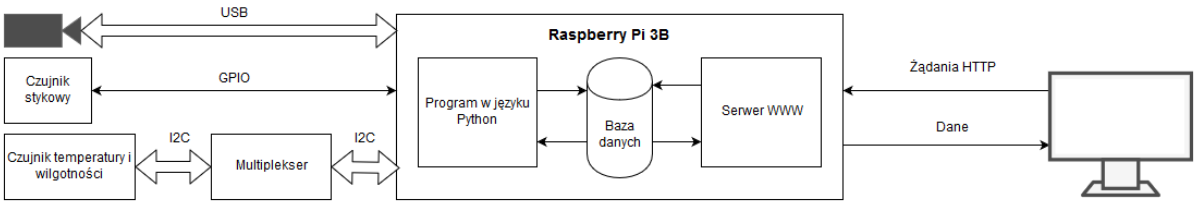
1 Architektura systemu



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 powinien być podłączony do wyprowadzenia GPIO i do masy, aby wykrywać zamknięcie i przerwanie obwodu. Raspberry Pi umożliwia podłączenie każdego wyprowadzenia poprzez wbudowany rezystor podciągający do napięcia zasilania (Vcc). Dzięki temu można wykrywać otwarcie okien lub drzwi jako oddalenie czujników i przerwanie obwodu – zmianę stanu odczytywaną na wyprowadzeniu z wysokiego na niski. W wykonanym prototypie dłuższy przewód podłączono do wyprowadzenia GPIO, a krótszy do masy na płytce prototypowej.

Wymogiem systemu jest również obsługa wielu czujników i kamer. Adres 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 została 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 czujnika do każdej z magistral. Innym rozwiązaniem tej sytuacji jest zastosowanie multipleksera I2C. Multiplekser zadziała jak przełącznik, który pozwoli na zapis i odczyt z jednym czujnikiem naraz. Przykładem takiego urządzenia jest TCA 9548A firmy Adafruit. Pozwala on na podłączenie do 8 urządzeń korzystających z magistrali I2C.

Do obsługi czujników i kamer 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.

W zaprojektowanym systemie bezpieczeństwa program obsługujący urządzenia działa niezależnie od programu udostępniającego dane użytkownikowi. Ponadto są one napisane w innych językach. Konieczny jest zatem sposób komunikacji i wymiany danych pomiędzy nimi. Dane są przechowywane w relacyjnej bazie danych, która poprzez umożliwia dostęp

do nich system zarządzania bazą danych (SZBD).

Wymogiem systemu jest udostępnianie wyników pomiaru systemu użytkownikowi poprzez stronę WWW. Konieczny jest zatem serwer WWW, na którym będzie umieszczona strona. Jednym z najbardziej popularnych aplikacji do prowadzenia serwerów WWW jest Apache. System Raspbian umożliwia pobranie pakietu, który po zainstalowaniu będzie działał jako serwer WWW. Umieszczona na serwerze strona będzie komunikowała się z serwerem poprzez żądania HTTP. Szkielet strony w postaci pliku HTML będzie wypełniony danymi z czujników po zrealizowaniu żądań wysyłanych asynchronicznie (technika AJAX) do serwera. Za obsługę żądań po stronie serwerowej będą odpowiedzialne mikroserwisy w postaci skryptów w języku PHP. Język PHP jest dobrze zintegrowany z serwerem Apache i jest jednym z najbardziej popularnych na serwerach WWW. Skrypty łączą się z bazą danych, w której są przechowywane dane zebrane z kamer i czujników. Następnie zwracają dane w odpowiedzi na żądania klienta.

2 Baza danych

Relacyjna baza danych składa się relacji (tabel), które są połączone związkami. Umożliwia ona programistyczny dostęp do danych poprzez oparty na transakcjach silnik bazy danych. Dzięki temu możliwe jest, aby jeden program zapisywał dane do bazy, a drugi niezależnie je odczytywał. W projekcie użyty został system zarządzania bazą danych MySQL. Zapewnia on dobrą integrację z użytym jako serwer WWW programem Apache.

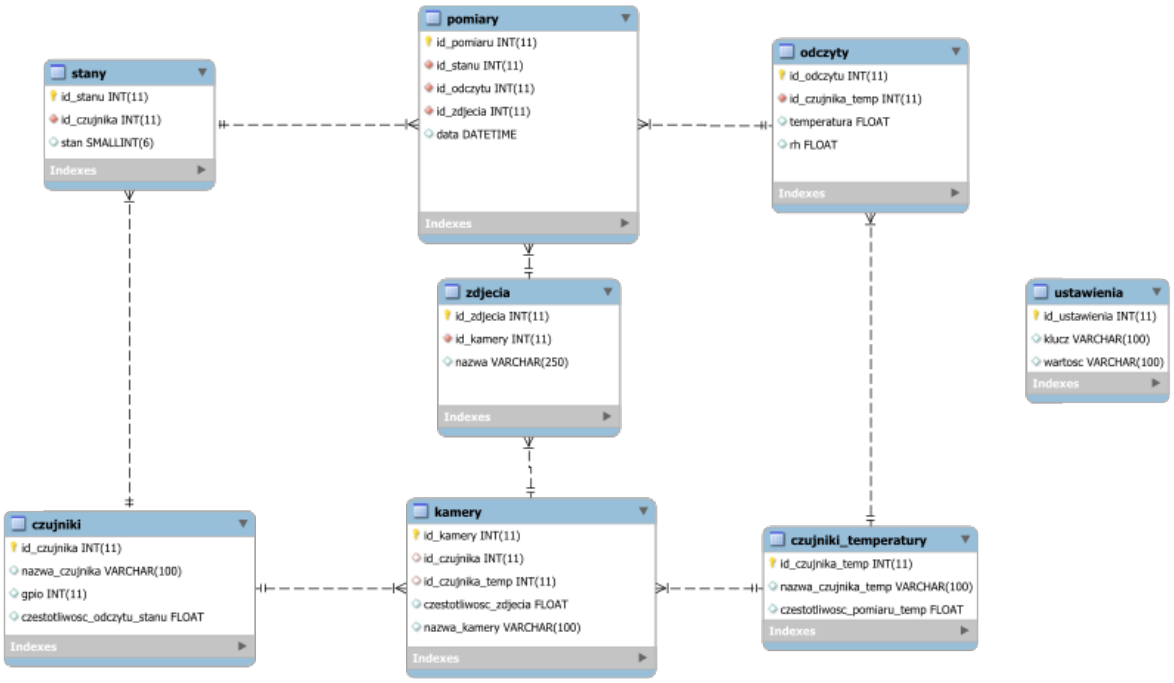
2.1 Projekt bazy danych

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. Informacje o urządzeniach mogą zawierać nazwę identyfikującą ich położenie lub przeznaczenie np. "Kamera w kuchni" lub "Czujnik - drzwi wejściowe". Każde urządzenie będzie jednak różniło się przechowywanymi informacjami. Tabela z czujnikami temperatur przechowuje dodatkowo informacje o numerze kanału multipleksera, do którego jest podłączona, a tabela z czujnikami stykowymi o numerze wyprowadzenia GPIO, do którego jest podłączony. 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 jednoznaczny identyfikator zdjęcia, który pozwoli zlokalizować je w pamięci urządzenia. Tabela z pomiarami powinna również posiadać datę wpisu.

System może jednak obsługiwać wiele kamer i czujników. Oznacza to, że między urządzeniem a pomiarem zachodzi związek wiele do wielu. Polega on na tym, że jedna kamera jest przypisana do wielu pomiarów, a jednocześnie pomiar może mieć wiele kamer. Jest on niemożliwy do zrealizowania przy użyciu kluczy obcych. Konieczne jest zatem wprowadzenie pośredniej tabeli, która zrealizuje w praktyce związek wiele do wielu. Taką tabelą dla przykładu kamery i pomiaru są zdjęcia. Każda kamera może wykonać wiele zdjęć, ale każde zdjęcie ma przypisaną tylko jedną kamerę. Podobnie każdy pomiar odpowiada tylko jednemu zdjęciu, ale jedno zdjęcie może odpowiadać wielu pomiarom. Wynika to z tego,

że wpisy do tabeli pomiary są niezależne od wykonywanych zdjęć. Dla czujników temperatury taką tabelą realizującą związek wiele do wielu jest tabela pomiary, a dla czujników stykowych - stany.



Rysunek 2: Schemat bazy danych na poziomie logicznym

Rysunek 2 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.

Poza wspomnianymi wyżej relacjami istnieje również relacja ustawienia, która nie jest w związku z innymi. 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.

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).

2.2 Fizyczna 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 (and. 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.

3 Program obsługujący kamery i czujniki

Program obsługujący czujniki i kamery musi cyklicznie wykonywać pomiary, odczyty i zdjęcia, a wyniki zapisywać do bazy danych. W związku potrzebne jest rozwiązanie, które pozwoli cyklicznie wykonywać funkcje w ramach jednego programu. Istnieją mechanizmy wewnątrz języka Python, które umożliwiają wykonywanie zadań po minięciu pewnego czasu. Wykorzystanie modułu sched wymagałoby jednak cyklicznego umieszczania zadania w kolejce po jego wykonaniu. Innym modułem jest schedule, który pozwala w jednej linijce zaplanować cykliczne wykonanie zadania. Poniżej przykładowe wywołanie szeregowania przy pomocy tego modułu.

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

3.1 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 Broadcomm, 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 podciągającego 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_UP)
```

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

3.2 Obsługa czujników podłączonych do magistrali I2C

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.

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

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

Do obsługi żądań konieczna była odpowiedni pakiet. Pierwszym zastosowanym pakietem był pakiet python-smbus. Korzysta on ze sterownika wbudowanego w jądro systemu Linux. Została przeprowadzona próba komunikacji w trybie Hold Master Mode ze względu na dostępne komendy protokołu I2C. Zakończyła się ona błędem o kodzie io errno5. Wynikał on z tego, że pakiet smbus obsługuje magistralę I2C zgodnie ze standardem SMBus. Posiada on bardziej ścisłe reguły dotyczące czasu trwania transakcji na magistrali. Przy użyciu pakietu nie było możliwe zastosowanie trybu No Hold Master Mode, ponieważ nie dostarczał on komendy odpowiadającej odczytowi rejestru urządzenia o wskazanym adresie. Wszystkie dostępne komendy zawierały zapis do urządzenia przed odczytem.

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 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 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 klasy Grupa.

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 z karty katalogowej.

Czujnik Si 7021 wykonuje również pomiar temperatury potrzebny do wyznaczenia wilgotności względnej. Wynik tego pomiaru jest przechowywany w urządzeniu. Można go odczytać, wysyłając komendę o kodzie 0xE0. Całą transakcję można zrealizować przy użyciu jednej komendy i2c_read_i2c_block_data, ponieważ nie ma potrzeby oczekiwania na konwersję pomiaru temperatury. Wysyła ona żądanie wystawienia wyniku pomiaru temperatury. Następnie odczytywana jest podana liczba bajtów. Bajty te są zapisane do tablicy i użyte do obliczenia wartości temperatury.

3.3 Obsługa kamer USB

Raspberry Pi 3 B posiada 4 gniazda USB, do których mogą podłączone kamery USB. Kamery USB są obsługiwane przy pomocy aplikacji fswebcam. Jest to samodzielny program pozwalający na wywoływanie zdjęć z kamer USB podłączonych do komputerem z systemem typu Unix. Wywołanie tego programu z poziomu skryptu w języku Python

wymaga użycia modułu subprocessing. Daje on możliwość otwierania programów w osobnych procesach. Do otwarcia fswebcam użyta została klasa Popen, ponieważ daje możliwość wywołania metody wait(), pozwalającej zaczekać na zakończenie wykonywania się procesu.