

# **LAPORAN PRAKTIKUM INTERNET OF THINGS**

## **MQQT-SIMULATOR**



**Valencia L Tobing  
11323046  
D3 Teknologi Informasi**

**INSTITUT TEKNOLOGI DEL  
FAKULTAS VOKASI  
2024/2025**

## Penjelasan Code :

### 1. Main.py

- Fungsi default\_settings

```
mqtt-simulator > main.py > ...
1 import argparse
2 from pathlib import Path
3 from simulator import Simulator
4
5 def default_settings():
6     base_folder = Path(__file__).resolve().parent.parent
7     settings_file = base_folder / 'config/settings.json'
8     return settings_file
```

Pada bagian ini, kita mengimpor modul yang diperlukan untuk menjalankan program. `argparse` digunakan untuk mengelola argumen dari baris perintah, memungkinkan pengguna memberikan input saat program dijalankan. Fungsi ini juga menetapkan lokasi default untuk file pengaturan yang digunakan oleh simulator. Pertama, kita mendapatkan jalur folder dasar dengan `Path(__file__).resolve().parent.parent`, yang mengarah dua tingkat ke atas dari lokasi file saat ini. Selanjutnya, jalur tersebut digabungkan dengan `'config/settings.json'` untuk membentuk jalur lengkap ke file pengaturan. Fungsi ini mengembalikan jalur file tersebut sebagai objek `Path`.

- File Validation Function

```
def is_valid_file(parser, arg):
    settings_file = Path(arg)
    if not settings_file.is_file():
        return parser.error(f"argument -f/--file: can't open '{arg}'")
    return settings_file
```

Fungsi ini digunakan untuk memeriksa apakah file yang diberikan sebagai argumen benar-benar ada dan dapat diakses. Parameter `parser` adalah objek `ArgumentParser`, sedangkan `arg` adalah argumen yang dimasukkan oleh pengguna. Di dalam fungsi, argumen diubah menjadi objek `Path` dan diperiksa apakah itu adalah file yang valid menggunakan metode `is_file()`. Jika file tidak ditemukan, fungsi ini akan menampilkan pesan kesalahan dan menghentikan eksekusi program. Jika file valid, fungsi akan mengembalikannya.

- Argument Parser Setup

```
parser = argparse.ArgumentParser()
parser.add_argument("-f", "--file", dest='settings_file', type=lambda x: is_valid_file(parser, x), help='settings file', default=default_settings())
args = parser.parse_args()
```

Pada bagian ini, kita membuat objek `ArgumentParser` untuk mengatur argumen yang diterima oleh program. Kita menambahkan argumen `-f` atau `--file`, yang digunakan untuk menerima jalur file pengaturan dari pengguna. Dengan `dest='settings_file'`, nilai argumen ini disimpan dalam atribut `settings_file` pada objek `args`. Tipe argumen ditentukan menggunakan `lambda` yang memanggil fungsi `is_valid_file`, sehingga validasi dilakukan saat argumen diterima. Jika tidak ada argumen yang diberikan, fungsi `default_settings()` akan dipanggil untuk menggunakan jalur file pengaturan default.

- Argument Parsing and Simulator Execution

```
simulator = Simulator(args.settings_file)
simulator.run()
```

Setelah semua argumen ditentukan, kita memanggil `parser.parse_args()` untuk memproses argumen yang dimasukkan pengguna di baris perintah. Hasilnya disimpan dalam variabel `args`. Selanjutnya, kita membuat instance dari kelas `Simulator`, dengan mengoper file pengaturan yang diperoleh dari `args.settings_file`. Terakhir, kita memanggil metode `run()` pada objek `simulator` untuk memulai simulasi berdasarkan pengaturan yang telah diberikan.

## 2. Simulator.py

- Constructor (`__init__` Method)

```
class Simulator:
    def __init__(self, settings_file):
        self.default_client_settings = ClientSettings(
            clean=True,
            retain=False,
            qos=2,
            time_interval=10
        )
        self.topics = self.load_topics(settings_file)
```

Metode ini adalah konstruktor untuk kelas `Simulator`. Ia menerima satu argumen, `settings_file`, yang merupakan jalur ke file pengaturan. Di dalam konstruktor, kita menginisialisasi `default_client_settings` dengan objek `ClientSettings` yang memiliki nilai default untuk atribut seperti `clean`, `retain`, `qos`, dan `time_interval`. Selanjutnya, kita memanggil metode `load_topics` untuk memuat dan menginisialisasi daftar topik berdasarkan pengaturan dalam file JSON yang diberikan, dan menyimpannya dalam atribut `self.topics`.

- Read Client Settings Method

```
def read_client_settings(self, settings_dict: dict, default: ClientSettings):
    return ClientSettings(
        clean=settings_dict.get('CLEAN_SESSION', default.clean),
        retain=settings_dict.get('RETAIN', default.retain),
        qos=settings_dict.get('QOS', default.qos),
        time_interval=settings_dict.get('TIME_INTERVAL', default.time_interval)
    )
```

Metode ini bertugas untuk membaca pengaturan klien dari sebuah dictionary. Ia menerima dua argumen: `settings_dict`, yang merupakan dictionary berisi pengaturan, dan `default`, yang adalah objek `ClientSettings` dengan nilai default. Metode ini mengembalikan objek `ClientSettings` baru yang diisi dengan nilai dari `settings_dict`, menggunakan nilai default jika pengaturan tertentu tidak ada dalam dictionary. Ini memberikan fleksibilitas dalam mengonfigurasi pengaturan klien untuk setiap topik.

- Load Topics Method

```
def load_topics(self, settings_file):
    topics = []
    with open(settings_file) as json_file:
        config = json.load(json_file)
        broker_settings = BrokerSettings(
            uri=config.get('BROKER_URI', 'localhost'),
            port=config.get('BROKER_PORT', 1883),
            protocol=config.get('PROTOCOL_VERSION', 4) or None
        )
        broker_client_settings = self.read_client_settings(config, default=broker_settings)
        # read each registered topic
        for topic in config['TOPICS']:
            topic_data = topic['DATA']
            topic_payload_root = topic.get('PAYLOAD_ROOT', {})
            topic_client_settings = self.read_client_settings(topic, default=broker_client_settings)
            if topic['TYPE'] == 'single':
                # create single topic with name = {topic['NAME']}
                topic_url = topic['PREFIX']
                topics.append(topic(broker_settings, topic_url, topic_data, topic_payload_root, topic_client_settings))
            elif topic['TYPE'] == 'multiple':
                # create multiple topics with names = {topic['NAME']} + {id}
                for id in range(topic['NAME_START'], topic['NAME_END'] + 1):
                    topic_url = topic['PREFIX'] + '/' + str(id)
                    topics.append(topic(broker_settings, topic_url, topic_data, topic_payload_root, topic_client_settings))
            elif topic['TYPE'] == 'list':
                # create multiple topics with names = {topic['NAME']} + {id}
                for id in range(topic['NAME_START'], topic['NAME_END'] + 1):
                    topic_url = topic['PREFIX'] + '/' + str(id)
                    topics.append(topic(broker_settings, topic_url, topic_data, topic_payload_root, topic_client_settings))
    return topics
```

Metode ini bertanggung jawab untuk memuat dan mengonfigurasi topik dari file pengaturan JSON. Pertama, kita membuka file pengaturan dan memuat isinya ke dalam dictionary config. Selanjutnya, kita mengonfigurasi broker\_settings dengan mengambil URL, port, dan versi protokol dari dictionary tersebut, menggunakan nilai default jika diperlukan. Kemudian, kita membaca pengaturan klien broker dengan metode read\_client\_settings.

Setelah itu, kita iterasi melalui setiap topik yang terdaftar dalam config['TOPICS']. Untuk setiap topik, kita memeriksa jenisnya (single, multiple, atau list) dan membuat URL topik sesuai format yang ditentukan. Objek Topic baru dibuat dengan pengaturan broker, URL topik, data topik, payload root, dan pengaturan klien topik, sebelum akhirnya ditambahkan ke daftar topics. Metode ini mengembalikan daftar semua topik yang telah dikonfigurasi.

- Run Method

```
def run(self):
    for topic in self.topics:
        print(f'Starting: {topic.topic_url} ...')
        topic.start()
    for topic in self.topics:
        # workaround for Python 3.12
        topic.join()
```

Metode ini bertugas untuk menjalankan simulasi dengan memulai setiap topik yang telah dimuat. Kita iterasi melalui semua topik dalam `self.topics`, mencetak pesan yang menunjukkan bahwa topik sedang dimulai, dan memanggil metode `start()` pada setiap objek `Topic`. Setelah semua topik dimulai, kita melakukan `join` pada setiap topik untuk memastikan bahwa eksekusi utama menunggu hingga semua topik selesai beroperasi. Ini memberikan cara yang teratur untuk mengelola siklus hidup topik dalam simulasi.

- Stop Method

```
def stop(self):
    for topic in self.topics:
        print(f'Stopping: {topic.topic_url} ...')
        topic.stop()
```

Metode ini bertujuan untuk menghentikan semua topik yang sedang berjalan. Kita iterasi melalui setiap topik dalam `self.topics`, mencetak pesan yang menunjukkan bahwa topik sedang dihentikan, dan kemudian memanggil metode `stop()` pada setiap objek `Topic`. Metode ini menyediakan cara untuk membersihkan dan menghentikan semua aktivitas terkait topik saat simulasi selesai, memastikan bahwa semua sumber daya dilepaskan dengan benar.

### 3. Topic.py

- Importing Modules

```
mqtt-simulator > topic.py > topic
1 import time
2 import json
3 import threading
4 import paho.mqtt.client as mqtt
5 from data_classes import BrokerSettings, ClientSettings
6 from topic_data import TopicDataNumber, TopicDataBool, TopicDataRawValue, TopicDataMathExpression
```

Di bagian ini, kita mengimpor modul dan kelas yang diperlukan untuk kelas `Topic`. Modul `time` digunakan untuk menunda eksekusi dalam loop. Modul `json` digunakan untuk mengonversi data ke format JSON sebelum dipublikasikan. `threading` memungkinkan kelas ini berjalan sebagai thread terpisah. `paho.mqtt.client` adalah pustaka MQTT yang digunakan untuk mengelola koneksi dan komunikasi dengan broker MQTT. Kelas `BrokerSettings` dan `ClientSettings` diimpor dari modul `data_classes`, sementara berbagai kelas data topik diimpor dari modul `topic_data`, yang menangani jenis data spesifik yang akan diterbitkan.

- Constructor (`__init__` Method)

```
class Topic(threading.Thread):
    def __init__(self, broker_settings: BrokerSettings, topic_url: str, topic_data: List[object], topic_payload_root: object, client_settings: ClientSettings):
        threading.Thread.__init__(self)

        self.broker_settings = broker_settings

        self.topic_url = topic_url
        self.topic_data = self.load_topic_data(topic_data)
        self.topic_payload_root = topic_payload_root

        self.client_settings = client_settings

        self.loop = False
        self.client = None
        self.payload = None
```

Metode ini adalah konstruktor untuk kelas `Topic`. Ia menerima beberapa argumen, termasuk pengaturan broker, URL topik, data topik, payload root, dan pengaturan klien. Dalam konstruktor, kita memanggil `__init__` dari kelas `Thread` untuk memulai thread. Selanjutnya, kita menyimpan argumen yang diterima dalam atribut instance. Variabel `loop` diinisialisasi sebagai `False` dan akan digunakan untuk mengontrol eksekusi thread. `client` dan `payload` diinisialisasi sebagai `None`, yang akan diisi saat koneksi ke broker dibuat dan payload siap.

- Load Topic Data Method

```
def load_topic_data(self, topic_data_object):
    topic_data = []
    for data in topic_data_object:
        data_type = data['TYPE']
        if data_type == 'int' or data_type == 'float':
            topic_data.append(TopicDataNumber(data))
        elif data_type == 'bool':
            topic_data.append(TopicDataBool(data))
        elif data_type == 'raw_values':
            topic_data.append(TopicDataRawValue(data))
        elif data_type == 'math_expression':
            topic_data.append(TopicDataMathExpression(data))
        else:
            raise NameError(f"Data TYPE '{data_type}' is unknown")
    return topic_data
```

Metode ini bertanggung jawab untuk memuat data topik dari objek yang diterima. Ia iterasi melalui setiap item dalam topic\_data\_object, memeriksa tipe data, dan membuat objek yang sesuai berdasarkan tipe tersebut. Misalnya, jika tipe adalah int atau float, objek TopicDataNumber dibuat. Jika tipe adalah bool, objek TopicDataBool dibuat, dan seterusnya. Jika tipe data tidak dikenali, akan muncul kesalahan. Metode ini mengembalikan daftar objek data topik yang telah dimuat.

- Connect Method

```
def connect(self):
    self.loop = True
    clean_session = None if self.broker_settings.protocol == mqtt.MQTTv5 else self.client_settings.clean
    self.client = mqtt.Client(self.topic_url, protocol=self.broker_settings.protocol, clean_session=clean_session)
    self.client.on_publish = self.on_publish
    self.client.connect(self.broker_settings.url, self.broker_settings.port)
    self.client.loop_start()
```

Metode ini digunakan untuk menghubungkan ke broker MQTT. Pertama, kita mengatur self.loop menjadi True untuk memungkinkan eksekusi loop dalam metode run. Jika protokol broker adalah MQTT v5, sesi bersih (clean session) diatur menjadi None; jika tidak, nilainya diambil dari pengaturan klien. Selanjutnya, kita membuat instance mqtt.Client dan mengatur callback on\_publish untuk menangani peristiwa penerbitan. Setelah itu, kita melakukan koneksi ke broker menggunakan URL dan port yang ditentukan, lalu memulai loop MQTT dengan loop\_start().

- Disconnect Method

```
def disconnect(self):
    self.loop = False
    self.client.loop_stop()
    self.client.disconnect()
```

Metode ini digunakan untuk memutuskan koneksi dari broker MQTT. Kita mengatur self.loop menjadi False untuk menghentikan eksekusi loop dalam metode run. Selanjutnya, kita menghentikan loop dengan loop\_stop() dan memutuskan koneksi dengan broker menggunakan disconnect().

- Generate Payload Method

```
def generate_payload(self):
    payload = {}
    payload.update(self.topic_payload_root)
    has_data_active = False
    for data in self.topic_data:
        if data.is_active:
            has_data_active = True
            payload[data.name] = data.generate_value()
    if not has_data_active:
        self.disconnect()
        return
    return payload
```

Metode ini bertanggung jawab untuk menghasilkan payload yang akan diterbitkan. Pertama, ia membuat dictionary kosong untuk payload dan memperbarui dengan `topic_payload_root`. Kemudian, ia iterasi melalui setiap objek data dalam `self.topic_data`, memeriksa apakah data tersebut aktif. Jika aktif, ia menyimpan nilai yang dihasilkan oleh metode `generate_value()` dari objek data ke dalam payload. Jika tidak ada data yang aktif, metode ini akan memutuskan koneksi dengan broker dan mengembalikan `None`. Jika ada data aktif, ia mengembalikan payload yang telah disusun.



#### 4. Utils.py

- Fungsi `should_run_with_probability`

```
mqtt-simulator > utils.py > ...  
1  import random  
2  
3  def should_run_with_probability(probability: float):  
4      random_number = random.random()  
5      return random_number < probability  
6
```

##### **TujuanFungsi:**

Fungsi ini dirancang untuk menentukan apakah suatu proses atau tindakan harus dijalankan berdasarkan probabilitas yang ditentukan. Ini berguna dalam situasi di mana Anda ingin mengontrol frekuensi eksekusi suatu tindakan secara acak, misalnya dalam simulasi atau pengambilan keputusan berbasis probabilitas.

##### **Parameter:**

`probability`: Ini adalah parameter bertipe float yang mewakili probabilitas (dalam rentang 0.0 hingga 1.0) bahwa fungsi akan mengembalikan True. Misalnya, jika `probability` adalah 0.7, maka ada 70% kemungkinan fungsi akan menghasilkan True.

##### **MenghasilkanAngkaAcak:**

`random_number = random.random()`: Di sini, fungsi `random.random()` dari modul `random` digunakan untuk menghasilkan angka acak antara 0.0 (inklusif) dan 1.0 (eksklusif). Angka ini akan digunakan untuk menentukan apakah tindakan harus dilakukan berdasarkan probabilitas yang diberikan.

##### **LogikaPengembalian:**

`return random_number < probability`: Fungsi membandingkan angka acak yang dihasilkan dengan parameter `probability`. Jika `random_number` lebih kecil dari `probability`, fungsi mengembalikan True; jika tidak, mengembalikan False. Dengan cara ini, probabilitas tindakan dijalankan sesuai dengan nilai yang diberikan.



### Cara Menjalankan Kode :

1. Pertama-tama jalankan command **python -m venv venv** pada cmd di vscode.ini digunakan untuk membuat sebuah lingkungan virtual Python (virtual environment) baru dengan nama venv.

```
(.venv) D:\PERKULIAHAN\Semester 3\IOT\mqtt-simulator-master\mqtt-simulator>python -m venv venv
```

2. Lalu masukkan command **venv\Scripts\activate** untuk mengaktifkan lingkungan virtual Python yang sebelumnya telah dibuat.

```
(.venv) D:\PERKULIAHAN\Semester 3\IOT\mqtt-simulator-master\mqtt-simulator>venv\Scripts\activate
```

3. Kemudian masukkan command **pip install -r requirements.txt** untuk untuk menginstal semua paket Python yang tercantum dalam file requirements.txt.

```
(.venv) D:\PERKULIAHAN\Semester 3\IOT\mqtt-simulator-master\mqtt-simulator>venv\Scripts\activate
(.venv) D:\PERKULIAHAN\Semester 3\IOT\mqtt-simulator-master\mqtt-simulator>pip install -r requirements.txt
Collecting paho-mqtt==1.3.0 (from -r requirements.txt (line 1))
  Using cached paho-mqtt-1.3.0.tar.gz (50 kB)
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Building wheels for collected packages: paho-mqtt
  Building wheel for paho-mqtt (pyproject.toml) ... done
  Created wheel for paho-mqtt: filename=paho_mqtt-1.3.0-py3-none-any.whl size=54007 sha256=1341a28b1f41121d8a9c088f0e0800c15b922166118f1107971
  Stored in directory: c:\users\user\appdata\local\pip\cache\wheels\70\76\01\76d44c9ed0130a06f6a0a00fffa2e088471db75d05ff95
Successfully built paho-mqtt
Installing collected packages: paho-mqtt
Successfully installed paho-mqtt-1.3.0
Python 3.9.0 (tags/v3.9.0:mb96fa6, May 8, 2021) available
```

4. Masukkan command **py mqtt-simulator/main.py** untuk untuk menjalankan skrip Python yang terletak di dalam direktori mqtt-simulator dengan nama file main.py.

```
(venv) D:\PERKULIAHAN\Semester 3\IOT\mqtt-simulator-master\mqtt-simulator>py mqtt-simulator/main.py
Starting: lamp/1 ...
Starting: lamp/2 ...
Starting: air_quality ...
Starting: temperature/roof ...
Starting: temperature/basement ...
Starting: freezer ...
Starting: location ...
```