Simulator.py

Internet of Things

Disusun Oleh :

Nama : Tasya Simamora

NIM : 11323059

Diploma III Teknologi Informasi

Institut Teknologi Del

Sitoluama, Laguboti 2024

Code:

simulator.py

mqtt-simulator > simulator.py > Simulator > read_client_settings

```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:

Kelas *Simulator*

Kelas ini bertugas mengelola simulasi komunikasi berbasis message broker dengan protokol MQTT. Kelas ini mengatur topik-topik (topics) yang akan digunakan dalam komunikasi berdasarkan file konfigurasi JSON. Saat diinisialisasi, kelas ini membuat pengaturan klien MQTT bawaan menggunakan objek *ClientSettings* dan memuat daftar topik menggunakan metode *load_topics*. Setiap topik dikonfigurasi berdasarkan jenisnya (misalnya, single, multiple, atau list) untuk menentukan bagaimana pesan akan diteruskan ke broker MQTT.

Konstruktor *_init_*

Konstruktor kelas bertugas menginisialisasi nilai bawaan untuk pengaturan klien MQTT dengan membuat objek *ClientSettings* yang memiliki properti seperti *clean*, *retain*, *qos*, dan *time_interval*. Selanjutnya, file konfigurasi dibaca melalui metode *load_topics*, yang menghasilkan daftar objek *Topic* berdasarkan isi file JSON tersebut.

Metode *read_client_settings*

Metode ini membaca pengaturan klien MQTT dari sebuah dictionary dan menghasilkan objek *ClientSettings*. Metode ini memastikan bahwa setiap properti seperti *CLEAN_SESSION, RETAIN, QOS*, dan *TIME_INTERVAL* disesuaikan dengan nilai yang diberikan dalam file konfigurasi. Jika tidak ada nilai yang ditemukan, maka nilai bawaan dari parameter *default* akan digunakan. Hasil akhirnya adalah objek *ClientSettings* dengan nilai yang sudah dikonfigurasi.

Metode *load_topics*

Metode ini membaca file JSON konfigurasi dan membuat daftar topik berdasarkan pengaturan yang ada. Pertama, pengaturan broker MQTT seperti URL, port, dan versi protokol disimpan dalam objek *BrokerSettings*. Pengaturan klien umum dibaca menggunakan metode *read_client_settings*. Selanjutnya, metode ini memproses setiap topik dalam konfigurasi. Jika tipe topik adalah *single*, maka topik dibuat dengan format */{PREFIX}.* Jika tipe multiple, maka beberapa topik dibuat berdasarkan rentang *RANGE_START* hingga *RANGE_END*, dengan format */{PREFIX}/{id}.* Sementara itu, tipe *list* menghasilkan beberapa topik berdasarkan elemen dalam *LIST*, dengan format */{PREFIX}/{item}.* Semua topik yang dihasilkan disimpan dalam daftar dan dikembalikan oleh metode ini.

Metode *run*

Metode ini digunakan untuk memulai semua topik yang telah dikonfigurasi. Untuk setiap topik dalam daftar *self.topics*, metode *start()* dipanggil untuk memulai komunikasi. Setelah itu, metode *join()* dipanggil untuk memastikan semua topik berjalan secara bersamaan. Metode ini penting untuk menjalankan simulasi komunikasi dengan broker MQTT secara serentak.

Metode *stop*

Metode ini digunakan untuk menghentikan semua topik yang sedang berjalan. Untuk setiap topik dalam daftar *self.topics*, metode *stop()* dipanggil untuk mengakhiri komunikasi. Metode ini memastikan semua topik dihentikan dengan rapi.