

LAPORAN PRAKTIKUM IOT



Nokatri Sitinjak 11323039

INSTITUT TEKNOLOGI DEL
FAKULTAS VOKASI
D III TEKNOLOGI INFORMASI
2024/2025

Mqtt terdiri dari beberapa yaitu:

➤ 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
9
10 def is_valid_file(parser, arg):
11     settings_file = Path(arg)
12     if not settings_file.is_file():
13         return parser.error(f"argument -f/--file: can't find '{arg}'")
14     return settings_file
15
16 parser = argparse.ArgumentParser()
17 parser.add_argument('-f', '--file', dest='settings_file',
18                     help='config file path', required=False)
19 args = parser.parse_args()
20
21 simulator = Simulator(args.settings_file)
22 simulator.run()
```

Skrip Python ini menggunakan modul argparse untuk mengelola argumen baris perintah dan memulai simulasi broker MQTT dengan kelas Simulator. Fungsi default_settings mengembalikan jalur ke file pengaturan default (settings.json) yang terletak di folder config. Fungsi is_valid_file memeriksa apakah jalur yang diberikan sebagai argumen merupakan file yang valid, dan jika tidak, akan mengeluarkan pesan kesalahan. Skrip ini kemudian membuat objek ArgumentParser untuk menambahkan argumen -f/--file, yang memungkinkan pengguna menentukan file pengaturan; jika tidak ada yang diberikan, file default digunakan. Setelah memvalidasi file, skrip membuat instansi dari kelas Simulator dengan file pengaturan yang ditentukan dan memanggil metode run() untuk memulai simulasi. Dengan demikian, skrip ini memfasilitasi eksekusi simulasi dengan pengaturan yang dapat disesuaikan.

➤ Simulator.py

```
import json
from topic import Topic
from data_classes import BrokerSettings, ClientSettings

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)

    def read_client_settings(self, settings_dict: dict):
        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)
        )

    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', 'MQTT')
            )
```

Kelas Simulator dalam kode Python ini dirancang untuk memuat dan mengelola topik-topik dalam simulasi broker MQTT. Saat diinisialisasi, kelas ini mengambil

file pengaturan JSON yang berisi konfigurasi broker dan klien. Metode `load_topics` bertugas membaca pengaturan tersebut dan membuat objek `Topic` berdasarkan tipe topik yang ditentukan (seperti `single`, `multiple`, atau `list`). Selain itu, kelas ini memiliki metode `run` untuk memulai semua topik dan menunggu prosesnya, serta metode `stop` untuk menghentikan semua topik yang sedang berjalan. Dengan demikian, kelas ini menyediakan struktur yang efisien untuk mensimulasikan interaksi dengan broker MQTT.

➤ **Topik.py**

```
import time
import json
import threading
import paho.mqtt.client as mqtt
from data_classes import BrokerSettings, ClientSettings
from topic_data import TopicDataNumber, TopicDataBool, TopicDataRawValue, TopicDataMa

class Topic(threading.Thread):
    def __init__(self, broker_settings: BrokerSettings, topic_url: str, topic_data: list):
        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

    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':
```

Kode yang diberikan adalah implementasi kelas `Topic` dalam Python yang menggunakan `threading` dan pustaka `Paho MQTT` untuk mengelola publikasi data ke topik MQTT. Kelas ini mewarisi dari `threading.Thread`, memungkinkan setiap topik untuk berjalan di thread terpisah. Dalam konstruktor, kelas ini menerima pengaturan broker, URL topik, data topik, akar payload, dan pengaturan klien. Metode `load_topic_data` memuat data topik berdasarkan tipe yang ditentukan, sedangkan metode `connect` dan `disconnect` menghubungkan dan memutuskan koneksi dari broker MQTT. Metode `run` dijalankan saat thread dimulai, menghubungkan ke broker dan terus menerbitkan payload yang dihasilkan ke topik dengan interval waktu tertentu. Setelah data dipublikasikan, metode `on_publish` mencetak waktu dan URL topik, sementara metode `generate_payload` menghasilkan payload yang akan dipublikasikan, menggabungkan data dari akar payload dan nilai yang dihasilkan dari data aktif. Dengan demikian, kelas ini menyediakan mekanisme yang efisien untuk publikasi data secara teratur ke broker MQTT.

➤ **Utils.py**

```
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
```

Fungsi `should_run_with_probability` dalam kode Python ini digunakan untuk menentukan apakah suatu aksi harus dijalankan berdasarkan probabilitas yang diberikan. Fungsi ini menerima satu argumen, `probability`, yang merupakan nilai float

antara 0 dan 1, di mana 0 berarti tidak ada kemungkinan untuk menjalankan aksi, sementara 1 berarti aksi pasti akan dijalankan. Dengan menggunakan modul random, fungsi ini menghasilkan angka acak antara 0.0 dan 1.0. Kemudian, ia membandingkan angka acak tersebut dengan nilai probabilitas; jika angka acak lebih kecil dari probabilitas, fungsi mengembalikan True, menandakan bahwa aksi dapat dijalankan, dan jika tidak, mengembalikan False. Dengan cara ini, fungsi ini memungkinkan pengambilan keputusan berbasis probabilitas, berguna dalam berbagai aplikasi seperti simulasi, permainan, atau pengujian, di mana tindakan acak diperlukan

Cara untuk menjalankan

Jadi kali ini saya akan membuat mqtt simulator yang dimana cara menjalankannya itu kita harus menjalankan yang dibawah ini

```
D:\Semester 3\IOT\week13\mqtt-simulator-main>python -m venv venv
```

Biasanya jika python nya gada maka code ini ga akan bisa jalan jadi pastikan kalau python nya sudah di install

Selanjutnya jika code nya udah berhasil kita bisa lanjut ke code selanjutnya seperti yang dibawah ini:

```
D:\Semester 3\IOT\week13\mqtt-simulator-main>venv\Scripts\activate
```

Bagian ini berfungsi untuk memfasilitasi pengembangan dan pengujian aplikasi IoT yang menggunakan protokol MQTT sebagai mekanisme komunikasi data. Dengan menggunakan simulator, pengembang dapat menghemat waktu dan sumber daya dengan melakukan pengujian awal pada lingkungan virtual sebelum menerapkannya pada perangkat fisik

```
(venv) D:\Semester 3\IOT\week13\mqtt-simulator-main>pip install -r requirements.txt
Collecting paho-mqtt==1.5.0 (from -r requirements.txt (line 1))
  Using cached paho_mqtt-1.5.0-py3-none-any.whl
Installing collected packages: paho-mqtt
Successfully installed paho-mqtt-1.5.0

[notice] A new release of pip is available: 24.2 -> 24.3.1
[notice] To update, run: python.exe -m pip install --upgrade pip
```

Setelah itu langkah selanjutnya itu adalah menjalankan “**pip install -r requirements.txt**” hal ini berguna untuk menginstal dan mengelola paket-paket Python. Dalam konteks ini, perintah “python.exe -m pip install --upgrade pip” akan memperbarui pip ke versi terbaru yang tersedia, yaitu dari versi 24.2 ke versi 24.3.1.

```
(venv) D:\Semester 3\IOT\week13\mqtt-simulator-main>py mqtt-simulator/main.py
starting: lamp/1 ...
starting: lamp/2 ...
starting: air_quality ...
starting: temperature/roof ...
starting: temperature/basement ...
starting: freezer ...
starting: location ...
11:36:29] Data published on: lamp/2
11:36:29] Data published on: temperature/basement
11:36:29] Data published on: location
11:36:29] Data published on: freezer
11:36:32] Data published on: temperature/roof
11:36:34] Data published on: air_quality
11:36:34] Data published on: lamp/1
11:36:35] Data published on: location
11:36:35] Data published on: lamp/2
11:36:35] Data published on: lamp/2
11:36:36] Data published on: temperature/basement
11:36:40] Data published on: freezer
11:36:40] Data published on: freezer
11:36:42] Data published on: temperature/roof
11:36:42] Data published on: temperature/roof
11:36:45] Data published on: air_quality
11:36:45] Data published on: air_quality
11:36:45] Data published on: air_quality
```

Langkah selanjutnya adalah menjalankan “**py mqtt-simulator/main.py**” digunakan untuk menjalankan simulator MQTT atau aplikasi IoT yang terkait dengan simulator tersebut. Hal ini dapat membantu pengembang dalam menguji dan mengembangkan aplikasi IoT yang menggunakan protokol MQTT