AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Technologie Internetu Rzeczy
Eclipse StreamSheets

Marcin Kozub, Adrian Chrobot, Rafał Kamiński

1. Opis projektu

Projekt realizuje Proof of Concept dla Eclipse StreamSheets. Napisaliśmy skrypty dla N sensorów w Pythonie, które wysyłają pseudo-losowe dane przez MQTT Brokera do StreamSheeta. StreamSheet jest w stanie publikować na osobny „podtopic” /state zmianę stanu sensora, co ma później odzwierciedlenie w przesyłanych danych. W StreamSheet stworzyliśmy również wizualizacje dostarczanych danych od sensorów.

2. Opis działania

Utworzyliśmy trzy streamsheety, każdy zbierający dane z innego typu czujników, poprzez nasłuchiwanie na konkretnych topicach i agregujące je do postaci tabeli i wykresów.

Wykorzystaliśmy również funkcjonalność dashboardu, wprowadzając zmianę statusu sensorów, poprzez publikację danych na konkretny topic po kliknięciu przycisku.

3. Użyte technologie

* Eclipse Paho MQTT Python
* Eclipse Streamsheets

4. Przykładowa implementacja skryptu wybranego sensora

Klasa abstrakcyjna dla każdego sensora:

```
class Sensor(ABC):
    def __init__(self, broker: str, port: int, sender_topic: str,
client_id: str):
        self.sender_topic = sender_topic
        self.port = port
        self.client_id = client_id
        self.broker = broker
        self.client = self.__connect_mqtt()

    def __connect_mqtt(self) -> mqtt_client:
        def on_connect(client_id: mqtt_client, userdata, flags, rc:
int):
            if rc == 0:
                print("Connected to MQTT Broker!")
            else:
                print("Failed to connect, return code %d\n", rc)

        client = mqtt_client.Client(client_id=self.client_id)
        client.on_connect = on_connect
        client.connect(self.broker, self.port)
```

```
        return client

    def _check_status(self, status: int):
        if status != 0:
            print(f"Failed to send message to topic
{self.sender_topic}")

    @abstractmethod
    def publish(self, data: str):
        ...

    @abstractmethod
    def subscribe(self, client: mqtt_client):
        ...

    @abstractmethod
    def _get_random_data(self) -> str:
        ...
```

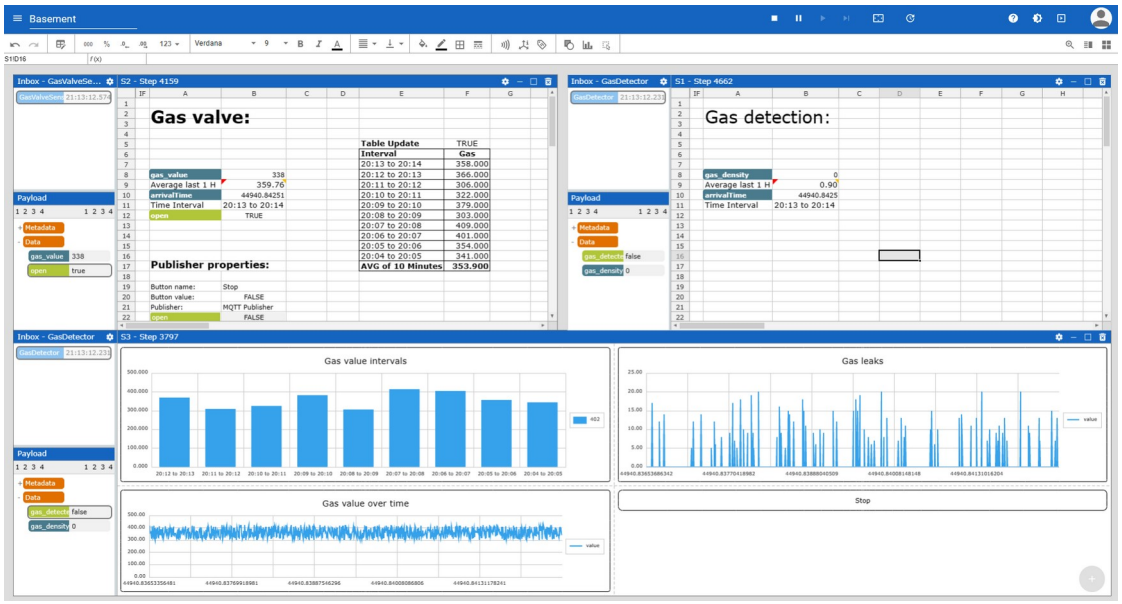
Klasa dla wybranego sensora Light:

```
class Light(Sensor):
    is_turn_on = True
    color_temperatures = ["COOLEST", "COOL", "NEUTRAL", "WARM", "WARMEST"]
    color_temperature: str = "COOLEST"
    brightness: int = 0

    def __init__(self, broker: str, port: int, sender_topic: str,
client_id: str):
        super().__init__(broker, port, sender_topic, client_id)

    def publish(self, data: str):
        self.subscribe(self.client)
        self.client.loop_start()

        while True:
            random_data = self._get_random_data()
            result = self.client.publish(
                self.sender_topic, random_data
            )
            status = result[0]
            self._check_status(status)
            sleep(SLEEP_TIME)

    def subscribe(self, client: mqtt_client):
        def on_message(client, userdata, msg):
            # Should receive JSON of format :
            # {
            # 'turn_on': True/False,
            # 'brightness_value': INTEGER,
            # 'color_value': STRING
            # }

            m = msg.payload.decode("utf-8")
            m = json.loads(m)
```

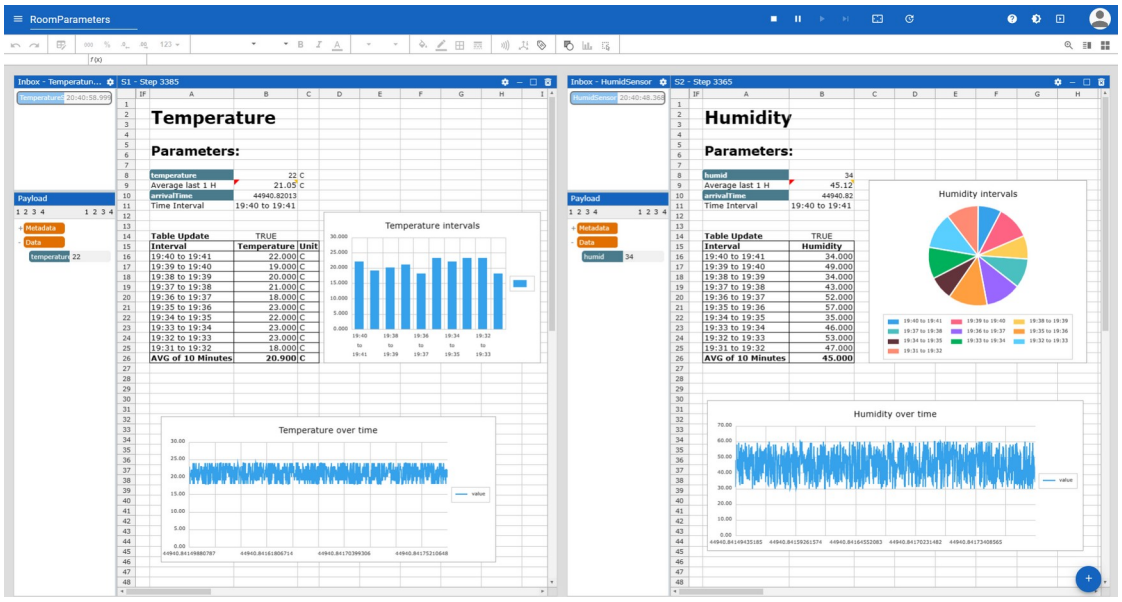
```
            print(f"Received `{m}` from `{msg.topic}` topic")




            try:
                if m["turn_on"] == False:
                    self.is_turn_on = False
                else:
                    self.is_turn_on = True

                try:
                    self.brightness = int(m["brightness_value"])
                except ValueError:
                    self.brightness = 0

                self.color_temperature = m["color_value"]
            except KeyError:
                # inappropriate data sent
                pass

        topic = self.sender_topic + "/state"
        client.subscribe(topic)
        client.on_message = on_message

    def _get_random_data(self) -> str:
        # brightness [%]
        # color_temperature in [coolest, cool, neutral, warm, warmest]

        data = dict()
        data["brightness_value"] = self.brightness
        data["color_value"] = self.color_temperature
        data["turn_on"] = self.is_turn_on
        print(data)
        return json.dumps(data)
```

Każda klasa danego sensora inicjalizuje się łącząc się z brokerem, posiadając przekazany adres ip z portem oraz topic główny, na którym ma wysyłać swoje dane. Cały czas przesyła pseudo-losowe dane odpowiednie dla swojego typu czujnika oraz biorąc pod uwagę aktualny stan, w którym się znajduje. Również non-stop nasłuchuje na swój „podtopic” /state, dzięki czemu jest gotowy na reakcję w każdej chwili, gdy użytkownik wyśle nowy stan dla sensora. W przykładzie powyżej, możemy wysłać stan, w którym turn_on będzie False, to spowoduje wyłączenie światła.

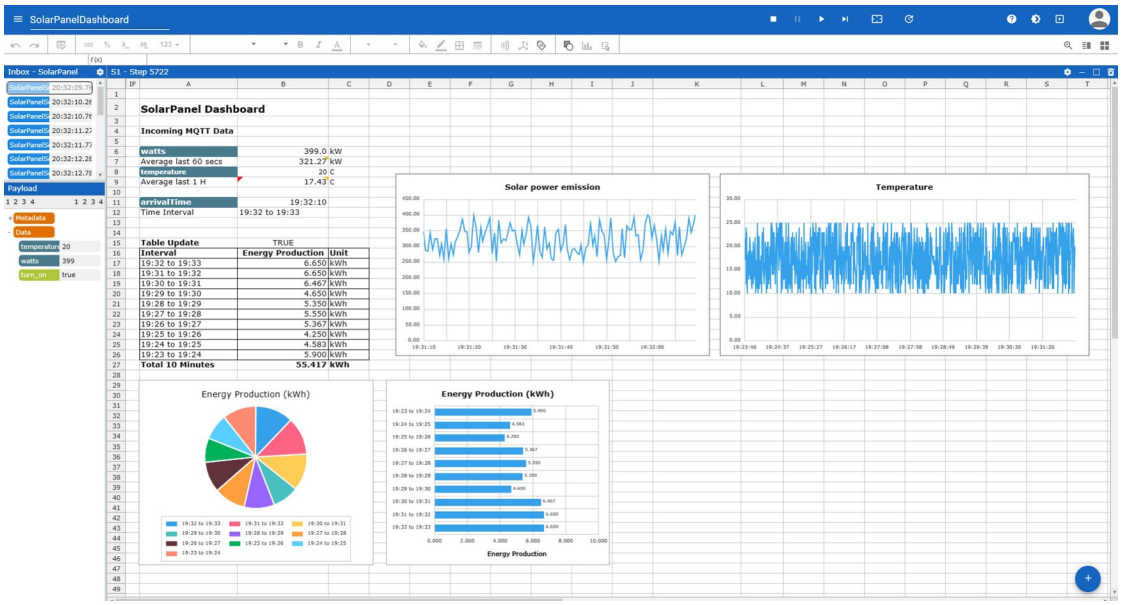
5. Utworzone arkusze



Streamsheet wraz z dashboardem, zbierający dane dotyczące gazu i pozwalający na wyłączanie i włączanie zaworu.



Streamsheet zbierający dane o temperaturze i wilgotności.



Streamsheet zbierający dane o energii wytwarzanej z paneli fotowoltaicznych.

6. Propozycje dalszego rozwoju projektu

Istnieje możliwość łatwego dodawania nowych czujników według schematu oraz utworzenia dashboardów pozwalających na łatwiejsze zarządzanie czujnikami. Dodatkowo można wprowadzić warunkowe zmiany statusów czujnika, poprzez publikację na konkretne topici, gdy dla przykładu średnia ilość wyprodukowanej energii przekroczy jakąś wartość.

7. Użycie komercyjne

Istnieje kilka wersji programu. Darmowa - z której skorzystaliśmy w tym POC – Proffesional, Business oraz Enterprise. Każda kolejna zawiera więcej opcji oraz wsparcia ze strony producenta.

8. Repozytorium ze skryptami

https://github.com/Cozoob/IOT-project