

# Stacja pogodowa: Pogodownik 3000

## **Autorzy:**

Haniszewski Kamil,  
Susłowicz Mateusz,  
Trętowicz Tymoteusz,  
Ziobrowski Dawid

## **Prowadzący:**

dr inż. Donat Orski

## Spis treści

<b>1</b>	<b>Wymagania projektowe</b>	<b>3</b>
1.1	Wymagania funkcjonalne . . . . .	3
1.2	Wymagania нефункционалне . . . . .	3
<b>2</b>	<b>Opis architektury systemu</b>	<b>4</b>
<b>3</b>	<b>Opis implementacji zastosowanych rozwiązań</b>	<b>5</b>
3.1	Implementacja pomiaru danych pogodowych . . . . .	6
3.2	Implmentacja konsumenta . . . . .	7
3.3	Implementacja węzła końcowego /add_sensor_log . . . . .	8
<b>4</b>	<b>Opis działania i prezentacja interfejsu</b>	<b>9</b>
4.1	Interfejs . . . . .	9
4.2	Procedura startowa stacji pogodowej . . . . .	11
4.3	Procedura startowa Consumera . . . . .	11
4.4	Procedura startowa strony internetowej . . . . .	12
<b>5</b>	<b>Opis wkładu pracy</b>	<b>12</b>
<b>6</b>	<b>Podsumowanie</b>	<b>13</b>
<b>7</b>	<b>Literatura</b>	<b>13</b>

# 1 Wymagania projektowe

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

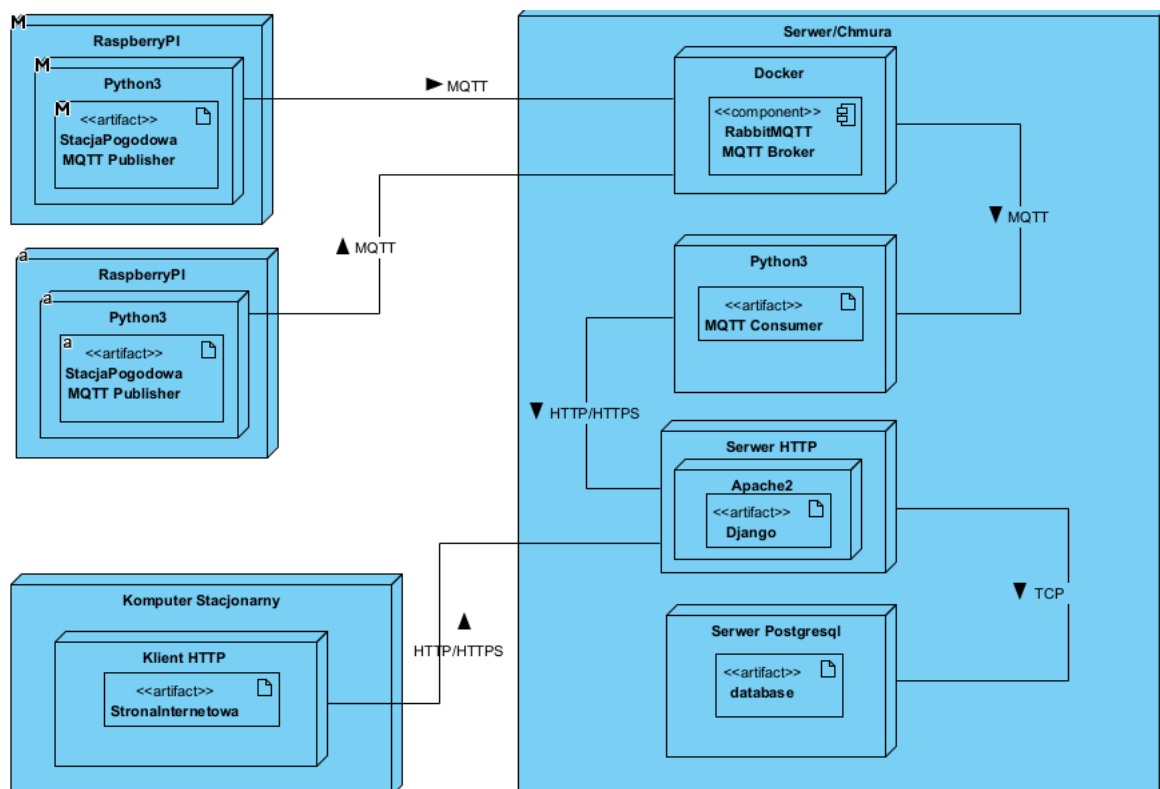
## 1.1 Wymagania funkcjonalne

- Zbieranie, przetwarzanie i zapisywanie danych pogodowych ze stacji (RaspberryPi),
- Zbieranie i wyświetlanie danych dotyczących temperatury,
- Zbieranie i wyświetlanie danych dotyczących wilgotności powietrza,
- Zbieranie i wyświetlanie danych dotyczących wysokości nad poziomem morza (w metrach),
- Zbieranie i wyświetlanie danych dotyczących ciśnienia atmosferycznego,
- Zbieranie i wyświetlanie danych dotyczących temperatury,
- Wyliczanie wartości średnich na podstawie zebranych 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 нефunkcjonalne

- 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 praktykami framework'a Django,

## 2 Opis architektury systemu



Pojedyńcza stacja pogodowa to system RaspberryPi z czujnikiem BME280 oraz programem napisanym w języku Python3 publikującym dane pogodowe. Taka stacja pełni rolę 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 internetowej we frameworku Django oraz implementacje komunikacji z bazą danych.

### 3.1 Implementacja pomiaru danych pogodowych

Poniższy kod jest częścią programu włączonego na stacjach pogodowych (RaspberryPi). Poniższy fragment kodu jest wykonywany we wcześniej ustalonych odstępach czasu, oraz ma za zadanie konfigurację czujnika *BME280*, zebranie danych z czujnika oraz ich publikację.

(5\_PIR\_Pogoda\_Producer/sender.py)

```
1  def bme280():
2      i2c = busio.I2C(board.SCL, board.SDA)
3      bme280 =
4          adafruit_bme280.Adafruit_BME280_I2C(i2c, 0x76)
5
6      # konfiguracja czujnika BME280
7      bme280.sea_level_pressure
8          = 1013.25
9      bme280.standby_period
10         = adafruit_bme280.STANDBY_TC_500
11      bme280.iir_filter
12         = adafruit_bme280.IIR_FILTER_X16
13      bme280.overscan_pressure
14         = adafruit_bme280.OVERSCAN_X16
15      bme280.overscan_humidity
16         = adafruit_bme280.OVERSCAN_X1
17      bme280.overscan_temperature
18         = adafruit_bme280.OVERSCAN_X2
19
20      dic = {}
21      # zbieranie danych
22      dic["sensor"] =
23          int(TERMNINAL_ID) if TERMNINAL_ID else 0
24      dic["temperature"] = bme280.temperature
25      dic["humidity"] = bme280.humidity
26      dic["pressure"] = bme280.pressure
27      dic["altitude"] = bme280.altitude
28      dic["timestamp"] = str(datetime.datetime.now())
29
30      print(json.dumps(dic))
31      # publikacja danych na wcześniej ustalonym kanale
32      channel.basic_publish(
33          exchange="",
34          routing_key="pogoda",
35          body=json.dumps(dic)
36      )
37
```

## 3.2 Implementacja konsumenta

Poniższy kod jest częścią MQTT Consumer'a. Jest on wykonywany za każdym razem gdy konsumer otrzyma wiadomość od brokera. Celem tego kodu jest odczytanie wiadomości, 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)

```
1  def rabbit_callback(ch, method, properties, body):
2      try:
3          # tworzenie adresu serwera HTTP
4          api_addr = config['webservice']['api_address']
5          # odczytanie treści wiadomości
6          recv = json.loads(body)
7          # wysłanie zapytania HTTP POST z wiadomością
8          resp = requests.post(
9              f'{api_addr}/add_sensor_log',
10             data=recv
11         )
12
13         print(f"Received message: {recv}")
14         print(resp.text)
15
16     except json.decoder.JSONDecodeError as err:
17         print(f"An error occurred: {err}")
18
```

### 3.3 Implementacja węzła końcowego /add\_sensor\_log

Poniższy kod przedstawia implementację 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 komunikat sukcesu z kodem 201. W przeciwnym wypadku oznacza to że obiekt serializacji jest nie poprawny więc zwracany jest komunikat błędu z kodem 400.

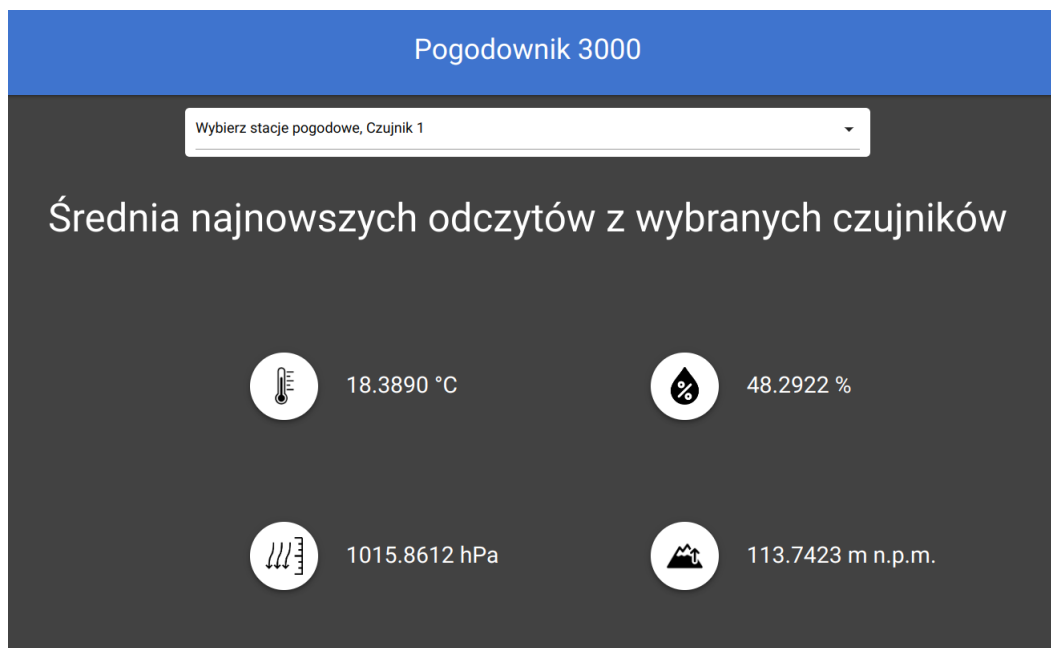
(5\_PIR\_Pogoda\_Django/sensors/views.py#add\_sensor\_log)

```
1  @api_view(['POST'])
2  def add_sensor_log(request):
3      # jeżeli zapytanie jest inne niż HTTP POST...
4      if request.method != 'POST':
5          # ...odpowiedz błędem z kodem 400
6          return Response(
7              serializer.errors,
8              status=status.HTTP_400_BAD_REQUEST
9          )
10
11     serializer = SensorLogSerializer(data=request.data)
12
13     # jeżeli serializacja danych się powiodła...
14     if serializer.is_valid():
15         # zapisz do bazy danych
16         serializer.save()
17         # odpowiedz kodem znaczącym sukces
18         # i zakończ procedure
19         return Response(
20             serializer.data,
21             status=status.HTTP_201_CREATED
22         )
23
24     # jeżeli procedura się nie zakończyła wcześniej
25     # oznacza to że serializacja danych się nie powiodła
26     # ...odpowiedz błędem z kodem 400
27     return Response(
28         serializer.errors,
29         status=status.HTTP_400_BAD_REQUEST
30     )
31
```

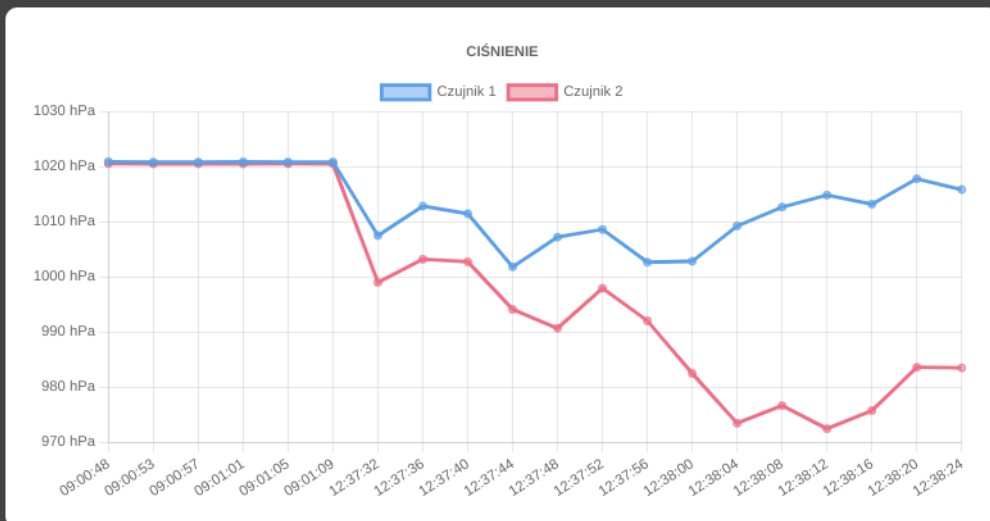
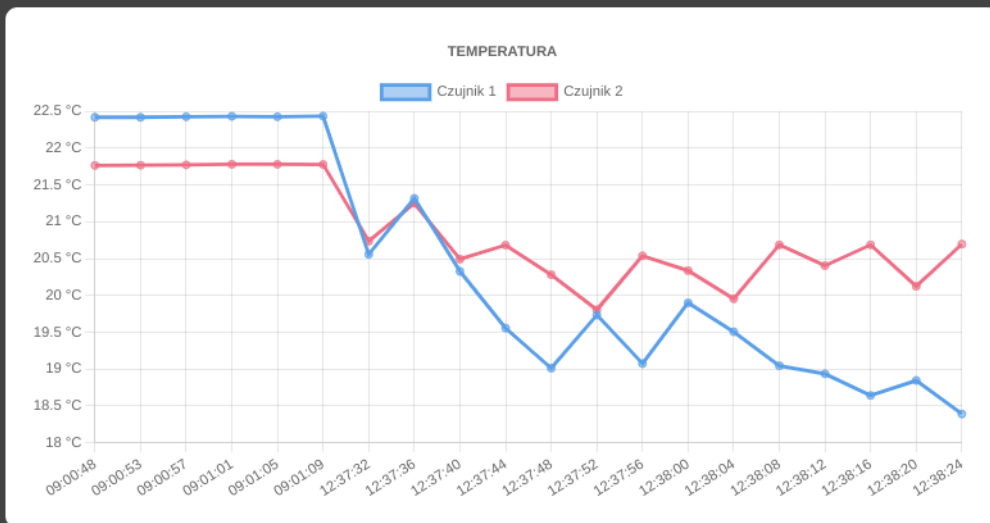


## 4 Opis działania i prezentacja interfejsu

### 4.1 Interfejs



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.



## 4.2 Procedura startowa stacji pogodowej

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

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

## 4.3 Procedura startowa Consumera

Na serwerze ze środowiskiem uruchomieniowym Python3:

- (Opcjonalnie) Pobrać projekt, jeżeli nie jest się w jego posiadaniu.  
W terminalu z powłoką systemu wykonać:  
`$> git clone https://github.com/mamyxk/5_PIR_Pogoda_Consumer.git`
- Wejść do katalogu:  
`$> cd 5_PIR_Pogoda_Consumer`
- Ustawić skrypt startowy jako wykonywalny:  
`$> chmod +x start.sh`
- Stworzyć plik konfiguracyjny `app.conf.json` (Schemata znajduje się w pliku `app.conf.json.sample`).
- Wykonać skrypt startowy:  
`$> ./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 terminalu z powłoką systemu wykonać:  

```
$> git clone https://github.com/mamyxk/5_PIR_Pogoda_Django.git
```
- Wejść do katalogu:  

```
$> cd 5_PIR_Pogoda_Django
```
- Ustawić skrypt startowy jako wykonywalny:  

```
$> chmod +x start.sh
```
- Wykonać skrypt startowy:  

```
$> ./start.sh
```

## 5 Opis wkładu pracy

Cały zespół 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 poszerzony o większą ilość zbieranych 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 framework'a Django
- Wykłady z kursu *Podstawy internetu rzeczy*