



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

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

System powinien być zrealizowany w postaci komputera jedнопłytkowego z systemem Linux, na którym będzie działał serwer WWW. Odpowiada on za dostęp do strony WWW i zebranych danych. Kontrolę otwarcia drzwi i okien można zrealizować poprzez czujniki stykowe, które należy podłączyć do wyprowadzeń GPIO komputera. Jako czujniki stykowe będzie można również wykorzystać specjalizowane czujniki dla systemów alarmowych np. czujniki ruchu, zalania wodą, detektory dymu i gazu. Dostępne na rynku zintegrowane czujniki warunków środowiskowych (np. temperatury i wilgotności) wymagają obsługi magistrali (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 bardzo dużej pojemności pamięci. Wystarczające będzie okresowe wykonywanie zdjęć oraz dodatkowo wyzwalanie zdjęcia w przypadku otwarcia drzwi lub okna – zadziałania czujnika stykowego przypisanego do odpowiedniej kamery. Większość funkcji systemu możliwa jest do zrealizowania na prostszej platformie (np. Arduino), jednak jak pokazały ostatnie badania,[?] jest ona podatna na włamania i ataki sieciowe. Zastosowanie systemu Linux umożliwi osiągnięcie większego bezpieczeństwa i skalowalności systemu.

Interfejs użytkownika musi prezentować aktualny stan systemu i umożliwiać dostęp do ostatnio wykonanych zdjęć, odczytów z czujników środowiskowych oraz stanu czujników stykowych (otwarty/zamknięty). Ponadto użytkownik powinien mieć dostęp do przeszłych zdjęć i pomiarów zebranych w archiwum. Historia pomiarów z 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 w systemie z kamer i czujników powinny być więc zapisane w systemie tak, by umożliwić do nich dostęp dla serwera WWW, na którym będzie działać strona WWW. Pomiarzy 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:

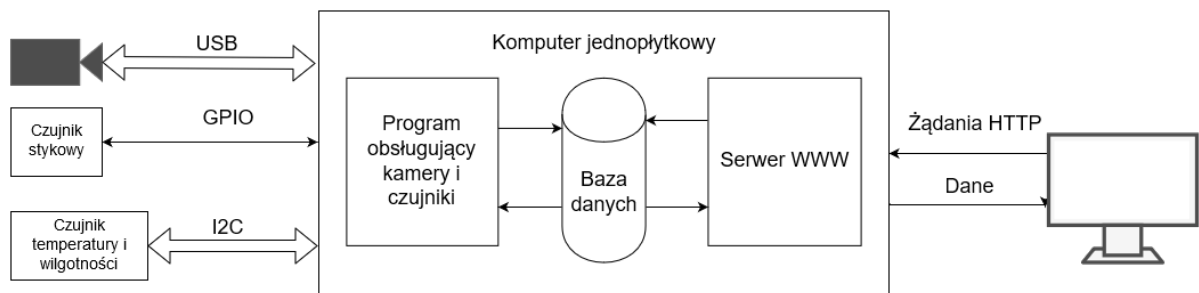
- wykorzystanie komputera jedнопłytkowego,
- użycie systemu Linux,
- użycie serwera WWW,

- 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 sieć 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 przedstawiono na rysunku 1. Zostanie ona omówiona w dalszych podrozdziałach.



Rys. 1: Architektura systemu bezpieczeństwa.

## 2.2. Przegląd dostępnych platform

Jednym z najbardziej popularnych komputerów jednopłytkowych jest platforma Raspberry Pi rozwijana przez Raspberry Pi Foundation jako narzędzie edukacyjne do nauczania programowania i elektroniki. Wokół tego projektu zebrała się duża grupa amatorów i entuzjastów dzielących się swoimi projektami. Jednym z konkurentów platformy Raspberry Pi jest komputer jednopłytkowy BeagleBone rozwijany przez firmę 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 w jednej obudowie 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 1.

	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 [?] [?] [?]

Przewagą komputera Raspberry Pi 3B jest szybszy procesor o większej liczbie rdzeni, co pozwala na równoległe wykonywanie większej liczby wątków, oraz większa pojemność pamięci RAM. Ponadto ma on możliwość podłączenia zarówno do sieci przewodowej (Ethernet) jak i bezprzewodowej (WLAN). Może być to przewagą w przypadku, gdy system miałby być zainstalowany z daleka od routera. 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 o projektowanego systemu wybrano platformę Raspberry Pi 3B.

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 wymagania – 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 bazującego na stronie WWW interfejsu użytkownika.

### 2.3. Czujniki warunków środowiskowych

Wymogiem technicznym wobec systemu jest również obsługa czujników warunków środowiskowych. Najważniejszymi parametrami, które należy zmierzyć w warunkach domowych są temperatura i wilgotność względna powietrza. Wpływają one na jakość powietrza w pomieszczeniach, odczuwalny poziom komfortu oraz na poziom zachorowań wśród domowników. 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 2.

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

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

Na podstawie danych przedstawionych w tabeli 2 wybrano czujnik Si7021 jako użyty w projekcie czujnik środowiskowy. Zapewnia on pomiar temperatury i wilgotności względnej w zakresie wartości panujących w warunkach pokojowych oraz ma najniższą cenę. Przewagą czujnika SHT31 była możliwość zmiany adresu I2C przez podłączenie wyprowadzenia adresowego do stanu wysokiego lub niskiego. Byłoby to jednak rozwiązanie możliwe do zastosowania przy użyciu tylko dwóch czujników w systemie.

## 2.4. Czujniki stykowe

Magnetyczne czujniki zbliżeniowe mogą być użyte do określenia pozycji drzwi lub okien i tym samym realizować nad nimi nadzór. 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 otwarty, a po zbliżeniu magnesu następuje jego zamknięcie i przepływ prądu. Do realizacji pracy wybrano ten czujnik ze względu na łatwość montażu – czujnik posiada taśmę samoprzylepną i otwory na śruby montażowe. Zamiast kontaktronów możliwe będzie również dołączenie innych czujników dwustanowych stosowanych w systemach alarmowych np. czujniki ruchu, zalania wodą, detektory dymu i gazu.

## 2.5. Kamery cyfrowe

Kamery USB zastosowane w systemie mają pozwolić na realizację domowego monitoringu wizyjnego 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. Koszt typowej kamery IP wynosi ok. 400 zł.[?]

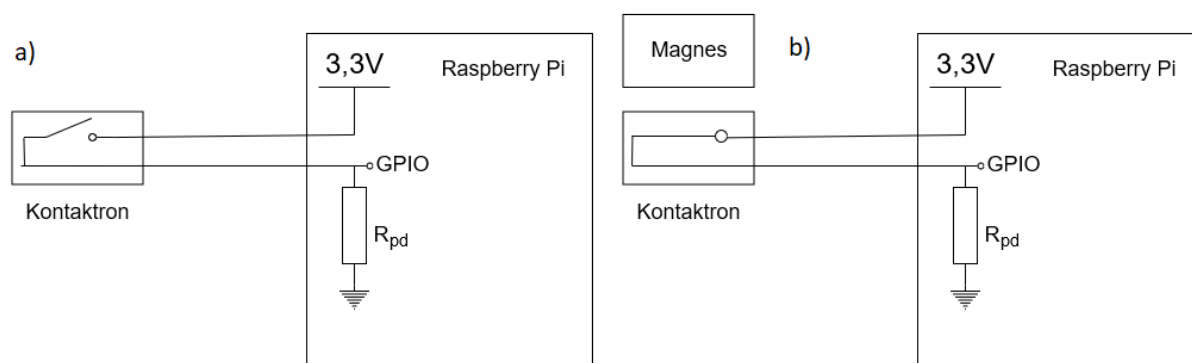
W systemie zastosowano dwie kamery z różnego przedziału cenowego. Kamera Titanium Onyx posiada matrycę CMOS, która umożliwia wykonywanie zdjęć o rozdzielczości do 5 megapikseli przy zastosowaniu interpolacji. Co ważne, kamera ta jest kompatybilna z UVC (*USB Video Class*) – sterownikiem wbudowanym w system Linux. Kamera posiada również przełącznik włączający 3 diody LED, które mogą być przydatne w przypadku wykonywania zdjęć

nocnych. Koszt tej kamery to ok. 50 zł. Drugą zastosowaną 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 stałogniskowa w przeciwieństwie do kamery Titanium Onyx. Koszt tej kamery to ok. 110 zł.

## 2.6. Projekt części sprzętowej systemu

Połączenie między komputerem Raspberry Pi 3B a dołączonymi czujnikami zrealizowano na prototypowej płytce stykowej. Umożliwia ona tworzenie prostych układów bez konieczności lutowania połączeń. Połączenia między płytką prototypową a wyprowadzeniami GPIO Raspberry Pi są wykonane przez ekspander wyprowadzeń ProtoPi Plus.

W systemie zostały zastosowane magnetyczne czujniki zbliżeniowe MC-38. Kontaktron został podłączony do wyprowadzenia GPIO poprzez z użyciem wbudowanego rezystora ściągającego do masy (ang. *pull-down*) oraz napięcia zasilania (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 2 został przedstawiony schemat podłączenia czujnika stykowego do platformy Raspberry Pi (z pominięciem ekspandera ProtoPi Plus).



Rys. 2: Schemat podłączenia czujnika MC-38 w przypadku oddalonego (a) i zbliżonego (b) magnesu.

Wymogiem wobec 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 jest stały i wynosi 0x40. Nie mają one możliwości programowej 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 kolizji i sytuacji, w której nie można jednoznacznie określić, z którego czujnika został odczytany wynik pomiaru. Raspberry Pi 3 B posiada możliwość 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 wybranym w danej chwili. Przykładem takiego układu jest TCA 9548A firmy Adafruit. Pozwala on na podłączenie do 8 urządzeń korzystających ze wspólnej magistrali I2C.

Multiplekser TCA 9548A jest podłączony do wyprowadzeń GPIO2 i GPIO 3 w Raspberry Pi, które mogą być aktywowane jako odpowiednio: linia SDA i SCL magistrali I2C. Ponadto wyprowadzenia GPIO2 i GPIO3 są podłączone przez wbudowane rezystory podciągające

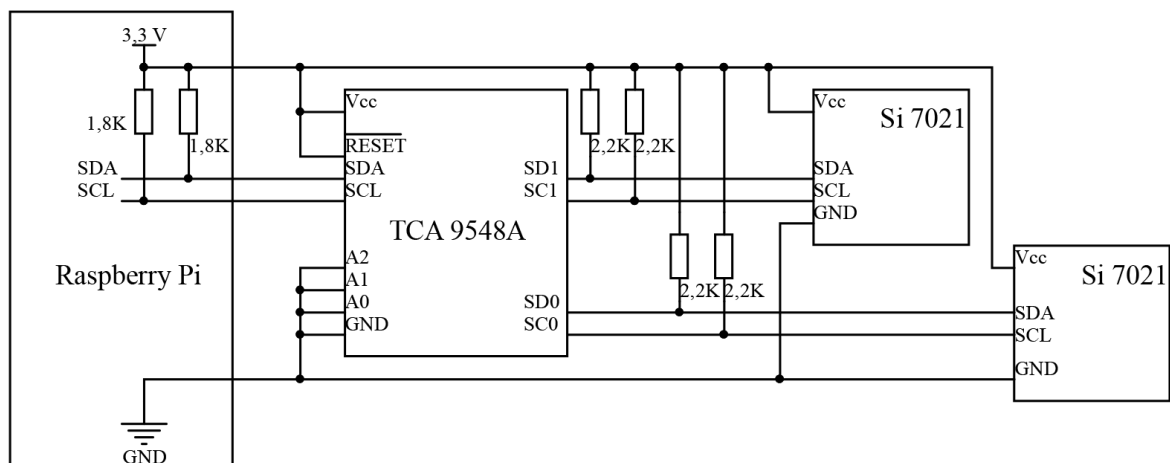


do napięcia zasilania 3,3 V[?], co jest zgodne z zaleceniami podanymi w karcie katalogowej multiplexera.[?] Multiplexer jest zasilany z wyprowadzenia 3,3 V platformy Raspberry Pi i podłączony do jej masy. Wyprowadzenia adresowe A2, A1 i A0 są podłączone do masy, co oznacza, że adres I2C multiplexera to 0x70. Nieużywane wyprowadzenie  $\overline{\text{RESET}}$  jest z kolei podłączone do zasilania poprzez rezystor podciągający, 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 10 k $\Omega$  podciągające linie SDA i SCL do napięcia zasilania. Producent multiplexera TCA 9548A podaje z kolei w karcie katalogowej wzór 1 pozwalający na wyznaczenie minimalnej wartości rezystorów podciągających.[?] Napięcie  $V_{\text{DPU}}$  jest napięciem linii sygnałowych, które jest równe napięciu zasilania  $V_{\text{CC}}$ . Napięcie  $V_{\text{OL(max)}}$  to maksymalne napięcie wyjściowe, a prąd  $I_{\text{OL}}$  to prąd wyjściowy czujnika. Obie wartości są podane w karcie katalogowej czujnika Si7021.[?]

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

Im większa rezystancja rezystora podciągającego tym mniejszy pobór prądu. Rezystor podciągający tworzy jednak wraz z pojemnością magistrali  $C_b$  filtr dolnoprzepustowy RC. W przypadku długich przewodów połączeniowych 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 przewidzenia, więc rezystancja powinna być zbliżona do minimalnej wymaganej. W ramach projektowanego systemu użyte zostały rezystory podciągające 2,2 k $\Omega$ . Rysunek 3 przedstawia schemat połączeń między komputerem Raspberry Pi a multiplexserem TCA 9548A i dwoma czujnikami Si7021. Przedstawiono poglądowo tylko dwa czujniki, ale system może być rozbudowany o dodatkowe czujniki. Przetestowano działanie systemu przy maksymalnie 4 czujnikach, jednak multiplexer posiada w sumie 8 kanałów, do których można dołączyć urządzenia komunikujące się przy użyciu magistrali I2C.



Rys. 3: Schemat podłączenia multiplexera TCA9548A i czujników Si7021.

Podłączenie większej liczby kamer USB do portów Raspberry Pi może spowodować chwilowe przekroczenie dopuszczalnego poboru mocy z gniazd USB. Rozwiązaniem tego problemu jest użycie aktywnego rozgałęziacza (ang. *hub*) USB z zewnętrznym zasilaniem. Zwiększa on

jednocześnie zasięg podłączonych przewodowo kamer. Należy uważać jednak, by rozgałęziacz nie zasilął wstecznie (ang. *backfeed*) komputera Raspberry Pi, ponieważ grozi to uszkodzeniem układu. Z tego powodu w projekcie użyto dedykowanego rozgałęziacza firmy The Pi Hut.

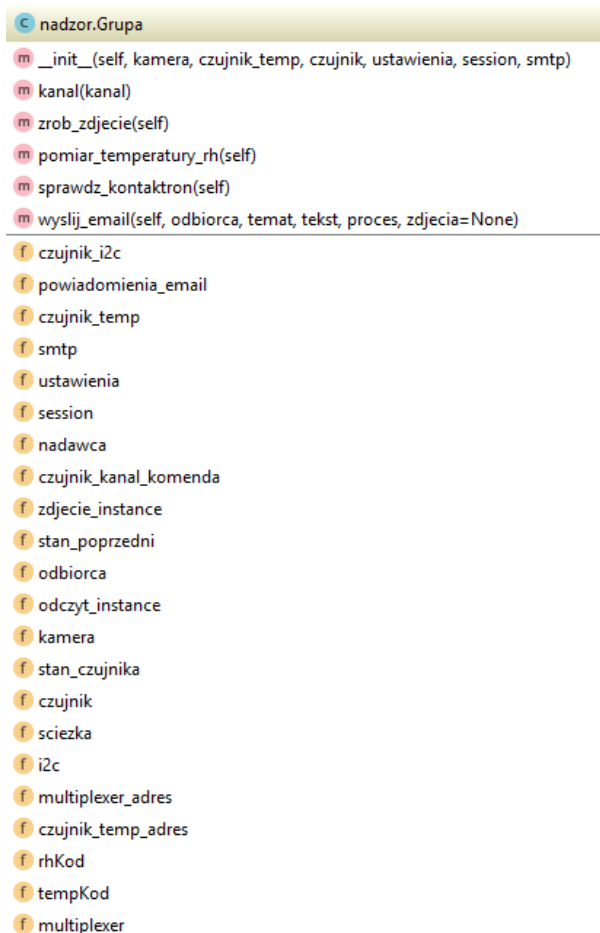
### 3. Obsługa kamer i czujników

Do napisania programu obsługującego czujniki i kamery wybrałem język Python, ponieważ umożliwia on poprzez wiele dostępnych modułów 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, przez które będą wysyłane 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.czesotliwosc_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()`.



Rys. 4: Diagram UML klasy Grupa

Rysunek 4 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 wypro- wadzenia 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 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)
```

Użyтыми 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 `ścieżka`. 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 multipleksa 8 bitowego kodu odpowiadającego numerowi kanału. Kod ten tworzony jest przez ustawienie bitu o pozycji równej

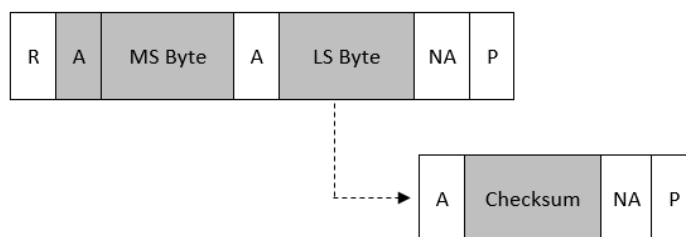
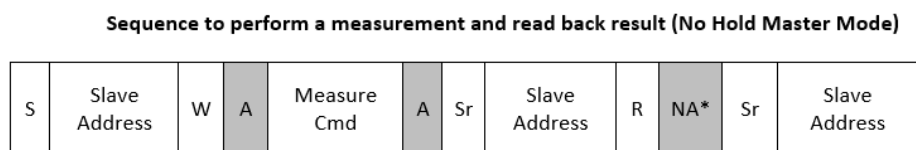
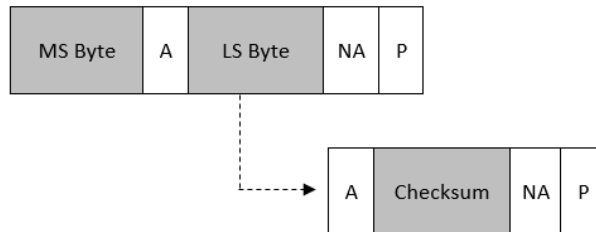
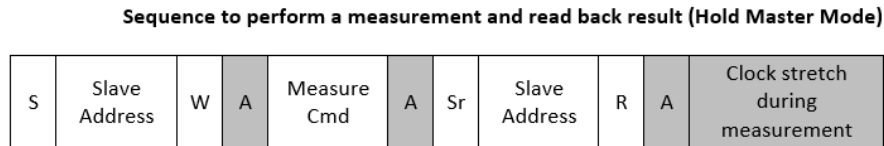
numerowi kanału jako 1. Pozostałe pola powinny być ustawione jako 0. Operację tą wykonuje pomocnicza funkcja `kanal`. Zwraca ona 8-bitową komendę odpowiadającą numerowi kanału. Definicja kodu została pokazana w tabeli 3. Po wysłaniu przez I2C komendy i znaku STOP kanał jest aktywowany. Dalsze komendy 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 5. 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.



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

Do programowej komunikacji przez I2C potrzebna była odpowiednia biblioteka. Pierwszym zastosowanym modulem 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 powtórnego 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 wysłania 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 2 podanego w karcie katalogowej.[?]  $Kod_{RH}$  oznacza tablicę bajtów, do której został zapisany wynik pomiaru.

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

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 3.[?]

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

Obsługa wyjątków w komunikacji przy użyciu magistrali I2C 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`. 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 były zamknięte), to znaczy, że nastąpiło otwarcie czujników. Zgodnie z wymogami projektowanego systemu następuje 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. Prostszy 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 `smtpplib` na początku działania programu `nadzor.py`:

```
def init_smtp(sender, password):
    smtp_server = smtpplib.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 z 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.

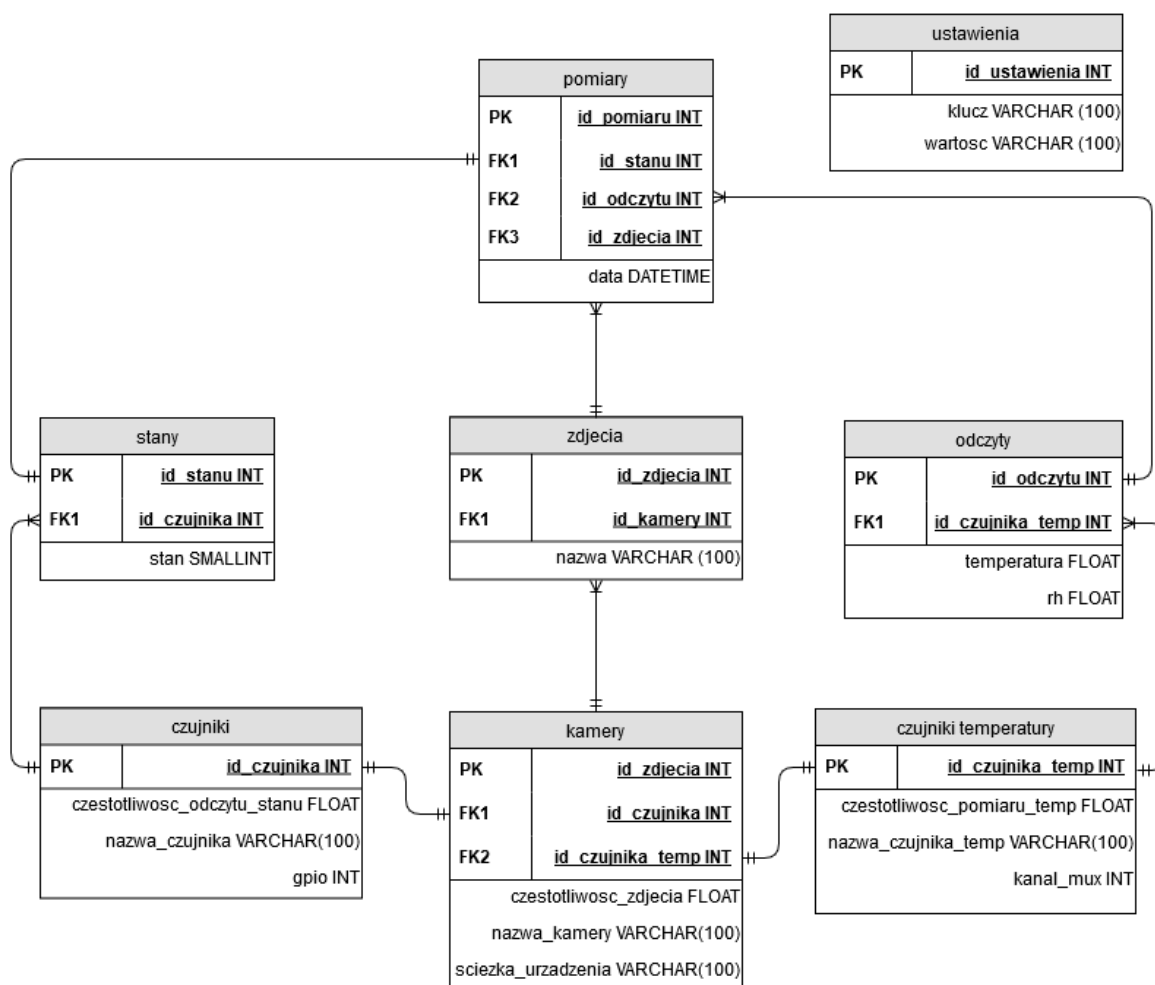
Projekt bazy danych należy rozpocząć od zdefiniowania, jakie obiekty mają być przechowywane w bazie. W projektowanym systemie potrzebne są informacje o czujnikach i kamerach podłączonych do Raspberry Pi oraz pomiarach wykonanych przez nie. Wszystkie urządzenia mają nazwę identyfikującą ich położenie 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 6. 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 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



Rys. 6: Schemat bazy danych na poziomie logicznym

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 `temperatura` i `rh`. Wpisy w tabeli `zdjęcia` zawierają 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 urzą-

dzeń. 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 związkowi 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`.<sup>[?]</sup>

```
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`, `zdzjecia` 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 wobec systemu jest umożliwienie użytkownikowi dostępu do zebranych danych przez stronę WWW. Konieczny jest zatem serwer HTTP, na którym będzie umieszczona strona. Jednym z najpopularniejszych środowisk do prowadzenia serwerów WWW jest Apache. Jest on dostępny w wersji dla systemu Raspbian, który działa na platformie Raspberry Pi w zaprojektowanym systemie. Umieszczona na serwerze strona będzie komunikowała się z serwerem przez żądania HTTP. Szkielet strony w postaci pliku HTML będzie wypełniony danymi zebranymi przez kamery i czujniki po zrealizowaniu żądań wysyłanych asynchronicznie (przy użyciu techniki AJAX) do serwera. Za obsługę żądań po stronie serwerowej będą odpowiedzialne mikrousługi w postaci skryptów w języku PHP. Skrypty te łączą się z bazą danych, w której są przechowywane dane zebrane z kamer i czujników. Następnie zwracają wyniki zapytań do bazy w odpowiedzi na żądanie klienta. Język PHP został wybrany do obsługi żądań po stronie serwerowej, ponieważ jest dobrze zintegrowany z serwerem Apache i jest jedną z najbardziej popularnych technologii na serwerach WWW.

Dostęp do prezentowanych danych powinien być zabezpieczony. Autoryzacja użytkownika na stronie została wykonana w formie uwierzytelnienia prostego (ang. *basic authentication*). Wymaga ono jedynie stworzenia na serwerze pliku z hasłem przy użyciu narzędzia `htpasswd` będącego częścią pakietu Apache. Plik z hasłem jest przechowywany poza folderem, w którym są przechowywane pliki strony internetowej, dzięki czemu jest poza zasięgiem ewentualnego ataku. Hasło przechowywane jest w formie 128-bitowego skrótu (ang. *hash*) stworzonego przy pomocy funkcji skrótu MD5.[?] Dzięki temu hasło nie jest przechowywane w formie prostego tekstu, który mogłoby od razu odczytany w wypadku zdobycia dostępu do pliku przez niepożądaną osobę. Z przesłanego przez użytkownika hasła również tworzony jest skrót, który jest porównywany ze skrótem hasła przechowywanym w pliku. Jeśli skróty są równe, oznacza to, że przesłane hasło jest poprawne. Komunikacja między serwerem a klientem odbywa się w formie nieszyfrowanej HTTP, przez co klient wysyła swoje hasło i login niezaszyfrowane w nagłówku żądania HTTP.[?] Tym samym osoba podsłuchująca komunikację mogłaby poznać dane logowania. Jest to słabość zabezpieczeń systemu w obecnej postaci, którą należałoby w przyszłości wyeliminować, używając szyfrowania SSL w komunikacji przy użyciu protokołu HTTP. Innym usprawnieniem byłoby użycie funkcji skrótu z rodziny SHA-2 lub SHA-3 do generowania skrótów haseł przechowywanych na serwerze. Są one w tej chwili uważane za niemożliwe do złamania.

### 5.2. Żądania HTTP i ich obsługa – technika AJAX

Po pozytywnym uwierzytelnieniu użytkownika w systemie udostępniony jest plik `index.html`. Zawiera on szkielet strony oraz dołączone skrypty JavaScript w tym skrypt `main.js`, który zawiera funkcje realizujące żądania HTTP. Służą one do pobierania danych z systemu oraz wprowadzania ustawień dokonanych przez użytkownika. Wykorzystania została do tego technika AJAX (ang. *Asynchronous JavaScript and XML*). Jest to technika tworzenia aplikacji internetowych, w której żądania są wykonywane asynchronicznie. Technika ta jest oparta o architekturę serwer-klient, w której rolę klienta pełni przeglądarka. Po wysłaniu żądania przeglądarka kontynuuje działanie, a w tym czasie serwer realizuje żądanie. Po zakończeniu obsługi żądania



serwer wysyła odpowiedź do przeglądarki, która odbiera dane i przetwarza je. Dzięki temu treść strony może być odświeżana bez konieczności jej przeładowywania. Jest to przydatne do odświeżania danych w tle tak, by użytkownik obserwował aktualny stan systemu.

Do przygotowania żądań HTTP wykorzystywana jest funkcja `$.ajax()` z biblioteki jQuery. Przygotowuje ona obiekt `XMLHttpRequest`, reprezentujący żądanie HTTP. Funkcja pozwala również określić adres URL, do którego ma być skierowane żądanie, typ żądania (GET, POST, DELETE itp.), dołączone dane oraz pozostałe pola odpowiadające nagłówkom protokołu HTTP. Żądania te będą kierowane do dedykowanych programów w języku PHP działających na serwerze WWW. Wykonują one zapytania do bazy danych i zwracają ich wyniki lub dokonują wpisów do niej. Działanie poszczególnych skryptów zostanie omówione w dalszych podrozdziałach.

### 5.2.1. Strona główna i archiwum

Podstawowym żądaniem wywoływanym przez funkcję `main.js` jest żądanie pobrania aktualnego stanu systemu. Jest ono wywoływane w funkcji `main()` przez funkcję `getData()`:

```
function getData() {  
    return $.ajax({  
        url: "get_data.php?q=",  
        type: "GET",  
        contentType: "text/plain"  
    })  
}
```

Funkcja przygotowuje żądanie typu GET do mikrousługi `get_data.php` z pustym parametrem `q`. Informuje on o tym, że mają być przesłane tylko dane o aktualnym stanie systemu. Funkcja zwraca obiekt typu `Promise` (ang. *obietnica*), który reprezentuje ostateczne zakończenie lub niepowodzenie operacji asynchronicznej.[?]

Po otrzymaniu żądania program `get_data.php` odczytuje ze zmiennej `$_GET` wartość parametru `q`. Posłuży on do określenia, jakie zapytania mają być wysłane do bazy danych. Następnie nawiązywane jest połączenie z bazą danych przy pomocy biblioteki PDO:

```
$conn = new PDO("mysql:host=$servername;dbname=$db", $username, $password);
```

Zmienna `$conn` reprezentująca połączenie z bazą danych jest użyta do wykonania zapytania o wszystkie wpisy z tabeli `kamery`. Jeśli parametr `q` jest pusty, to dla każdej kamery jest wykonywane zapytanie do tabeli `pomiary` o ostatnio wykonany wpis:

```
$stmt = $conn->query("SELECT *  
FROM pomiary NATURAL JOIN zdjecia NATURAL JOIN stany NATURAL JOIN odczyty  
WHERE zdjecia.id_kamery = $id_kamery  
GROUP BY pomiary.id_pomiaru  
ORDER BY pomiary.id_pomiaru DESC LIMIT 1  
");
```

Wynik tego zapytania jest zapisywany w tablicy pod indeksem równym unikalnemu identyfikatorowi kamery. Po dokonaniu zapytań dla wszystkich kamer istniejących w systemie, program

zwraca odpowiedź w formie JSON (ang. *JavaScript Object Notation*). Jest to jeden z powszechnie stosowanych formatów tekstowej wymiany danych. W nagłówku odpowiedzi należy przekazać informację o typie MIME przekazywanych danych[?] – `Content-Type: application/json`. Następnie dane są kodowane do postaci JSON przy użyciu funkcji `json_encode` i zwrócone przez funkcję `echo`.

Zwrócone dane można obsłużyć po stronie klienta (przeglądarki) na kilka sposobów. Jednym z najnowszych sposobów dostępnych w wersji języka JavaScript ECMAScript 2017 jest składnia `async/await`. Jest ona obsługiwana przez większość współczesnych przeglądarek w najnowszej wersji.[?] Pozwala ona na wykonywanie asynchronicznego kodu w sposób przypominający tradycyjne, sekwencyjne działanie. Zaletą takiego rozwiązania jest poprawienie czytelności kodu.

Definicja funkcji działającej w sposób asynchroniczny musi być poprzedzona słowem kluczowym `async`. W ciele funkcji można wówczas wywoływać funkcje zwracające obiekty `Promise` ze słowem kluczowym `await`. Powoduje to wstrzymanie działania funkcji do momentu rozwiązania obietnicy:

```
async function main() {
  try {
    const result = await getData();
    displayHomeContent(result);
    (...)
  } catch (e) {
    console.log(e.responseText);
    (...)
  }
  (...)
}
```

Funkcja `getData()` jest wywoływana w bloku `try/catch` pozwalającym na obsługę wyjątków w przypadku niepowodzenia asynchronicznego żądania HTTP. W przypadku pozytywnego zakończenia żądania dane zwrócone w odpowiedzi są zapisywane do zmiennej i mogą być użyte przez funkcję `displayHomeContent()` wyświetlającą dane na stronie głównej.

Procedura pobierania danych o archiwalnych stanach systemu jest podobna. Przesłanie żądania do serwera wykonuje funkcja `getAllData()`. Wywołuje ona żądanie typu GET do mikrousługi `get_data.php` z atrybutem `q` równym `archiwum`. Po stronie serwerowej żądanie obsługiwane jest podobnie do żądania z pustym atrybutem `q`. Różnica polega na tym, że dla każdej kamery w systemie zwracana jest cała historia jej zdjęć przy użyciu poniższego zapytania do bazy danych:

```
$stmt = $conn->query("
SELECT *
FROM zdjecia NATURAL JOIN pomiary NATURAL JOIN stany NATURAL JOIN odczyty
WHERE zdjecia.id_kamery = $id_kamery
");
```

Następnie dane są zapisywane w formie JSON i zwracane do klienta przez funkcję `echo`. Po stronie przeglądarki obiekt `Promise` zwracany przez funkcję `getAllData()` również jest

obsługiwany przy pomocy składni `await/async`. Po pozytywnym zakończeniu pobierania danych tworzony jest widok archiwum.

### 5.2.2. Ustawienia

Przez stronę WWW użytkownik ma również dostęp do panelu administracyjnego pozwalającego na zmianę ustawień. Dostępne ustawienia to:

- włączenie powiadomień e-mail i podanie adresu e-mail odbiorcy powiadomień,
- zmiana nazwy urządzeń podłączonych do systemu,
- zmiana częstotliwości wykonywania zdjęć i pomiarów.

Po dokonaniu zmian w formularzu i zapisaniu zmian skrypt `main.js` zapisuje dane ze wszystkich elementów HTML typu `form`. Wykorzystywana jest przy tym funkcja `serializeArray()` z biblioteki jQuery. Zapisuje ona dane z formularza do tablicy obiektów o postaci: `{name: <Nazwa ustawienia>, value: <Wartość ustawienia>}`. W przypadku formularzy z danymi dotyczącymi ustawień nazw urządzeń i częstotliwości pomiarów obiekty te zostaną przekształcone do postaci: `{<Nazwa ustawienia>: <Wartość ustawienia>}`. Jest ona łatwiejsza do parsowania po stronie serwerowej, ponieważ bezpośrednio łączy nazwę kolumny z wartością, która ma być dla niej ustawiona. Wykorzystana do tego zostanie wbudowana funkcja obiektu `Array – reduce()`:

```
const data = $(element).serializeArray().reduce((a, x) => {
  a[x.name] = x.value;
  return a;
}, {});
```

Pierwszym argumentem funkcji `reduce()` jest anonimowa funkcja wywołana przy pomocy wyrażenia funkcji strzałkowej (`=>`). Jest ona wywoływana dla każdego elementu tablicy i wykorzystuje dwa argumenty: tablicę, która ma być spłaszczona (`a`) oraz aktualny element tablicy (`x`). Wartość ustawienia jest zapisywana przy kluczu ją opisującym, a wynik działania funkcji zapisywany jest w nowym obiekcie podanym jako drugi argument funkcji – `{}`. W ten sposób dane są zapisane w formie JSON. Przed dołączeniem do ciała funkcji muszą one być zmienione z obiektu do postaci ciągu znaków (`string`).

Każde z wymienionych wyżej ustawień posiada osobną funkcję przygotowującą żądanie typu POST do serwera. Jest to typ żądania HTTP używany do przesyłania danych z formularzy.[?] Wynika to z tego, że każde z tych danych jest inaczej przetwarzane po stronie serwera. Dla ustawień e-mail jest to funkcja `updateSettings()`. Wysyła ona żądanie typu POST do mikro-usługi `update_settings.php`. Program ten zaczyna od zdekodowania przesłanych danych w formie JSON do tablicy:

```
$input = json_decode( file_get_contents( 'php://input' ) )
```

Następnie dla każdego ustawienia (pary klucz/wartość) wykonywane jest zapytanie typu UPDATE do bazy danych o następującej treści:

```
UPDATE ustawienia
SET wartosc=:wartosc
WHERE klucz=:klucz
```

, gdzie `:wartosc` oraz `:klucz` zostaną uzupełnione odpowiednimi danymi w momencie wykonywania zapytania przez funkcję `execute()`. Użycie parametryzowanych zapytań zabezpiecza system przed atakami typu *SQL Injection*. Polegają one na umieszczeniu np. w treści formularza zapytania języka SQL, które umieszczone bezpośrednio w treści zapytania mogłyby narazić bazę danych np. na usunięcie którejś z tabel.[?]

Informacje o zmianie nazw urządzeń są przesyłane w postaci żądania HTTP typu POST przez funkcję `updateNames()` do mikrousługi `update_names.php`. Dla każdej kamery i przypisanych do niej czujników jest wysyłane osobne żądanie zawierające wprowadzone nazwy oraz identyfikator kamery. Program `update_names.php` po zdekodowaniu przesłanych danych wykonuje zapytanie do bazy danych o identyfikatory czujników, które są przypisane do kamery o przesłanym identyfikatorze. Następnie dla każdego urządzenia następuje aktualizacja nazw, jeśli przesłany ciąg znaków nie jest pusty. W tym przypadku również użyto parametryzowanych zapytań.

Zmiana preferencji wykonywania pomiarów i zdjęć jest przekazywana do serwera przy użyciu funkcji `updatePrefs()`. Przekazuje ona zebrane z formularza dane do mikrousługi `update_prefs.php`, załączając je do żądania typu POST. Dane te również są wysyłane osobno dla każdej grupy urządzeń z identyfikowanym kluczem głównym obiektu typu kamera. W tym wypadku dane są zmieniane w zbiorczym zapytaniu do bazy danych:

```
UPDATE
kamery NATURAL JOIN czujniki NATURAL JOIN czujniki_temperatury
SET
kamery.czesotliwosc_zdjecia = :czesotliwosc_zdjecia ,
czujniki.czesotliwosc_odczytu_stanu = :czesotliwosc_odczytu_stanu ,
czujniki_temperatury.czesotliwosc_pomiaru_temp = :czesotliwosc_pomiaru_temp
WHERE
id_kamery = :id_kamery
```

Każde z wykonywanych żądań zwraca obiekt `Promise`, który jest umieszczany na liście. Po umieszczeniu wszystkich obiektów liście wywoływana jest metoda `Promise.all()`. Czekając na rozwiązanie wszystkich obiektów `Promise`. Jeśli wszystkie zakończą się powodzeniem, wykonywana jest przekazana w pierwszym argumencie funkcja. Wyświetla ona powiadomienie o sukcesie aktualizacji ustawień. Jeśli jednak przynajmniej jedno żądanie zakończy się niepowodzeniem, wykonywana jest funkcja wyświetlająca powiadomienie o błędzie.

### 5.2.3. Eksport zdarzeń

Wymagania systemu mówią również o możliwości eksportu listy zdarzeń (stanów systemu) do pliku w formacie `csv`. Wysłanie żądania pobrania pliku jest realizowane inaczej niż w opisanej wyżej technice AJAX. Zostaje otwarte nowe okno przeglądarki, którego adres URL jest ustawiony jako `export_events.php?start=<Data początku>&end=<Data końca>`. Użytkownik może wybrać w interfejsie przedział dat, dla których mają być wyeksportowane zdarzenia.

Są one umieszczane jako wartości parametrów `start` i `end`. Otwarte okno jest puste i służy jedynie do wymuszenia otwarcia okna dialogowego do zapisu pliku, o który prosi użytkownik:

```
let url = 'export_events.php?start=';
if (startDate != null) url = url + startDateString;
url = url + '&end=';
if (endDate != null) url = url + endDateString;
window.open(url, '_blank');
```

Otwarcie nowego okna w przeglądarce z adresem URL programu `export_events.php` powoduje wywołanie jego działania po stronie serwerowej. Program pobiera wartości parametrów URL `start` i `end`, a następnie wykonuje zapytanie do bazy danych o pierwszy i ostatni rekord z tabeli `pomiary`. Jeśli użytkownik nie wybrał daty początku lub końca zakresu, dla którego mają być eksportowane zdarzenia, zostaną one domyślnie ustawione odpowiednio jako data pierwszego lub ostatniego wpisu w bazie danych. Takie podejście pozwala na eksport wszystkich zdarzeń poprzez pozostawienie wartości parametrów `start` i `end` pustych.

Następnie wykonywane jest zapytanie do bazy danych o wszystkie wpisy do tabeli `pomiary` wraz z odpowiadającym im odczytom urządzeń i informacjami o urządzeniach:

```
SELECT *
FROM pomiary
NATURAL JOIN zdjecia NATURAL JOIN kamery
NATURAL JOIN stany NATURAL JOIN czujniki
NATURAL JOIN odczyty NATURAL JOIN czujniki_temperatury
WHERE data BETWEEN :first_date AND :last_date
```

Dane zwrócone przez zapytanie są zapisywane do pliku w formacie `csv` o nazwie informującej o wybranym przedziale dat. W pliku wynikowym zapisywane są również nagłówki kolumn, dzięki czemu użytkownik otrzyma informację o tym, co przedstawiają dane. Po zakończeniu zapisu do pliku, jego zawartość jest zwrócona użytkownikowi przy pomocy funkcji `readfile`. Plik jest przechowywany na serwerze w folderze `log`. Dzięki temu kolejne zapytania o zdarzenie z tego samego zakresu dat nie wymagają przeprowadzenia zapytania do bazy danych. Przed przeprowadzeniem zapytania program sprawdza, czy istnieje już plik o nazwie odpowiadającej żadanemu przedziałowi dat. Jeśli tak, plik zostaje odczytany i zwrócony do klienta.

Aby zwrócone dane były potraktowane jako plik do zapisu jako załącznik, konieczne jest właściwe ustawienie nagłówków odpowiedzi:

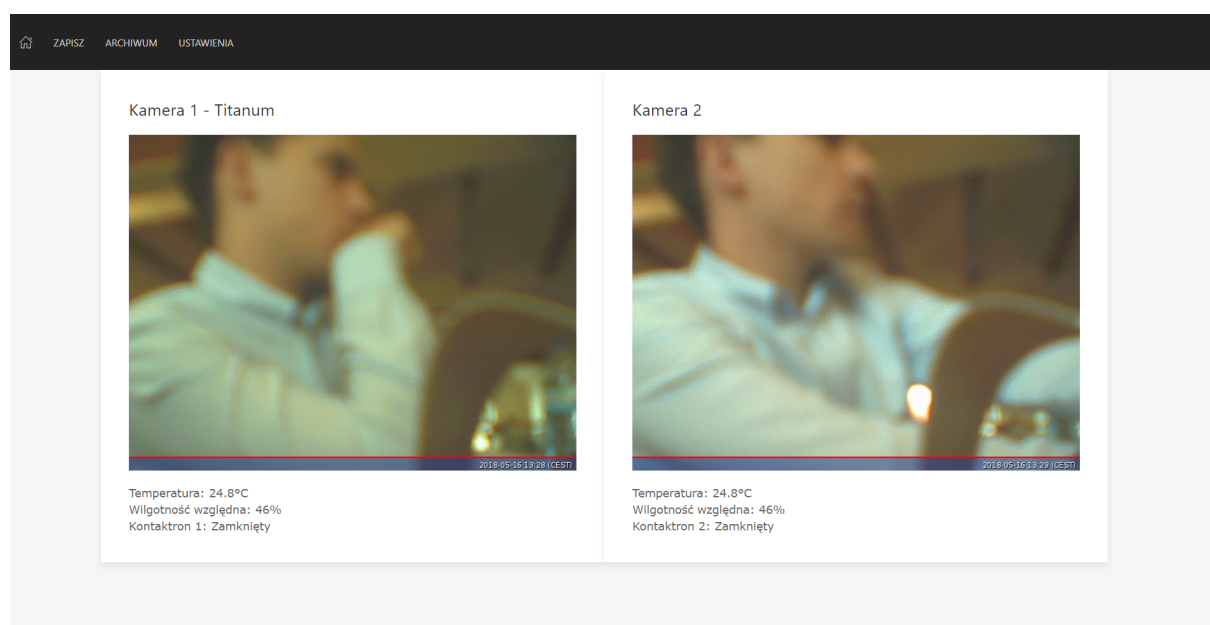
```
header("Content-Description: File Transfer");
header("Content-Type: application/octet-stream");
header('Content-Disposition: attachment; filename="'.basename($file).'"');
```

Odpowiadają one za przekazanie informacji klientowi, że następuje transfer pliku (`File Transfer`) typu binarnego (`application/octet-stream`), który ma zostać zapisany po otwarciu okna dialogowego do zapisu plików (`Content-Disposition: attachment`).[?][?]

### 5.3. Interfejs użytkownika

Interfejs użytkownika stanowi strona WWW typu *single-page application* (SPA), której szkielet jest opisany w pliku `index.html`. Idea SPA polega na stworzeniu strony WWW, która nie wymaga przeładowywania, żeby prezentować różną zawartość, a wszystkie dane mogą być udostępniane przez jeden plik HTML. Odświeżanie zawartości strony następuje w tle dzięki komunikacji z serwerem WWW. W projektowanym systemie komunikacja ta została wykonana zgodnie z techniką AJAX opisaną w poprzednich podrozdziałach. Do wykonania interfejsu użytkownika posłużyłem się biblioteką *UIKit*, która dostarcza gotowe rozwiązania takich elementów jak np. okna modalne czy elementy nawigacyjne.

Na stronie głównej przedstawionej na rysunku ?? domyślnie wyświetlane są ostatnie zdjęcia ze wszystkich kamer w systemie. Na karcie kamery znajdują się: nazwa kamery, ostatnie zdjęcie nią wykonane oraz dane z czujnika temperatury i czujnika stykowego przypisanego do kamery. Dane te są odświeżane domyślnie co 8 sekund (częściej niż wykonywane zdjęcia), aby stale prezentować użytkownikowi aktualny stan systemu.



Rys. 7: Strona główna interfejsu użytkownika

Pasek nawigacyjny u góry ekranu pozwala przejść do *Archiwum* prezentującego przeszłe zdjęcia podzielone według kamer, które je wykonały. Zdjęcia są przedstawiane w postaci miniatur umieszczonych na siatce. Po kliknięciu na nie zdjęcie powiększa się, zajmując cały ekran. Wówczas zdjęcia można również przeglądać w trybie galerii zdjęć. Galeria zdjęć została zaimplementowana przy użyciu obiektu *Lightbox* z biblioteki *UIKit*[?]

Wybór linku *Zapisz* otwiera okno modalne z dwoma polami do wyboru dat. Na rysunku ?? widać stan okna w przypadku wybrania początkowej daty. Wówczas w drugim kalendarzu do wyboru dat dostępny jest wyłącznie zakres dat następujący po wybranej dacie początku. Implementację kalendarza do wyboru dat dostarcza biblioteka *Pikaday*[?]. W momencie tworzenia obiektu można podać nazwę funkcji, która ma być wykonywana w momencie wyboru daty (zdarzenie `OnSelect`). W przypadku kalendarza do wyboru początkowej daty jest to funkcja `updateStartDate`, a dla kalendarza do wyboru końcowej daty `updateEndDate`:

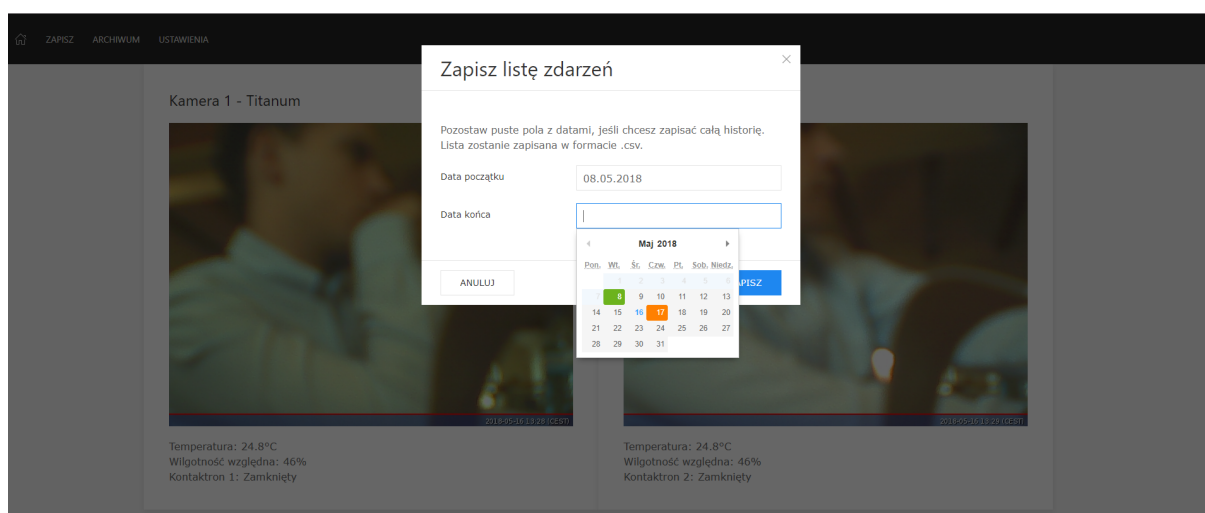


```

function updateStartDate(startDate, startPicker, endPicker) {
    startPicker.setStartRange(startDate);
    endPicker.setStartRange(startDate);
    endPicker.setMinDate(startDate);
}
function updateEndDate(endDate, startPicker, endPicker) {
    startPicker.setEndRange(endDate);
    startPicker.setMaxDate(endDate);
    endPicker.setEndRange(endDate);
}

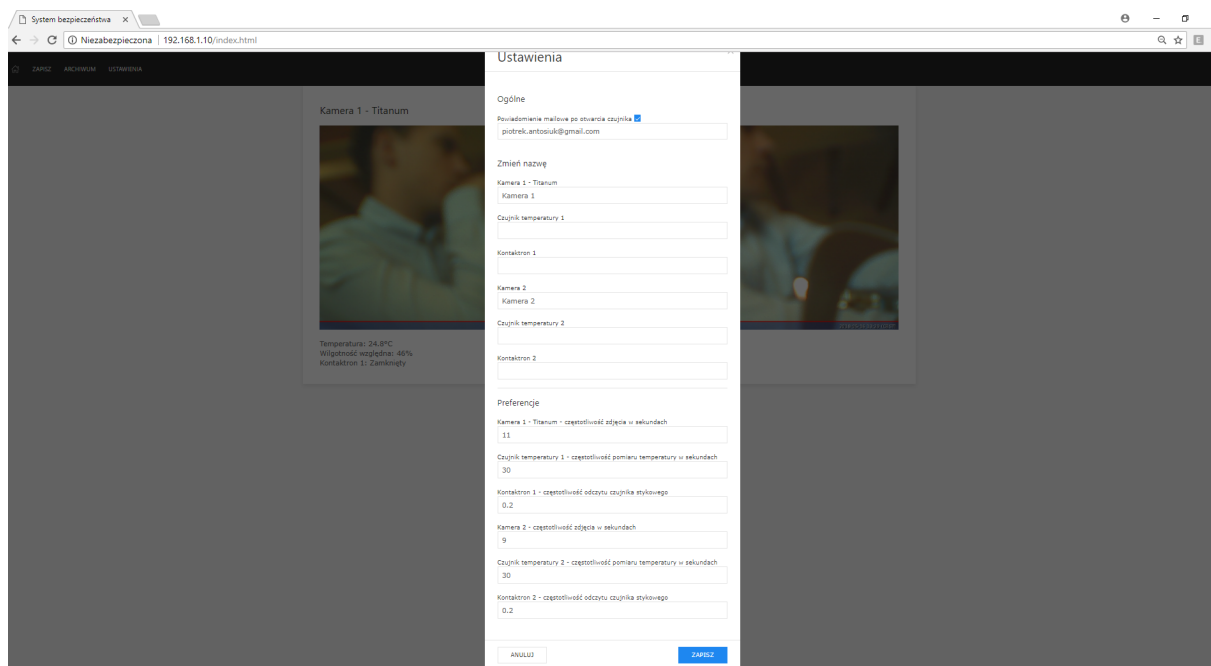
```

Pierwsza z nich odpowiada za ustawienie wybranej daty początkowej jako najmniejszej możliwej daty w kalendarzu do wyboru daty końcowej. Druga wykonuje analogiczną operację z wybraną datą końcową i kalendarzem do wyboru daty początkowej. Dzięki temu interfejs nie pozwala na wprowadzenie błędnych zakresów dat np. takich, w których data początku jest większa niż data końca.



Rys. 8: Okno modalne do eksportu listy zdarzeń

Otworzenie okna modalnego z panelem administracyjnym jest możliwe po kliknięciu w link *Ustawienia*. Okno ustawień składa się z kilku formularzy umożliwiających ich zmianę. Sekcja *Ogólne* pozwala na włączenie powiadomień e-mail. Po zaznaczeniu pola wyboru (ang. *checkbox*) tego ustawienia pojawia się również pole tekstowe, w którym można wpisać adres e-mail odbiorcy powiadomień. Kolejna sekcja *Zmień nazwę* składa się z pól tekstowych opisanych dotychczasowymi nazwami urzędzeń. Nazwa nie jest zmieniana, jeśli pole jest pozostawione puste. Ostatnia sekcja *Preferencje* umożliwia zmianę częstotliwości wykonywania zdjęć i pomiarów. Pola tekstowe opisane nazwa urzędzeń są domyślnie wypełnione dotychczasowymi wartościami. Wszystkie zmiany są wprowadzane dopiero po zatwierdzeniu ich przyciskiem *Zapisz*. Zamknięcie okna przyciskiem *Anuluj* powoduje usunięcie wpisanych wartości do formularza i przywrócenie go do pierwotnego stanu.



Rys. 9: Okno modalne ustawień



## **6. Testy systemu**

## **7. Wnioski i podsumowanie**

## 8. Bibliografia

### Literatura

- [1] Apache Software Foundation, Authentication and Authorization, Dokumentacja serwera HTTP Apache [przełączany 30 maja 2018]. Dostępny w: <https://httpd.apache.org/docs/2.4/howto/auth.html>
- [2] Apache Software Foundation, htpasswd - Manage user files for basic authentication, Dokumentacja serwera HTTP Apache [przełączany 30 maja 2018]. Dostępny w: <https://httpd.apache.org/docs/2.4/programs/htpasswd.html>
- [3] 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 Dostępny w: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1474709/>
- [4] Bayer M., Session Basics, Dokumentacja SQLAlchemy [przełączany 30 maja 2018]. Dostępny w: [http://docs.sqlalchemy.org/en/latest/orm/session\\_basics.html](http://docs.sqlalchemy.org/en/latest/orm/session_basics.html)
- [5] Bayer M., Declarative API, Dokumentacja SQLAlchemy [przełączany 2 czerwca 2018]. Dostępny w: <http://docs.sqlalchemy.org/en/latest/orm/extensions/declarative/api.html>
- [6] BeagleBoard: BeagleBoneBlack, *Embedded Linux Wiki* [przełączany 25 maja 2018]. Dostępny w: <https://elinux.org/Beagleboard:BeagleBoneBlack>
- [7] Bushell D., Rikkert R., Dokumentacja biblioteki Pikaday [przełączany 2 czerwca 2018]. Dostępny w: <https://github.com/dbushell/Pikaday>
- [8] Cabaj K., Mazur G., Nosek M., *Compromising an IoT device based on Harvard-architecture microcontroller*, XLII Sympozjum Wilga-2018, 3-10 czerwca 2018.
- [9] Comparison of single-board computers, *Wikipedia* [przełączany 2 czerwca 2018]. Dostępny w: [https://en.wikipedia.org/w/index.php?title=Comparison\\_of\\_single-board\\_computers&oldid=843993327](https://en.wikipedia.org/w/index.php?title=Comparison_of_single-board_computers&oldid=843993327)
- [10] Clarke J., SQL Injection Attacks and Defense, wyd. 2, Syngress, 2012, ISBN 978-1-59749-963-7
- [11] Dziedzic K., Kamera IP w sieci. Stwórz prywatny monitoring [przełączany 25 maja 2018]. Dostępny w: <http://www.komputerswiat.pl/poradniki/sprzet/kamery-internetowe/2015/07/kamera-ip-w-sieci.aspx>
- [12] Fielding R., Reschke J., Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content, RFC7231, IETF, czerwiec 2014, DOI: 10.17487/RFC7231 Dostępny w: <https://tools.ietf.org/html/rfc7231>
- [13] Fielding R., Reschke J., Hypertext Transfer Protocol (HTTP/1.1): Authentication, RFC 7235, IETF, czerwiec 2014, DOI: 10.17487/RFC7235. Dostępny w: <https://tools.ietf.org/html/rfc7235>

- [14] 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. Dostępny w: <https://tools.ietf.org/html/rfc2045>
- [15] Freed N., Borenstein N., Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types, RFC 2046, IETF, listopad 1996, DOI: 10.17487/RFC2046. Dostępny w: <https://tools.ietf.org/html/rfc2046>
- [16] Hoperf Electronic, Karta katalogowa czujnika temperatury i wilgotności TH02 [przełączany 2 czerwca 2018]. Dostępny w: [http://www.hoperf.com/upload/sensor/TH02\\_V1.1.pdf](http://www.hoperf.com/upload/sensor/TH02_V1.1.pdf)
- [17] joan2937, Dokumentacja funkcji `i2c_read_device()` [przełączany 25 maja 2018]. Dostępny w: [http://abyz.me.uk/rpi/pigpio/python.html#i2c\\_read\\_device](http://abyz.me.uk/rpi/pigpio/python.html#i2c_read_device)
- [18] Kostyrko K., Łobzowski A., *Klimat. Pomiary i regulacje*, Agenda wydawnicza PAK, Warszawa, wrzesień 2002.
- [19] Mouser Electronics, Karta katalogowa czujnika temperatury i wilgotności SHT31D [przełączany 25 maja 2018]. Dostępny w: [http://www.mouser.com/ds/2/682/Sensirion\\_Humidity\\_Sensors\\_SHT3x\\_Datasheet\\_digital-971521.pdf](http://www.mouser.com/ds/2/682/Sensirion_Humidity_Sensors_SHT3x_Datasheet_digital-971521.pdf)
- [20] Mozilla Developer Network, szopenkrk, Promise [przełączany 30 maja 2018]. Dostępny w: <https://developer.mozilla.org/pl/docs/Web/JavaScript/Referencje/Obiekty/Promise>
- [21] Mozilla Developer Network, Bab64, doubleOrt, freezy, hirokiky, jswisher, spygi i in., `async function` [przełączany 30 maja 2018]. Dostępny w: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async\\_function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function)
- [22] Python Software Foundation, Dokumentacja modułu Python `email.mime` [przełączany 24 maja 2018]. Dostępny w: <https://docs.python.org/3/library/email.mime.html>
- [23] Raspberry Pi Foundation, Schematy peryferiów Raspberry Pi 3 Model B [przełączany 24 maja 2018]. Dostępny w: [https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/rpi\\_SCH\\_3b\\_1p2\\_reduced.pdf](https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/rpi_SCH_3b_1p2_reduced.pdf)
- [24] Raspberry Pi Foundation, Specyfikacja Raspberry Pi 3 Model B [przełączany 24 maja 2018]. Dostępny w: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>
- [25] Reschke J. Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP), RFC 6266, IETF, czerwiec 2011, DOI: 10.17487/RFC6266. Dostępny w: <https://tools.ietf.org/html/rfc6266>
- [26] Silicon Labs, Karta katalogowa czujnika temperatury i wilgotności Si7021 [przełączany 24 maja 2018]. Dostępny w: <https://www.silabs.com/documents/public/data-sheets/Si7021-A20.pdf>

- [27] Texas Instruments Inc., Karta katalogowa multipleksera TCA9548A [przełączany 24 maja 2018]. Dostępny w: <http://www.ti.com/lit/ds/symlink/tca9548a.pdf>
- [28] Torvalds L., Opis protokołu SMBus [przełączany 20 maja 2018]. Dostępny w: <https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/plain/Documentation/i2c/smbus-protocol>
- [29] YOOSTheme GmbH, Dokumentacja biblioteki UIKit [przełączany 2 czerwca 2018]. Dostępny w: <https://getuikit.com/docs/introduction>

## **9. Załączniki**