

LAPORAN PRAKTIKUM PENGEMBANGAN APLIKASI MOBILE

MQTT



**Febrina Elisabeth Sihombing
11323060
D3 Teknologi Informasi**

**INSTITUT TEKNOLOGI DEL
FAKULTAS VOKASI
2024/2025**

1.

```
(.venv) D:\PERKULIAHAN\Semester 3\IOT\mqtt-simulator-master\mqtt-simulator>python -m venv venv
```

Perintah ini digunakan untuk membuat *virtual environment* bernama venv menggunakan modul venv. Tujuannya adalah untuk membuat lingkungan kerja Python terisolasi, sehingga instalasi paket-paket Python tidak memengaruhi sistem global.

2.

```
(.venv) D:\PERKULIAHAN\Semester 3\IOT\mqtt-simulator-master\mqtt-simulator>venv\Scripts\activate
```

Kemudian perintah ini, *virtual environment* yang dibuat tadi diaktifkan. Setelah aktif, semua perintah terkait Python akan berjalan di lingkungan venv. Indikatornya adalah adanya prefiks (.venv) di terminal Anda.

3.

```
(.venv) D:\PERKULIAHAN\Semester 3\IOT\mqtt-simulator-master\mqtt-simulator>venv\Scripts\activate
(.venv) D:\PERKULIAHAN\Semester 3\IOT\mqtt-simulator-master\mqtt-simulator>pip install -r requirements.txt
Collecting paho-mqtt==1.5.0 (from -r requirements.txt (line 1))
Using cached paho-mqtt-1.5.0.tar.gz (99 kB)
Installing build dependencies ... done
Getting requirements to build wheel ... done
Preparing metadata (pyproject.toml) ... done
Building wheels for collected packages: paho-mqtt
Building wheel for paho-mqtt (pyproject.toml) ... done
Created wheel for paho-mqtt: filename=paho_mqtt-1.5.0-py3-none-any.whl size=64937 sha256=1341ca28b1fa331261d8a9ec889bf005ef609800c15bf592b9a6338f2107e727
Stored in directory: c:\users\asus\appdata\local\pip\cache\wheels\7b\9e\10\41c66db1509abff62dabae6ff620686b471d675dc5ff5b5
Successfully built paho-mqtt
Installing collected packages: paho-mqtt
Successfully installed paho-mqtt-1.5.0
[notice] A new release of pip is available: 24.2 -> 24.3.1
```

Dari log yang muncul, terlihat bahwa salah satu paket yang diinstal adalah paho-mqtt versi 1.5.0. Paket ini digunakan untuk mengimplementasikan protokol MQTT, yang sering digunakan untuk komunikasi pada perangkat IoT.

Proses instalasi berjalan lancar, dan semua dependensi berhasil diinstal. Oh iya, ada notifikasi bahwa versi baru dari pip tersedia, tapi saya belum sempat memperbaruinya karena itu opsional.

4.

```
(.venv) D:\PERKULIAHAN\Semester 3\IOT\mqtt-simulator-master\mqtt-simulator>py mqtt-simulator/main.py
```

Kemudian kita menjalankan konfigurasi ini

5.

```
(.venv) D:\PERKULIAHAN\Semester 3\IOT\mqtt-simulator-master\mqtt-simulator>py mqtt-simulator/main.py
Starting: lamp/1 ...
Starting: lamp/2 ...
Starting: air_quality ...
Starting: temperature/roof ...
Starting: temperature/basement ...
Starting: freezer ...
Starting: location ...
```

program MQTT simulator berhasil dijalankan dan menginisialisasi beberapa sensor/topic MQTT:

1. lamp/1 - untuk lampu pertama
2. lamp/2 - untuk lampu kedua
3. air_quality - untuk kualitas udara
4. temperature/roof - untuk suhu di atap
5. temperature/basement - untuk suhu di basement
6. freezer - untuk freezer
7. dll

Penjelasan Code :

- Main.py

```
import argparse
from pathlib import Path
from simulator import Simulator
```

- argparse: Modul ini digunakan untuk mempermudah pembuatan antarmuka baris perintah (CLI) dengan menangani parsing argumen.
- pathlib.Path: Modul ini digunakan untuk bekerja dengan jalur file dan direktori.
- simulator.Simulator: Modul Simulator diimpor dari paket simulator, yang kemungkinan besar berisi logika utama untuk simulasi yang dijalankan.

```
def default_settings():
    base_folder = Path(__file__).resolve().parent.parent
    settings_file = base_folder / 'config/settings.json'
    return settings_file
```

Fungsi ini mengembalikan jalur ke file konfigurasi default (settings.json). Ini menentukan folder dasar dari lokasi file skrip Python yang sedang dijalankan, kemudian menambahkan subfolder config dan file settings.json ke jalur tersebut.

```
def is_valid_file(parser, arg):
    settings_file = Path(arg)
    if not settings_file.is_file():
        return parser.error(f"argument -f/--file: can't open '{arg}'")
    return settings_file
```

Fungsi ini memeriksa apakah file yang diberikan melalui argumen baris perintah ada dan valid. Jika tidak ada atau bukan file, maka akan menghasilkan pesan kesalahan.

```
parser = argparse.ArgumentParser()
parser.add_argument('-f', '--file', dest='settings_file', type=lambda x: is_valid_file(parser, x), help='settings file', default=default_settings())
args = parser.parse_args()
```

ini dimulai dengan membuat parser argumen menggunakan ArgumentParser untuk menangani argumen yang diberikan saat menjalankan skrip dari baris perintah. Selanjutnya, dengan menggunakan parser.add_argument(), ditambahkan argumen -f atau --file yang berfungsi untuk menerima jalur ke file konfigurasi. Jika argumen ini tidak diberikan, maka fungsi default_settings() akan dipanggil untuk menggunakan konfigurasi default. Setelah itu, args = parser.parse_args() digunakan untuk mengambil argumen yang diberikan dari baris perintah dan menyimpannya dalam variabel args. Terakhir, objek Simulator dibuat dengan menggunakan settings_file yang diterima dari argumen atau default, dan metode run() dipanggil untuk menjalankan simulasi.

- Simulator.py

```
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()
```

Langkah pertama yang dilakukan adalah dengan mengimpor beberapa library dan modul penting, yaitu json yang digunakan untuk memproses file dengan format JSON, Topic yang diimpor dari modul topic untuk mendefinisikan topik yang akan digunakan dalam simulasi, serta dua kelas BrokerSettings dan ClientSettings dari modul data_classes, yang menyimpan pengaturan untuk broker dan klien. Kelas Simulator kemudian didefinisikan untuk menangani logika simulasi. Pada konstruktor `__init__`, objek ClientSettings dibuat dengan pengaturan default, dan fungsi `load_topics` dipanggil untuk memuat topik-topik dari file konfigurasi. Fungsi `load_topics` membuka file JSON yang berisi pengaturan broker, klien, dan berbagai topik yang perlu diproses, yang meliputi tiga jenis topik: single, multiple, dan list. Topik-topik ini kemudian dimasukkan ke dalam daftar topics. Fungsi `read_client_settings` digunakan untuk membaca pengaturan klien dari file konfigurasi dan mengembalikannya sebagai objek ClientSettings. Setelah topik-topik dimuat, fungsi `run` digunakan untuk memulai simulasi dengan menjalankan setiap topik yang ada, sementara fungsi `stop` digunakan untuk menghentikan simulasi dengan memanggil metode `stop()` pada setiap topik. Secara keseluruhan, skrip ini memungkinkan simulasi topik MQTT yang dapat dijalankan dan dihentikan sesuai pengaturan yang dimuat dari file konfigurasi.

- Topic.py

```
import time
import json
import threading
import paho.mqtt.client as mqtt
from data_classes import BrokerSettings, ClientSettings
from topic_data import TopicDataNumber, TopicDataBool, TopicDataRawValue, TopicDataMathExpression

class Topic(threading.Thread):
    def __init__(self, broker_settings: BrokerSettings, topic_url: str, topic_data: list[object], topic_payload_root: object, client_settings: ClientSettings):
        threading.Thread.__init__(self)

        self.broker_settings = broker_settings

        self.topic_url = topic_url
        self.topic_data = self.load_topic_data(topic_data)
        self.topic_payload_root = topic_payload_root

        self.client_settings = client_settings

        self.loop = False
        self.client = None
        self.payload = None

    def load_topic_data(self, topic_data_object):
        topic_data = []
        for data in topic_data_object:
            data_type = data['TYPE']
            if data_type == 'int' or data_type == 'float':
                topic_data.append(TopicDataNumber(data))
            elif data_type == 'bool':
                topic_data.append(TopicDataBool(data))
            elif data_type == 'raw values':
                topic_data.append(TopicDataRawValue(data))
            elif data_type == 'math expression':
                topic_data.append(TopicDataMathExpression(data))
            else:
                raise NameError(f"Data TYPE '{data_type}' is unknown")
        return topic_data

    def connect(self):
        self.loop = True
        clean_session = None if self.broker_settings.protocol == mqtt.MQTTv5 else self.client_settings.clean
        self.client = mqtt.Client(self.topic_url, protocol=self.broker_settings.protocol, clean_session=clean_session)
        self.client.on_publish = self.on_publish

    def disconnect(self):
        self.loop = False
        self.client.loop_stop()
        self.client.disconnect()

    def run(self):
        self.connect()
        while self.loop:
            self.payload = self.generate_payload()
            self.client.publish(topic=self.topic_url, payload=json.dumps(self.payload), qos=self.client_settings.qos, retain=self.client_settings.retain)
            time.sleep(self.client_settings.time_interval)

    def on_publish(self, client, userdata, result):
        print(f"[{time.strftime('%H:%M:%S')}] Data published on: {self.topic_url}")

    def generate_payload(self):
        payload = {}
        payload.update(self.topic_payload_root)
        has_data_active = False
        for data in self.topic_data:
            if data.is_active:
                has_data_active = True
                payload[data.name] = data.generate_value()
        if not has_data_active:
            self.disconnect()
            return
        return payload
```

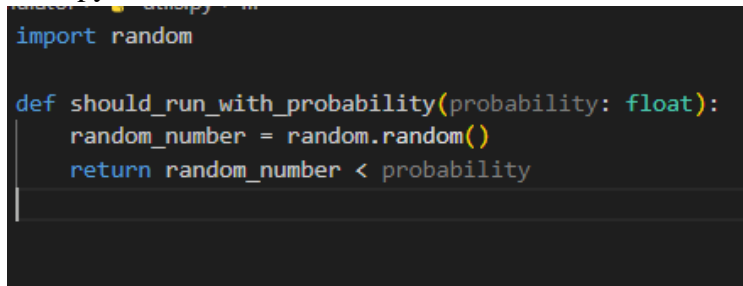
Code ini dimulai dengan mengimpor pustaka dan modul seperti time untuk mengelola jeda waktu, json untuk memproses data dalam format JSON, dan threading untuk menjalankan simulasi secara paralel menggunakan thread. Selain itu, pustaka paho.mqtt.client digunakan untuk berinteraksi dengan broker MQTT, sementara modul lain yang diimpor dari data_classes dan topic_data menyediakan kelas dan struktur data yang dibutuhkan untuk topik-topik MQTT. Selanjutnya di kelas utama didefinisikan Kelas utama Topic, yang merupakan turunan dari threading.Thread. Kelas ini bertanggung jawab untuk menangani setiap topik yang akan dipublikasikan ke broker MQTT. Di dalam konstruktor __init__, berbagai pengaturan dan data terkait topik dimuat, termasuk pengaturan broker (broker_settings), pengaturan klien (client_settings), serta data topik yang diambil melalui metode load_topic_data(). Metode ini memeriksa tipe data untuk setiap entri dalam topic_data dan memuat objek yang sesuai, seperti TopicDataNumber, TopicDataBool, atau TopicDataRawValue, tergantung pada tipe yang ditemukan.

Metode `connect()` digunakan untuk menghubungkan klien MQTT ke broker dengan pengaturan yang diberikan. Klien akan menggunakan protokol MQTT yang ditentukan dalam `broker_settings`, dan sesi bersih (`clean_session`) diatur sesuai dengan pengaturan klien. Setelah terhubung, metode `run()` mulai menjalankan thread, yang dalam loop-nya akan menghasilkan dan mengirim payload secara berkala berdasarkan interval waktu yang ditentukan dalam `client_settings`. Payload yang dikirim merupakan hasil dari metode `generate_payload()`, yang menghasilkan data dari `topic_data` jika data tersebut aktif. Setelah payload dibuat, data tersebut dipublikasikan ke broker menggunakan metode `client.publish()`.

Selain itu, setiap kali data dipublikasikan ke broker, metode `on_publish()` akan dipanggil, yang mencetak waktu dan topik yang telah dipublikasikan. Jika tidak ada data yang aktif untuk dipublikasikan, metode `disconnect()` akan dipanggil untuk menghentikan koneksi. Secara keseluruhan, kelas `Topic` ini memungkinkan simulasi pengiriman data ke broker MQTT dengan pengaturan dan data yang dapat dikonfigurasi, serta penanganan publikasi yang dilakukan dalam thread terpisah untuk memungkinkan proses paralel.

40 mini

- `Utils.py`



```
import random

def should_run_with_probability(probability: float):
    random_number = random.random()
    return random_number < probability
```

Fungsi `should_run_with_probability(probability: float)` dirancang untuk menentukan apakah sebuah proses atau tindakan harus dijalankan berdasarkan probabilitas tertentu. Fungsi ini menerima argumen `probability`, yang merupakan angka desimal antara 0 dan 1, yang mewakili kemungkinan terjadinya suatu peristiwa.

Di dalam fungsi, pertama-tama dibuat sebuah angka acak antara 0 dan 1 menggunakan `random.random()`, yang menghasilkan angka desimal acak dalam rentang `[0, 1)`. Kemudian, fungsi memeriksa apakah angka acak yang dihasilkan lebih kecil daripada nilai `probability` yang diberikan. Jika angka acak tersebut lebih kecil dari probabilitas, maka fungsi akan mengembalikan nilai `True`, yang berarti proses atau tindakan tersebut harus dijalankan. Sebaliknya, jika angka acak lebih besar atau sama dengan probabilitas, maka fungsi akan mengembalikan `False`, yang berarti proses tersebut tidak dijalankan.

Fungsi ini sangat berguna dalam situasi di mana Anda ingin melakukan sesuatu dengan peluang tertentu, misalnya, untuk mensimulasikan kejadian acak, pengambilan sampel, atau pengujian dengan probabilitas tertentu.

