LAPORAN PRAKTIKUM

INTERNET OF THINGS

SIMULATOR PY

Winda Vanecya Sembiring
11323013
D3 Teknologi Informasi

INSTITUT TEKNOLOGI DEL
FAKULTAS VOKASI
2024-2025

A. Code

$ code simulator.py 2

C: > Users > Winda Sembiring > Downloads > simulator.py > Simulator > load_topics

```python
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, 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)
        )

    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

    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()

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

Penjelasan Code :

- Import Modul

```
C: > Users > Winda Sembiring > Downloads > simulator.py > Simulator > load_topics
1   import json
2   from topic import Topic
3   from data_classes import BrokerSettings, ClientSettings
```

Bagian ini mengimpor modul dan kelas yang diperlukan untuk menjalankan simulasi. json digunakan untuk memproses data dalam format JSON, yang akan digunakan untuk mengonfigurasi pengaturan broker MQTT dan topik-topik yang akan digunakan dalam simulasi. Topic adalah kelas yang mengelola topik komunikasi dalam MQTT, memungkinkan untuk memanipulasi topik, mengirim, dan menerima data. Sementara itu, BrokerSettings dan ClientSettings adalah kelas yang berfungsi untuk menyimpan dan mengatur pengaturan broker MQTT dan klien MQTT, seperti URL broker, port, serta opsi komunikasi lainnya.

- Mendefenisikan Kelas Simulator

```
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)
```

Kelas Simulator adalah kelas utama yang mengelola proses simulasi komunikasi MQTT. Pada saat inisialisasi objek dari kelas ini, file pengaturan diberikan sebagai parameter settings_file. Konstruktor kelas ini mengatur pengaturan default untuk klien dengan menggunakan objek ClientSettings, yang mencakup parameter seperti sesi bersih (clean=True), pesan yang tidak dipertahankan (retain=False), pengaturan QoS (Quality of Service) dengan tingkat 2, dan interval waktu antar pesan yang dikirimkan sebesar 10 detik. Selain itu, konstruktor ini juga memuat topik-topik dari file pengaturan dengan memanggil metode load_topics.

❖ Metode 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)
    )
```

Metode ini digunakan untuk membaca pengaturan klien dari sebuah dictionary (settings_dict) yang memuat nilai-nilai pengaturan yang dibaca dari file JSON. Jika suatu pengaturan tidak ditemukan dalam dictionary, maka nilai default yang diberikan pada parameter default akan digunakan. Dengan cara ini, kita memastikan bahwa pengaturan klien dapat disesuaikan, namun jika tidak ada nilai yang ditentukan, pengaturan default tetap digunakan

❖ Menerapkan metode 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)
```

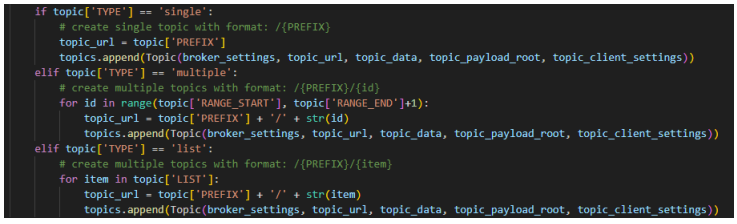
Metode load_topics bertugas untuk membaca file pengaturan yang berformat JSON dan mengonfigurasi topik-topik yang akan digunakan dalam simulasi. Pertama, file JSON dibuka dan isinya dibaca menggunakan json.load. Kemudian, pengaturan broker MQTT seperti URL, port, dan versi protokol ditetapkan dengan nilai default jika tidak ada nilai yang diberikan dalam file pengaturan. Selain itu, metode ini juga menggunakan read_client_settings untuk mendapatkan pengaturan klien broker, yang selanjutnya akan digunakan untuk setiap topik yang dimuat.

❖ Membaca dan mengelola topik

```
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)
```

Kode ini, dilakukan iterasi terhadap setiap elemen yang ada dalam daftar config['TOPICS']. Daftar ini merupakan kumpulan topik yang didefinisikan dalam file konfigurasi JSON yang telah dibaca sebelumnya. Setiap topik dalam daftar ini berisi informasi yang diperlukan untuk konfigurasi komunikasi MQTT. Untuk setiap topik, pertama-tama kode mengambil data yang terkait dengan topik tersebut, yang disimpan dalam variabel topic_data. Data ini biasanya berisi payload atau informasi yang akan dikirimkan melalui topik MQTT. Selanjutnya, kode mencoba untuk mengambil nilai dari kunci 'PAYLOAD_ROOT' dalam topik. Jika kunci tersebut ada, maka nilai tersebut akan digunakan sebagai struktur dasar untuk payload, tetapi jika tidak ada, maka sebuah dictionary kosong akan digunakan sebagai nilai default. Hal ini berguna untuk memastikan bahwa setiap topik memiliki struktur yang tepat untuk data yang dikirimkan. Selain itu, kode juga memanggil metode read_client_settings untuk membaca pengaturan klien spesifik untuk topik ini. Metode ini akan mencari pengaturan dalam data topik tersebut, dan jika tidak ada, akan menggunakan pengaturan default yang telah dibaca sebelumnya untuk broker MQTT. Dengan cara ini, setiap topik dapat dikonfigurasi dengan pengaturan klien yang tepat, yang memastikan komunikasi berjalan sesuai dengan kebutuhan masing-masing topik. Secara keseluruhan, bagian kode ini berfungsi untuk menyiapkan dan mengkonfigurasi semua topik yang akan digunakan dalam simulasi komunikasi MQTT, dengan mempertimbangkan data, struktur payload, dan pengaturan klien yang diperlukan

- Membuat topik berdasarkan jenis



```
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))
```

Pada bagian kode ini, dilakukan pemrosesan topik berdasarkan jenis yang didefinisikan dalam konfigurasi. Ada tiga jenis topik yang bisa dipilih: single, multiple, dan list. Pertama, kode memeriksa apakah jenis topik adalah 'single'. Jika jenisnya adalah 'single', maka topik akan dibuat dengan format URL yang sederhana menggunakan nilai dari PREFIX yang didefinisikan dalam konfigurasi. Misalnya, jika PREFIX adalah "sensor", maka topik yang dibuat adalah "/sensor". Setelah itu, objek Topic dibuat dengan pengaturan yang sesuai

(broker, URL topik, data, struktur payload, dan pengaturan klien), dan objek Topic ini ditambahkan ke dalam daftar topics.Selanjutnya, jika jenis topik adalah 'multiple', maka kode akan membuat banyak topik dengan format URL yang terdiri dari PREFIX diikuti oleh sebuah angka yang berada dalam rentang yang ditentukan oleh RANGE_START dan RANGE_END. Rentang ini diperoleh dengan menggunakan loop for yang akan menghasilkan nilai id mulai dari RANGE_START hingga RANGE_END. Sebagai contoh, jika RANGE_START adalah 1 dan RANGE_END adalah 5, maka topik yang dibuat akan memiliki format "/sensor/1", "/sensor/2", dan seterusnya hingga "/sensor/5". Setiap topik yang dihasilkan akan menjadi objek Topic yang ditambahkan ke dalam daftar topics.Terakhir, jika jenis topik adalah 'list', maka kode akan membuat topik berdasarkan daftar LIST yang diberikan dalam konfigurasi. Setiap elemen dalam daftar ini akan digabungkan dengan PREFIX untuk membentuk URL topik. Misalnya, jika LIST berisi [1, 2, 3] dan PREFIX adalah "device", maka topik yang dihasilkan adalah "/device/1", "/device/2", dan "/device/3". Sama seperti sebelumnya, setiap topik yang dibuat akan menjadi objek Topic yang ditambahkan ke dalam daftar topics.Dengan demikian, bagian kode ini bertujuan untuk membuat topik-topik yang berbeda sesuai dengan jenis yang ditentukan, dan setiap topik ini kemudian dimasukkan dalam daftar topics untuk digunakan lebih lanjut dalam simulasi komunikasi MQTT.

- 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()
```

Metode run menjalankan simulasi komunikasi untuk setiap topik yang telah dimuat sebelumnya. Untuk setiap topik, proses dimulai dengan memanggil topic.start(), yang akan memulai pengiriman data untuk topik tersebut. Setelah itu, topic.join() dipanggil untuk memastikan bahwa proses komunikasi selesai dengan baik sebelum melanjutkan ke topik berikutnya

❖ Stop

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

Metode stop digunakan untuk menghentikan komunikasi pada setiap topik yang sedang berjalan. Hal ini penting untuk memastikan bahwa semua proses selesai dengan aman dan tidak ada pengiriman atau penerimaan data yang terganggu saat simulasi dihentikan. Untuk setiap topik, topic.stop() dipanggil untuk mengakhiri komunikasi.