LAPORAN PRAKTIKUM
Internet of Things

MQTT Server

Anno Deritman Siregar
11323024
DIII-Teknologi Informasi

INSTITUT TEKNOLOGI DEL
FAKULTAS VOKASI
Tahun Ajaran 2024/2025

1. Imports

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

mengimpor modul json untuk memproses data JSON dan dua kelas yaitu BrokerSettings dan ClientSettings dari modul data_classes. Pada kelas Topic dari modul topic. Kelas ini digunakan untuk mengonfigurasi pengaturan broker, klien, dan topik MQTT dalam simulasi.

2. Konstruktor

```
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)
```

Terdapat beberapa modul dan kelas yang diimpor untuk mendukung pengelolaan konfigurasi MQTT. Pertama, modul json digunakan untuk membaca dan mengolah file konfigurasi dalam format JSON. Kemudian, dua kelas dari modul data_classes, yaitu BrokerSettings dan ClientSettings, digunakan untuk menyimpan pengaturan terkait broker MQTT dan klien MQTT. Kelas Topic yang diimpor dari modul topic bertanggung jawab untuk mengelola topik-topik MQTT yang akan digunakan dalam simulasi. Dengan demikian, kode ini mempersiapkan dan mengelola pengaturan broker dan klien serta topik-topik MQTT yang akan diproses lebih lanjut dalam simulasi.

3. Metode 1

```
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 membaca pengaturan klien MQTT dari dictionary settings_dict, yang biasanya berasal dari file JSON. Jika pengaturan tertentu tidak ditemukan di dictionary tersebut, maka nilai default dari parameter default akan digunakan. settings_dict.get()

digunakan untuk mengambil nilai pengaturan dari dictionary, dengan nilai default yang diberikan jika pengaturan tersebut tidak ada.

4. Metode 2

```
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
        )
```

program membaca file konfigurasi dalam format JSON yang berisi pengaturan broker dan topik-topik MQTT. Data dari file JSON ini dimuat ke dalam variabel config. Kemudian, program mengekstrak pengaturan broker, seperti URL, port, dan versi protokol MQTT, yang disimpan dalam objek broker_settings. Jika pengaturan ini tidak ada dalam file, maka nilai default akan digunakan. Selanjutnya, pengaturan klien broker yang digunakan oleh program diambil dengan memanggil metode read_client_settings, yang menghasilkan objek broker_client_settings yang berisi pengaturan klien default untuk broker yang telah dibaca.

```
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':
```

- topic_data: Mengambil data yang terkait dengan topik (biasanya pengaturan atau informasi payload).
- topic_payload_root: Mengambil pengaturan PAYLOAD_ROOT untuk topik, jika ada.
- topic_client_settings: Membaca pengaturan klien untuk topik ini, dengan menggunakan pengaturan broker sebagai default.

5. Membuat Topik Berdasarkan Tipe

```
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))
topics
```

- single: Jika tipe topik adalah single, maka hanya satu topik yang dibuat dengan format URL /PREFIX.

- multiple: Jika tipe topik adalah multiple, maka beberapa topik akan dibuat dengan ID yang bervariasi dalam rentang yang ditentukan oleh RANGE_START dan RANGE_END. Format URL-nya adalah /PREFIX/ID.
- list: Jika tipe topik adalah list, maka topik dibuat berdasarkan elemen dalam daftar LIST. Format URL-nya adalah /PREFIX/ITEM.

6. Metode 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()
```

- run: Menjalankan simulator dengan memulai semua topik.
- topic.start(): Memanggil metode start() pada setiap topik, yang diasumsikan akan menjalankan proses terkait topik tersebut, misalnya mulai mempublikasikan atau berlangganan.
- topic.join(): Menunggu agar setiap topik selesai (jika menggunakan threading atau proses lain yang berjalan secara paralel).

7. Metode Stop

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

- stop: Menghentikan semua topik dengan memanggil metode stop() pada setiap topik. Biasanya digunakan untuk menghentikan proses yang sedang berjalan.