

**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:  
 ...

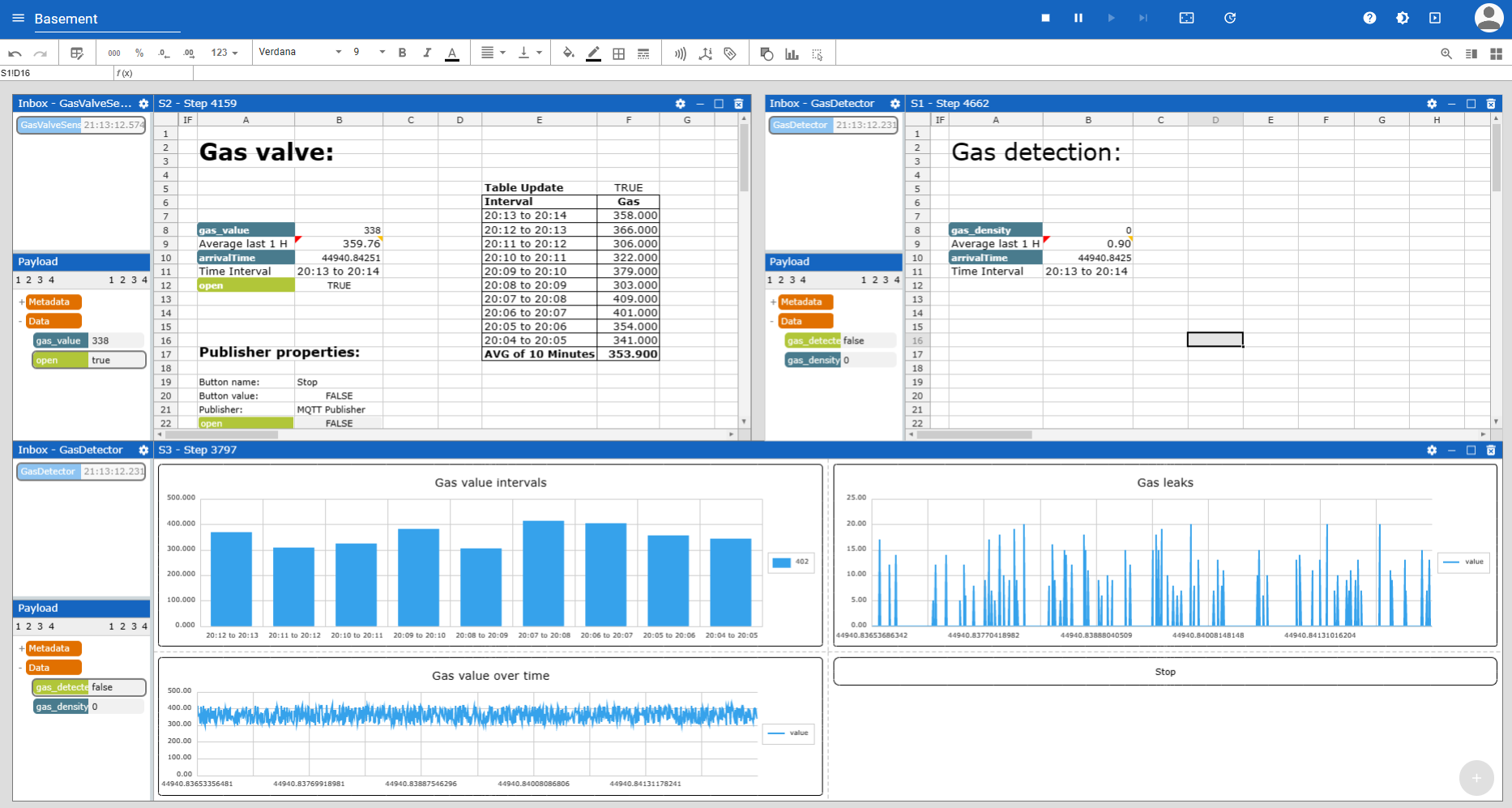
Klasa dla wybranego sensora Light:

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")

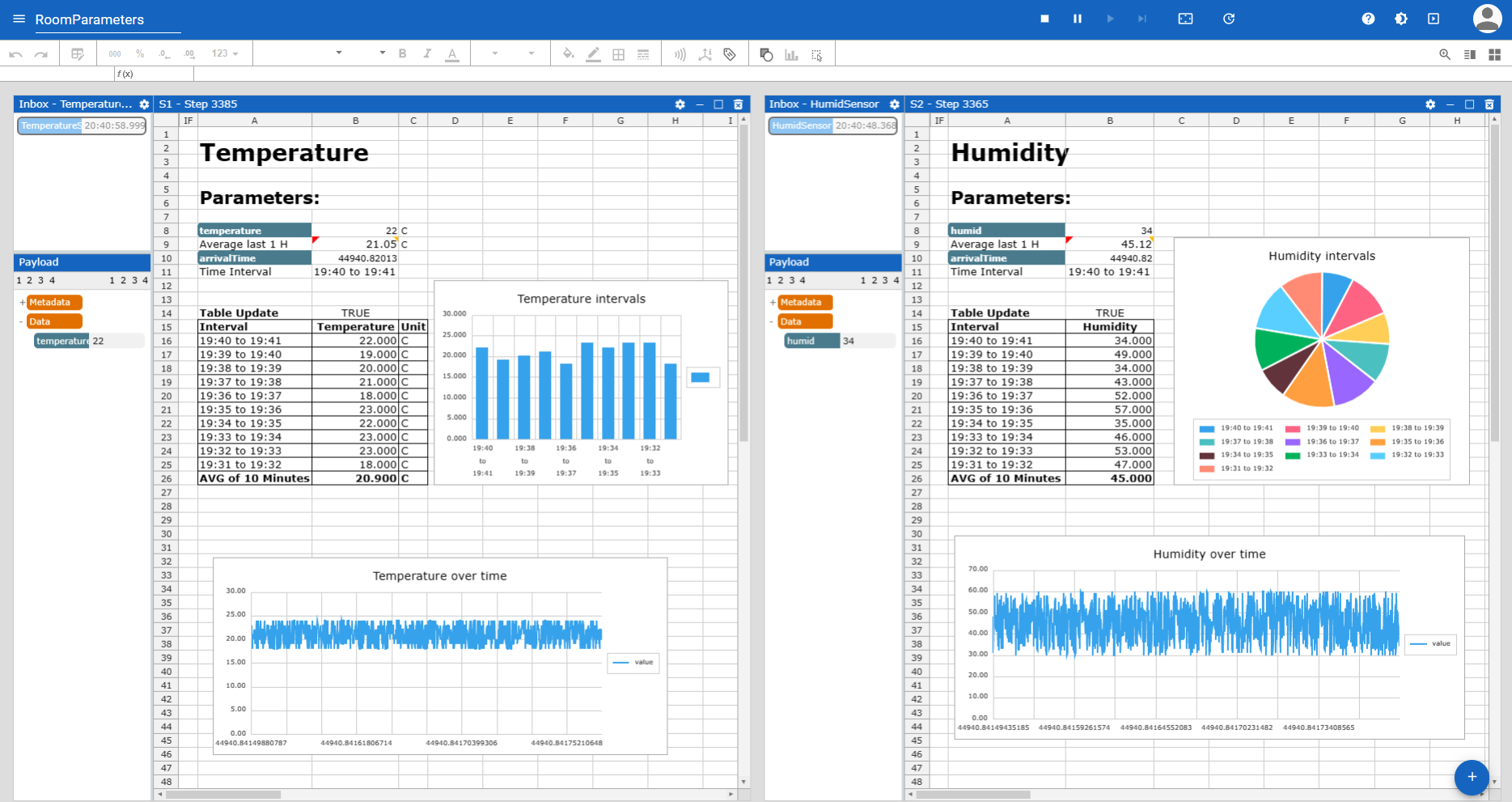
try:  
 if m["turn\_on"] == False:  
 self.is\_turn\_on = False  
 else:  
 self.is\_turn\_on = True  
  
 try:  
 self.brightness = int(m["brightness\_value"])  
 except ValueError:  
 self.brightness = 0  
  
 self.color\_temperature = m["color\_value"]  
 except KeyError:  
 # inappropriate data sent  
 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.

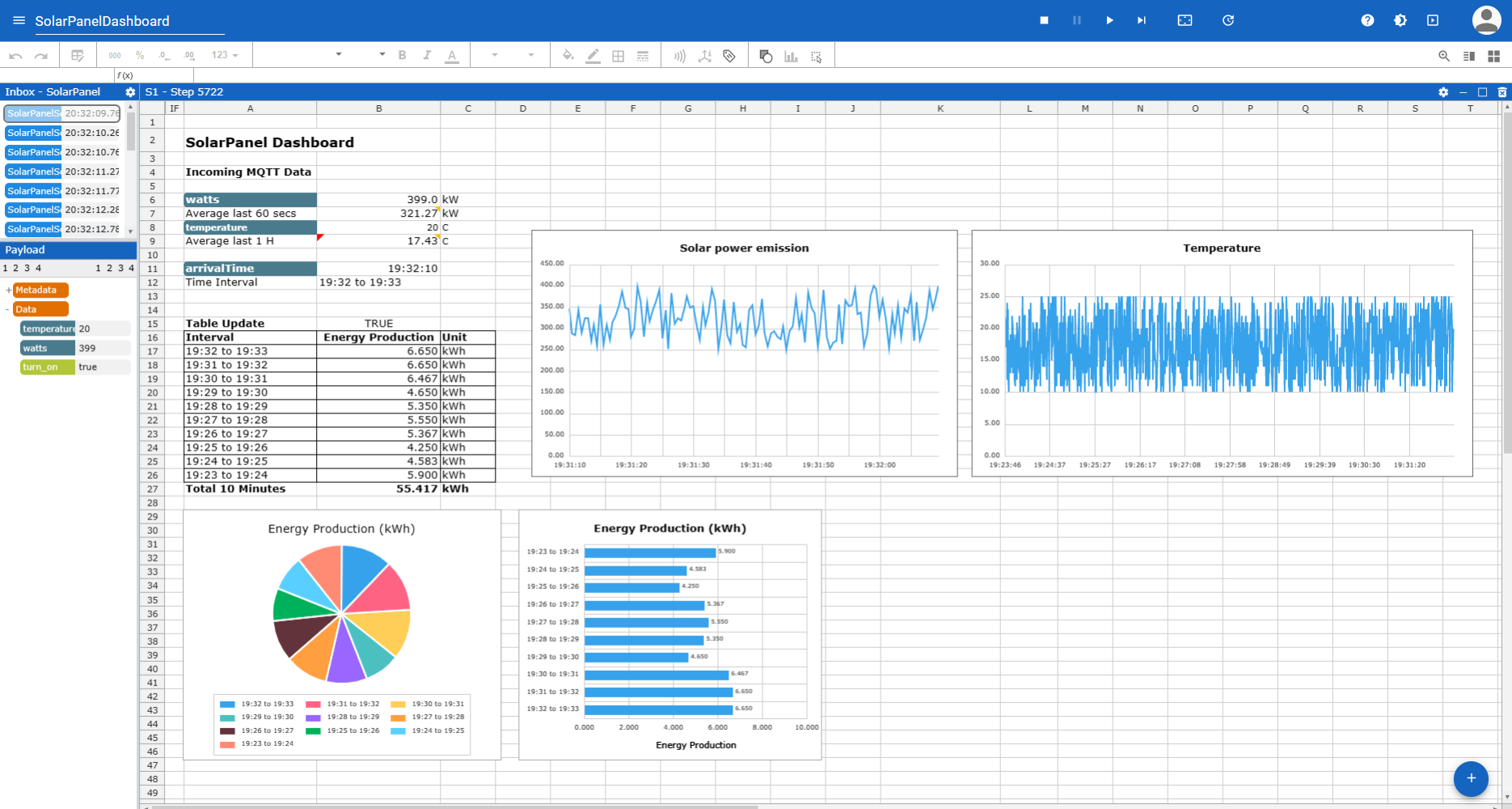
1. **Utworzone arkusze**



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.

1. **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.
2. **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.

1. **Repozytorium ze skryptami**<https://github.com/Cozoob/IOT-project>