LAPORAN PRAKTIKUM
Internet of Things

MQTT SERVER

Santo Martogi Simangunsong
11323017
D3 Teknologi Informasi

INSTITUT TEKNOLOGI DEL
FAKULTAS VOKASI

➔ Code simulator.py

C: > Users > ASUS > mqtt-simulator-master > mqtt-simulator > {} simulator.py

```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()
```

Kode di atas ini merupakan implementasi dari sebuah simulator yang mengatur topik-topik MQTT untuk komunikasi menggunakan protokol MQTT.

➔ Import Statements

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

- json digunakan untuk memuat pengaturan dari file JSON.
- Topic itu merupakan class yang diimpor dari modul topic, yang kemungkinan berfungsi untuk mewakili dan mengelola topik-topik MQTT.
- BrokerSettings dan ClientSettings merupakan kelas yang diimpor dari modul data_classes, digunakan untuk menyimpan pengaturan broker dan klien MQTT.

➔ Class Simulator

```
class Simulator:
    def __init__(self, settings_file):
```

Class Simulator di atas bertanggung jawab untuk mengelola seluruh simulasi. Konstruktor __init__ menerima nama file pengaturan dan memuat topik-topik berdasarkan pengaturan tersebut.

➔ Pengaturan Klien Default

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

Kode di atas membuat objek default_client_settings dari kelas ClientSettings dengan pengaturan default untuk koneksi MQTT, yaitu clean=True (session bersih), retain=False (pesan tidak disimpan di broker), qos=2 (Quality of Service level 2, memastikan pesan dikirimkan sekali dan hanya sekali), dan time_interval=10 detik (interval waktu antara pengiriman pesan). Kemudian, kode tersebut memanggil metode load_topics(settings_file) untuk memuat daftar topik dari file pengaturan (settings_file) dan menyimpannya dalam variabel 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)
    )
```

Fungsi read_client_settings bertujuan untuk membaca pengaturan klien dari sebuah dictionary (settings_dict) dan mengembalikan objek ClientSettings dengan nilai-nilai yang diambil dari dictionary tersebut. Jika sebuah pengaturan tidak ditemukan dalam dictionary, maka akan menggunakan nilai default dari objek default. Misalnya, atribut seperti CLEAN_SESSION, RETAIN, QOS, dan TIME_INTERVAL akan dicek dalam dictionary, dan jika tidak ada, nilainya diambil dari default.

➔ 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.MQTTv31
        )
```

Fungsi load_topics membaca file konfigurasi (dalam format JSON) yang diberikan melalui parameter settings_file untuk memuat pengaturan topik MQTT. Pertama, file dibuka dan datanya dibaca sebagai objek JSON menggunakan json.load. Kemudian, fungsi membuat instance BrokerSettings untuk menyimpan pengaturan broker seperti URL, port, dan versi protokol MQTT, dengan nilai default seperti 'localhost' untuk URL, 1883 untuk port, dan 4 (atau MQTTv311) untuk protokol jika tidak ada nilai di file JSON. Hasil akhirnya adalah daftar topik yang akan digunakan.

➔ 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()
```

Kode tersebut adalah bagian dari sebuah metode bernama run yang bertugas menjalankan thread untuk setiap objek topic dalam daftar self.topics. Pertama, setiap topik dimulai dengan memanggil metode start() pada objek thread tersebut, sambil mencetak pesan ke konsol tentang topik yang sedang dimulai.

➔ Metode stop

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

Fungsi stop() ini digunakan untuk menghentikan setiap topik yang ada di dalam daftar self.topics. Fungsi ini pertama-tama mencetak pesan yang memberitahukan bahwa topik sedang dihentikan, dengan menampilkan URL topik yang bersangkutan. Kemudian, untuk setiap objek topic dalam daftar self.topics, fungsi stop() dipanggil pada objek tersebut, yang berarti proses atau aktivitas terkait topik tersebut akan dihentikan.

TERIMA KASIH