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

2.2. Przegląd dostępnych platform

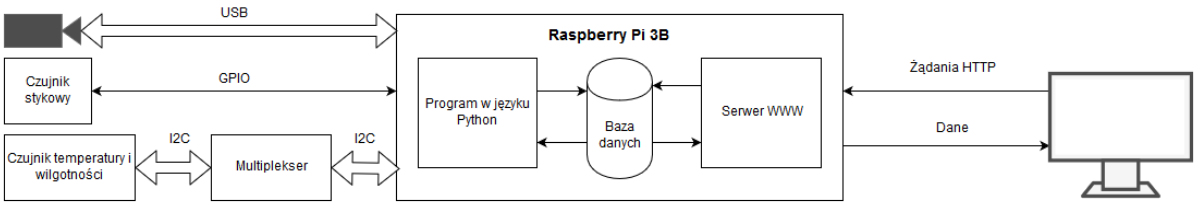
2.3. Czujniki warunków środowiskowych

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 (kontaktron) należy umieścić na framudze, a magnes na drzwiach lub oknach. Obwód kontaktronu jest domyślnie rozwarty, a po zbliżeniu magnesu zostaje zamknięty. Czujnik zo-

stał 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 aktywacją 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 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, poprzez które będą wysyłanie 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 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.

3. Program procedury obsługi kamer i czujników

Program nadzor.py 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 wewnątrz 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 ona przeznaczona do planowania cyklicznego wykonywania zadań. Przykładowe wywołanie szeregowania przy użyciu `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 funkcji jako argument `do()`.

Program jest napisany przy użyciu metodyki obiektowej. Takie podejście pozwala powiązać czujnik z kamerą, która będzie wykonywała zdjęcia w przypadku jego otwarcia. Klasa Grupa reprezentuje te dwa urządzenia oraz dodatkowo przypisany do nich czujnik temperatury. Zawiera ona metody, które wykonujące zdjęcie, pomiar temperatury i wilgotności 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.

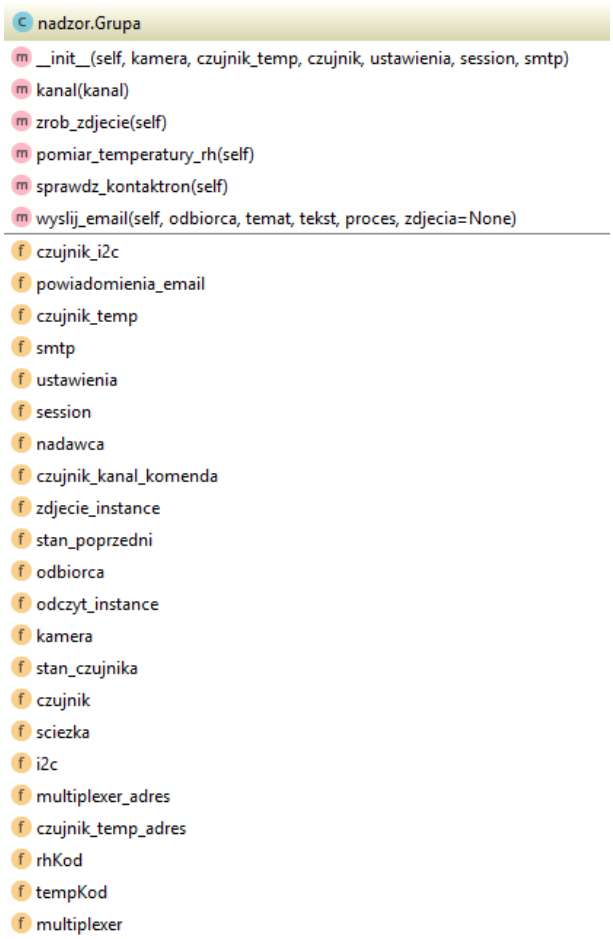
3.1. Obsługa kamer USB

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

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



Rysunek 2: Diagram UML klasy Grupa

- "`-r`" – rozdzielczość ,
- "`-d`" – nazwa wirtualnego węzła kamery,
- "`-q`" – tryb cichy,
- ścieżka, gdzie zdjęcie ma być zapisane.

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 przetestowana możliwa rozdzielczość 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`. Na tej podstawie podejmowana jest decyzja o nazwie zdjęcia, która będzie zapisana do bazy danych.

3.2. Obsługa czujników podłączonych do magistrali I2C

Multiplekser TCA9548A umożliwia podłączenie 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 jego numerowi. Kod tworzony jest przez ustawienie bitu o pozycji równej numerowi kanału jako 1. Definicja bajtu sterującego została pokazana w tabeli 1 Pozostałe pola powinny być ustawione jako 0. Operację tą wykonuje pomocnicza funkcja `kanal`. Zwraca ona 8 bitową komendę odpowiadającą numerowi kanału. Po wysłaniu komendy i znaku STOP kanał jest aktywowany. Kolejne komendy można 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	Kanał 0 nieaktywny
							1	Kanał 0 aktywny
X	X	X	X	X	X	0	X	Kanał 1 nieaktywny
						1		Kanał 1 aktywny
X	X	X	X	X	0	X	X	Kanał 2 nieaktywny
					1			Kanał 2 aktywny
X	X	X	X	0	X	X	X	Kanał 3 nieaktywny
				1				Kanał 3 aktywny
X	X	X	0	X	X	X	X	Kanał 4 nieaktywny
			1					Kanał 4 aktywny
X	X	0	X	X	X	X	X	Kanał 5 nieaktywny
		1						Kanał 5 aktywny
X	0	X	X	X	X	X	X	Kanał 6 nieaktywny
	1							Kanał 6 aktywny
0	X	X	X	X	X	X	X	Kanał 7 nieaktywny
1								Kanał 7 aktywny

Tabela 1: Definicja bajtu sterującego [1]

Pierwszą komendą, którą należy wysłać do czujnika to komenda pomiaru wilgotności względnej. W dokumentacji czujnika Si7021 opisane są dwie metody pomiaru - Hold Master Mode oraz No Hold Master Mode. Pierwsza z nich oznacza, że urządzenie nadrzędne wysyła żądanie pomiaru wilgotności. Urządzenie podrzędne potwierdza otrzymanie żądanie i dokonuje pomiaru. Wymaga to zastosowania rozciągania zegara (ang. *clock stretching*), które polega na utrzymywania przez urządzenie podrzędne linii zegarowej SCK w stanie niskim. Dzieje się to do momentu zakończenia pomiaru

przez urządzenie i wystawieniu jego wyniku do rejestru. 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.

Do programistycznej komunikacji poprzez I2C potrzebna była odpowiednia bibliotek. Pierwszym zastosowanym modułem był python-smbus. Korzysta on ze sterownika wbudowanego w jądro systemu Linux. Próba przeprowadzenia pomiaru w trybie Hold Master Mode 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. 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.

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()` wynik 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. Kod_{RH} oznacza tablicę bajtów, do której został zapisany wynik pomiaru.

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

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 funkcji `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.

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

Obsługa wyjątków w połączeniu z multiplekserem 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. Po zakończeniu pomiaru wynik zostaje zapisany w bazie danych.

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 płytce 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)
```

Pierwszym argumentem tej funkcji jest numer wyprowadzenia GPIO, drugi określa, czy wyprowadzenia ma być skonfigurowane jako wejście (`GPIO.IN`) czy wyjście (`GPIO.OUT`). Trzeci argument to znacznik określający, czy ma zostać użyty rezystor ściągający czy podciągający.

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 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 następuję wywołanie zdjęcia - funkcji obsługującej kamery. Dodatkowo program przechodzi do wysłania powiadomienia email.

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. Prostszym rozwiązaniem jest utworzenie konta email w zewnętrznej usłudze (np. Gmail) i korzystanie z jej serwera SMTP.

Połączenie z serwerem SMTP Google jest nawiązane 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`. Komunikacja odbywa się na standardowym dla tego połączenia porcie 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, wysłana będzie 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 otworzony 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.

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 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.

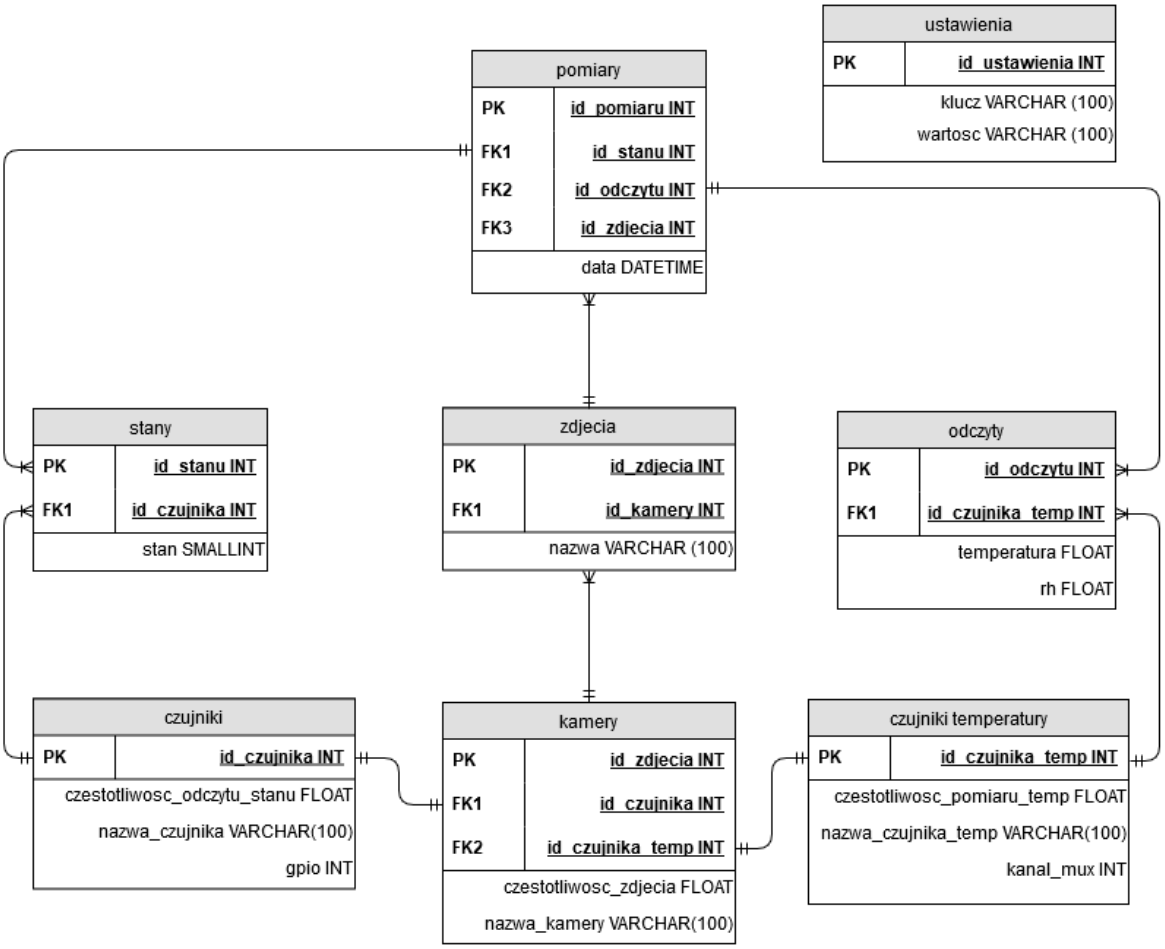
4.2. Projekt bazy danych

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 multipleksera, 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 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 zdję-



Rysunek 3: Schemat bazy danych na poziomie logicznym

cia 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 związkom 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 kolejność 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.

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.

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

Literatura

[1] Karta katalogowa multipleksera TCA9548A: `http://www.ti.com/lit/ds/symlink/tca9548a.pdf`

9. Załączniki