LAPORAN PRAKTIKUM

Internet Of Things

Romian A. Tambunan

11323007

D3 Teknologi Informasi

INSTITUT TEKNOLOGI DEL

FAKULTAS VOKASI

TAHUN AJARAN 2024/2025

```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}
```

```python
class Simulator:
    def load_topics(self, settings_file):
        with open(settings_file) as json_file:
            broker_settings = BrokerSettings(
            )
            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 ini adalah implementasi dari simulator untuk menangani pengiriman data melalui protokol MQTT, yang umum digunakan untuk komunikasi antar perangkat

CLASS SIMULATOR

```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)

    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)
        )
```

Penjelasan:

__init__(self, settings_file):

menginisialisasi simulator dengan file pengaturan (settings_file).

self.default_client_settings: Mengatur pengaturan klien default misalnya clean=True, retain=False

self.topics: Memuat semua topik yang ada

read_client_settings(self, settings_dict, default):

Fungsi ini membaca pengaturan klien dari file JSON dan menggabungkannya dengan pengaturan default jika ada nilai yang tidak ditemukan.

FUNGSI LOAD_TOPICS

Fungsi load_topics adalah tempat utama yang membaca file JSON dan memproses pengaturan topik-topik MQTT

```
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
```

Penjelasan:

broker_settings: Membaca pengaturan broker seperti URL, port, dan versi protocol

broker_client_settings: Membaca pengaturan klien untuk broker yang diset secara global

MENJALANKAN SIMULATOR metode run dan stop

```
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

run(self): menjalankan semua topik yang ada di dalam self.topics

Pesan Starting: {topic.topic_url} dicetak untuk menunjukkan bahwa simulasi dimulai untuk setiap topik.

stop(self): Menutup atau menghentikan komunikasi pada semua topik yang ada di dalam self.topics.

Memanggil topic.stop() untuk setiap topik untuk menghentikan pengiriman data atau koneks

Pesan Stopping: {topic.topic_url} dicetak untuk menunjukkan bahwa topik dihentikan