika temperatury. Są one liczbami zmiennoprzecinkowymi i dlatego są zapisywane w polach typu FLOAT. Jeśli pomiar został wykonany błędnie, zostanie dokonany wpis do tabeli `odczyty` zawierający wartości NULL w polach `temperatur` i `rh`. Wpisy w tabeli `zdjęcia` za-

wierają nazwę pliku ze zdjęciem wykonanym przez kamerę. Jeśli jednak zdjęcie nie zostanie wykonane prawidłowo, w polu `nazwa` zostanie wpisana wartość NULL.

Tabela `pomiary` przedstawia status części systemu w czasie, zawierając klucze obce aktualnych w danym momencie zdjęć, odczytów i stanów, pochodzących ze zgrupowanych urządzeń. Jedno zdjęcie lub odczyt temperatury i wilgotności może być przypisane do wielu pomiarów, ale do jednego pomiaru jest przypisany tylko jeden stan czujnika stykowego. Wynika to z tego, że w programie `nadzor.py` wpisy do tabeli `pomiary` są wykonywane po odczycie stanu. Następuje on domyślnie co 0,1 sekundy, aby wykrywać oddalenie czujników. Data jest przechowywana w polu o typie DATETIME.

Poza wspomnianymi wyżej relacjami istnieje również relacja `ustawienia`, która nie jest w związku z żadną inną. Jest to tabela przechowująca różne ustawienia systemu, które nie dotyczą bezpośrednio urządzeń. Posiada ona dwie kolumny: `klucz` i `wartosc`. Kolumna `klucz` jest opisem ustawienia, a `wartosc` przechowuje informację o nim np. adres e-mail, na który mają być wysyłane powiadomienia.

Ponadto kamera, czujnik temperatury i czujnik stykowy, który ma wywoływać zdjęcie zostały zgrupowane. Każda kamera przechowuje klucze przypisanych do niej czujników. Odpowiada to związkom jeden do jednego między relacjami `kamery` i `czujniki` oraz `kamery` i `czujniki_temperatury`.

4.3. Praktyczna realizacja bazy danych

Baza danych została zaimplementowana przy użyciu systemu zarządzania bazą danych MySQL. Do stworzenia tabel został napisany skrypt w języku Python. Wykorzystuje on moduł `SQLAlchemy`, który umożliwia działanie na relacjach bazy danych jak na obiektach programistycznych. Takie odwzorowanie nazywa się mapowaniem obiektowo-relacyjnym (ang. *Object-Relational Mapping* ORM). Ułatwia ono wprowadzanie kolejnych zmian do struktury bazy danych, relacji i związków między nimi.

Bazę danych należy stworzyć, wywołując polecenie w panelu zarządzania bazą danych przy użyciu polecenia:

```
CREATE DATABASE nadzor;
```

Plik zawierający program tworzący tabele w bazie danych nosi nazwę `baza.py`. W pierwszej kolejności należy stworzyć klasę `Base`, która zapewnia mapowanie do tabel w bazie danych. Proces tworzenia tabel w bazie polega na stworzeniu klasy odpowiadającej każdej tabeli w bazie, która dziedziczy po klasie `Base`.[**?**]

```
class Pomiary(Base):
    __tablename__ = 'pomiary'
    id_pomiaru = Column(Integer, primary_key=True)
    id_stanu = Column(Integer, ForeignKey('stany.id_stanu', ondelete='
   CASCADE'), nullable=False)
    id_odczytu = Column(Integer, ForeignKey('odczyty.id_odczytu', ondelete=
   'CASCADE'), nullable=False)
    id_zdjecia = Column(Integer, ForeignKey('zdjecia.id_zdjecia', ondelete=
   'CASCADE'), nullable=False)
    data = Column(DATETIME)
    stany = relationship(Stany)
    odczyt = relationship(Odczyty)
```

```
    zdjecia = relationship(Zdjecia)
```

Klasa Pomiary posłuży jako przykład tworzenia tabeli w bazie danych. Należy zdefiniować atrybut `__tablename__`, który jest unikalną nazwą tabeli. Następnie tworzone są kolumny przez wywołanie konstruktora klasy Column. W konstruktorze należy przekazać argument będący typem danych przechowywanych w kolumnie. Dla kolumny będącej kluczem głównym należy ustawić znacznik `primary_key` jako `True`. Do zdefiniowania związku z inną relacją należy podać jako argument obiekt klasy `ForeignKey` z informacją, która kolumna w danej tabeli służy jako klucz obcy. Następnie należy zdefiniować związek, wywołując funkcję `relationship()`. Przyjmuje ona jako argument nazwę klasy. Oznacza to, że definicja tej klasy musi być stworzona wcześniej w kodzie programu.

Dla wszystkich użytych czujników i kamer należy również stworzyć rekordy w bazie danych, odpowiadające im. Odpowiada za to program `insert.py`. Proces tworzenia nowego wpisu w tabeli polega na stworzeniu słownika zawierającego nazwy kolumn i wartości, które mają być im przypisane. Następnie wywoływana jest funkcja `get_or_create()`, która najpierw sprawdza, czy istnieje rekord w bazie danych o takich wartościach. Jeśli już istnieje, to zwraca go. Jeśli nie istnieje, to tworzy nowy rekord. Dzięki temu do programu można dopisywać kolejne wywołania, tworzące nowe rekordy w bazie danych i uruchamiać program, który nie stworzy duplikatów już istniejących rekordów. Poniżej przedstawiono przykład stworzenia nowego wpisu w tabeli `kamery` zgodnie z opisaną wyżej procedurą. Argumentami funkcji `get_or_create()` są:

- obiekt klasy `Session`, która zarządza transakcjami z bazą danych[**?**],
- nazwa klasy odpowiadającej tabeli, dla której ma być stworzony obiekt,
- słownik argumentów kluczowych poprzedzony operatorem `**`.

```
titanum_1_kwargs = {
    "nazwa_kamery": "Kamera 1 - Titanum",
    "sciezka_urzadzenia": "/dev/video0",
    "id_czujnika": kontaktron_1.id_czujnika,
    "id_czujnika_temp": si7021_zielony.id_czujnika_temp,
    "czestotliwosc_zdjecia": 10.0
}
titanum_1 = get_or_create(session, Kamery, **titanum_1_kwargs)
```

W podobny sposób dokonywane są zapisy do tabel `stany`, `zdjęcia` i `odczyty`. Wykonywane są one w programie `nadzor.py` odpowiednio w funkcjach: `sprawdz_kontaktron()`, `pomiar_temperatury_rh` i `zrob_zdjecie`. Przykładowo, wyniki pomiarów temperatury i wilgotności zostają zapisane do słownika przy kluczach odpowiadających nazwom odpowiednich kolumn w tabeli. Pozostałym polem w słowniku jest identyfikator czujnika stykowego, dla którego został wykonany odczyt. Następnie wynik zostaje zapisany do bazy danych poprzez funkcję `create()`. Jej argumenty są takie same jak opisanej wyżej funkcji `get_or_create()`. Zwrócony obiekt odpowiadający nowemu wpisowi w bazie danych jest zapisywany w odpowiednim polu klasy Grupa.

```
odczyt = {
  "id_czujnika_temp": self.czujnik_temp.id_czujnika_temp,
  "temperatura": temp,
  "rh": rh
}
self.odczyt_instance = create(self.session, Odczyty, **odczyt)
```

Odczyt czujników stykowych jest najczęściej wykonywanym pomiarem, więc po stworzeniu nowego wpisu w tabeli `stany` powinien zostać stworzony wpis do tabeli pomiary. Zostaje do niej wpisany rekord zawierający identyfikator właśnie wykonanego odczytu stanu oraz identyfikatory ostatnich wykonanych zdjęć oraz pomiarów temperatury odczytane z pól obiektu klasy Grupa. Wraz z nimi zostaje zapisana data wpisu odczytana przy pomocy funkcji `datetime.now()` zapisana w formacie *dzień-miesiąc-rok godzina-minuta-sekunda.mikrosekunda*.

```
pomiar = {
  "id_stanu": stan_instance.id_stanu,
  "id_odczytu": self.odczyt_instance.id_odczytu,
  "id_zdjecia": self.zdjecie_instance.id_zdjecia,
  "data": data
}
pomiar_instance = create(self.session, Pomiary, **pomiar)
```

5. Strona i serwer WWW

5.1. Serwer WWW

Wymogiem systemu jest umożliwienie użytkownikowi dostępu do zebranych danych poprzez stronę WWW. Konieczny jest zatem serwer HTTP, na którym będzie umieszczona strona. Jednym z najbardziej środowisk do prowadzenia serwerów WWW jest Apache. Jest on dostępny w wersji dla systemu Raspbian, który działa na Raspberry Pi. Umieszczona na serwerze strona będzie komunikowała się z serwerem poprzez żądania HTTP. Szkielet strony w postaci pliku HTML będzie wypełniony danymi zebranymi przez kamery i czujniku po zrealizowaniu żądań wysyłanych asynchronicznie (technika AJAX) do serwera. Za obsługę żądań po stronie serwerowej będą odpowiedzialne mikroserwisy w postaci skryptów w języku PHP. Skrypty łączą się z bazą danych, w której są przechowywane dane zebrane z kamer i czujników. Następnie zwracają dane w odpowiedzi na żądania klienta. Język PHP jest dobrze zintegrowany z serwerem Apache i jest jedną z najbardziej popularnych technologii na serwerach WWW.

5.2. Żądania HTTP i ich obsługa – technika AJAX

5.2.1. Strona główna i archiwum

5.2.2. Eksport zdarzeń

5.2.3. Ustawienia

5.3. Interfejs użytkownika

6. Testy systemu

7. Wnioski i podsumowanie

8. Bibliografia

Literatura

[1] Arundel A. V., Sterling E. M., Biggin J. H. i Sterling T. D., Indirect health effects of relative humidity in indoor environments, *Environmental Health Perspectives* 1986;65:351-361 https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1474709/

[2] Bayer M., Session Basics, Dokumentacja SQLAlchemy http://docs.sqlalchemy.org/en/latest/orm/session_basics.html

[3] Bayer M., Declarative API, Dokumentacja SQLAlchemy http://docs.sqlalchemy.org/en/latest/orm/extensions/declarative/api.html

[4] BeagleBoard.org Foundation, Specyfikacja BeagleBone Black https://elinux.org/Beagleboard:BeagleBoneBlack

[5] Dziedzic K., Kamera IP w sieci. Stwórz prywatny monitoring http://www.komputerswiat.pl/poradniki/sprzet/kamery-internetowe/2015/07/kamera-ip-w-sieci.aspx

[6] Freed N., Borenstein N., Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies, RFC 2045, IETF, listopad 1996, DOI: 10.17487/RFC2045. https://tools.ietf.org/html/rfc2045

[7] Freed N., Borenstein N., Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, RFC 2046, IETF, listopad 1996, DOI: 10.17487/RFC2046. https://tools.ietf.org/html/rfc2046

[8] Hoperf Electronic, Karta katalogowa czujnika temperatury i wilgotności TH02: http://www.hoperf.com/upload/sensor/TH02_V1.1.pdf

[9] joan2937, Dokumentacja funkcji `i2c_read_device()` http://abyz.me.uk/rpi/pigpio/python.html#i2c_read_device

[10] Mouser Electronics, Karta katalogowa czujnika temperatury i wilgotności SHT31D: http://www.mouser.com/ds/2/682/Sensirion_Humidity_Sensors_SHT3x_Datasheet_digital-971521.pdf

[11] Porównanie komputerów jednopłytkowych (ang.) https://en.wikipedia.org/wiki/Comparison_of_single-board_computers

[12] Python Software Foundation, Dokumentacja modułu Python email.mime https://docs.python.org/3/library/email.mime.html

[13] Raspberry Pi Foundation, Schematy peryferiów Raspberry Pi 3 Model B https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/rpi_SCH_3b_1p2_reduced.pdf

[14] Raspberry Pi Foundation, Specyfikacja Raspberry Pi 3 Model B https://www.raspberrypi.org/products/raspberry-pi-3-model-b/

[15] Reschke J. Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP), RFC 6266, IETF, czerwiec 2011, DOI: 10.17487/RFC6266. https://tools.ietf.org/html/rfc6266

[16] Silicon Labs, Karta katalogowa czujnika temperatury i wilgotności Si7021: https://www.silabs.com/documents/public/data-sheets/Si7021-A20.pdf

[17] Texas Instruments Inc., Karta katalogowa multipleksera TCA9548A: http://www.ti.com/lit/ds/symlink/tca9548a.pdf

[18] Torvalds L., Opis protokołu SMBus https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/plain/Documentation/i2c/smbus-protocol

9. Załączniki