Laporan Praktikum

Internet of Things

MQTT Simulator

Nama :

Antonia Tiopani Manalu 11323014

Prodi :

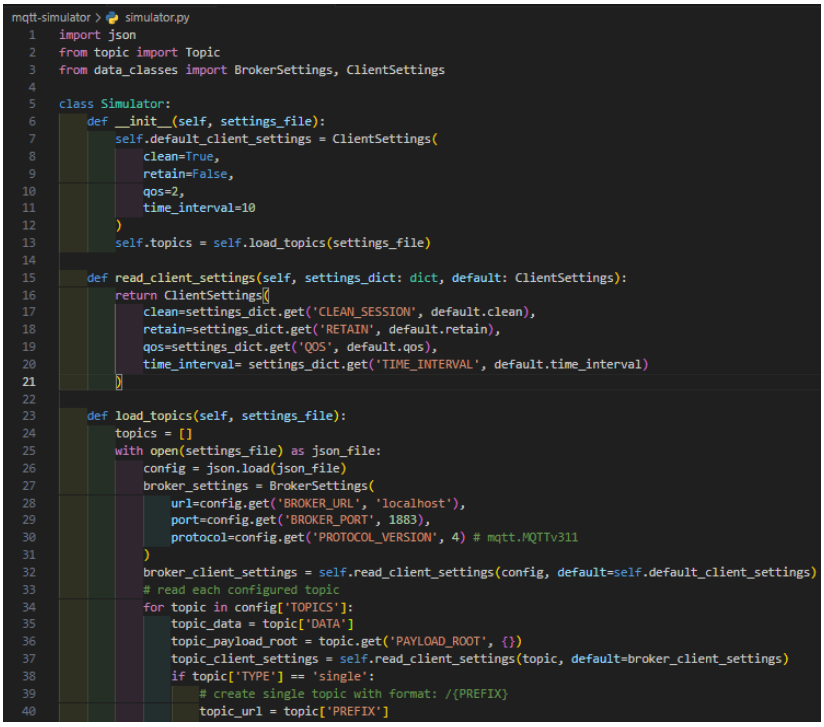
D3 Teknologi Informasi

INSTITUT TEKNOLOGI DEL

FAKULTAS VOKASI

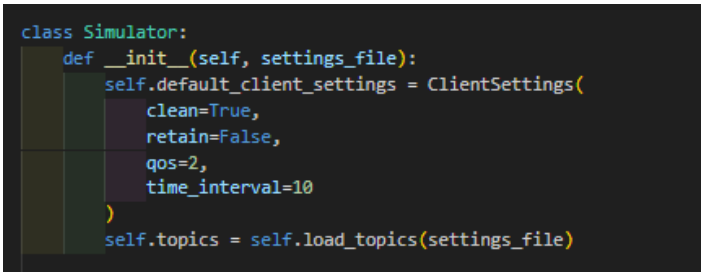
2024/2025

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()
```

Penjelasan:



```python
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)
```

Pada kelas simulator **(__init__)** sebagai fungsi pengaturan default untuk klien MQTT. Membaca konfigurasi dari file JSON untuk menghasilkan daftar objek Topic. Memiliki konfigurasi **clean= true** digunakan untuk menentukan apakah sesi MQTT adalah sesi bersih. Jika True, maka broker

akan menghapus data sesi sebelumnya saat klien terhubung. **Retain= false** yang digunakan untuk menentukan apakah pesan MQTT akan di-retain oleh broker. Jika False, broker tidak menyimpan pesan terakhir untuk topik tersebut. **Qos=2** digunakan untuk menentukan tingkat kualitas layanan Quality of Service untuk pesan MQTT

0: Kirim pesan tanpa konfirmasi (at most once).

1: Kirim pesan dengan konfirmasi (at least once).

2: Kirim pesan dengan konfirmasi lengkap (exactly once).

time_interval=10 sebagai interval waktu dalam detik antara pengiriman pesan melalui topik.

```
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)
    )
```

Pada code ini berfungsi untuk membaca pengaturan klien MQTT dari konfigurasi JSON. Jika pengaturan tidak ditemukan dalam file JSON, nilai default akan digunakan. **settings_dict** dictionary yang berisi pengaturan dari file JSON. **default** merupakan nilai default untuk pengaturan klien MQTT. Sebuah objek ClientSettings dengan properti seperti clean, retain, qos, dan 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
```

Pada code ini digunakan sebagai fungsi load_topics digunakan untuk membaca file pengaturan (file JSON) yang berisi konfigurasi topik-topik MQTT. Berdasarkan data tersebut, fungsi ini membuat daftar (list) objek Topic yang merepresentasikan setiap topik MQTT yang telah dikonfigurasi. **settings_file** merupakan nama file atau path file JSON yang berisi pengaturan. File ini digunakan untuk mendefinisikan topik, pengaturan broker MQTT, dan pengaturan klien. **topics** merupakan sebuah list kosong bernama topics diinisialisasi untuk menyimpan semua topik MQTT yang akan dibuat. **open()** digunakan untuk membuka file JSON. **json.load()** digunakan untuk membaca dan mengubah data mejadi dictionary Phyton. Dari dictionary **config**, nilai pengaturan broker seperti BROKER_URL, BROKER_PORT, dan PROTOCOL_VERSION dibaca, dan jika nilai tidak ditemukan maka nilai default seperti 'localhost' untuk URL untuk port akan digunakan. Pengaturan ini digunakan untuk membuat instance BrokerSettings.

Fungsi read_client_settings dipanggil untuk membaca pengaturan default klien dari config atau menggunakan nilai default yang telah didefinisikan sebelumnya. Semua topik yang terdaftar dalam config['TOPICS'] diproses satu per satu. Nilai DATA (payload) akan dibaca dan disimpan ke dalam `topic_data`. Jika ada, `PAYLOAD_ROOT` diambil dari konfigurasi; jika tidak, akan diatur ke {} sebagai nilai default. Pengaturan klien untuk topik akan dibaca menggunakan `read_client_settings`.

- Tipe single
  Satu topik dibuat menggunakan PREFIX langsung. Objek Topic untuk topik tersebut ditambahkan ke list topics
- Tipe multiple
  Topik dibuat dengan rentang ID yang ditentukan oleh `RANGE_START` dan `RANGE_END`.
- Tipe list
  Topik dibuat menggunakan elemen-elemen dalam LIST.

Setelah semua topik selesai, maka akan berisi objek-objek Topic dikembalikan.

```
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 run pada kelas Simulator digunakan untuk menginterasi melalui dafatar topik dan memulai setiap topik dengan **start()** dan dijalankan secara treads terpisah, dan **join()** digunakan untuk mengunggu hinga setiap topik selesai menjalankan tugasnya.

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

Fungsi stop pada kelas simulator adalah untuk menginterasikan melalui daftar topik yang telah dimuat(self.topics) pada setiap objek Topic, dan sebelum menghentikan topik, fungsi yang mencetak pesan menunjukan bahwa topik tersebut sedang dihentikan.