

LAPORAN PRAKTIKUM INTERNET OF THINGS



11323052

Lasro Pesta Natalia Tamba

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

1. def __init__

```
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)
```

Kode di atas merupakan bagian dari fungsi yang berjalan otomatis saat kita membuat objek dari kelas **simulator**. Dimana ini berfungsi untuk menyimpan pengaturan dasar dan daftar topik.

Pertama, ada pengaturan default klien MQTT yang disimpan di **self.default_client_settings**.

- **clean=True** artinya koneksi baru akan dimulai dari nol tanpa menyimpan status lama.
- **retain=False** berarti pesan tidak akan disimpan di broker setelah dikirim.
- **qos=2** memastikan pesan dikirim dan diterima tepat satu kali tanpa duplikasi.
- **time_interval=10** adalah waktu tunggu default dalam hitungan detik.

Kemudian, fungsi **load_topics** dipanggil dengan memasukkan nama file pengaturan (**settings_file**). Fungsi ini membaca file JSON tersebut, memuat data seperti pengaturan broker dan daftar topik, lalu mengembalikan daftar topik yang sudah siap digunakan. Daftar topik ini disimpan ke dalam atribut **self.topics**.

2. def read_client_settings

```
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)
    )
```

Kode di atas merupakan fungsi yang digunakan untuk membaca pengaturan klien dari sebuah dictionary (**settings_dict**). Jika ada pengaturan tertentu yang tidak ditemukan dalam dictionary, fungsi ini akan menggunakan nilai default yang sudah ditentukan sebelumnya melalui parameter **default**.

Fungsi ini membuat dan mengembalikan sebuah objek **ClientSettings**. Setiap atribut dalam objek **ClientSettings** diisi menggunakan data dari **settings_dict**, misalnya:

- **CLEAN_SESSION** diambil dari dictionary untuk menentukan apakah sesi bersih digunakan. Kalau tidak ada, akan menggunakan nilai default dari **default.clean**.
- Hal yang sama berlaku untuk atribut lain seperti **RETAIN**, **QOS**, dan **TIME_INTERVAL**.

Dengan cara ini, fungsi memastikan bahwa setiap atribut memiliki nilai, baik dari pengaturan yang diberikan pengguna (jika ada) atau dari nilai bawaan. Hasil akhirnya adalah sebuah objek **ClientSettings** yang siap digunakan dengan pengaturan yang sesuai.

3. def load_topics

```
def load_topics(self, settings_file):
    topics = []
    with open(settings_file) as json_file:
        config = json.load(json_file)
        broker_settings = BrokerSettings(
            url=config.get('BROKER_URL', 'localhost'),
            port=config.get('BROKER_PORT', 1883),
            protocol=config.get('PROTOCOL_VERSION', 4) # mqtt.MQTTv311
        )
        broker_client_settings = self.read_client_settings(config, default=self.default_client_settings)
        # read each configured 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 format: /{PREFIX}
                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 format: /{PREFIX}/{id}
                for id in range(topic['RANGE_START'], topic['RANGE_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 format: /{PREFIX}/{item}
                for item in topic['LIST']:
                    topic_url = topic['PREFIX'] + '/' + str(item)
                    topics.append(Topic(broker_settings, topic_url, topic_data, topic_payload_root, topic_client_settings))
    return topics
```

Kode di atas memiliki fungsi yang bertujuan untuk membaca file pengaturan yang berisi daftar topik dan membuat objek-objek topik berdasarkan pengaturan tersebut.

Pertama, fungsi ini membuka dan membaca file JSON yang diberikan melalui parameter **settings_file**. Kemudian setelah file dibaca, isinya akan dimasukkan ke dalam variabel **config**. Lalu di dalam **config**, terdapat informasi tentang pengaturan broker (seperti URL, port, dan versi protocol) yang digunakan untuk membuat objek **BrokerSettings**. Jika informasi ini tidak ada, fungsi akan menggunakan nilai default (misalnya, URL default adalah 'localhost').

Setelah itu, fungsi ini memanggil **read_client_settings** untuk mengambil pengaturan klien broker yang akan digunakan di setiap topik. Pengaturan ini adalah pengaturan

dasar yang akan diterapkan pada semua topik, kecuali jika diubah di masing-masing topik.

Selanjutnya, fungsi ini mengakses daftar **TOPICS** dari file JSON. Setiap topik dalam daftar ini akan diproses satu per satu. Setiap topik memiliki beberapa informasi penting seperti tipe topik (**single**, **multiple**, atau **list**), data topik (**DATA**), dan root payload (**PAYLOAD_ROOT**).

- Jika tipe topik adalah **single**, fungsi akan membuat satu topik dengan format URL seperti **/PREFIX**.
- Jika tipe topik adalah **multiple**, maka fungsi akan membuat beberapa topik berdasarkan rentang ID yang ditentukan (dari **RANGE_START** hingga **RANGE_END**).
- Jika tipe topik adalah **list**, fungsi akan membuat beberapa topik berdasarkan daftar item yang ada di **LIST**.

Untuk setiap topik yang dibuat, fungsi ini membuat objek **Topic** dengan mengirimkan pengaturan broker, URL topik, data topik, root payload, dan pengaturan klien untuk topik tersebut.

Setelah semua topik diproses, fungsi ini mengembalikan daftar **topics** yang berisi semua objek topik yang telah dibuat.

4. def run

```
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()
```

Fungsi kode **run** ini digunakan untuk memulai dan menjalankan semua topik yang sudah dimuat dalam daftar **self.topics**.

Pertama, fungsi ini akan memproses setiap topik dalam daftar **self.topics**. Untuk setiap topik, akan ditampilkan pesan yang memberitahukan bahwa topik tersebut sedang dimulai, lalu fungsi **start()** dipanggil untuk memulai topik. Fungsi **start()** biasanya digunakan untuk menjalankan topik di thread terpisah, misalnya untuk mengirimkan atau menerima pesan dari broker.

Setelah itu, fungsi ini kembali melalui setiap topik dan memanggil **join()** pada setiap topik. Fungsi **join()** digunakan untuk memastikan bahwa program utama menunggu hingga semua topik selesai dijalankan sebelum melanjutkan ke langkah berikutnya. Ini penting jika topik-topik tersebut berjalan di thread terpisah karena **join()** memastikan proses utama tidak melanjutkan sampai semua topik selesai bekerja.

Ada catatan dalam kode yang mengatakan bahwa penggunaan **join()** ini adalah **workaround untuk Python 3.12**, artinya ada masalah atau perubahan tertentu di versi Python tersebut yang membuat penggunaan **join()** diperlukan untuk agar semuanya berjalan dengan benar.

5. def stop

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

Pada kode fungsi **stop** ini bertujuan untuk menghentikan semua topik yang telah dijalankan sebelumnya.

Di dalam fungsi, pertama-tama dilakukan iterasi (perulangan) melalui setiap topik yang ada di dalam daftar **self.topics**. Untuk setiap topik, akan dicetak pesan yang memberitahukan bahwa topik tersebut sedang dihentikan, dengan mencantumkan URL topik (**topic.topic_url**). Setelah itu, fungsi **stop()** dipanggil pada setiap topik.

Fungsi **stop()** ini kemungkinan besar digunakan untuk menghentikan aktivitas yang sedang dilakukan oleh topik, seperti menghentikan pengiriman pesan atau berhenti berlangganan dari topik di broker.

Dengan kata lain, fungsi **stop** memastikan bahwa semua topik yang aktif dihentikan dengan benar.

Untuk menjalankan MQTT simulator kita harus melakukan hal berikut:

1. Membuat virtual environment di python, seperti berikut:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB
PS D:\KULIAH\Sem 3\IoT\w12\mqtt-simulator-master> python -m venv venv
```

2. Setelah itu kita aktifkan virtual enviroentemnya, seperti berikut:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB
PS D:\KULIAH\Sem 3\IoT\w12\mqtt-simulator-master> source venv/bin/activate
```

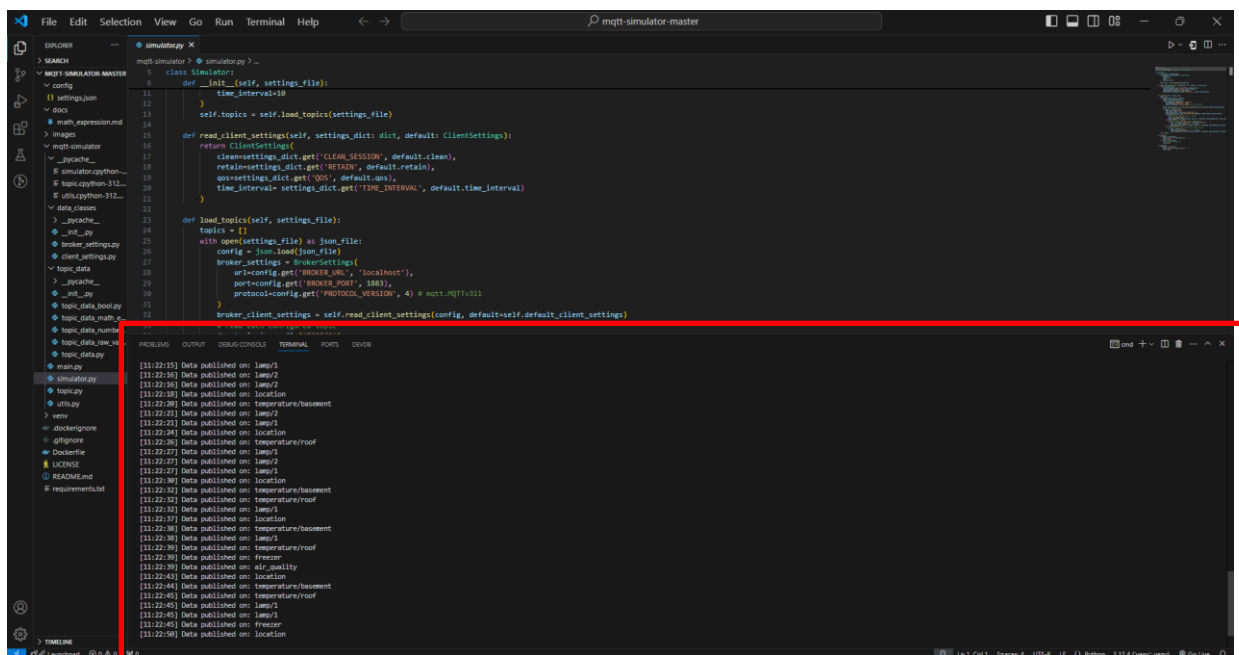
3. Setelah itu download requirements.txt, sepert berikut:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB
PS D:\KULIAH\Sem 3\IoT\w12\mqtt-simulator-master> pip install -r requirements.txt
```

4. Lalu jalankan mqtt seperti berikut:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS DEVDB
PS D:\KULIAH\Sem 3\IoT\w12\mqtt-simulator-master> python3 mqtt-simulator/main.py
```

HASIL :



The screenshot shows a VS Code editor with a file explorer on the left containing files like `mqtt-simulator-master`, `config`, `data_classes`, `mqtt-simulator`, `requirements.txt`, and `utils.py`. The main editor displays the `mqtt-simulator/main.py` file, which contains a `mqtt_simulator` class. The terminal at the bottom shows the output of the program, which is a series of log messages indicating data published to various topics. The messages are as follows:

```
[11:22:15] Data published on: lamp1
[11:22:16] Data published on: lamp2
[11:22:18] Data published on: location
[11:22:20] Data published on: temperature/basement
[11:22:21] Data published on: lamp2
[11:22:21] Data published on: lamp1
[11:22:24] Data published on: location
[11:22:26] Data published on: temperature/roof
[11:22:27] Data published on: lamp1
[11:22:27] Data published on: lamp2
[11:22:27] Data published on: lamp1
[11:22:27] Data published on: location
[11:22:28] Data published on: temperature/basement
[11:22:32] Data published on: temperature/roof
[11:22:32] Data published on: lamp1
[11:22:37] Data published on: location
[11:22:38] Data published on: temperature/basement
[11:22:38] Data published on: lamp1
[11:22:39] Data published on: temperature/roof
[11:22:39] Data published on: freezer
[11:22:39] Data published on: air_quality
[11:22:43] Data published on: location
[11:22:44] Data published on: temperature/basement
[11:22:45] Data published on: temperature/roof
[11:22:45] Data published on: lamp1
[11:22:45] Data published on: lamp2
[11:22:45] Data published on: freezer
[11:22:50] Data published on: location
```