Stacja pogodowa: Pogodownik 3000

Autorzy:
Haniszewski Kamil,
Susłowicz Mateusz,
Trętowicz Tymoteusz,
Ziobrowski Dawid

Prowadzący:
dr inż. Donat Orski

Spis treści

1 Wymagania projektowe

System ma na celu udostępnienie użytkownik informacji dotyczących warunków pogodowych. Wartości temperatury, wysokości na poziomem morza, ciśnienia atmosferycznego oraz wilgotności są zbierane z systemów RasberryPi z czujnikami *BME280*. Stacje wysyłają zebrane informację w ustalonych odstępach czasu.

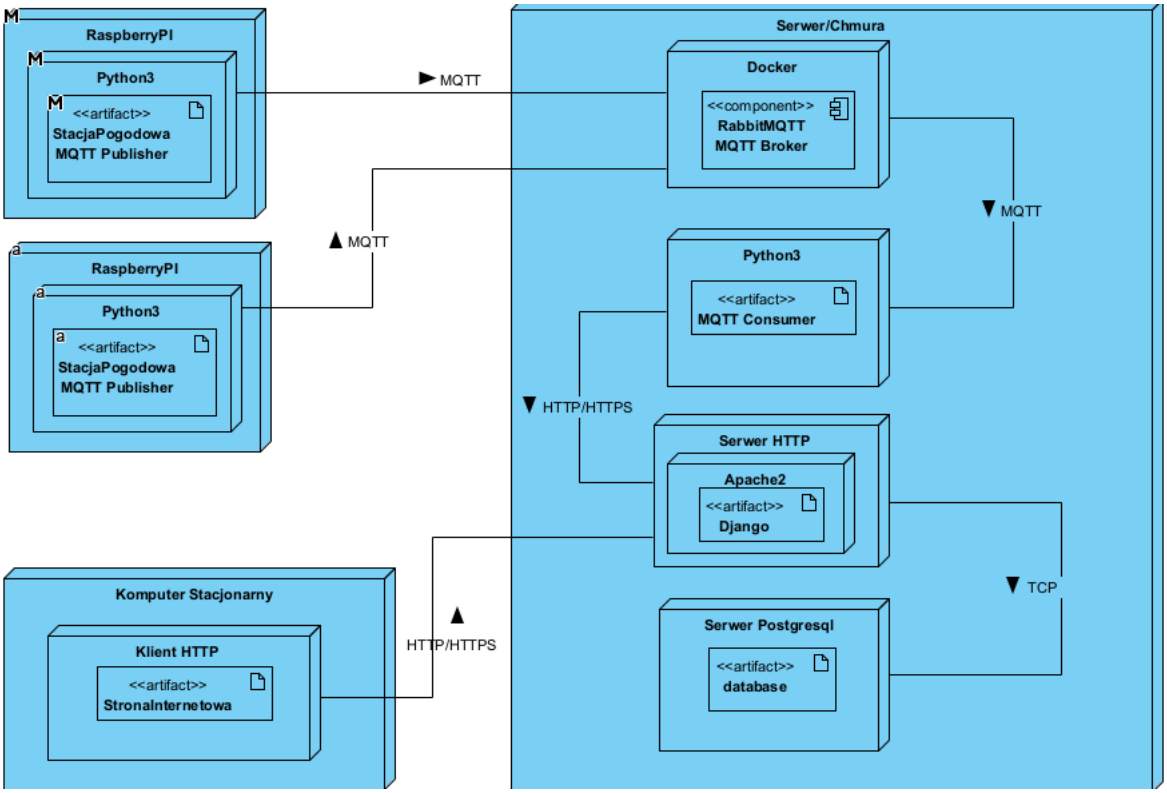
1.1 Wymagania funkcjonalne

- Zbieranie, przetwarzanie i zapisywanie danych pogodowych ze stacji (RasberryPi),
- Zbieranie i wyświetlanie danych dotyczących temperatury,
- Zbieranie i wyświetlanie danych dotyczących wilgotności powietrza,
- Zbieranie i wyświetlanie danych dotyczących wyoskości nad poziomem morza (w metrach),
- Zbieranie i wyświetlanie danych dotyczących ciśnienia atmosferycznego,
- Zbieranie i wyświetlanie danych dotyczących temperatury,
- Wyliczanie wartości średnich na podstawie zebrnaych danych,
- Wyświetlanie wykresów bazujących na zebranych informacjach z wybranej stacji.
- Wyświetlanie wykresów bazujących na zebranych informacjach z wartości średnich ze wszystkich stacji.

1.2 Wymagania niefunkcjonalne

- Dostępność strony prezentującej dane w Polskiej wersji językowej,
- Dostępność strony prezentującej dane na przeglądarkach: Google Chrome, Chromium, Microsoft Edge, Safari, Mozilla Firefox, Opera.
- Zgodność z protokołem MQTT,
- Zgodność ze standardami i dobrymi parktykami framework'a Django,

2 Opis architektury systemu



Rys. 1: Diagram rozmieszczenia architektury.

Pojedyńcza stacja pogodowa to system RasberryPi z czujnikiem BME280 oraz programem napisanym w języku Python3 publikującym dane pogodowe. Taka stacja pełni role nadawcy w protokole MQTT (*MQTT Publisher*).

Dane ze stacji są odbierane przez broker wiadomości RabbitMQ. Działa on na serwerze w konetrze Dockera.

Konsumer (*MQTT Consumer*) napisany w języku Python3 wysyła odebrane informacje do serwera HTTP, który następnie zapisuje je do bazy danych (PostgreSQL). Serwer HTTP jest również odpowiedzialny za odbieranie zapytań i udostępnianie strony internetowej pod adresem: pogoda.nazaliczenie.pl.

3 Opis implementacji zastosowanych rozwiązań

Kod znajduje się w trzech repozytoriach:

1. `5_PIR_Pogoda_Producer` - Projekt zawiera skrypt startowy powłoki systemowej, który instaluje potrzebne dependencje oraz uruchamia program który dokonuje pomiarów pogody oraz je wysyła.

2. `5_PIR_Pogoda_Consumer` - Projekt zawiera dane konfiguracyjne oraz program działający jako MQTT Consumer.

3. `5_PIR_Pogoda_Django` - Projekt zawiera implementacje strony webowej we frameworku Django oraz implementacje komunikacji z bazą danych.

3.1 Implementacja pomiaru danych pogodowych

Poniższy kod jest częścią programu włączonego na stacjach pogodowych (RasberryPi). Poniższy fragment kodu jest wykonywany we wcześniej ustalonych odstępach czasu, oraz ma za zadanie konfigurację czujnika *BME280*, zebranie danych z czujnika oraz ich publikacje.
(`5_PIR_Pogoda_Producer/sender.py`)

```
    def bme280():
        i2c = busio.I2C(board.SCL, board.SDA)
        bme280 =
            adafruit_bme280.Adafruit_BME280_I2C(i2c, 0x76)

        # konfiguracja czujnika BME280
        bme280.sea_level_pressure
            = 1013.25
        bme280.standby_period
            = adafruit_bme280.STANDBY_TC_500
        bme280.iir_filter
            = adafruit_bme280.IIR_FILTER_X16
        bme280.overscan_pressure
            = adafruit_bme280.OVERSCAN_X16
        bme280.overscan_humidity
            = adafruit_bme280.OVERSCAN_X1
        bme280.overscan_temperature
            = adafruit_bme280.OVERSCAN_X2

        dic = {}
        # zbieranie danych
        dic["sensor"] =
            int(TERMNINAL_ID) if TERMNINAL_ID else 0
        dic["temperature"] = round(bme280.temperature, 2)
        dic["humidity"] = round(bme280.humidity, 2)
        dic["pressure"] = round(bme280.pressure, 2)
        dic["altitude"] = round(bme280.altitude, 2)
        dic["timestamp"] = str(datetime.datetime.now())

        print(json.dumps(dic))
        # publikacja danych na wczesniej ustalonym kanale
        channel.basic_publish(
            exchange="",
            routing_key="pogoda",
            body=json.dumps(dic)
        )

```

3.2 Implmentacja konsumera

Poniższy kod jest częścią MQTT Consumer'a. Jest on wykonywany za każdym razem gdy konsumer otrzyma wiadomość od brokera. Celem tego kodu jest odczytanie wiadomość, deserializacja treści z formatu JSON do obiektu pythona oraz następnie wysłanie zapytanie HTTP POST z do serwera na węzeł końcowy: `/add_sensor_log`.
(`5_PIR_Pogoda_Consumer/main.py`)

```
    def rabbit_callback(ch, method, properties, body):
        try:
            # tworzenie adresa serwera HTTP
            api_addr = config['webservice']['api_address']
            # odczytanie tresci wiadomosci
            recv = json.loads(body)
            # wyslanie zapytania HTTP POST z wiadomoscia
            resp = requests.post(
                f'{api_addr}/add_sensor_log',
                data=recv
            )

            print(f"Received message: {recv}")
            print(resp.text)

        except json.decoder.JSONDecodeError as err:
            print(f"An error occured: {err}")

```

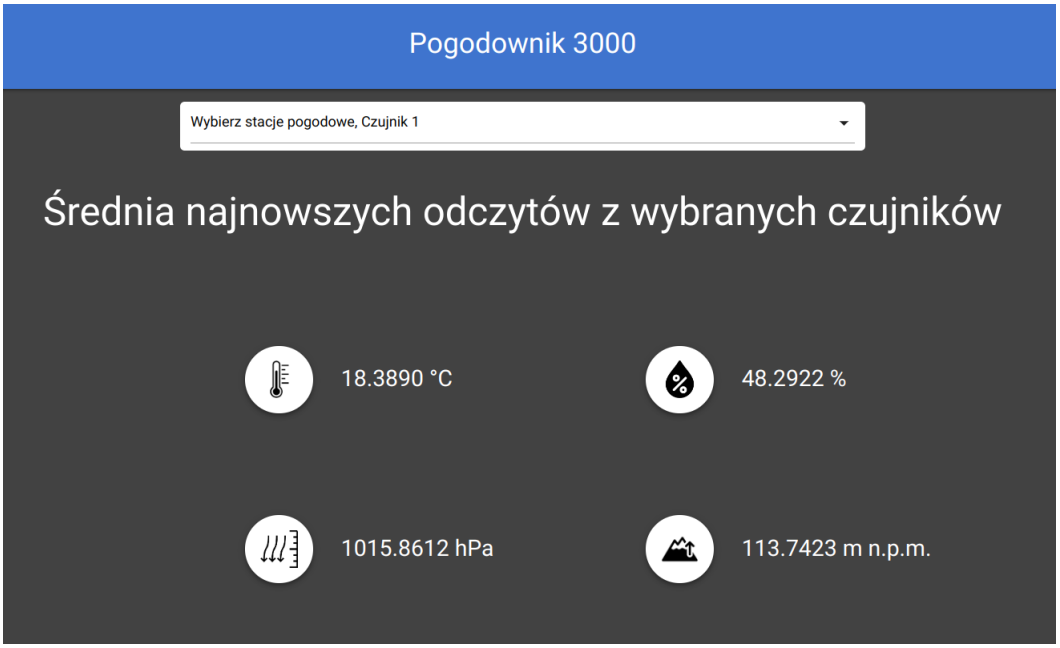
3.3 Implementacja węzła końcowego `/add_sensor_log`

Poniższy kod przedstawia implementacje węzła końcowego (*endpoint'u*) z użyciem frameworka Django. Kod ten się wykona za każdym razem gdy na adres serwera zostanie wysłane zapytanie typu HTTP POST na endpoint `/add_sensor_log`. Procedura sprawdza czy metoda zapytania się zgadza. Jeżeli nie zwracany jest komunikat błędu z kodem 400. W przeciwnym razie następuje stworzenie obiektu do serializacji danych z ciała zapytania. Jeżeli obiekt jest poprawny dane są zapisywane do bazy danych oraz zwracany jest komuniakt sukcesu z kodem 201. W przeciwnym wypadku oznacza to że obiekt serializacji jest nie poprawny więc zwracany jest komunikat błędu z kodem 400.
(`5_PIR_Pogoda_Django/sensors/views.py#add_sensor_log`)

```
    @api_view(['POST'])
    def add_sensor_log(request):
        # jezeli zapytanie jest inne niz HTTP POST...
        if request.method != 'POST':
            # ...odpowiedz bledem z kodem 400
            return Response(
                serializer.errors,
                status=status.HTTP_400_BAD_REQUEST
            )

        serializer = SensorLogSerializer(data=request.data)

        # jezeli serializacja danych sie powiodla...
        if serializer.is_valid():
            # zapisz do bazy danych
            serializer.save()
            # odpowiedz kodem znaczacym sukces
            # i zakocz procedure
            return Response(
                serializer.data,
                status=status.HTTP_201_CREATED
            )

        # jezeli procedura sie nie zakonczyla wczesniej
        # oznacza to ze serializacja danych sie nie powiodla
        # ...odpowiedz bledem z kodem 400
        return Response(
            serializer.errors,
            status=status.HTTP_400_BAD_REQUEST
        )

```

4 Opis działania i prezentacja interfejsu

4.1 Interfejs



Rys. 2: Interfejs główny, przedstawiający wybór czujnika oraz średnią ostatnich odczytów.

Interfejs umożliwia wybór stacji pogodowej z której chcemy zobaczyć pomiary. Wyświetlane dane, czyli temperatura, wilgotność, ciśnienie atmosferyczne oraz wysokość nad poziomem morza są uaktualniane w czasie rzeczywistym.

Rys. 3: Interfejs główny, część dlasza, przedstawiająca wykres wybranych odczytów (tutaj ciśnienia i temperatury).

4.2 Procedura startowa stacji pogodowej

Na włączonym systemie RasberryPi z czujnikiem BME280 z zainstalowanym środowiskiem uruchomieniowym Python3:

- (Opcjonalnie) Pobrać projekt, jeżeli nie jest się w jego posiadaniu.
  W termianlu z powłoką systemu wykonać:
  `$> git clone https://github.com/mamyxk/5_PIR_Pogoda_Producer.git`
- Wejść do katalogu:
  `$> cd 5_PIR_Pogoda_Producer`
- Ustawić skrpyt startowy jako wykonywalny:
  `$> chmod +x start.sh`
- Uzupełnić skrpyt startowy pofunymi danymi (adres ip serwera, port, nazwa kanału, nazwa użytkownika, hasło).

```
1     #ID stacji pogodowej
2     TERMINAL_ID=1
3     # Ip serwera na ktore powinny byc wysylane dane
4     RABBIT_IP=127.0.0.1
5     # Port na ktory powinny byc wysylane dane
6     RABBIT_PORT=8000
7     # Nazwa kalanu na ktorym beda publikowane dane
8     RABBIT_VHOST=pogoda
9     # Nazwa uzytkownika polaczenia do serwera
10    RABBIT_USER=user
11    # Haslo polaczenia do serwera dla uzytkownika
12    RABBIT_PASS=pass
13
```

- Wykonać skrypt startowy:
  `$> ./start.sh`

4.3 Procedura startowa Consumera

Na serwerze ze środowiskiem uruchomieniowym Python3:

- (Opcjonalnie) Pobrać projekt, jeżeli nie jest się w jego posiadaniu.
  W termianlu z powłoką systemu wykonać:
  `$> git clone https://github.com/mamyxk/5_PIR_Pogoda_Consumer.git`
- Wejść do katalogu:
  `$> cd 5_PIR_Pogoda_Consumer`

- Ustawić skrpyt startowy jako wykonywalny:
  `$> chmod +x start.sh`
- Stworzyć plik konfiguracyjny `app.conf.json` (Schemta znajduje się w pliku `app.conf.json.sample`).
- Wykonać skrypt startowy:
  `$> ./start.sh`

4.4 Procedura startowa strony internetowej

Na serwerze ze środowiskiem uruchomieniowym Python3 oraz zkonfigurowaną bazą danych PostgreSQL:

- (Opcjonalnie) Pobrać projekt, jeżeli nie jest się w jego posiadaniu.
  W termianlu z powłoką systemu wykonać:
  `$> git clone https://github.com/mamyxk/5_PIR_Pogoda_Django.git`
- Wejść do katalogu:
  `$> cd 5_PIR_Pogoda_Django`
- Ustawić skrpyt startowy jako wykonywalny:
  `$> chmod +x start.sh`
- Wykonać skrypt startowy:
  `$> ./start.sh`

5 Opis wkładu pracy

Cały zespó brał udział w planowaniu rozwiązań, implementacji i automatyzacji. Wkład każdej z osób był kluczowy i każda z osób była zdeterminowana by dostarczyć projekt na jak najwyższym poziomie.

Pan Dawid Ziobrowski zajmował się implementacją strony internetowej zarówno w aspekcie wizualnym jak i funkcjonalnym.
Pan Kamil Haniszewski zajmował się implementacją Brokera, Consumer oraz konfiguracją sieciową, jak również częściową implementacją strony internetowej.
Pan Mateusz Susłowicz zajmował się implementacją Producera oraz QA całego systemu.
Pan Tymoteusz Trętowicz zajmował się implementacją Producera, tworzeniem dokumentacji jak i częściową implementacją.

6 Podsumowanie

Projekt uważamy za zakończony sukcesem. Mógłby zostać poszerzny o większą ilość zbieraych danych, przez dokładniejsze czujniki, bądź czujniki innego rodzaju. Strona internetowa może zostać rozszerzona o bardziej interaktywne design jak i więcej możliwości prezentacji danych.

7 Literatura

- Dokumentacja RabbitMQ
- Dokumentacja framewrok'a Django
- Wykłady z kursu *Podstawy internetu rzeczy*