



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

<b>1. Wstęp teoretyczny</b>	<b>3</b>
1.1. Współczesne systemy monitoringu i bezpieczeństwa . . . . .	3
1.2. Koncepcja "inteligentnego domu" . . . . .	3
1.3. Motywacje i cel pracy . . . . .	3
<b>2. Architektura systemu</b>	<b>3</b>
2.1. Wymagania techniczne i funkcjonalne . . . . .	3
2.2. Przegląd dostępnych platform . . . . .	3
2.3. Czujniki warunków środowiskowych . . . . .	3
2.4. Czujniki stykowe . . . . .	3
2.5. Kamery cyfrowe . . . . .	3
2.6. Projekt części sprzętowej systemu . . . . .	3
<b>3. Program procedury obsługi kamer i czujników</b>	<b>5</b>
3.1. Obsługa kamer USB . . . . .	5
3.2. Obsługa czujników podłączonych do magistrali I2C . . . . .	6
3.3. Obsługa czujników stykowych . . . . .	8
3.4. Wysyłanie powiadomień e-mail . . . . .	9
<b>4. Baza danych</b>	<b>10</b>
4.1. Założenia wstępne . . . . .	10
4.2. Projekt bazy danych . . . . .	10
4.3. Praktyczna realizacja bazy danych . . . . .	12
<b>5. Strona i serwer WWW</b>	<b>13</b>
5.1. Serwer WWW . . . . .	13
5.2. Żądania HTTP i ich obsługa – technika AJAX . . . . .	13
5.2.1. Strona główna i archiwum . . . . .	13
5.2.2. Eksport zdarzeń . . . . .	13
5.2.3. Ustawienia . . . . .	13
5.3. Interfejs użytkownika . . . . .	13
<b>6. Testy systemu</b>	<b>13</b>
<b>7. Wnioski i podsumowanie</b>	<b>13</b>
<b>8. Bibliografia</b>	<b>13</b>
<b>9. Załączniki</b>	<b>13</b>

# 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

## 2.2. Przegląd dostępnych platform

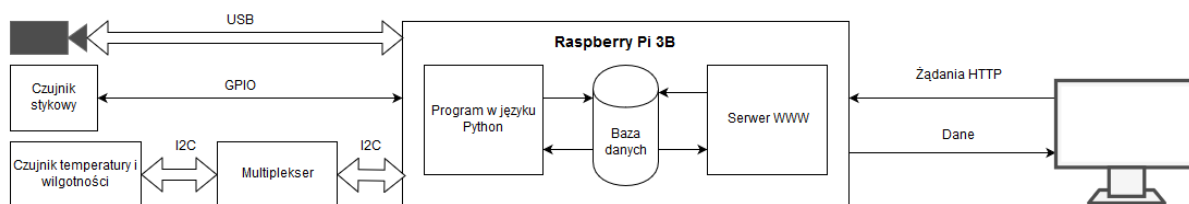
## 2.3. Czujniki warunków środowiskowych

Wymogiem technicznym systemu jest również obsługa czujników przy użyciu magistrali I2C. W ramach wykonanego systemu została zapewniona obsługa czujników temperatury i wilgotności. Wybrany został czujnik Si7021 firmy Adafruit. Zapewnia on pomiar temperatury w zakresie od -10 do 85°C oraz wilgotności względnej od 0 do 80%. Są to wartości wystarczające w warunkach pracy w zamkniętym pomieszczeniu.

## 2.4. Czujniki stykowe

## 2.5. Kamery cyfrowe

## 2.6. Projekt części sprzętowej systemu



Rysunek 1: Schemat architektury systemu

Rysunek 1 przedstawia projekt architektury systemu. System oparty jest o komputer jednopłytkowy (ang. single-board computer) Raspberry Pi 3B. Na komputerze działa dystrybucja Raspbian systemu operacyjnego Linux. Połączenia między czujnikami stykowymi oraz czujnikami I2C są wykonane na płytce prototypowej. Wyprowadzenia GPIO Raspberry Pi są rozszerzone do płytki poprzez moduł ProtoPi Plus. Dzięki temu ułatwione było prototypowanie i testowanie systemu bez konieczności lutowania połączeń. Kamery USB są podłączone do gniazd USB komputera.

W systemie zostały zastosowane czujniki zbliżeniowe magnetyczne MC-38. Można je wykorzystać do określenia pozycji drzwi albo okien. Część z przewodami (kontakttron) należy umieścić na framudze, a magnes na drzwiach lub oknach. Obwód kontakttronu jest domyślnie rozwarty, a po zbliżeniu magnesu zostaje zamknięty. Czujnik

został podłączony do napięcia 3,3V oraz wyprowadzenia GPIO, aby wykrywać zamknięcie i przerwanie obwodu. Wówczas zbliżone czujniki oznaczają zamknięty obwód i wysoki stan na wyprowadzeniu. W przypadku oddalenia czujników i otwarcia obwodu oczekiwany jest stan niski na wyprowadzeniu. W tym celu konieczny jest rezystor ściągający napięcie odczytywane na wyprowadzeniu do masy. Raspberry Pi umożliwia programistyczną aktywację wbudowanego rezystora ściągającego. Dzięki temu można wykrywać otwarcie okien lub drzwi jako oddalenie czujników i przerwanie obwodu – zmianę stanu z wysokiego na niski.

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 0x70. 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. W założeniach projektowych jest wymaganie obsługi do 4 czujników temperatury każdego przypisanego do jednej kamery. Nie jest więc możliwe rozwiązanie polegające na podłączeniu po jednym czujniku do każdej z magistral. 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.

Do napisania programu obsługującego czujniki i kamery wybrałem język Python, ponieważ umożliwia on dzięki wielu dostępnym modułom wysokopoziomą 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, poprzez które będą wysyłane powiadomienia.

W zaprojektowanym systemie bezpieczeństwa program obsługujący urządzenia działa niezależnie od programu udostępniającego poprzez serwer WWW dane użytkownikowi. Konieczny jest zatem sposób komunikacji i wymiany danych pomiędzy nimi. Relacyjna baza danych pozwala na przechowywanie danych w tabelach, do których dostęp zagwarantowany jest poprzez odpowiedni interfejs programistyczny.

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 czujniki 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.

### 3. Program procedury obsługi kamer i czujników

Program obsługujący czujniki i kamery musi cyklicznie wykonywać pomiary, odczyty i zdjęcia, a wyniki zapisywać do bazy danych. Potrzebne jest narzędzie, które pozwoli okresowo wykonywać funkcje w ramach jednego programu. Istnieją mechanizmy wewnętrznie języka Python(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 on przeznaczony do planowania cyklicznego wykonywania zadań w określonych interwałach. Przykładowe wywołanie szeregowania przy użyciu `schedule` przedstawiono poniżej.

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

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

Program obsługujący czujniki początkowo wykonuje zapytanie do bazy danych o wszystkie czujniki, czujniki temperatury i kamery. Następnie przechodzi w pętli przez wyniki, tworząc obiekty klasy `Grupa`. Klasa `Grupa` reprezentuje kamerę oraz przypisane do niej czujniki. Zawiera ona metody, które pozwalają wykonywać zdjęcie, pomiar temperatury i wilgotność oraz odczyt stanu wyprowadzenia GPIO, do którego jest podłączony czujnik stykowy. Zarządza ona również wysłaniem wiadomości email z powiadomieniem w przypadku otwarcia czujnika.

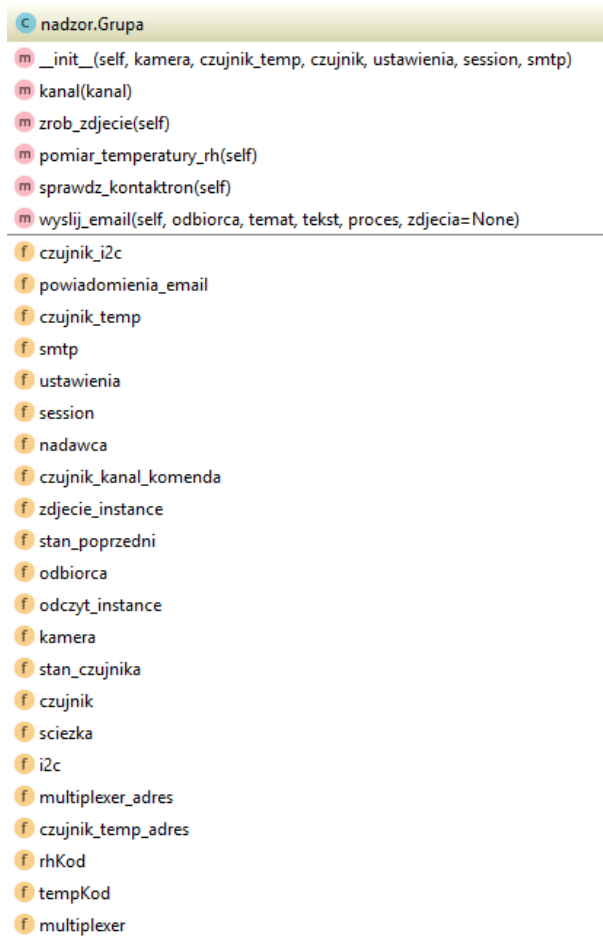
Rysunek 2 przedstawia diagram UML klasy `Grupa`. Skrótem *m* oznaczone są metody klasy, a *f* - atrybuty. Działanie poszczególnych metod jest opisane w poniższych podrozdziałach.

#### 3.1. Obsługa kamer USB

Kamery USB są obsługiwane przy pomocy aplikacji `fswebcam`. Jest to samodzielna aplikacja pozwalająca na wywoływanie zdjęć z kamer USB podłączonych do komputera z systemem typu Unix. Wywołanie tego programu z poziomu programu w języku Python wymaga użycia modułu `subprocessing`. Daje on możliwość otwierania programów w osobnych procesach. Do otwarcia `fswebcam` użyta została klasa `Popen`. Do wywoływanego programu można przekazać sekwencję argumentów. Użyтыми argumentami programu `fswebcam` są rozdzielczość, nazwa wirtualnego węzła kamery oraz ścieżka, gdzie zdjęcie ma być zapisane.

Rozdzielczość zdjęcia jest ustawiona jako 640x480 pikseli. Jest to maksymalna przetestowana możliwa rozdzielczość kamery Titanium Onyx. Kolejną komendą jest nazwa wirtualnego węzła kamery (ang. *virtual device node*). Jest to plik, który system Linux tworzy podczas pierwszego podłączenia urządzenia. Plik ten jest przechowywany w folderze `/dev`. Użycie tej komendy pozwala na wywołanie zdjęcia przez konkretną kamerę. Zdjęcie jest zapisywane w miejscu wskazanym przez atrybut klasy `Grupa` o nazwie `sciezka` (`/var/www/html/img`). Dzięki temu dostęp do zdjęć ma strona internetowa umieszczona w serwerze WWW.

Po wywołaniu procesu funkcja kontynuuje działanie, zapisując nazwę zdjęcia (będącą aktualną datą) w tabeli `zdjecia` wraz z identyfikatorem kamery, która zostało wy-



Rysunek 2: Diagram UML klasy Grupa

konane. Rodzi to pewien problem związany z tym, że nie istnieje sprawdzenie błędów pochodzących z procesu. W bazie danych zostaje zapisana nazwa zdjęcia, które mogło zostać niewykonane. Jest to pozornie problem, natomiast brak znalezienia zdjęcia o wskazanej nazwie przez stronę WWW będzie jasną informacją dla użytkownika, że wystąpił błąd. Dzięki takiemu podejściu funkcja nie blokuje dalszego wykonywania zaszeregowanych programów oczekiwaniem na zakończenie procesu. Wykonywanie funkcji w osobnym wątku doprowadziłoby do tego, że zdjęcie byłoby przypisane do późniejszych pomiarów w zależności od czasu zakończenia procesu a nie wywołania jego wykonania. Funkcja `zrob_zdjecie` zwraca obiekt klasy `Popen` reprezentujący proces z wykonywanym programem `fswebcam`.

### 3.2. Obsługa czujników podłączonych do magistrali I2C

Konfiguracja pomiaru odbywa się w następujący sposób. Do multiplexera (identyfikowanego poprzez *handler*) przesyłany jest 8-bitowy kod odpowiadający numerowi kanałowi. Aktywacja kanału oznacza ustawienie bitu o pozycji równej numerowi kanałowi jako 1. Po wysłaniu komendy i znaku STOP kanał jest aktywowany. Kolejne komendy można adresować, używając adresu czujnika temperatury. Pierwszą komendą, którą należy wysłać to komenda pomiaru wilgotności względnej. Dostępne są dwa tryby pomiaru tej wielkości: Hold Master Mode oraz No Hold Master Mode. Pierwszy z nich

oznacza, że urządzenie nadrzędne wysyła żądanie pomiaru wilgotności. Urządzenie podrzędne potwierdza otrzymanie żądanie i dokonuje pomiaru. Wymaga to zastosowania rozciągania zegara (clock stretching), które polega na utrzymywaniu przez urządzenie podrzędne linii zegarowej SCL w stanie niskim. Dzieje się to do momentu zakończenia pomiaru przez urządzenie i wystawieniu jego wyniku do rejestru. Drugi tryb różni się tym, że po potwierdzeniu otrzymania żądania wystawiany jest symbol zakończenia komunikacji. Znając maksymalny czas konwersji pomiaru, urządzenie nadrzędne może odczytać wynik z rejestru po jego zakończeniu.

Do programistycznej komunikacji poprzez I2C potrzebna była odpowiednia biblioteka. Pierwszym zastosowanym modulem był python-smbus. Korzysta on ze sterownika wbudowanego w jądro systemu Linux. W dokumentacji czujnika Si7021 opisane są dwie metody pomiaru - Hold Master Mode oraz No Hold Master Mode. Pierwszy z nich polega na wysłaniu do urządzenia kodu pomiaru (0xE5) oraz żądania odczytu po potwierdzeniu przez urządzenie odbioru. Wtedy urządzenie dokonuje pomiaru, stosując jednocześnie tzw. rozszerzanie zegara (ang. clock stretching). Polega ono na tym, że urządzenie podrzędne utrzymuje sygnał na linii zegara SCK w stanie niskim. Próba przeprowadzenia pomiaru w tym trybie przy pomocy funkcji `i2c_smbus_read_word_data()` oraz `i2c_smbus_read_i2c_block_data()` zakończyła się błędem o kodzie `io errno5`. Wynikał on z tego, że pakiet smbus obsługuje magistralę I2C zgodnie ze standardem SMBus – rozszerzeniem I2C. Posiada on bardziej ścisłe reguły dotyczące czasu trwania transakcji na magistrali.

Drugi tryb pomiaru (No Hold Master Mode) różni się tym, że po wysłaniu kodu pomiaru (0xF5) oraz żądania odczytu urządzenie nie potwierdza odbioru do momentu zakończenia konwersji pomiaru. Następnie należy odczytać dwubajtowy wynik pomiaru z rejestru czujnika. Należałoby zatem wysłać osobno bajt z kodem pomiaru, a następnie odczekać na wykonanie pomiaru, którego maksymalny czas jest opisany w specyfikacji, wysłać żądanie odczytu i odebrać wynik. Niestety wymienione funkcje odczytu pakietu python-smbus są poprzedzone wysłaniem żądania do urządzenia. Inna dostępna funkcja `i2c_smbus_read_byte()` pozwala na odczyt wyłącznie jednego bajtu naraz. Dwukrotne wysłanie żądania przy pomocy tej funkcji prowadzi do dwukrotnego odczytania jednego bajtu pomiaru.

Innym pakietem umożliwiającym komunikację poprzez magistralę I2C jest pigpio. Opiera on swoje działanie na bibliotece napisanej w języku C. Przed rozpoczęciem działania z 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 `@reboot` umieszczony przed wpisem programu `nadzor.py` zapewnia, że program zostanie uruchomiony za każdym razem, gdy włączany 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. Przy użyciu tego modułu możliwe jest przeprowadzenie pomiaru wilgotności względnej w trybie No Hold Master Mode. W pierwszej kolejności wysyłana jest przy pomocy funkcji `i2c_write_byte()` 8-bitowa komenda pomiaru. Następnie program jest usypiany na 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. Następnie odczytywany jest poprzez funkcję `i2c_read_device()` wy-

nik pomiaru składający się z dwóch bajtów. Funkcja zwraca liczbę odczytanych bajtów i bajty w formie tablicy. Bajty te można następnie wykorzystać do obliczenia wilgotności względnej na podstawie wzoru 1 podanego w karcie katalogowej.

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

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 komendy `i2c_read_i2c_block_data` bez obawy o rozciąganie zegara, ponieważ nie ma potrzeby oczekiwania na konwersję pomiaru temperatury. Dwa bajty wyniku pomiaru są zapisane do tablicy i użyte do obliczenia wartości temperatury na podstawie wzoru 2.

$$Temperatura(^{\circ}C) = \frac{(Kod_{temp}[0] * 256 + Kod_{temp}[1]) * 175,72}{65536} - 46,85 \quad (2)$$

Obsługa wyjątków w połączeniu z multiplexerem jest przeprowadzona poprzez wypisanie do konsoli komunikatu o typie urządzenia, dla którego nastąpił błąd (multiplexer albo czujnik). W takim wypadku wartościom temperatury i wilgotności są przypisywane wartości `None`. Po zakończeniu pomiaru wynik zostaje zapisany w tabeli odczyty przy pomocy funkcji `create()` z pliku `baza.py`. Zwrócony obiekt odpowiadający nowemu rekordowi w bazie danych zostaje zapisany w polu `odczyt_instance` obiektu.

### 3.3. Obsługa czujników stykowych

Korzystanie z wyprowadzeń GPIO w języku Python jest możliwe na komputerze Raspberry Pi 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 płycie drukowanej. Drugi sposób oznacza numeracją kanałów 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 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`. Użycie rezystora ściągającego jest wywoływane 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)
```

`self.czujnik.gpio` - to atrybut przechowujący numer wyprowadzenia GPIO. `GPIO.IN` - konfiguruje wyprowadzenie jako wejście. `pull_up_down=GPIO.PUD_UP` - to flaga określająca zastosowanie rezystora podciągającego.



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 obiektu stan\_czujnika wartości 1. Jeśli stan jest niski, zmiennej przypisywane jest 0. Następnie sprawdzany jest poprzedni stan wyprowadzenia przechowywany jako atrybut obiektu klasy Grupa. Jeśli poprzednio stan był wysoki (czujniki były zbliżone), to znaczy, że nastąpiło otwarcie czujników. Zgodnie z wymogami systemu następuje wywołanie zdjęcia - funkcji obsługującej kamery. Do zmiennej zostaje zapisana referencja obiektu klasy Popen, w którym został otworzony program wykonujący zdjęcie. Zostanie on użyty w funkcji wysyłanie powiadomień mailowych.

Wynik odczytu stanu zostaje zapisany do słownika przy kluczu "stan" odpowiadającemu kolumnie w tabeli danych. Pozostałym polem w słowniku jest identyfikator czujnika stykowego, dla którego został wykonany odczyt. Następnie wynik zostaje zapisany do bazy danych poprzez funkcję create(). Stan czujnika odczytany w tym wywołaniu funkcji zostaje zapisany w polu stan\_poprzedni obiektu klasy Grupa.

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 odczytane z atrybutów obiektu identyfikatory ostatnich wykonanych zdjęć oraz pomiarów temperatury. Wraz z nimi zostaje zapisana data wpisu odczytana przy pomocy funkcji datetime.now() zapisana w formacie dzień-miesiąc-rok godzina-minuta-sekunda.

### 3.4. Wysyłanie powiadomień e-mail

Funkcjonalnością systemu jest również wysyłanie powiadomienia mailowego po otwarciu czujnika. Jest ono realizowane poprzez serwer SMTP Google. Ta decyzja projektowa wynika z tego, że konfiguracja własnego serwera SMTP w ramach Raspberry Pi wymagałaby posiadania stałego adresu dostarczanego przez dostawcę usług internetowych, do którego będzie przypisana domena. Prostszy rozwiązaniem jest utworzenie konta email w zewnętrznej usłudze (np. Gmail).

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 Gmaila jest stosowanie szyfrowanego połączenia, które jest stworzone przez funkcję SMTP\_SSL poprzez standardowy port 465. Następnie następuje uwierzytelnienie w serwerze poprzez przesłanie adresu email konta, z którego mają być wysyłane wiadomości oraz hasła do niego. Następnie zostaje zwrócona referencja obiektu reprezentującego połączenie z serwerem SMTP.

Przy tworzeniu obiektu zostają zapisane do atrybutów odczytane z bazy danych ustawienia dotyczące powiadomień – znacznik włączenia powiadomień oraz adres email ich odbiorcy. Jeśli powiadomienia są włączone, czyli znacznik jest ustawiony jako "on" oraz adres email nie jest pusty, będzie wysłana wiadomość email.

Przygotowany jest tekst wiadomości informujący, że został otwarty czujnik o nazwie przypisanej do niego. W temacie czujnika zostaje wpisana informacja o otwarciu czujnika i dacie otwarcia odczytanej w momencie wywołania funkcji. Następnie zostaje utworzony nowy wątek, w którym jest wywoływana funkcja `wyslij_email()`. Argumentami tej funkcji są adres email odbiorcy, temat wiadomości, jej tekst oraz nazwa zdjęcia, która będzie dołączona jako załącznik. Argumentem jest też referencja obiektu `Popen`, który został stworzony do wywołania w podprocesie programu wykonującego zdjęcie. Jest on użyty do tego, aby wstrzymać działanie funkcji `wyslij_email()` do czasu zakończenia wykonywania zdjęcia. Jest to osiągnięte przy użyciu funkcji `wait()` tego obiektu.

Do stworzenia wiadomości email jest wykorzystany obiekt klasy `MIMEMultipart`. W polach `Subject`, `To` i `From` zostają wpisane odpowiednio temat, odbiorca i nadawca wiadomości email. Następnie funkcja otwiera wszystkie pliki ze zdjęciami, których nazwy zostały przekazane w liście jako argument. Dla każdego z nich 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 wyświetlony jako załącznik do pobrania w kliencie poczty elektronicznej.

## 4. Baza danych

### 4.1. Założenia wstępne

Relacyjna baza danych składa się z relacji (tabel), które są połączone związkami. W takim modelu organizacji bazy danych łatwo przedstawić rzeczywiste obiekty, których dane ma przedstawiać. 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 poprzez obecność w jednej z tabel uczestniczących w związku klucza obcego, który pozwala jednoznacznie zidentyfikować wiersz z drugiej tabeli.

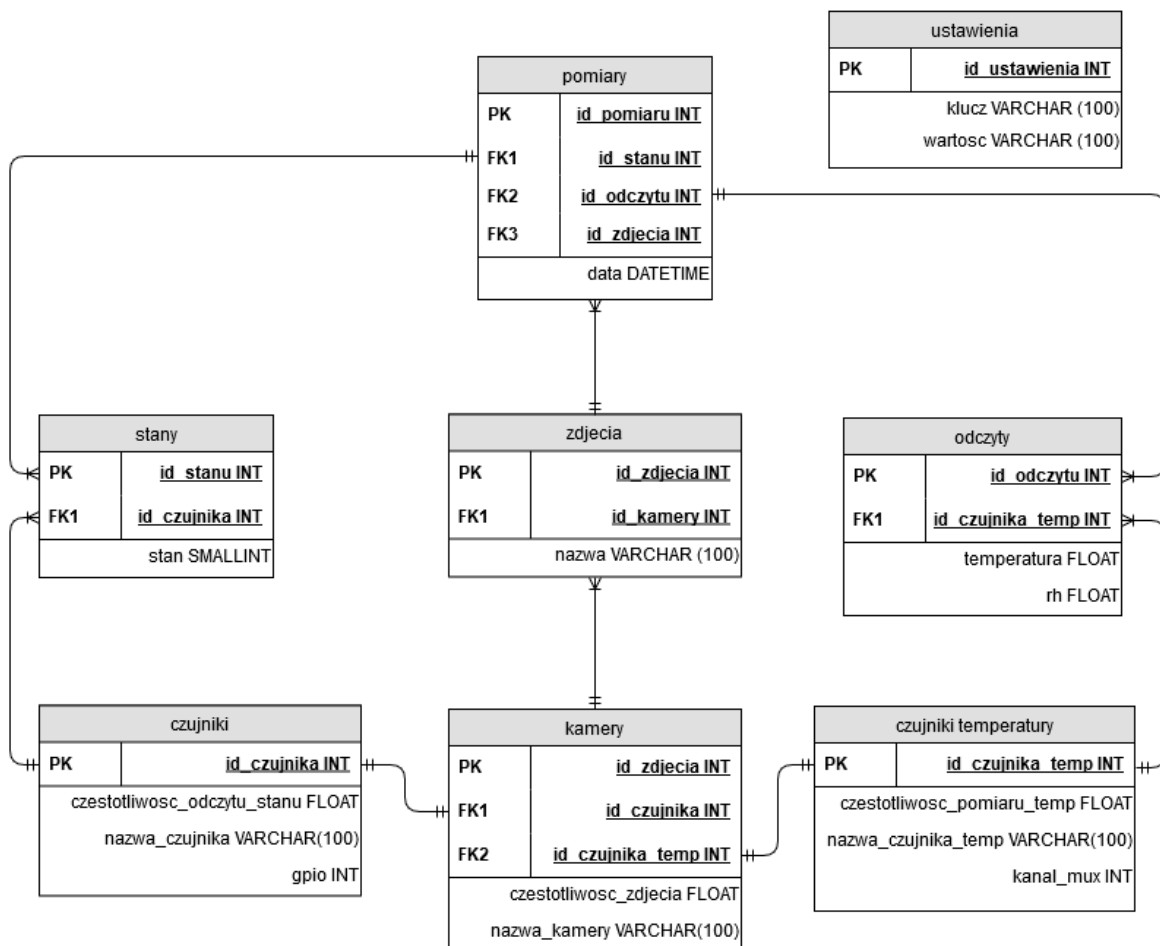
Programistyczny dostęp do baz danych jest możliwy poprzez interfejs do opartego na transakcjach silnik bazy danych. Transakcje to zestaw operacji, które powinny być wykonane w całości lub wcale. Dzięki nim możliwe jest zachowanie integralności danych w sytuacji, gdy kilka klientów (programów) zapisuje dane do bazy danych. W projekcie użyty został wolnodostępny system zarządzania bazą danych MySQL.

### 4.2. Projekt bazy danych

Projekt bazy danych należy rozpocząć od zdefiniowania, jakie obiekty mają być przechowywane w bazie. W projektowanym systemie niezbędne 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 multiplexera, do którego jest podłączona, a czujnik stykowy o numerze wyprowadzenia GPIO, do którego jest podłączony. 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 składającej się z czujnika, czujnika stykowego i kamery informacje o ostatnich wykonanych pomiarach. Wpisy do tej tabeli muszą być wykonywane tak często, jak wpisy do najczęściej wykonywany pomiar. Pozostałe wielkości będą ostatnio zmierzonymi. Pozwala to na jasne dopasowanie zdjęcia oraz zmierzonej temperatury i wilgotności do odczytu stanu.



Rysunek 3: Schemat bazy danych na poziomie logicznym

Rysunek 3 przedstawia schemat bazy danych na poziomie logicznym. Poziom logiczny 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: nazwie, częstotliwości wywoływania pomiaru lub zdjęcia i sprzętowej lokalizacji urządzenia (numer wyprowadzenia, kanał multipleksera). Tabele stany, zdjęcia i odczyty zawierają informacje o realizacjach pomiarów wykonane przez te urządzenia. Tabela pomiary przedstawia status systemu w czasie, zawierając klucze aktualnych na ten moment zdjęć, odczytów i stanów.

Poza wspomnianymi wyżej relacjami istnieje również relacja ustawienia, która nie jest w związku z żadną inną. Jest to relacja, która pozwala przechowywać różne ustawienia systemu, które nie dotyczą bezpośrednio urządzeń. Posiada ona dwie kolumny klucz i wartość. Klucz jest opisem ustawienia, a wartość przechowuje informację o nim np. adres email, na który mają być wysyłane powiadomienia. Wartość ta może być zmieniona poprzez interfejs na stronie WWW, ale jest wykorzystywana przez program obsługujący kamery i czujniki. Przechowywanie tych danych w bazie zapewnia ich niezależność od wykonywania poszczególnego programu.

Ponadto kamera, czujnik temperatury i czujnik stykowy, który ma wywoływać zdjęcie kamery zostały zgrupowane w ten sposób, że kamera przechowuje klucze przypisanych do niej czujników. Odpowiada to związkowi jeden do wielu między kamerą a czujnikiem (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 mapowanie obiektowo-relacyjnym (ang. Object-Relational Mapping ORM). Ułatwia ono wprowadzanie kolejnych zmian do struktury bazy danych, relacji i związków między nimi.

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 poprzez wywołanie konstruktora klasy Column. Przyjmuje on argumenty dotyczące typu przechowywanego 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`, który przechowuje informację, która kolumna w której tabeli służy jako klucz obcy. Następnie należy zdefiniować związek, wywołując funkcję `relationship()`. Przyjmuje ona jako argument nazwę klasy. Oznacza to, że klasa ta musi być stworzona wcześniej w kodzie.

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 istnieje już, to zwraca go. Jeśli nie istnieje, to tworzy nowy rekord. Dzięki temu do programu można dodawać tworzenie kolejnych rekordów bez obaw o powstanie duplikatów już stworzonych.

## **5. Strona i serwer WWW**

### **5.1. Serwer 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**

## **9. Załączniki**