Laporan Praktikum

Internet of Things

Simolator.py

Nama : Feby Revalia Manalu

NIM : 11323050

Program Studi : D3 Teknologi Informasi

INSTITUT TEKNOLOGI DEL

FAKULTAS VOKASI

2024/2025

Simulator.py adalah sebuah modul atau skrip Python yang dirancang untuk mensimulasikan sistem berbasis protokol MQTT (Message Queuing Telemetry Transport). Tujuan utama dari simulator.py adalah menjalankan simulasi komunikasi antar perangkat atau sistem yang terhubung melalui topik-topik tertentu, seperti yang sering digunakan dalam Internet of Things (IoT).

1. **Simulator**:

- Dalam konteks ini, *simulator* adalah sebuah alat atau program yang mensimulasikan perilaku perangkat atau aplikasi dalam ekosistem tertentu, tanpa memerlukan perangkat keras atau sistem nyata.
- **`simulator.py`** menciptakan simulasi berbasis topik MQTT, yang melibatkan broker, klien, dan pesan.

2. **MQTT**:

- MQTT adalah protokol komunikasi yang ringan, dirancang untuk perangkat dengan sumber daya terbatas (misalnya sensor IoT) yang berkomunikasi melalui broker dan menggunakan topik untuk saling bertukar data.

Import Library dan Modul

```
1   import json
2   from topic import Topic
3   from data_classes import BrokerSettings, ClientSettings
```

- **json**: Digunakan untuk membaca file konfigurasi dalam format JSON.
- **Topic**: Kelas yang digunakan untuk mewakili topik MQTT.
- **BrokerSettings, ClientSettings**: Kelas data yang mewakili pengaturan broker MQTT dan klien MQTT.

Kelas Simulator

Kelas ini bertanggung jawab untuk membuat konfigurasi dari file JSON, membuat objek topic dan mengelola jalannya simulasi.

```
5   class Simulator:
6       def __init__(self, settings_file):
7           self.default_client_settings = ClientSettings(
8               clean=True,
9               retain=False,
10              qos=2,
11              time_interval=10
12          )
13          self.topics = self.load_topics(settings_file)
```

- **`settings_file`**: Nama file konfigurasi JSON.
- **`self.default_client_settings`**: Pengaturan klien MQTT default (clean session, tidak menyimpan pesan, QoS 2, interval waktu 10 detik).
- **`self.topics`**: List dari objek topik MQTT yang dimuat dari file konfigurasi.

Metode read_client_settngs

```python
    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)
        )
```

- Membaca pengaturan klien dari kamus konfigurasi.
- Menggunakan nilai default jika kunci tertentu tidak ada dalam konfigurasi.

Metode load_topics:

```python
    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
```

Membaca file JSON yang berisi konfigurasi broker dan topik. **Broker Settings**: Pengaturan broker MQTT seperti URL, port, dan versi protokol.
Topics: Membaca daftar topik dari file konfigurasi dan mengelompokkan berdasarkan jenis:

- **`single`**: Topik tunggal.
- **`multiple`**: Topik dalam rentang tertentu (misalnya, `/sensor/1` hingga `/sensor/10`).
- **`list`**: Topik yang ditentukan dalam daftar (misalnya, `/device/A`, `/device/B`).

bjnk

```python
    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
```

- Memulai semua topik untuk simulasi.
- **topic.start()**: Memulai proses simulasi untuk setiap topik.
- **topic.join()**: Menunggu semua proses selesai.

```
62      def stop(self):
63          for topic in self.topics:
64              print(f'Stopping: {topic.topic_url} ...')
65              topic.stop()
66
```

Menghentikan semua proses simulasi topik.

simulator.py adalah program Python untuk mensimulasikan komunikasi berbasis protokol MQTT, yang sering digunakan dalam pengembangan sistem **Internet of Things (IoT)**. Program ini membaca konfigurasi dari file JSON, membuat objek topik MQTT (single, multiple, atau list), dan menjalankan simulasi komunikasi, seperti mempublikasikan atau menerima data melalui topik tersebut.

Kegunaan Utama:

1. Menguji pengaturan broker dan klien MQTT.
2. Memvalidasi komunikasi antar perangkat virtual.
3. Melakukan debugging dan benchmarking performa sistem.

Ini adalah alat penting untuk pengembangan, pengujian, dan prototipe sistem IoT tanpa memerlukan perangkat fisik.